DB2® XML Extender

MQSeries® XML Functions and Stored Procedures, Release Notes

Version 7.2

DB2® XML Extender

MQSeries® XML Functions and Stored Procedures, Release Notes

Version 7.2

Note

Before using this information and the product it supports, read the information in “Appendix. Notices” on page 47.

First Edition (July, 2001)

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

Contents

About this document v
Who should read this book v
Prerequisite and related information v
How to send your comments v

MQSeries XML functions and stored procedures 1
Overview 1
Setting up 2
Software Requirements 2
Installing MQXML 2
Setting up the database for MQXML . . . 3
Disabling the database for MQXML . . . 3
MQSeries XML functions and stored procedures: Reference 4
Functions 4
Stored procedures for MQSeries message queues 25

Appendix. Notices 47
Trademarks 49

About this document

This document describes how to use the new features available in UDB DB2 V7.2 to easily integrate MQSeries messaging with XML Extender applications .The new MQSeries integration features provide support for XML messages using the XML Extender. These new functions provide seamless access to MQSeries messaging within standard SQL statements to support a wide range of applications ranging from simple event notification to operational data store creation.

Who should read this book

This book is intended for the following people:

- People who work with XML data in DB2 applications and who are familiar with XML concepts. Readers of this document should have a general understanding of XML and DB2.
- Application developers that are familiar with MQSeries.
- DB2 database administrators who are familiar with DB2 administration concepts, tools, and techniques.
- DB2 application programmers who are familiar with SQL and with one or more programming languages that can be used for DB2 applications.

Prerequisite and related information

This document refers to and assumes you are familiar with information in the following documents:

- *DB2 XML Extender Administration and Programming, Version 7: ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2sxe70.pdf.*
- *DB2 XML Extender Administration and Programming, Version 7.2 Release Notes: http://www.ibm.com/software/data/db2extenders/xmlext/library.html*
- *MQ Series Web site: http://www.software.ibm.com/MQSeries*
- *IBM DB2 Universal Database Release Version 7.2/Version 7.1 FixPak 3*

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this document or the MQXML product, send your comments by E-mail to db2xml@us.ibm.com. Put ″MQXML ″ in the subject line and be sure to include the name of the

book, the version of DB2 XML Extender, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

MQSeries XML functions and stored procedures

This document describes MQSeries XML functions and stored procedures and is a supplement of the following product documents:

- *IBM DB2 Universal Database® XML Extender Administration and Programming, Version 7 Release 1*
- *IBM DB2 XML Extender Release Notes, Version 7 Release 2*

It is assumed that you are familiar with the concepts in these documents.

XML Extender provides management for XML documents in DB2. It now supports MQSeries, a flexible messaging system that allows applications to communicate in a distributed, heterogeneous environment. XML Extender also provides MQSeries XML functions and stored procedures that allow you to use MQSeries message queues when querying, composing, and decomposing XML documents.

Overview

XML Extender provides two methods of storing and accessing XML data. Using the XML column method, you can store XML documents in a DB2 table while querying, updating, and retrieving the documents contents. The new MQ XML user-defined functions enable you to query XML documents and then publish the results to a message queue. Additionally, you can use the XML collection method to store the untagged contents of an XML document in one or multiple tables or compose XML documents from multiple tables. Using the new MQ XML stored procedures, you can retrieve an XML document from a message queue, decompose it into untagged data and store the data in DB2 tables. Likewise, you can compose an XML document from DB2 data and send the document to MQSeries message queue.

MQSeries supports three messaging models:

datagrams
: Messages are sent to a single destination with no reply expected.

publish/subscribe
: One or more publishers send a message to a publication service which distributes the message to interested subscribers.

request/reply
: Messages are sent to a single destination and the sender expects to receive a response.

You can use these three messaging models to distribute XML data and documents.

MQSeries can be used in numerous ways. Simple datagrams are exchanged to coordinate multiple applications, to exchange information, request services, and to provide notification of interesting events. Publish/subscribe is most often used to disseminate real-time information in a timely manner. The request/reply style is generally used as a simple form of pseudo-synchronous remote procedure call. More complex models can also be constructed by combining these basic styles.

The fundamental messaging techniques described here are used in a wide variety of ways. Because MQSeries is available across a very wide range of operating systems it provides an important mechanism to link together disparate applications, from either similar or dissimilar environments. The MQXML functions and stored procedures provide the ability to send XML documents between disparate applications.

Setting up

This section describes how to install and set up DB2 MQSeries XML user-defined functions (UDFs) and store procedures for use with DB2 XML Extender (referred to as MQXML in this document).

Software Requirements

Ensure that you have the following software installed before using MQXML functions and stored procedures.

- DB2 Universal Database Version 7.2
- DB2 XML Extender Version 7.2
- DB2 MQSeries Functions Version 7.2 (Available as an optional installation feature of DB2 Universal Database V7.2. Installation information is available in the DB2 Universal Database V7.2 Release Notes.)

  MQSeries Publish/Subscribe or MQSeries Integrator when using publishing functions.

Installing MQXML

Use the following steps to install MQXML:

1. Install required DB2 Universal Database and MQSeries Functions software
2. Install DB2 XML Extender 7.2 Web download for the operating system of your choice. The download is available at the following Web address:

   `http://www.ibm.com/software/data/db2/extenders/xmlext/index.html/install.txt`

   See db2xml/<operating_system>/<country> for your operating system and language in the installation package.

3. Use the verification steps provided in the install.txt file to verify that XML Extender has been properly installed and set up.

Setting up the database for MQXML

After you have installed and set up, MQXML, prepare the database to use with MQXML.

Preparing the database for use with MQXML

The following steps should be completed before using a database with MQXML:

1. If a database has not been created, create one to contain the XML documents.
2. Enable the database for DB2 XML Extender. See the documentation at: ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2sxe70.pdf.
3. Enable the database for DB2 MQ Functions, see the documentation at: ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2ire71.pdf chapter ″MQSeries″.
4. Bind the database to MQXML:
   a. Connect to the database by entering:
      ```
      db2 connect to <db_name
      ```
   b. Change the directory to <DXX_INSTALL>\bnd. For example:
      ```
      C:\dxx\bnd
      ```
   c. Enter the bind command:
      ```
      db2 bind mqxml.bnd
      ```
5. Disconnect from database by entering: db2 terminate
6. Run the enable_mqxml command to enable the database for MQXML:
   ```
   enable_MQXML -n db_name -u user_id -p password (-force)
   ```

Examples:

1. This example, shows how to enable MQXML functions and stored procedures. You are connected to the SAMPLE database, using userID ″user1″ and password ″password1″. Any error or warning messages will be printed to the console.
   ```
   enable_MQXML -n sample -u user1 -p password1
   ```
2. This example shows how the *force* option can be used to install or reinstall MQXML functions. No error is generated if the functions have already been installed.
   ```
   enable_MQXML -n sample -u user1 -p password -force
   ```

Disabling the database for MQXML

Enter the disable_MQXML command:

```
disable_MQXML -n db_name -u user_id -p password
```

Example: This example shows how to disable a database for the MQXML functions and store procedures. Any error or warning messages will be printed to the console.

```
disable_MQXML -n sample -u user1 -p password
```

- **Important Notes**

Temporary tablespace will be created with the enable_mqxml command. Ensure that the authorization ID of this command has SYSCTRL or SYSADM authority.

How to get more information on the Web:

See our Web site at:

```
http://www.ibm.com/software/data/db2/extenders/xmlext/index.html
```

The latest documentation can be accessed from:

```
http://www.ibm.com/software/data/db2/extenders/xmlext/library.html
```

MQSeries XML functions and stored procedures: Reference

The following sections describe the functions and stored procedures provided for use with MQSeries.

Functions

This section describes the MQSeries XML functions used with data in XML columns. With these functions you can send, retrieve, publish, and read messages containing XMLVARCHAR data.

Table 1. The MQSeries XML user-defined functions

Function	Purpose
"db2xml.MQReadXML" on page 6	Return a message at the head of a queue without removing it from the queue.
"db2xml.MQReadXMLAll" on page 9	Returns a table containing message data without removing messages from the queue.
"db2xml.MQReceiveXML" on page 12	Return and remove a message from the queue.
"db2xml.MQReceiveAllXML" on page 14	Return and remove message from the queue
"db2xml.MQSENDXML" on page 17	Send a message with no expected reply.
"db2xml.MQSENDXMLFILE" on page 20	Send a message that contains a file with no expected reply.

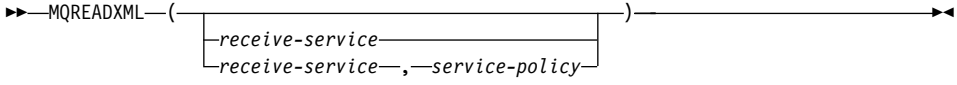
Table 1. The MQSeries XML user-defined functions (continued)

Function	Purpose
“db2xml.MQPUBLISHXML” on page 23	Send message to queue to be picked up by applications that monitor the queue.

db2xml.MQReadXML

Purpose: The MQREADXML function returns XMLVARCHAR data from the MQSeries location specified by *receive-service* using the quality of service policy *service-policy*. Performing this operation does not remove the message from the queue associated with *receive-service*. The message at the head of the queue will be returned. The return value is an XMLVARCHAR containing the messages. If no messages are available to be returned a NULL will be returned.

Format:



Parameters:

Table 2. MQReadXML parameters

Parameter	Data type	Description
receive-service	VARCHAR(48)	A string containing the logical MQSeries destination from which the message is to be received. If specified, the *receive-service* refers to a Service Point defined in the AMT.XML repository file. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes

Table 2. MQReadXML parameters (continued)

Parameter	Data type	Description
service-policy	VARCHAR(48)	A string containing the MQSeries AMI Service Policy used in the handling of this message. When the *service policy* is specified, it refers to a Policy defined in the AMT.XML repository file. A Service Policy defines a set of quality of service options that are applied to the messaging operation. These options include message priority and message persistence. If the *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of service-policy is 48 bytes.

Results: When a message in the queue has been read successfully, MQREADXML returns a db2xml.xmlvarchar. A NULL is returned if no messages are available.

Examples: Example 1: This example reads the message at the head of the queue specified by the default service (DB2.DEFAULT.SERVICE) using the default policy (DB2.DEFAULT.POLICY).

```
values MQREADXML()
```

This example returns the contents of the message as an XMLVARCHAR if successful. If no messages are available a NULL will be returned.

Example 2: This example reads the message at the head of the queue specified by the service MYSERVICE using the default policy (DB2.DEFAULT.POLICY).

```
values MQREADXML('MYSERVICE')
```

This example returns the contents of the message as an XMLVARCHAR if successful. If no messages are available a NULL will be returned.

Example 3: This example reads the message at the head of the queue specified by the service MYSERVICE using the policy MYPOLICY.

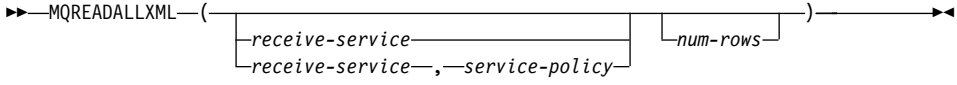
```
values MQREADXML('MYSERVICE','MYPOLICY')
```

This example returns the contents of the message as an XMLVARCHAR if successful. If no messages are available a NULL will be returned.

db2xml.MQReadXMLAll

Purpose: The MQREADALLXML function returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service* using the quality of service policy service-policy. Performing this operation does not remove the messages from the queue associated with receive-service. If *num-rows* is specified, then a maximum of num-rows messages will be returned. If num-rows is not specified then all available messages will be returned.

Format:



Parameters:

Table 3. MQReadXMLAll parameters

Parameter	Data type	Description
receive-service	VARCHAR(48)	A string containing the logical MQSeries destination from which the message is to be read. If specified, the*receive-service* must refer to a Service Point defined in the AMT.XML repository file. However, if *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes. For more information on *receive-service*, see the MQSeries Application Messaging Interface .

Table 3. MQReadXMLAll parameters (continued)

Parameter	Data type	Description
service-policy	VARCHAR(48)	A string containing the MQSeries AMI Service Policy used in the handling of this message. When the *Service policy* is specified, it refers to a Policy defined in the AMT.XML repository file. The maximum size of service-policy is 48 bytes. For additional information, refer to the MQSeries Application Messaging Interface manual.
num-rows	INTEGER	A positive integer containing the maximum number of messages to be returned by the function.

Results: The MQREADALLXML function returns a table containing messages and message metadata as described below.

Table 4. Result set table

Column Name	Data Type	Description
MSG	XMLVARCHAR	The contents of the MQSeries message.
CORRELID	VARCHAR(24)	A correlation ID that can be used to relate messages.
TOPIC	VARCHAR(40)	If the topic the message was published with, if available.
QNAME	VARCHAR(48)	The queue name the message was received at
MSGID	VARCHAR(24)	The MQSeries assigned unique identifier for this message
MSGFORMAT	VARCHAR(8)	The format of the message as defined by MQSeries. Typical strings have a format of MQSTR.

Examples: Example 1: All the messages from the queue that are specified by the default service (DB2.DEFAULT.SERVICE) are read using the default policy (DB2.DEFAULT.POLICY). The messages and all the metadata return in table format.

```
select * from table (MQREADALLXML()) t
```

Example 2: Messages from the head of the queue are specified by the service MYSERVICE by using the default policy (DB2.DEFAULT.POLICY). Only the *msg* and *correlid* columns are returned.

```
select t.MSG, t.CORRELID from table (MQREADALLXML('MYSERVICE')) t
```

Example 3: The head of the queue specified by the default service (DB2.DEFAULT.SERVICE) is read using the default policy (DB2.DEFAULT.POLICY) . Only messages with a *CORRELID* of '1234' are returned. All columns are returned.

```
select * from table (MQREADALLXML()) t where t.CORRELID = '1234'
```

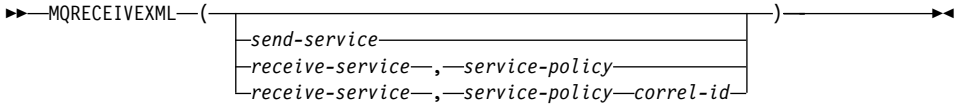
Example 4: The first 10 message from the head of the queue specified by the default service (DB2.DEFAULT.SERVICE) are read using the default policy (DB2.DEFAULT.POLICY) . All columns are returned.

```
select * from table (MQREADALLXML(10)) t
```

db2xml.MQReceiveXML

Purpose: The MQRECEIVEXML removes messages associated with *receive-service* from the queue.The function returns XMLVARCHAR data from the MQSeries location specified by the *receive-service* function which uses the quality of *service-policy*.

Format:



Parameters:

Table 5. MQReceiveXML parameters

Parameter	Data type	Description
receive-service	VARCHAR(48)	A string containing the logical MQSeries destination from which the message is to be received. When the *receive-service* is specified, it refers to a Service Point defined in the AMT.XML repository file. However, if *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.
service-policy	VARCHAR(48)	A string containing the MQSeries AMI Service Policy to be used in handling of this message. If specified, the service-policy must refer to a Policy defined in the AMT.XML repository file. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

Results: MQRECEIVEXML functions return a db2xml.XMLVARCHAR if messages are received from the queue successfully. A NULL is returned if no messages are available.If the *correl-id* is specified then the first message with a matching correlation identifier will be returned. If *correl-id* is not specified then the message at the head of the queue will be returned.

Examples: Example 1: This example receives the message at the head of the queue specified by the default service (DB2.DEFAULT.SERVICE) using the default policy (DB2.DEFAULT.POLICY).

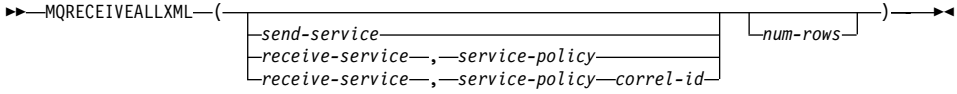
```
values MQRECEIVEXML()
```

This example returns the contents of the message as an XMLVARCHAR if successful. If no messages are available a NULL is returned.

db2xml.MQReceiveAllXML

Purpose: The MQRECEIVEALLXML removes the messages from the queue associated with *receive-service*. If thecorrel-id is specified then only those messages with a matching correlation identifier will be returned. If *correl-id* is not specified then the message at the head of the queue will be returned. If *num-rows* are specified, then a maximum of *num-rows* messages will be returned. If it is not specified then all available messages will be returned.

Format:



Parameters:

Table 6. MQReceiveAllXML parameters

Parameter	Data type	Description
receive-service	VARCHAR(512)	A string containing the logical MQSeries destination from which the message is to be received. If specified, the *receive-service*refers to a Service Point defined in the AMT.XML repository file. But, if *receive-service*is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of receive-service is 48 bytes.
service-policy	VARCHAR(48)	A string containing the MQSeries AMI Service Policy used to handle this message. The service-policy when specified, refers to a policy defined in the AMT.XML repository file. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

Table 6. MQReceiveAllXML parameters (continued)

Parameter	Data type	Description
correl-id	VARCHAR(24)	A string containing an optional correlation identifier to be associated with this message. The *correl-id* is often specified in request/reply scenarios to associate requests with replies. If it is not outlined no correlation id will be specified. The maximum size of *correl-id* is 24 bytes.
num-rows	INTEGER	A positive integer that contains the maximum number of messages returned by the function.

Results: When a message is successfully received from the queue, MQRECEIVEXML returns a db2xml.xmlvarchar. A NULL is returned when no messages are available. The messages are returned in a table as described below

Column Name	Data Type	Description
MSG	XMLVARCHAR	The contents of the MQSeries message.
CORRELID	VARCHAR(24)	A correlation ID that can be used to relate messages.
TOPIC	VARCHAR(40)	If the topic the message was published with, if available.
QNAME	VARCHAR(48)	The queue name the message was received at.
MSGID	CHAR(24)	The MQSeries assigned unique identifier for this message
MSGFORMAT	VARCHAR(8)	The format of the message as defined by MQSeries. Typical strings have a format of MQSTR.

Examples: Example 1: All messages received from the queue are specified by the default service (DB2.DEFAULT.SERVICE) using the default policy (DB2.DEFAULT.POLICY). The messages and all the metadata are returned as a table.

```
select * from table (MQRECEIVEALLXML()) t
```

Example 2: All the messages are received from the head of the queue and are specified by the service MYSERVICE using the default policy (DB2.DEFAULT.POLICY). Only the MSG and CORRELID columns are returned.

```
select t.MSG, t.CORRELID from table (MQRECEIVEALLXML('MYSERVICE')) t
```

Example 3: All the messages received from the head of the queue are specified by the service MYSERVICE using the policy MYPOLICY that match the correlation id '1234'. Only the MSG and CORRELID columns are returned.

```
select t.MSG, t.CORRELID from table
(MQRECEIVEALLXML('MYSERVICE','MYPOLICY','1234')) t
```

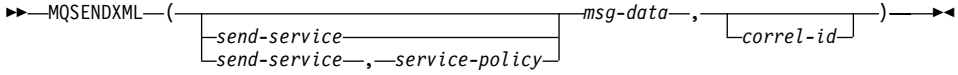
Example 4: The first 10 messages are received from the head of the queue and specified by the default service (DB2.DEFAULT.SERVICE) using the default policy (DB2.DEFAULT.POLICY) . All columns are returned.

```
 select * from table (MQRECEIVEALLXML(10)) t
```

db2xml.MQSENDXML

Purpose: The MQSENDXML function sends the data contained in msg-data to the MQSeries location specified by send-service using the service-policy. An optional user defined message correlation identifier may also be specified by correl-id. The function returns a '1' if successful.

Format:



Parameters:

Table 7. MQSENDXML parameters

Parameter	Data type	Description
msg-data	XMLVARCHAR	An XMLVARCHAR expression containing the data to be sent via MQSeries.
send-service	VARCHAR(48)	A string containing the logical MQSeries destination to which the message is to be sent. When the *send-service* is listed, it refers to a Service Point defined in the AMT.XML repository file. The DB2.DEFAULT.SERVICE is used when the *send-service* is not specified. The maximum size of *send-service* is 48 bytes.
service-policy	VARCHAR(48)	A string containing the MQSeries AMI Service Policy used to handle the message. When specified, the *service-policy* refers to a policy defined in the AMT.XML repository file. If the *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of service-policy is 48 bytes.

Table 7. MQSENDXML parameters (continued)

Parameter	Data type	Description
correl-id	VARCHAR(24)	A string containing an optional correlation identifier associated with the message. The correl-id is often specified in request/reply scenarios to associate requests with replies. If it is not specified, no correlation id will be shown. The maximum size of correl-id is 24 bytes.

Results: A successful message result in a value of '1'. The side effect of successfully executing this function is that a message containing msg-data will be sent to the location specified by send-service using the policy defined by service-policy.

Examples: Example 1: This example sends all of the XML documents contained in the order column of the order_tab table to the default service (DB2.DEFAULT.SERVICE) using the default policy (DB2.DEFAULT.POLICY). No correlation identifier is used.

```
Select MQSENDXML(order) from order_tab
```

If successful, this example returns the value '1'.

Example 2: This example sends all of the XML documents contained in the order column of the order_tab table to the service MYSERVICE using policy MYPOLICY with no correlation identifier

```
Select MQSENDXML('MYSERVICE', 'MYPOLICY', order) from order_tab
```

This example returns the value '1' if successful.

Example 3: This example sends the XML document for Midwestern customers from the customer field of the customer_tab table to the service MYSERVICE using policy MYPOLICY with correlation identifier "Midwestern".

```
Select MQSENDXML('MYSERVICE','MYPOLICY',customer,'MidWestern')From customer_tab where ter
```

This example returns the value '1' if successful.

Example 4: This example sends all of the XML documents contained in the order column of the order_tab table to the service MYSERVICE using the default policy (DB2.DEFAULT.POLICY) and no correlation identifier.

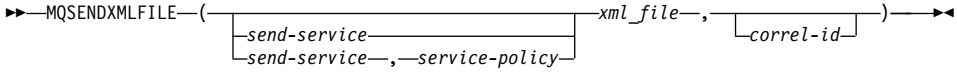
```
Select MQSENDXML('MYSERVICE',order) from order_tab;
```

This example returns the value '1' if successful

db2xml.MQSENDXMLFILE

Purpose: The MQSENDXMLFILE function sends the data contained in *xml_file* to the MQSeries location specified by send-service using the quality of service policy. An optional user defined message correlation identifier may be specified by correl-id. The function returns a '1' if successful.

Format:



Parameters:

Table 8. MQSENDXMLFILE parameter

Parameter	Data type	Description
xml_file	VARCHAR(80)	An XML file name with a maximum size of 48 bytes. The file contains the data to be sent via MQSeries.
send-service	VARCHAR(48)	A string containing the logical MQSeries destination to which the message is to be sent. When specified, the send-service refers to a Service Point defined in the AMT.XML repository file. . If *send-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of send-service is 48 bytes
service-policy	VARCHAR(48)	A string containing the MQSeries AMI service to be used in handling of this message. If specified, the *service-policy* refers to a Policy defined in the AMT.XML repository file. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes

Table 8. MQSENDXMLFILE parameter (continued)

Parameter	Data type	Description
correl-id	VARCHAR(24)	A string containing an optional correlation identifier to be associated with this message. The *correl-id* is often specified in request/reply scenarios to associate requests with replies. If not specified, no correlation id will be listed. The maximum size of *correl-id* is 24 bytes.

Results: If the function is successful, it results in a '1'. The side effect of successfully executing this function is that a message containing *msg-data* will be sent to the location specified by *send-service* using the policy defined by *service-policy*.

Examples: Example 1: XML documents contained in file ″c:\xml\test1.xml″ are sent to the default service (DB2.DEFAULT.SERVICE) using the default policy (DB2.DEFAULT.POLICY) with no correlation identifier.

```
Values MQSENDXMLFILE('c:\xml\test1.xml');
```

This example returns the value '1' if successful

Example 2: XML documents contained in file ″c:\xml\test2.xml″ are sent to the service MYSERVICE using policy MYPOLICY with no correlation identifier.

```
Values MQSENDXMLFILE('MYSERVICE', 'MYPOLICY', 'c:\xml\test2.xml');
```

This example returns the value '1' if successful

Example 3: XML documents contained in file ″c:\xml\test3.xml″are sent to the service MYSERVICE using policy MYPOLICY with correlation identifier ″Test3″.

```
Values MQSENDXML('MYSERVICE','MYPOLICY', 'c:\xml\test3.xml', 'Test3');
```

This example returns the value '1' if successful.

Example 4: XML documents contained in file ″c:\xml\test4.xml″ are sent to the service MYSERVICE using the default policy (DB2.DEFAULT.POLICY) and no correlation identifier.

```
Values MQSENDXMLFILE('MYSERVICE', 'c:\xml\test4.xml');
```

This example returns the value '1' if successful.

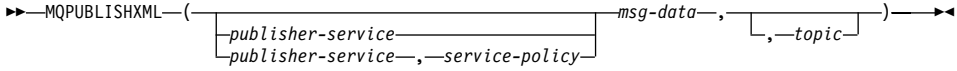
db2xml.MQPUBLISHXML

Purpose: The MQPUBLISHXML function publishes XMLVARCHAR data to MQSeries. This function requires the installation of either MQSeries Publish/Subscribe or MQSeries Integrator. See the following Web site for more information:

`http://www.software.ibm.com/MQSeries`

The MQPUBLISHXML function publishes the XML data contained in *msg-data* to the MQSeries publisher specified by *publisher-service* using the quality of service policy *service-policy*. The topic of the message is optionally specified by *topic*. An optional user defined message correlation identifier may be specified by *correl-id*. The function returns a '1' if successful.

Format:



Parameters:

Table 9. MQPUBLISHXML parameters

Parameter	Data type	Description
publisher-service	VARCHAR(48)	A string containing the logical MQSeries destination to which the message is to be sent. When specified, the *publisher-service* refers to a publisher Service Point defined in the AMT.XML repository file. If the*publisher-service* is not specified, then the DB2.DEFAULT.PUBLISHER will be used. The maximum size of publisher-service is 48 bytes.

Table 9. MQPUBLISHXML parameters (continued)

Parameter	Data type	Description
service-policy	VARCHAR(48)	A string containing the MQSeries AMI*service policy* to be used in handling this message. If specified, the *service-policy* refers to a policy which is defined in the AMT.XML repository file. The Service Policy also defines a set of quality of service options that should be applied to the messaging operation options. These options include message priority and message persistence the*service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of service-policy is 48 bytes. For more information, see the MQSeries Application Messaging Interface.
msg-data	XMLVARCHAR	An XMLVARCHAR expression containing the data to be sent via MQSeries.
topic	VARCHAR(40)	A string containing the topic that the message is to be published under. If no topic is specified, none will be associated with the message. The maximum size of topic is 40 bytes. Multiple topics may be listed within a topic string by separating each topic by ″:″.

Results: If successful, MQPUBLISHXML functions return a ’1’. A value of ’0’ is returned if the function is unsuccessful.

Examples: Example 1: All of the XML documents contained in the *order* column of the *order_tab* table are published by the default publisher service

(DB2.DEFAULT.PUBLISHER) using the default policy (DB2.DEFAULT.POLICY) with no correlation identifier. No topic is specified for the message.

```
Select MQPUBLISHXML(order) from order_tab
```

This example returns the value '1' if successful.

Example 2: XML documents are published for Midwestern customers from the customer field of the *customer_tab* table to the publisher service MYPUBLISHER under the topic "/Midwest/Customers". The default policy is used and no correlation identifier is specified.

```
Select MQPUBLISHXML('MYPUBLISHER',customer,'/Midwest/Customers') from customer_tab where
```

This example returns the value '1' if successful.

Stored procedures for MQSeries message queues

Table 10. The MQSeries XML user-defined functions

Function	Purpose
"db2xml.dxxmqGen()" on page 27	Invoke the dxxmqGen stored procedure to compose XML documents, using a DAD file as a input parameter.
"db2xml.dxxmqRetrieve()" on page 31	Invoke the dxxmqRetrieve stored procedure to compose XML documents, using a collection name as a input parameter.
"dxxmqShred()" on page 35	Invoke the dxxmqShred stored procedure to decompose an XML document using a DAD file as an input parameter.
"dxxmqShredAll()" on page 37	Invoke the dxxmqShredAll stored procedure to decompose multiple XML documents using a DAD file as an input parameter.
"dxxmqInsert()" on page 40	Invoke the dxxmqInsert stored procedure to decompose an XML document using a collection name as an input parameter.
"dxxmqInsertAll()" on page 43	Invoke the dxxmqInsertAll stored procedure to decompose multiple XML documents using a collection name as an input parameter.

Composition stored procedures for message queues

The composition stored procedures dxxmqGen() and dxxmqRetrieve() are used to generate XML documents using data in existing database tables and to send the generated XML documents to a message queue. The dxxmqGen()

stored procedure takes a DAD file as input; it does not require an enabled XML collection. The dxxmqRetrieve() stored procedure takes an enabled XML collection name as input.

- dxxmqGen
- dxxmqRetrieve

db2xml.dxxmqGen():

Purpose: Constructs an XML document from data that is stored in the XML collection tables specified in the DAD file, and sends the XML document to a MQ message queue. The stored procedure returns a string to indicate the status of the stored procedure.

To support dynamic query, dxxmqGen() takes an input parameter, *override*. Based on the input *overrideType*, the application can override the SQL_stmt for SQL mapping or the conditions in RDB_node for RDB_node mapping in the DAD file. The input parameter *overrideType* is used to clarify the type of the *override*. For details about the *override* parameter, see 29.

Format:

```
dxxmqGen(varchar(48)    serviceName,     /*input*/
         varchar(48)    policyName,      /*input*/
         varchar(80)    dadFileName,     /*input*/
         integer        overrideType,    /*input*/
         varchar(1024)  override,        /*input*/
         integer        maxRows          /*input*/
         integer        numRows,         /*output*/
         char(20)       status)          /*output*/
```

Parameters:

Table 11. dxxmqGen() parameters

Parameter	Description	IN/OUT parameter
serviceName	A string containing the logical MQSeries destination to which the message is to be sent. When the *serviceName* is listed, it refers to a service point defined in the AMT.XML repository file. The DB2.DEFAULT.SERIVCE is used when the *serviceName* is not specified. The maximum size of *serviceName* is 48 bytes.	IN
policyName	A string containing the MQSeries AMI Service Policy used to handle messages. When specified, the *policyName* refers to a policy defined in the AMT.XML repository file. If the *policyName* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *policyName* is 48 bytes.	IN
dadFileName	The name of the DAD file.	IN
overrideType	A flag to indicate the type of the following *override* parameter: • **NO_OVERRIDE**: No override. • **SQL_OVERRIDE**: Override by an SQL_stmt. • **XML_OVERRIDE**: Override by an XPath-based condition.	IN

Table 11. dxxmqGen() parameters (continued)

Parameter	Description	IN/OUT parameter
override	Overrides the condition in the DAD file. The input value is based on the *overrideType*. • **NO_OVERRIDE**: A NULL string. • **SQL_OVERRIDE**: A valid SQL statement. Using this *overrideType* requires that SQL mapping is used in the DAD file. The input SQL statement overrides the SQL_stmt in the DAD file. • **XML_OVERRIDE**: A string that contains one or more expressions in double quotation marks separated by ″AND″. Using this *overrideType* requires that RDB_node mapping is used in the DAD file.	IN
maxRows	The maximum number of rows in the result table.	IN
numRows	The actual number generated rows in the result table.	OUT
status	The text and codes returned that specify whether or not the stored procedure ran successfully, any error codes that are generated, and the number of XML documents which are received or sent to the message queue.	OUT

Examples: The following example fragment generates an XML document and sent it to the queue. It assumes that a MQ/AMI service, *myService*, and a policy, *myPolicy*, have been defined in the repository file. This file stores repository definitions in XML format.

```
#include "dxx.h"
#include "dxxrc.h"
               EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
               char           serviceName[48];  /* name of the MQ/AMI service*/
               char           policyName[48];   /* name of the MQ/AMI policy*/
               char           dadFileName[80];  /* name of the DAD file */
               char           override[2];      /* override, will set to NULL*/
               short          overrideType;     /* defined in dxx.h */
               short          max_row;          /* maximum number of rows */
               short          num_row;          /* actual number of rows */
               char           status[20]        /* status code or message */
```

```
 short          ovtype_ind;
 short          ov_inde;
 short          maxrow_ind;
 short          numrow_ind;
 short          dadFileName_ind;
 short          serviceName_ind;
 short          policyName_ind;
 short          status_ind;

 EXEC SQL END DECLARE SECTION;
 strcpy(dadFileName,"e/dxx/dad/litem3.dad");
 strcpy(serviceName,"myService");
 strcpy(policyName,"myPolicy");
 override[0] = '\0';
 overrideType = NO_OVERRIDE;
 max_row = 500;
 num_row = 0;
 status[0] = '/0';
 dadFileName_ind = 0;
 serviceName_ind = 0;
 policyName_ind = 0;
 maxrow_ind = 0;
 numrow_ind = -1;
 status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqGen(:serviceName:serviceName_ind;
       :policyName:policyName_ind,
       :dadFileName:dadFileName_ind,
       :overrideType:ovtype_ind,
       :override:ov_ind
       :max_row:maxrow_ind,
       :num_row:numrow_ind,
       :status:status_ind);
```

db2xml.dxxmqRetrieve():

Purpose: Enables the same DAD file to be used for both composition and decomposition. The stored procedure dxxmqRetrieve() also serves as a means for retrieving decomposed XML documents. As input, dxxmqRetrieve() takes a buffer containing the enabled XML collection name, the MQ/AMI service and policy names. It sends the composed XML document to a MQ Queue; it returns the number of rows sent to the queue and a status message.

To support dynamic query, dxxmqRetrieve() takes an input parameter, *override*. Based on the input *overrideType*, the application can override the SQL_stmt for SQL mapping or the conditions in RDB_node for RDB_node mapping in the DAD file. The input parameter *overrideType* is used to clarify the type of the *override*. For details about the *override* parameter, see 29.

The requirements of the DAD file for dxxmqRetrieve() are the same as the requirements for dxxmqGen(). The only difference is that the DAD is not an input parameter for dxxmqRetrieve(); the required parameter is instead the name of an enabled XML collection.

Format:

```
dxxmqRetrieve(varchar(48)    serviceName,        /*input*/
              varchar(48)    policyName,         /*input*/
              varchar(80)    collectionName,     /*input*/
              integer        overrideType,       /*input*/
              varchar(1024)  override,           /*input*/
              integer        maxrows,            /*input*/
              integer        numrows,            /*output*/
              char(20)       status)             /*output*/
```

Parameters:

Table 12. dxxmqRetrieve() parameters

Parameter	Description	IN/OUT parameter
serviceName	A string containing the logical MQSeries destination to which the message is to be sent. When the *serviceName* is listed, it refers to a Service Point defined in the AMT.XML repository file. The DB2.DEFAULT.SERVICE is used when the *serviceName* is not specified. The maximum size of *serviceName* is 48 bytes.	IN
policyName	A string containing the MQSeries AMI Service Policy used to handle messages. When specified, the *policyName* refers to a policy defined in the AMT.XML repository file. If the *policyName* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *policyName* is 48 bytes.	IN
collectionName	The name of an enabled collection.	IN
overrideType	A flag to indicate the type of the following *override* parameter: • **NO_OVERRIDE**: No override. • **SQL_OVERRIDE**: Override by an SQL_stmt. • **XML_OVERRIDE**: Override by an XPath-based condition.	IN

Table 12. dxxmqRetrieve() parameters (continued)

Parameter	Description	IN/OUT parameter
override	Overrides the condition in the DAD file. The input value is based on the *overrideType*. • **NO_OVERRIDE**: A NULL string. • **SQL_OVERRIDE**: A valid SQL statement. Using this *overrideType* requires that SQL mapping is used in the DAD file. The input SQL statement overrides the SQL_stmt in the DAD file. • **XML_OVERRIDE**: A string that contains one or more expressions in double quotation marks separated by ″AND″. Using this *overrideType* requires that RDB_node mapping is used in the DAD file.	IN
maxRows	The maximum number of rows in the result table.	IN
numRows	The actual number generated rows in the result table.	OUT
status	The text and codes returned that specify whether or not the stored procedure ran successfully, any error codes that are generated, and the number of XML documents which are received or sent to the message queue.	OUT

Examples: The following fragment is an example of a call to dxxmqRetrieve().

```
#include "dxx.h"
#include "dxxrc.h"
               EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
               char           serviceName[48];      /* name of the MQ/AMI service*/
               char           policyName[48];       /* name of the MQ/AMI policy*/
               char           collection[32];   /* name of the XML collection */
               char           override[2];          /* override, will set to NULL*/
               short          overrideType;         /* defined in dxx.h */
               short          max_row;              /* maximum number of rows */
               short          num_row;              /* actual number of rows */
               char           status[20]            /* status code or message */
               short          ovtype_ind;
               short          ov_inde;
               short          maxrow_ind;
               short          numrow_ind;
```

```
short          collection_ind;
short          serviceName_ind;
short          policyName_ind;
short          status_ind;

EXEC SQL END DECLARE SECTION;

/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(serviceName,"myService");
strcpy(policyName,"myPolicy");
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
status[0] = '\0';
serviceName_ind = 0;
policyName_ind = 0;
collection_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqRetrieve(:serviceName:serviceName_ind;
       :policyName:policyName_ind,
       :collection:collection_ind,
       :overrideType:ovtype_ind,
       :override:ov_ind,
       :max_row:maxrow_ind,
       :num_row:numrow_ind,
       :status:status_ind) ;
```

Decomposition stored procedures for message queues

The decomposition stored procedures dxxmqInsert(), dxxmqInsertAll(), dxxmqShred() and dxxmqShredAll() are used to break down or shred incoming XML documents from a message queue, and to store the data in new or existing database tables. The dxxmqInsert() and dxxmqInsertAll() stored procedures take an enabled XML collection name as input. The dxxmqShred() and dxxmqShredAll() stored procedures take a DAD file as input; they do not require an enabled XML collection.

- dxxmqShred
- dxxmqShredAll
- dxxmqInsert
- dxxmqInsertAll

dxxmqShred():

Purpose: Decomposes an incoming XML document from a message queue, based on a DAD file mapping, and stores the content of the XML elements and attributes in specified DB2 tables.

In order for dxxmqShred() to work, all tables specified in the DAD file must exist, and all columns and their data types that are specified in the DAD must be consistent with the existing tables. The stored procedure requires that the columns specified in the join condition, in the DAD, correspond to primary-foreign key relationships in the existing tables. The join condition columns that are specified in the RDB_node of the root element_node must exist in the tables.

Format:

```
dxxmqShred(varchar(48)   servicName,     /* input */
           varchar(48)   policyName,     /* input */
           varchar(80)   dadFileName,    /* input */
           varchar(10)   status)         /* output */
```

Parameters:

Table 13. dxxmqShred() parameters

Parameter	Description	IN/OUT parameter
serviceName	A string containing the logical MQSeries destination to which the message is to be sent. When the *serviceName* is listed, it refers to a Service Point defined in the AMT.XML repository file. The DB2.DEFAULT.SERVICE is used when the *serviceName* is not specified. The maximum size of *serviceName* is 48 bytes.	IN
policyName	A string containing the MQSeries AMI Service Policy used to handle messages. When specified, the *policyName* refers to a policy defined in the AMT.XML repository file. If the *policyName* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *policyName* is 48 bytes.	IN
dadFileName	The name of the DAD file.	IN

Table 13. dxxmqShred() parameters (continued)

Parameter	Description	IN/OUT parameter
status	The text and codes returned that specify whether or not the stored procedure ran successfully, any error codes that are generated, and the number of XML documents which are received or sent to the message queue.	OUT

Examples: The following fragment is an example of a call to dxxmqShred().

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char        serviceName[48];    /* name of the MQ/AMI service */
              char        policyName[48];     /* name of the MQ/AMI policy */
              char        dadFileName[80];    /* name of the DAD file */
              char        status[20];         /* status code or message */
              short       serviceName_ind;
              short       policyName_ind;
              short       dadFileName_ind;
              short       status_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(dadFileName,"e:/dxx/samples/dad/getstart_xcollection.dad");
              strcpy(serviceName, "myService");
              strcpy(policyName, "myPolicy");
              status[0] =\0;
              serviceName_ind=0;
              policyName_ind=0;
              dadFileName_ind=0;
              status_ind=-1;

              /* Call the store procedure */
              EXEC SQL CALL dxxmqShred(:serviceName:serviceName_ind,
                               :policyName:policyName_ind,
                               :dadFileName:dadFileName_ind,
                               :status:status_ind);
```

dxxmqShredAll():

Purpose: Decomposes all incoming XML documents from a message queue, based on a DAD file mapping, and stores the content of the XML elements and attributes in specified DB2 tables.

In order for dxxmqShred() to work, all tables specified in the DAD file must exist, and all columns and their data types that are specified in the DAD must be consistent with the existing tables. The stored procedure requires that the columns specified in the join condition, in the DAD, correspond to primary-foreign key relationships in the existing tables. The join condition columns that are specified in the RDB_node of the root element_node must exist in the tables.

Format:

```
dxxmqShredAll(varchar(48)   serviceName,     /* input */
              varchar(48)   policyName,     /* input */
              varchar(80)   dadFileName,    /* input */
              varchar(20)   status)         /* output */
```

Parameters:

Table 14. dxxmqShredAll() parameters

Parameter	Description	IN/OUT parameter
serviceName	A string containing the logical MQSeries destination to which the message is to be sent. When the *serviceName* is listed, it refers to a Service Point defined in the AMT.XML repository file. The DB2.DEFAULT.SERVICE is used when the *serviceName* is not specified. The maximum size of *serviceName* is 48 bytes.	IN
policyName	A string containing the MQSeries AMI Service Policy used to handle messages. When specified, the *policyName* refers to a policy defined in the AMT.XML repository file. If the *policyName* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *policyName* is 48 bytes.	IN
dadFileName	The name of the DAD file.	IN
status	The text and codes returned that specify whether or not the stored procedure ran successfully, any error codes that are generated, and the number of XML documents which are received or sent to the message queue.	OUT

Examples: The following fragment is an example of a call to dxxmqShredAll().

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char        serviceName[48];    /* name of the MQ/AMI service */
              char        policyName[48];     /* name of the MQ/AMI policy */
              char        dadFileName[80];    /* name of the DAD file */
              char        status[20];         /* status code or message */
              short       serviceName_ind;
              short       policyName_ind;
              short       dadFileName_ind;
              short       status_ind;
              EXEC SQL END DECLARE SECTION;
```

```
/* initialize host variable and indicators */
strcpy(dadFileName,"e:/dxx/samples/dad/getstart_xcollection.dad");
strcpy(serviceName, "myService");
strcpy(policyName, "myPolicy");
status[0] = \0;
serviceName_ind=0;
policyName_ind=0;
dadFileName_ind=0;
status_ind=-1;

/* Call the store procedure */
EXEC SQL CALL dxxmqShredAll(:serviceName:serviceName_ind,
                  :policyName:policyName_ind,
                  :dadFileName:dadFileName_ind,
                  :status:status_ind);
```

dxxmqInsert():

Purpose: Breaks down or shreds an incoming XML document from a message queue, and stores the data in new or existing database tables. dxxmqInsert uses a collection name, rather than a DAD file name, to determine how to store the data.

Format:

```
dxxmqInsert(varchar(48)  serviceName,       /* input */
            varchar(48)   policyName,      /* input */
            varchar(80)   collectionName,  /* input */
            varchar(20)   status)          /* output */
```

Parameters:

Table 15. dxxmqInsert() parameters

Parameter	Description	IN/OUT parameter
serviceName	A string containing the logical MQSeries destination to which the message is to be sent. When the *serviceName* is listed, it refers to a Service Point defined in the AMT.XML repository file. The DB2.DEFAULT.SERVICE is used when the *serviceName* is not specified. The maximum size of *serviceName* is 48 bytes.	IN
policyName	A string containing the MQSeries AMI Service Policy used to handle messages. When specified, the *policyName* refers to a policy defined in the AMT.XML repository file. If the *policyName* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *policyName* is 48 bytes.	IN
collectionName	The name of an enabled XML collection.	IN
status	The text and codes returned that specify whether or not the stored procedure ran successfully, any error codes that are generated, and the number of XML documents which are received or sent to the message queue.	OUT

Examples: In the following fragment example, the dxxmqInsert() call retrieves the input XML document `order1.xml` from a message queue defined by *serviceName*, decomposes the document, and inserts data into the SALES_ORDER collection tables according to the mapping that is specified in the DAD file with which it was enabled.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char          serviceName[48];    /* name of an XML collection */
              char          policyName[48];     /* name of an XML collection */
              char          collection[48];     /* name of an XML collection */
```

```
char          status[10];         /* name of an XML collection */

short                 serviceName_ind;
short                 policyName_ind;
short                 collection_ind;
short                 status_ind;
EXEC SQL END DECLARE SECTION;

/* initialize host variable and indicators */
strcpy(serviceName, "myService");
strcpy(policyName, "myPolicy");
strcpy(collection,"sales_ord")
status[0] = \0;
serviceName_ind = 0;
policyName_ind = 0;
collection_ind = 0;
status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqInsert(:serviceName:serviceName_ind;
                :policyName:policyName_ind,
                :collection:collection_ind,
                :status:status_ind);
```

dxxmqInsertAll():

Purpose: Breaks down or shreds all incoming XML documents from a message queue, and stores the data in new or existing database tables. dxxmqInsert uses a collection name, rather than a DAD file name, to determine how to store the data.

Format:

```
dxxmqInsertAll(varchar(48)  serviceName,       /* input */
               varchar(48)   policyName,      /* input */
               varchar(48)   collectionName,  /* input */
               varchar(20)   status)          /* output */
```

Parameters:

Table 16. dxxmqInsertAll() parameters

Parameter	Description	IN/OUT parameter
serviceName	A string containing the logical MQSeries destination to which the message is to be sent. When the *serviceName* is listed, it refers to a Service Point defined in the AMT.XML repository file. The DB2.DEFAULT.SERVICE is used when the *serviceName* is not specified. The maximum size of *serviceName* is 48 bytes.	IN
policyName	A string containing the MQSeries AMI Service Policy used to handle messages. When specified, the *policyName* refers to a policy defined in the AMT.XML repository file. If the *policyName* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *policyName* is 48 bytes.	IN
collectionName	The name of an enabled XML collection.	IN
status	The text and codes returned that specify whether or not the stored procedure ran successfully, any error codes that are generated, and the number of XML documents which are received or sent to the message queue.	OUT

Examples: In the following fragment example, the dxxmqInsert() call retrieves all input XML documents from a message queue defined by *serviceName*, decomposes the documents, and inserts data into the SALES_ORDER collection tables according to the mapping that is specified in the DAD file with which it was enabled.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char          serviceName[48];    /* name of an XML collection */
              char          policyName[48];     /* name of an XML collection */
              char          collection[48];     /* name of an XML collection */
```

```
char          status[10];         /* name of an XML collection */

short                 serviceName_ind;
short                 policyName_ind;
short                 collection_ind;
short                 status_ind;
EXEC SQL END DECLARE SECTION;

/* initialize host variable and indicators */
strcpy(serviceName, "myService");
strcpy(policyName, "myPolicy");
strcpy(collection,"sales_ord")
status[0] = '\0';
serviceName_ind = 0;
policyName_ind = 0;
collection_ind = 0;
status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqInsertAll(:serviceName:serviceName_ind;
                :policyName:policyName_ind,
                :collection:collection_ind,
                :status:status_ind);
```

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49012
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other Countries, or both:

DB2
DB2 Universal Database
IBM
MQSeries

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.