IBM® DB2® XML Extender

Release Notes

Version 7, Fixpak 4

IBM® DB2® XML Extender

Release Notes

Version 7, Fixpak 4

Contents

About the Release Notes for IBM® DB2® XML Extender Version 7.2

This document contains information about the DB2 XML Extender 7.2 Fixpack 4, supplementing information in *DB2 Universal Database® XML Extender Administration and Programming, Version 7* manual, released for DB2 Universal Database Version 7, Fixpack 2 together with release notes for Fixpack 3.

The information in these Release Notes includes all topics except installation. For up-to-date information on installing DB2 XML Extender, see the Version 7.2 Installation Notes in readme.txt, in the root directory of the product.

Updates to Fixpack 4

Fixpack 4 has been updated on the website and these release notes reflect the changes. For the most recent version of DB2 XML Extender Fixpack 4 go to:

`www.ibm.com/software/data/db2/extenders/xmlext/support/fixpak.html`

The following changes have been made to DB2 XML Extender Fixpack 4:

- The memory leak for XML column, (APAR IY24331) and for XML collection has been fixed.
- It is now possible to shred into more than 4KB rows (JR16352) and into columns that have decimal fields (JR16352).

Updates for the DB2 Universal Database XML Extender Administration and Programming Version 7.

This document provides updates to information about topics in the *DB2 Universal Database XML Extender Administration and Programming Version 7*, the online error messages, and Fixpack 3 release notes.

Migration

If you have been using XML Extender version 7.2 Fixpack 3 or prior, you must complete the following steps before using an existing XML-enabled database with XML Extender V7.2 Fixpack 4:

- From the DB2 command line, enter:
  - db2 connect to <database_name>
  - db2 bind migv71.bnd
- Then run,
  - migv71 <database_name>

Failing to do the above can cause ″dxxadm disable_db ...″ to fail.

There are two parts to the migration script:

1. A Stored procedures (SP) migration that allows you to get two new CLOB stored procedures.
2. User defined functions (UDFs) migration that allow you to get the parallel capability for the scalar UDFs.

If the migration is successful, you will have two additional CLOB stored procedures and the scalar UDF will be run in parallel.

However, if the stored procedures migration fails, then the migration is terminated, and no UDF migration is carried out. If the stored procedures migration is successful, the UDF migration will be continued.

If the UDF migration is unsuccessful, you will still have two new CLOB stored procedures and your UDFs will still work, but they will not run in parallel.

iSeries users should refer to the cover letter for`PTF 5722DE1 V5R1M0 SI02317` for migration instructions.

Using DB2 XML Extender stored procedures across different platforms

Chapter
XML Extender Stored Procedures

Section
Calling XML Extender stored procedures

Update
Paragraph beginning ″ In general, call the XML Extender using...″

You can now use XML Extender in different operating systems from a single client application, if you write the stored procedure names in uppercase. To call the stored procedures in this way, use the *result_colname* and *valid_colname* versions of the composition stored procedures described in theFixpack 3 Release Notes. This method gives you the following benefits:

- You can use these stored procedures in DB2 Universal Database Extended Enterprise Edition (EEE) environments because you can include many columns in the result table. The versions of the stored procedures that do not support `result_colname` and `valid_colname` require exactly one column in the result table. DB2 UDB EEE does not support tables that contain a single column of a type derived from LOB.
- You can use a declared temporary table as your result table. Your temporary table is identified by a schema that is set to ″session″. Declared temporary tables enable you to support multi-user client environments.

It is strongly recommended that you use uppercase when calling the DB2 XML Extender stored procedures to access the stored procedures consistently across platforms.

New Composition stored procedures

Two more composition stored procedures have been developed. These are:

- db2xml.dxxGenXMLCLOB
- db2xml.dxxRetrieveXMLCLOB

These new stored procedures are similar to db2xml.dxxGenXML and db2xml.dxxRetrieveXML except that the XML document is returned in a CLOB and does not require a result table.

By using these stored procedures, you no longer need temporary or permanent tables for composed documents. This simplifies programming, especially in a multi-user client environment, and also reduces the instruction pathlength and improves throughput.

db2xml.dxxGenXMLClob and db2xml.dxxRetrieveXMLClob have the following benefits:

- They can be used in DB2 Universal Database EEE.
- They are supported on Windows, UNIX, and iSeries(these stored procedures are planned for z/OS).

dxxGenXMLClob

Purpose

As input, dxxRetrieveXMLClob takes a buffer containing the DAD. It constructs XML documents using data that is stored in the XML collection tables that are specified by the `<Xcollection>` in the DAD and returns the first and typically the only XML document generated into the *resultDoc* CLOB.

Format

```
dxxGenXMLClob(CLOB(100k)                        DAD                     /*input*/
              integer                           overrideType,          /*input*/
              varchar(varchar_value)    override,              /*input*/
              CLOB(2G)                          resultDoc,          /*output*/
              integer                           valid,             /*output*/
              integer                           numDocs,          /*output*/
              long                              returnCode,      /*output*/
              varchar(1024)                     returnMsg),     /*output*/
```

Where *varchar_value* is 32672 for Windows and UNIX and 16366 for iSeries and z/OS.

Parameters

Table 1. dxxGenXMLClob parameters

Parameter	**Description**	**IN/OUT parameter**
DAD	A CLOB containing the DAD file.	IN
overrideType	A flag to indicate the type of *override* parameter: **NO_OVERRIDE** No override. **SQL_OVERRIDE** Override by an SQL_stmt **XML_OVERRIDE** Override by an XPath-based condition.	IN
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 be 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 the word and. Using this *overrideType* requires that RDB_node mapping be used in the DAD file	IN
resultDoc	A CLOB that contains the composed XML document.	OUT
valid	valid is set as follows: • If VALIDATION=YES then valid=1 for successful validation or valid=0 for unsuccessful validation. • If VALIDATION=NO then valid=NULL.	OUT
numDocs	The number of XML documents that would be generated from the input data. **Note:** Currently only the first document is returned.	OUT
returnCode	The return code from the stored procedure.	OUT
returnMsg	The message text that is returned in case of error.	OUT

dxxRetrieveXMLClob

Purpose

dxxRetrieveXMLClob enables document composition from relational data. This stored procedure also serves as a means for retrieving decomposed XML documents.

The requirements for using dxxRetrieveXMLClob are the same as the requirements for dxxGenXMLClob. The only difference is that the DAD is not an input parameter for dxxRetrieveXMLClob, but it is the name of an enabled XML collection.

Format

```
dxxGenXMLClob(CLOB(100k)                         DAD                    /*input*/
              integer                           overrideType,          /*input*/
              varchar(varchar_value)    override,             /*input*/
              CLOB(2G)                          resultDoc,          /*output*/
              integer                           valid,             /*output*/
              integer                           numDocs,          /*output*/
              long                              returnCode,      /*output*/
              varchar(1024)                     returnMsg),     /*output*/
```

Where *varchar_value* is 32672 for Windows and UNIX and 16366 for iSeries and z/OS.

Parameters

Table 2. dxxRetrieveXMLClob parameters

Parameter	**Description**	**IN/OUT parameter**
collectionName	The name of an enabled XML collection.	IN
overrideType	A flag to indicate the type of *override* parameter: **NO_OVERRIDE** No override. **SQL_OVERRIDE** Override by an SQL_stmt **XML_OVERRIDE** Override by an XPath-based condition.	IN
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 be 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 the word and. Using this *overrideType* requires that RDB_node mapping be used in the DAD file	IN
resultDoc	The maximum number of rows in the result table.	IN
valid	valid is set as follows: • If VALIDATION=YES then valid=1 for successful validation or valid=0 for unsuccessful validation. • If VALIDATION=NO then valid=NULL.	OUT
numDocs	The number of XML documents that would be generated from the input data. NOTE: currently only the first document is returned.	OUT
returnCode	The return code from the stored procedure.	OUT
returnMsg	The message text that is returned in case of error.	OUT

MQSeries XML functions and stored procedures

Note: Supported Information: MQ is not supported on the z/OS or iSeries.

MQ XML stored procedures allow you to retrieve XML documents from message queues, decompose them 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.

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.

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 CLOB data.

Table 3. The MQSeries XML user-defined functions

Function	Purpose
"db2xml.MQReadXMLCLOB" on page 8	Return a message at the head of a queue without removing it from the queue.
"db2xml.MQReadAllXMLCLOB" on page 9	Returns a table containing message data without removing messages from the queue.
"db2xml.MQRcvXMLCLOB" on page 11	Return and remove a message from the queue.
"db2xml.MQRcvAllXML" on page 12	Return and remove message from the queue
"db2xml.MQSENDXML" on page 14	Send a message with no expected reply.
"db2xml.MQSendXMLFILECLOB" on page 16	Send a message that contains a file with no expected reply.
"db2xml.MQPublishXML" on page 18	Send message to queue to be picked up by applications that monitor the queue.

db2xml.MQReadXMLCLOB

Purpose

The MQREADXMLCLOB function returns XMLCLOB 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 XMLCLOB containing the messages. If no messages are available to be returned a NULL will be returned.

Format

```
►►──MQReadXMLCLOB──(──┬──────────────────────────────────────┬──)──────────────►◄
                      ├─receive-service──────────────────────┤
                      └─receive-service──,──service-policy───┘
```

Parameters

Table 4. MQReadXMLCLOB 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
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, MQREADXMLCLOB returns a db2xml.xmlclob. A NULL is returned if no messages are available.

db2xml.MQReadAllXMLCLOB

Purpose

The MQReadAllXMLCLOB 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

```
►►──MQReadAllXMLCLOB──(──┬──────────────────────────────────────┬──┬──────────┬──)──────────►◄
                         ├─receive-service──────────────────────┤  └─num-rows─┘
                         └─receive-service──,──service-policy───┘
```

Parameters

Table 5. MQReadAllXMLCLOB 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 .
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 MQReadAllXMLCLOB function returns a table containing messages and message metadata as described below.

Table 6. MQReadAllXMLCLOB Result set table

Column Name	Data Type	Description
MSG	XMLCLOB	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.

db2xml.MQRcvXMLCLOB

Purpose

The MQRcvXMLCLOB 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

```
►►──MQRcvXMLCLOB──(──┬─────────────────────────────────────────────────┬──)──────────►◄
                     ├─receive-service─────────────────────────────────┤
                     ├─receive-service──,──service-policy──────────────┤
                     └─receive-service──,──service-policy──correl-id───┘
```

Parameters

Table 7. MQRcvXMLCLOB 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

MQReceiveXMLCLOB functions return a db2xml.XMLCLOB 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. However, if the *correl-id* is not specified then the message at the head of the queue will be returned.

db2xml.MQRcvAllXML

Purpose

The MQRcvAllXMLCLOB removes the messages from the queue associated with *receive-service*. If the correl-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

```
►►──MQRcvAllXMLCLOB──(──┬──────────────────────────────────────────────────┬──┬────────────┬──)──────────►◄
                        ├─receive-service──────────────────────────────────┤  └─num-rows─┘
                        ├─receive-service──,──service-policy───────────────┤
                        └─receive-service──,──service-policy──correl-id────┘
```

Parameters

Table 8. MQRcvAllXMLCLOB 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. 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.
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.

Table 8. MQRcvAllXMLCLOB parameters (continued)

Parameter	Data type	Description
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, MQRcvAllXML returns a db2xml.xmlclob A NULL is returned when no messages are available. The messages are returned in a table as described below

Table 9. MQRcvAllXML result set table

Column Name	Data Type	Description
MSG	XMLCLOB	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.

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

```
►►──MQSENDXML──(──┬─────────────────────────────────┬──msg-data──,──┬─────────────┬──)──────────►◄
                  ├─send-service────────────────────┤               └─correl-id───┘
                  └─send-service──,──service-policy─┘
```

Parameters

Table 10. MQSendXML parameters

Parameter	Data type	Description
msg-data	XMLCLOB	An XMLCLOB 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.
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 results 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*.

db2xml.MQSendXMLFILECLOB

Purpose

The MQSendXMLFILECLOB 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

```
►►──MQSendXMLFILECLOB──(──┬─────────────────────────────────┬──xml_file──,──┬──────────┬──)──────────►◄
                          ├─send-service────────────────────┤               └─correl-id─┘
                          └─send-service──,──service-policy─┘
```

Parameters

Table 11. MQSENDXMLFILE parameter

Parameter	**Data type**	**Description**
xml_file	VARCHAR(80)	An XML file name with a maximum size of 80 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
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*.

db2xml.MQPublishXML

Purpose

The MQPUBLISHXML function publishes XMLVARCHAR and XMLCLOB 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

```
►►──MQPublishXML──(──┬─────────────────────────────────────────┬──msg-data──,──┬───────┬──)──────►◄
                     ├─publisher-service───────────────────────┤               └─topic─┘
                     └─publisher-service──,──service-policy────┘
```

Parameters

Table 12. 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 12. 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, the MQPublishXML functions return a '1'. A value of '0' is returned if the function is unsuccessful.

New composition stored procedures for message queues

The composition stored procedures dxxmqGenCLOB and dxxmqRetrieveCLOB are used to generate XML documents using data in existing database tables and to send the generated XML documents to a message queue. The dxxmqGenCLOB stored procedure takes a DAD file as input. It does not require an enabled XML collection. The dxxmqRetrieveCLOB stored procedure takes an enabled XML collection name as input.

db2xml.dxxmqGenCLOB

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, dxxmqGenCLOB 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*.

Format

```
dxxmqGenCLOB(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 13. dxxmqGenCLOB 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 13. dxxmqGenCLOB 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

db2xml.dxxmqRetrieveCLOB

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, dxxmqRetrieveCLOB 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*.

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

Format

```
dxxmqRetrieveCLOB(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 14. dxxmqRetrieveCLOB 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

Table 14. *dxxmqRetrieveCLOB parameters (continued)*

Parameter	Description	IN/OUT parameter
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
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

Decomposition stored procedures for message queues

The decomposition stored procedures dxxmqInsertCLOB, dxxmqInsertAllCLOB, dxxmqShredCLOB, and dxxmqShredAllCLOB are used to break down or shred incoming XML documents from a message queue, and to store the data into new or existing database tables. The dxxmqInsertCLOB and dxxmqInsertAllCLOB stored procedures take an enabled XML collection name as input. The dxxmqShredCLOB and dxxmqShredAllCLOB stored procedures take a DAD file as input; they do not require an enabled XML collection.

db2xml.dxxmqShredCLOB

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 dxxmqShredCLOB 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

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

Parameters

Table 15. dxxmqShredCLOB 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

db2XML.dxxmqShredAllCLOB

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 dxxmqShredAllCLOB 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

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

Parameters

Table 16. dxxmqShredAllCLOB 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

db2XML.dxxmqInsertCLOB

Purpose

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

Format

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

Parameters

Table 17. dxxmqInsertCLOB 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

Performance improvements

The following performance improvements have been made for composition and decomposition.

- The length of the override parameter has been has been increased from 1KB to 32KB for Unix and Windows. On iSeries and zSeries it is16KB.

The 1KB override imposed a restriction on the length of the SQL statement for SQL composition. The restriction encouraged the use of database views to reduce the length of the required SQL statement. However, that database views can sometimes incur additional pathlength because of view materialization. With a long override, the strong need for views is reduced. Note that this override parameter does not apply to the MQSeries stored procedures. The override for those stored procedures is still 1KB i.e. Vachar(1024).

- The requirement for an intermediate result table has been removed.

By using these stored procedures:

- You reduce the instruction pathlength because there is no need to create result tables.
- You simplify your programming.

Use the stored procedures that require an intermediate result table if you want to produce more than one document.

- The user defined functions for XML column have been enhanced for performance

The DB2 XML Extender user-defined functions will now keep small (512KB) XML documents in memory while processing them. This reduces Input/Output activity and the contention for the disk that is used for temporary files.

The definition of the DB2 XML Extender scalar (non-table) user-defined functions has been changed so that they can be run in parallel. This provides significant performance improvements in the execution of queries that refer to the User Defined Functions more than once.You have to run the migration script program to get the parallel capability for the scalar UDFs. If you already have columns enabled using the scalar UDFs, you have disable all your columns, run the migration script and then reenable the columns.

Using DAD files

The following section outlines changes that affect how you use DAD files..

SQL composition: using columns with the same name

Selected variables with the same name, even if from diverse tables, must be identified by a unique alias so that every variable in the select clause of the SQL statement is different. The following example shows how you would give columns that have the same names unique aliases.

```
<SQL_stmt>select o.order_key as oorder_key,key customer_name, customer_email,
p.part_key p.order_key as porder_key,color
qty, price, tax, ship_id, date, mode from order_tab o.
part_tab p ORDER BY order_key, part_key</SQL_stmt>
```

SQL composition: using columns with random values

If a SQL statement in a DAD has a random value, you have to give the random value function an alias in order to use it in the ORDER BY clause.This is because it is not associated with any column in a given table. For example, see the alias for Generate_unique at the end of the ORDER BY clause below.

```
<SQL_stmt>select o.order_key, customer_name,customer_email,p.part_key,color,
qty,price,tax,ship_id, date, mode from order_tab o, part_tab p,
table(select substr(char(timestamp(generate_unique
())),16)asship_id, date, mode,
part_key from ship_tab) s where o.order_key=1 and p.price>2000
and o.order_key=o.order_key and s.part_key ORDER BY order_key, part_key,
ship_id</SQL_stmt>
```

RDB node composition: restrictions

The following restrictions apply:

- The condition associated with any non-root_node RDB node DAD must compare against a literal.
- The condition associated with a root_node describes the relationship between the tables involved in the RDB node composition. For example, a primary foreign key relationship.
- Each equality in the condition associated with a top-level RDB_node specifies the join relationship between columns of two tables and is applied separately from the other equalities. In other words, all the predicates connected by AND do not apply simultaneously for a single join condition, thereby simulating an outer join during document composition. The parent-child relationship between each pair of tables is determined by their relative nesting in the DAD. For example:

```
<condition>order_tab.order_key=part_tab.order_key AND
part_tab.part_key=ship_tab.part_key</condition>
```

Composition and decomposition limits

Additional limits for XML Extender objects have been introduced. The following table lists the objects that have been introduced and their respective limits.

Table 18. Limits for XML Extender objects

Object	Limit
Maximum number of rows inserted into a table in a decomposition XML collection	10240 rows from each decomposed XML document
Maximum length of the name attribute in elements_node or attribute_node within a DAD	63 bytes
Maximum bytes in XMLFile path name specified as a parameter value	512 bytes

Defect fixes

The following section describes reported defects from previous versions of DB2 XML Extender and outlines how these problems have been solved.

XML RDB Node Decomposition:

The following changes have been made:

- The maximum number of rows that can be decomposed per table has been increased to 10K rows from 1K row
- For a subtree of the DAD with element_nodes and attribute_nodes that map to same table the following change has been made:
  - Attribute nodes no longer have to be the first children of the lowest common ancestor of the element nodes that map to the same table.

– Attribute nodes can appear anywhere in subtree, as long as they are not involved in a join condition.

XML RDB Node Composition: Multiple overrides are now allowed

In the previous version of DB2 XML Extender multiple, overrides on the same path were not supported. Only the first override was taken and the rest were ignored. Currently, all overrides specified will be accepted.

Example 1: You can specify mulitple XML overrides on the same location path to refine set conditions in your search. In the following example, we compose an XML document from the two tables using the test.dad file.

The following example shows you how to write multiple XML override code allowing you to constrain your search results.

Table 19. Department Table

Department Number	Department Name
10	Engineering
20	Operations
30	Marketing

Table 20. Employee Table

Employee Number	Department Number	Salary
123	10	$98,000.00
456	10	$87,000.00
111	20	$65,000.00
222	20	$71,000.00
333	20	$66,000.00
500	30	$55,000.00

The DADfile **test.dad** illustrated below contains a condition comparing the variable deptno with the value 10. To override this condition so that the search is expanded to greater than 10 and less than 30 you must set the override parameter when calling *dXXGenXML* as follows:

″/ABC.com/Department>10 AND /ABC.com/Department<30″

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "C:\dxx_xml\test\dtd/dad.dtd">
<DAD>
<dtdid>E:\dtd\lineItem.dtd</dtdid>
<validation>NO</validation>
<Xcollection>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "C:\dxx_sml\test\dtd\LineItem.dtd"</doctype>
<root_node>
<element_node name="ABC.com">
<RDB_node>
<table name="dept" key="deptno"/>
<table name="empl" key="emplno"/>
<condition>dept deptno=empl.deptno</condition>
</RDB_node>

   <element_node name="Department" multi_occurrence="YES">
   <text_node>
   <RDB_node>
    <table name="dept"/>
    <column name="deptno"/>
    <condition>deptno=10</condition>
   </RDB_node>
  </text_node>

<element_node name="Employees" multi_occurrence="YES">
 <text_node>
  <RDB_node>
   <table name="dept"/>
    <column name="deptnot"/>
     <condition>deptno=10</condition>
   </RDB_node>
  </text_node>

<element_node name="Employees" multi_occurence="YES">
    <element_node name="EmployeeNo">
       <text_node>
         <RDB_node>
             <table  name="empl"/>
              <column name="emplno"/>
               <condition>emplno<500</condition>
            </RDB_node>
          </text_node>
         </element_node>
      <element_node name="Salary">
         <text_node>
           <RDB_node>
              <table name="empl"/>
              <column name="salary"/>
              <condition>salary>5000.00</condition>
            </RDB_node>
          </text_node>
        </element_node>
    </element_node>
    </element_node>
    </element_node>
    </root_node>
    </Xcollection>
```

To compose an XML document without an override, enter **tests2x mydb test.dad result_tab** or you can invoke dxxGenXML without setting an override. This will generate a document similar to this:

```
<?xml version="1.0">
<!DOCTYPE Order SYSTEM "C:\dxx_xml\test\dtd\LineItem.dtd">
<ABC.com>
<Department>10
  <Employees>
   <EmployeeNo>123</EmployeeNO>
   <Salary>98,000.00</Salary>
   </Employees>
   <Employees>
    <EmployeeNo>456</EmployeeNo>
    <Salray>87,000.00</Salary>
   </Employees>
   </Department>
   </ABC.COM>
```

To override the DAD file you can invoke dxxGenXML as mentioned above, or you can run the**tests2x mydb test.dad result_tab -o 2 ″/ABC.com/Department>10 AND /ABC.com/Department<30″** with these conditions, to generate the following document.

```
<?xml version="1.0">
<!DOCTYPE Order SYSTEM "C:\dxx_xml\test\dtd\LineItem.dtd">
```

```
<ABC.com>
 <Department>20
  <Employees>
   <EmployeeNo>111</EmployeeNo>
   <Salary>65,000.00</Salary>
  </Employees>
   <EmployeeNo>222</EmployeeNo>
    <Salary>71,000.00</Salary>
    </Employees>
   <Employees>
    <EmployeeNo>333</EmployeeNo>
     <Salary>66,000.00</Salary>
    </Employees>
   </Department>
   </ABC.com>
```

XML RDB Node Composition: orderBy implemented

The orderBy option—the order data is sorted—was previously not supported. You can now control the way the sibling elements are sorted by using the orderBy option. In the sample dad called *orderBy.dad* below, orderBy is used to sort the contents of the output document by location desc, and itemno.

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<Xcollection>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Catalog SYSTEM "d:\dtd\test.dtd"</doctype>
<root_node>
<element_node name="Catalog">
    <RDB_node>
       <table name="stocks" orderBy="location desc, itemno asc"/>
    </RDB_node>
     <element_node name="Product" multi_occurrence="YES">
      <element_node name="ItemNo">
        <text_node>
          <RDB_node>
            <table name="stocks"/>
            <column name="itemno"/>
          </RDB_node>
        </text_node>
      </element_node>
         <element_node name="WarehouseLocation">
        <text_node>
          <RDB_node>
            <table name="stocks"/>
           < column name="location"/>
          </RDB_node>
        </text_node>
      </element_node>
    </element_node>
  </element_node>
</root_node>
</Xcollection>
</DAD>
```

By invoking dxxGenXML with the DAD illustrated above, the following document will be generated. Alternatively, you can use **tests2x mydb orderby.dad result_tab**. This will also generate the document.

```
!DOCTYPE Catalog SYSTEM "d:\dtd\test.dtd">
<Catalog>
  <Product>
<WarehouseLocation>Z</WarehouseLocation>
 < /Product>
  <Product>
```

```
<ItemNo>33</ItemNo>
    <WarehouseLocation>Y</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>77</ItemNo>
   <WarehouseLocation>Y</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>44</ItemNo>
    <WarehouseLocation>X</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>55/ItemNo>
    <WarehouseLocation>Q</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>66/ItemNo>
    <WarehouseLocation>Q</WarehouseLocation>
  </Product>
</Catalog>
```

By changing the orderBy specification to the following **orderBy=″location asc, itemno desc**, the document below will be generated.

```
<?xml version="1.0"?>
<!DOCTYPE Catalog SYSTEM "d:\dtd\test.dtd">
<Catalog>
  <Product>
    <ItemNo>66</ItemNo>
    <WarehouseLocation>Q</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>55</ItemNo>
    <WarehouseLocation>Q</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>44</ItemNo>
    <WarehouseLocation>X</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>77</ItemNo>
    <WarehouseLocation>Y</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>33</ItemNo>
    <WarehouseLocation>Y</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>22</ItemNo>
    <WarehouseLocation>Z</WarehouseLocation>
  </Product>
  <Product>
    <ItemNo>11</ItemNo>
    <WarehouseLocation>Z</WarehouseLocation>
  </Product>
</Catalog>
```

XML Composition: Successful completion messages are now returned

Complete messages are now returned for XML composition stored procedures. For example, DXXQ020I XML is successfully generated after an XML document is composed. Previously, messages were not being returned.

XML Composition: Composition from rows that have null values is now supported

You can now use columns that have null values to compose XML documents. Previously, using such columns to compose XML documents caused the XML Extender to fail.

Example: The following example illustrates how you can generate an XML document from a table*MyTable* which has a row containing a null value in column *Col1*. The dad used in the example is called *nullcol.dad*.

```
<?xml version="1.0"?>
 <!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
 <DAD>
 <validation>NO validation>NO>
<Xcollection>
<SQL_stmt>SELECT 1 as X, Col1 FROM MyTable order by X, Col1</SQL_stmt>
<prolog>?xml version="1.0"?prolog>?xml version="1.0"?>
 <doctype>!DOCTYPE Order SYSTEM "e:\t3xml\x.dtd">
 <root_node>
  <element_node name="MyColumn">
<element_node name="Column1" multi_occurrence="YES">
       <text_node>
       <column name="Col1"/>
       </text_node>
     </element_node>
   </element_node>
 </root_node>
 </Xcollection>
 </DAD>
```

Run: **tests2x mydb nullcol.dad result_tab** or use dxxGenXML to produce the following document. Note that the third *Column1* element represents a null value.

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "e:\t3xml\x.dtd">
<MyColumn>
  <Column1>1</Column1>
 <Column1>3</Column1>
 <Column1></Column1>
</MyColumn>
```

- XML Composition:

The case (upper, lower) treatment of the result_colname and the valid colname values has been improved.

Encoding Declarations Supported by XML Extender

All code pages are now supported on all UNIX and Window operating systems that are supported by XML Extender.

Table 21. Encoding declarations supported by XML Extender

Category	Encoding	Code page
Unicode	UTF-8	1208
	UTF-16	1200

Table 21. Encoding declarations supported by XML Extender (continued)

Category	Encoding	Code page
ASCII	iso-8859-1	819
	ibm-1252	1252
	iso-8859-2	912
	iso-8859-5	915
	iso-8859-6	1089
	iso-8859-7	813
	iso-8859-8	916
	iso-8859-9	920
MBCS	gb2312	1386
	ibm-932, shift_jis78	932
	Shift_JIS	943
	IBM-eucCN	1383
	ibm-1388	1388
	IBM-eucJP, EUC-JP	954, 33722
	ibm-930	930
	ibm-939	939
	ibm-1390	1390
	ibm-1399	1399
	ibm-5026	5026
	ibm-5035	5035
	euc-tw, IBM-eucTW	964
	ibm-937	937
	euc-kr, IBM-eucKR	970
	big5	950

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.

Trademarks

DB2
DB2 Universal Database
IBM
MQSeries