IBM InfoSphere Data Replication
Version 10.1.3

Replication and Event Publishing Guide and Reference

SC19-3637-00

IBM InfoSphere Data Replication
Version 10.1.3

Replication and Event Publishing Guide and Reference

SC19-3637-00

Note

Before using this information and the product that it supports, read the information in “Notices and trademarks” on page 567.

Contents

Chapter 1. Data conversion for Q Replication and Event Publishing

In Q Replication and Event Publishing, data passes between various servers and programs, and sometimes the data must be converted between different code pages.

Code pages for Q Replication

In Q Replication, data passes between various servers and programs, and sometimes the data must be converted. For example, the programs might have different code pages, or the platform or processor might handle numeric values differently.

Data is automatically converted at the column level as needed, even if the source server and target server are in different code pages. Endianess and floating point representations conversions are handled.

Recommendation: If possible, avoid any data conversion by using matching code pages for the following programs or servers:
- Q Capture program
- Q Apply program
- Source server
- Target server

If the source server and target server must use different code pages, then use matching code pages for the Q Capture program and the source server and use matching code pages for the Q Apply program and the target server.

When data from the source table is replicated to the target table or stored procedure, the Q Capture program sends the data over WebSphere® MQ in the format of the source table. The source data is converted, if required, at the target.

Linux UNIX Windows On Linux, UNIX, and Windows, when the Q Apply program receives the source data from the receive queue, it converts each column in the transaction and all other data in the message to its own code page. The target server expects the data to be in the code page and floating point representation of the Q Apply program.

z/OS On z/OS, the Q Apply program does not convert the source data to its own codepage. Instead, for each operation the Q Apply program tags the source codepage of the data for DB2. DB2 then directly converts the data from the source codepage to the target codepage.

If the source table is defined in EBCDIC or ASCII and the target table is defined in UTF-8, SQL errors could occur if the target column is too small to hold the expansion of bytes that occurs when EBCDIC or ASCII data is converted to UTF-8. To avoid this error, the column in the target table should be defined to hold 3 bytes for every character stored at the source. For example, if the source column is defined as CHAR(*x*), the target column should be defined as CHAR(3*x*).

z/OS **Recommendation:** zSeries® represents floating point values differently than Intel based CPUs, which might cause some data to be lost. Avoid using a float column as a key.

Restriction: The code pages for the Q Capture and Q Apply programs cannot be UTF-16.

If you plan to replicate data between servers with different code pages, check the *IBM DB2 Administration Guide: Planning* to determine if the code pages that you are using are compatible.

Code pages for event publishing

In event publishing, the data is converted from the code page of the source server to an XML message as UTF-8 (code page 1208, which is a standard encoding scheme for XML), or to a delimited message in a user-specified code page (default code page 1208).

When the user application sends an administration message (for example, a subscription deactivated message) to the Q Capture program in XML format, the XML parser converts the message to the code page of the Q Capture program.

Chapter 2. Setting up user IDs and passwords

To use the Q replication and event publishing programs, you need to set up user IDs and passwords for accessing DB2® servers on distributed systems.

Authentication requirements on Linux, UNIX, and Windows

Q Replication does not require you to use any specific type of authentication. You should use the compatible authentication type that best meets your business needs.

The following list provides more detail:

DB2 sources and targets

- Q Replication is a database application program.
- The underlying DB2 client-server facilities are as transparent to Q Replication as they are to any database application.
- Q Replication connects to DB2 databases using the traditional ID and password combination. You can use any authentication type that is compatible with this.
- Q Replication has no requirement on authentication type other than this.

Oracle sources

- Q Capture for Oracle databases is an Oracle C++ Call Interface (OCCI) database application program.
- Q Capture uses the usual Oracle user credentials that are required to connect to an Oracle database. The operating system user credentials are not used when connecting.
- The value of the **`capture_server`** or **`apply_server`** parameter corresponds to an alias that is defined in a tnsnames.ora file. The value of the **`capture_schema`** or **`apply_schema`** parameter corresponds to the user ID that is provided when connecting.
- You must create a password file with the asnpwd utility. When Q Capture for Oracle connects to the Oracle database, it retrieves the password from this password file.

Authorization requirements for Q Replication and Event Publishing

The user IDs that run the Q Replication and Event Publishing programs need authority to connect to servers, access or update tables, and perform other operations.

Authorization requirements for the Q Capture program

All user IDs that run a Q Capture program must have authorization to access the DB2 system catalog, access and update all Q Capture control tables, read the DB2 log, and run the Q Capture program packages.

z/OS For a list of authorization requirements on z/OS®, see DB2 authorizations for the Q Capture and Capture programs.

Linux UNIX Windows The following list summarizes the DB2 requirements and operating system requirements for Linux, UNIX, and Windows:

Requirements

User IDs that run a Q Capture program must have the following authorities and privileges:

- DBADM or SYSADM authority.
- WRITE privilege on the directory that is specified by the **capture_path** parameter. The Q Capture program creates diagnostic files in this directory.
- Windows Authority to create global objects.

In a partitioned database environment, the user IDs must be able to connect to database partitions and read the password file.

Authorization requirements for the Q Apply program

All user IDs that run a Q Apply program must have authorization to access the DB2 system catalog, access and update targets, access and update the Q Apply control tables, run the Q Apply program packages, and read the Q Apply password file.

z/OS For a list of authorization requirements on z/OS, see DB2 authorizations for the Q Apply and Apply programs.

Linux UNIX Windows

The following list summarizes the DB2 requirements and operating system requirements for Linux, UNIX, and Windows, and for non-DB2 targets.

Requirements

User IDs that run a Q Apply program must have the following authorities and privileges:

- DBADM or SYSADM authority.
- SELECT privilege for the source tables if the Q Apply program will be used to load target tables.
- WRITE privilege on the directory that is specified by the **apply_path** parameter. The Q Apply program creates diagnostic files in this directory.
- Windows Authority to create global objects.

If the Q Apply program uses the LOAD from CURSOR option of the LOAD utility to load target tables, the Q Apply server must be a federated server, and you must create nicknames, server definitions, and user mappings on the Q Apply server. The user ID that is supplied in the user mappings must have privilege to read from nicknames on the federated Q Apply server and read from the source tables.

Requirements for non-DB2 targets

User IDs that run a Q Apply program must have the following authorities and privileges:

- CREATE TABLE and CREATE INDEX on the remote database.
- WRITE privilege on nicknames in the federated database and, through user mappings, WRITE privilege on the non-DB2 target.

Authorization requirements for the Replication Center and ASNCLP program

User IDs that administer Q Replication and Event Publishing with the Replication Center or ASNCLP command-line program require certain database authorizations at the source and target servers.

You must have at least one user ID on all databases that are involved in the replication configuration, and that user ID must have the authority to perform a variety of administrative tasks at the Q Capture server, Q Apply server, and Monitor control server if you use the Replication Alert Monitor.

The following authorities and privileges are required:

- CONNECT privilege for the Q Capture server, Q Apply server, and Monitor control server
- All required table, table space, and index privileges to create control tables at the Q Capture server, Q Apply server, and Monitor control server
- All required table, table space, and index privileges to create targets at the Q Apply server
- SELECT, UPDATE, INSERT, and DELETE privileges for all control tables on the Q Capture server, Q Apply server, and Monitor control server
- z/OS SELECT privilege for the following DB2 for z/OS system catalog tables:
  - SYSIBM.SYSCHECKS
  - SYSIBM.SYSCOLUMNS
  - SYSIBM.SYSDATABASE
  - SYSIBM.SYSDUMMY1
  - SYSIBM.SYSINDEXES
  - SYSIBM.SYSINDEXPART
  - SYSIBM.SYSKEYCOLUSE
  - SYSIBM.SYSKEYS
  - SYSIBM.SYSKEYTARGETS
  - SYSIBM.SYSPARMS
  - SYSIBM.SYSRELS
  - SYSIBM.SYSROUTINES
  - SYSIBM.SYSSTOGROUP
  - SYSIBM.SYSTABCONST
  - SYSIBM.SYSTABLEPART
  - SYSIBM.SYSTABLES
  - SYSIBM.SYSTABLESPACE
  - SYSIBM.SYSTRIGGERS
- Privileges to bind plans on each DB2 database involved in replication or publishing, including the Q Capture server, Q Apply server, and Monitor control server
- The following stored procedure privileges:

  z/OS

  EXECUTE authority on the following stored procedures:
  - SYSIBM.SQLPROCEDURECOLS

    – SYSPROC.DSNWZP
    – SYSPROC.ADMIN_INFO_SSID

Linux UNIX Windows
: Privileges to create stored procedures using a shared library, and to call stored procedures.

To simplify administration with the Replication Center, you can use the Manage Passwords and Connectivity window to store user IDs for servers or systems, as well as to change the IDs that you stored and to test connections. To open the window, right-click the **Replication Center** folder and select **Manage Passwords for Replication Center**.

Connectivity requirements for Q Replication and Event Publishing

To replicate or publish data in a distributed environment, you must set up and configure connectivity. In most cases, you must also be able to connect to remote DB2 databases or subsystems to use the ASNCLP command-line program, Replication Center, or Replication Alert Monitor, to load target tables, or to insert signals to activate or deactivate Q subscriptions or publications.

Connectivity requirements for DB2 databases or subsystems differ depending on your replication or publishing environment:

- The ASNCLP or Replication Center must be able to make the following connections:
  - To the Q Capture server to administer Event Publishing.
  - To the Q Capture server and Q Apply server to administer Q Replication.
  - To the Monitor control server to set up the Replication Alert Monitor.
- If you plan to have the Q Apply program automatically load targets with source data by using the EXPORT utility, the Q Apply program must be able to connect to the Q Capture server. This connection requires a password file that is created with the **`asnpwd`** command.
- The Q Capture program must be able to connect to partitioned databases. This connection requires a password file that is created with the **`asnpwd`** command.
- If you are using the administration tools, system commands, or SQL to administer replication from a remote workstation, the remote workstation must be able to connect to the Q Capture server, Q Apply server, or Monitor control server.

Managing user IDs and passwords for remote servers (Linux, UNIX, Windows)

Replication and Event Publishing require a password file in some cases to store user IDs and passwords for connecting to remote servers.

About this task

A password file is required in the following cases:

- The Apply program requires a password file to access data on remote servers (the Capture program does not require a password file).
- The Q Apply program requires a password file to connect to the Q Capture server for Q subscriptions that use the EXPORT utility to load targets.

- The Q Capture program requires a password file to connect to multiple-partition databases.
- If the Q Capture program runs remotely from the source database or the Q Apply program runs remotely from the target database, the programs require password files to connect to the remote database.
- The **asntdiff** and **asntrep** commands require password files to connect to databases where the utilities are comparing or repairing table differences.
- The Replication Alert Monitor requires a password file to connect to any Q Capture, Capture, Q Apply, or Apply server that you want to monitor.

Important note about compatibility of password files: Password files that are created by the **asnpwd** command starting with Version 9.5 Fix Pack 2 use a different encryption method and cannot be read by older versions of the replication programs and utilities. If you share a password file among programs and utilities that are at a mixed level, with some older than these fix packs, do not recreate the password file by using an **asnpwd** command that is at these fix packs or newer. Replication programs and utilities at these fix packs or newer can continue to work with older password files. Also, you cannot change an older password file to use the later encryption method; you must create a new password file.

In general, replication and Event Publishing support the following scenarios:

- Creating a password file with one version and using it with a newer version. For example, you can create a password file under V8.2 and use it with V9.1 and V9.5.
- Creating a password file with one fix pack and using it with a newer fix pack within the same version. For example, you can create a password file with V9.1 Fix Pack 3 and use it with V9.1 Fix Pack 5.
- Creating a password file on one system and using it on another system as long as the following criteria are met:
  - The systems use the same code page.
  - The systems are all 32 bit or all 64 bit.

Encrypted password files are not supported for x64 Windows until 9.5 Fix Pack 2 or later.

Procedure

To manage user IDs and passwords for remote servers, follow these guidelines:

- Create an encrypted password file for replication and event publishing programs that are running on Linux, UNIX, and Windows by using the **asnpwd** command. The password file must be stored in the path that is set by the following parameters:

Table 1. Password file requirements

Program	Parameter
Apply	**apply_path**
Q Apply	**apply_path**
Q Capture	**capture_path**
Replication Alert Monitor	**monitor_path**
asntdiff or asntrep command	**DIFF_PATH**

- If the Q Apply program and Replication Alert Monitor are running on the same system, they can share the same password file. If you want the programs to share a password file, specify the same path and file name for the programs, or use symbolic links to share the same password file in the different directories.
- The Replication Center does not use the password file that is created with the **asnpwd** command to connect to remote servers. The first time that the Replication Center needs to access a database or subsystem, you are prompted for a user ID and password, which is stored for future use. You can use the Manage Passwords and Connectivity window to store user IDs for servers or systems, as well as to change the IDs that you stored and to test connections. To open the window, right-click the **Replication Center** folder and select **Manage Passwords for Replication Center**.

Chapter 3. Setting up WebSphere MQ for Q Replication and Event Publishing

Q Replication and Event Publishing use WebSphere MQ, formerly known as MQ Series, to transmit transactional data and exchange other messages.

Recommendation: Create the queue managers, queues, and other objects for the messaging infrastructure before you create replication objects such as control tables and Q subscriptions, and before you start the Q Capture and Q Apply programs. The following tools and samples are available to help:

- MQ Script Generators that you can use by clicking the Script Generator icon on the Replication Center menu bar
- The CREATE MQ SCRIPT command in the ASNCLP command-line program
- z/OS The ASNQDEFQ sample job in the SASNSAMP data set
- Linux UNIX Windows The asnqdefq sample script in the `SQLLIB/samples/repl/q` directory

For more information on WebSphere MQ, see the WebSphere MQ Information Center at http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp.

WebSphere MQ objects required for Q Replication and Event Publishing

Depending on the type of replication or publishing that you plan to perform, you need various WebSphere MQ objects.

For detailed information about creating WebSphere MQ objects, see the WebSphere MQ Information Center at http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp.

WebSphere MQ objects required for unidirectional replication (remote)

Unidirectional Q Replication or Event Publishing between remote servers requires a queue manager and queues for the Q Capture program and for the Q Apply program.

Because the servers are distributed, you also need transmission queues and channels for transmitting transactions and communicating across a network.

The following lists show the objects that are required for unidirectional replication between two remote servers:

Non-channel objects on source system

- A queue manager
- A remote queue definition to serve as the send queue (this queue points to a receive queue on the target system)
- A local queue to serve as the administration queue
- A local queue to serve as the restart queue

Non-channel objects on target system

- A queue manager
- A local queue to serve as the receive queue
- A remote queue definition to serve as the administration queue (this queue points to an administration queue on the source system)
- A model queue definition for any temporary local spill queues that the Q Apply program creates and uses while it loads target tables

  Note: If you have multiple Q Apply programs that share the same queue manager, each Q Apply program must have its own model queue with a name that is unique under that queue manager.

Channel from source to target

- A sender channel that is defined within the source queue manager
- An associated local transmission queue
- A matching receiver channel that is defined within the target queue manager

Channel from target to source

- A sender channel that is defined within the target queue manager
- An associated local transmission queue
- A matching receiver channel that is defined within the source queue manager

Figure 1 on page 11 shows the WebSphere MQ objects that are required for unidirectional Q Replication between remote servers.

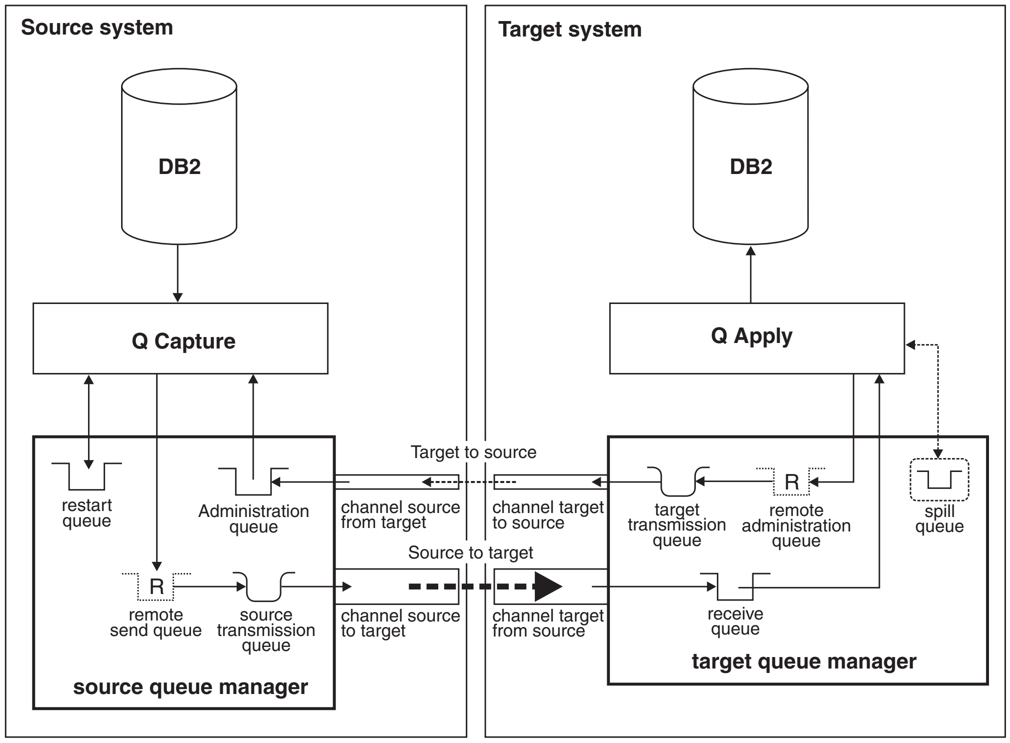


Figure 1. WebSphere MQ objects that are required for unidirectional Q Replication between remote servers. Objects that are required for the Q Capture program are defined within the queue manager on the source system. Objects that are required for the Q Apply program are defined within the queue manager on the target system. Two channel objects are required to create a transmission path between the source and target systems for data messages and informational messages from the Q Capture program. Two channel objects are also required to create a transmission path from the target system to the source system for control messages from the Q Apply program.

WebSphere MQ objects required for unidirectional replication (same system)

When a Q Capture program replicates data to a Q Apply program on the same system, you need only one queue manager. You can use the same local queue for the send queue and receive queue, and the two programs can share one local administration queue.

You do not need remote queue definitions, transmission queues, or channels.

The following list shows the WebSphere MQ objects that are required for unidirectional Q Replication or event publishing on the same system:

- One queue manager that is used by both the Q Capture program and Q Apply program
- One local queue to serve as both the send queue and receive queue
- One local queue to serve as the administration queue for both the Q Capture program and Q Apply program
- One local queue to serve as the restart queue
- A model queue that Q Apply uses to create dynamic spill queues to store changes to source tables during the target table loading process.

Figure 2 shows the WebSphere MQ objects that are required for unidirectional Q Replication on the same system.

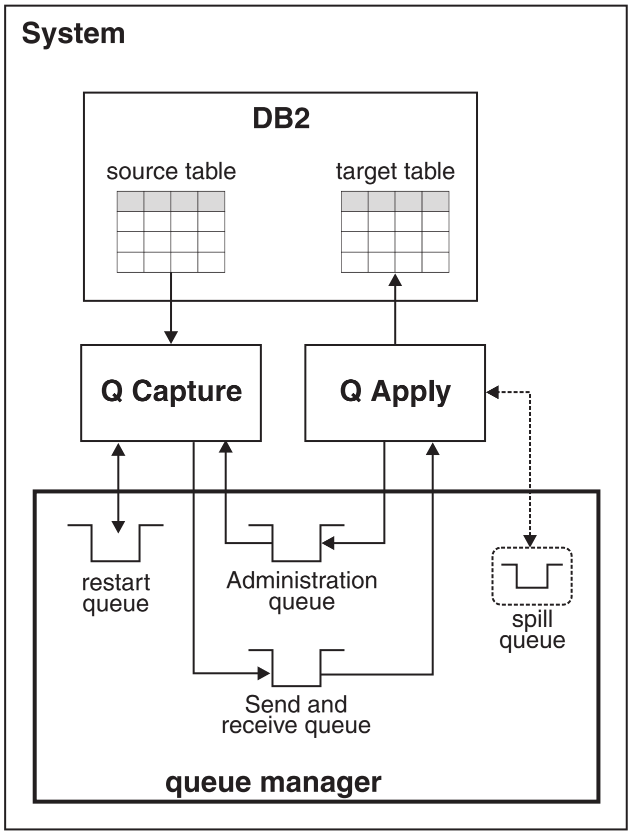


Figure 2. WebSphere MQ objects that are required for unidirectional Q Replication on the same system. When the Q Capture program and Q Apply program run on the same system, only one queue manager is required. One local queue serves as both send queue and receive queue, and another local queue serves as the administration queue for both the Q Capture program and Q Apply program.

When you create control tables for both a Q Capture program and Q Apply program that are replicating on the same system, you specify the same queue manager for both sets of control tables. When you create a replication queue map, you can specify the same local queue for both the send queue and receive queue. The same administration queue that you specify when you create the Q Capture control tables can also be specified as the Q Apply administration queue when you create a replication queue map.

WebSphere MQ objects required for event publishing

Event publishing between remote servers requires a queue manager and queues for the Q Capture program and for the user application. Because the servers are distributed, you also need transmission queues and channels for transmitting transactional data and communicating across a network.

The following lists show the objects that are required for event publishing between two remote servers:

Non-channel objects on source system

- A queue manager

- A remote queue to serve as the send queue (this queue points to a receive queue on the target system)
- A local queue to serve as the administration queue
- A local queue to serve as the restart queue

Non-channel objects on target system

- A queue manager
- A local queue to serve as the receive queue
- A remote queue to serve as the administration queue (this queue points to an administration queue on the source system)

Channel from source to target

- A sender channel that is defined within the source queue manager
- An associated local transmission queue
- A matching receiver channel that is defined within the target queue manager

Channel from target to source

- A sender channel that is defined within the target queue manager
- An associated local transmission queue
- A matching receiver channel that is defined within the source queue manager

Figure 3 on page 14 shows the WebSphere MQ objects that are required for event publishing between remote servers.

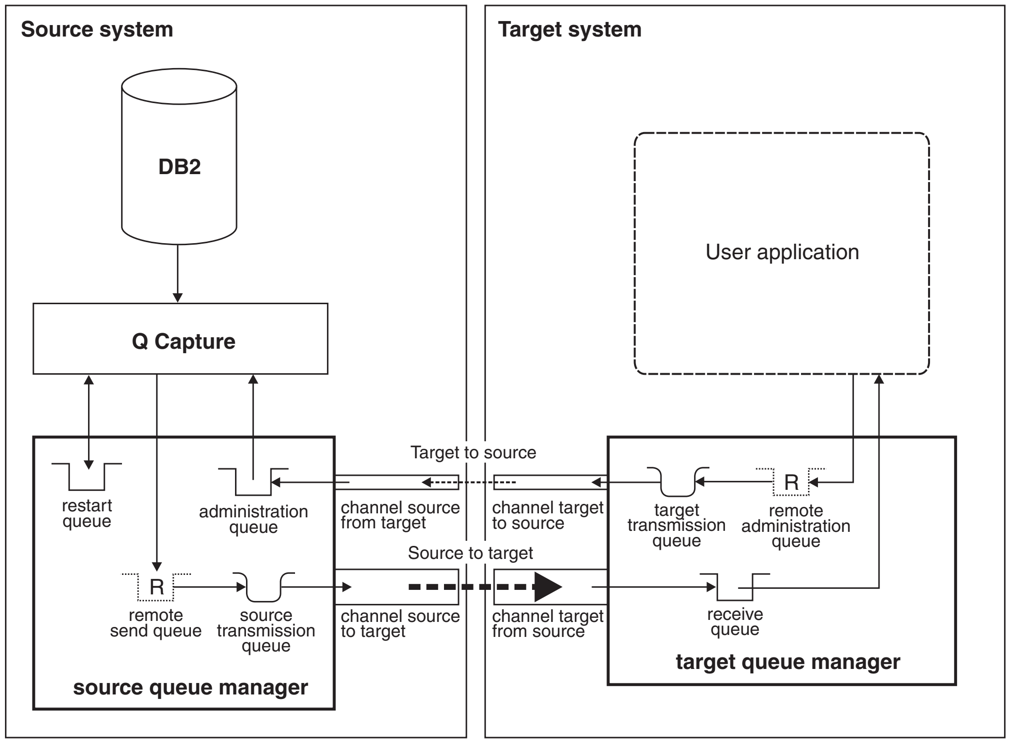


Figure 3. WebSphere MQ objects that are required for event publishing between remote servers. Objects that are required for the Q Capture program are defined within the queue manager on the source system. Objects that are required for the user application are defined within the queue manager on the target system. Two channel objects are required to create a transmission path between the source and target systems for data messages and informational messages from the Q Capture program. Two channel objects are also required to create a transmission path from the target system to the source system for control messages from the user application.

If you create multiple channels from the Q Capture program to the user application, you will need multiple transmission queues to hold messages that are awaiting transit.

WebSphere MQ objects required for bidirectional or peer-to-peer replication (two remote servers)

To replicate transactions in both directions between two servers, you define two sets of the same WebSphere MQ objects that are required for unidirectional replication. There is one exception: Only one queue manager is required on each system.

For example, assume that you plan to replicate transactions between Server A and Server B in both directions. You create the WebSphere MQ objects that link the Q Capture program at Server A with the Q Apply program at Server B. You also create the WebSphere MQ objects that link the Q Capture program at Server B with the Q Apply program at Server A. Server A and Server B each connect to a single queue manager on the systems where they run.

The following lists show the objects that are required for bidirectional or peer-to-peer replication between two remote servers. Because the queue manager is not part of the replication server but runs on the same system, the objects are grouped by system:

Non-channel objects at System A

- A queue manager
- A remote queue definition to serve as the send queue (this queue points to a receive queue at System B)
- A local queue to serve as the administration queue
- A local queue to serve as the restart queue
- A local queue to serve as the receive queue
- A remote queue definition to serve as the administration queue (this queue points to an administration queue at System B)
- A model queue definition for any temporary local spill queues that the Q Apply program creates and uses while it loads target tables

Non-channel objects at System B

- A queue manager
- A remote queue definition to serve as the send queue (this queue points to a receive queue at System A)
- A local queue to serve as the administration queue
- A local queue to serve as the restart queue
- A local queue to serve as the receive queue
- A remote queue definition to serve as the administration queue (this queue points to an administration queue at System A)
- A model queue definition for any temporary local spill queues that the Q Apply program creates and uses while it loads target tables

Channel objects

Channel objects from System A to System B

- A sender channel that is defined within the queue manager at System A
- An associated local transmission queue at System A
- A matching receiver channel that is defined within the queue manager at System B

Channel objects from System B to System A

- A sender channel that is defined within the queue manager at System B
- An associated local transmission queue at System B
- A matching receiver channel that is defined within the queue manager at System A

Figure 4 on page 16 shows the WebSphere MQ objects that are required for bidirectional or peer-to-peer Q Replication between two remote servers.

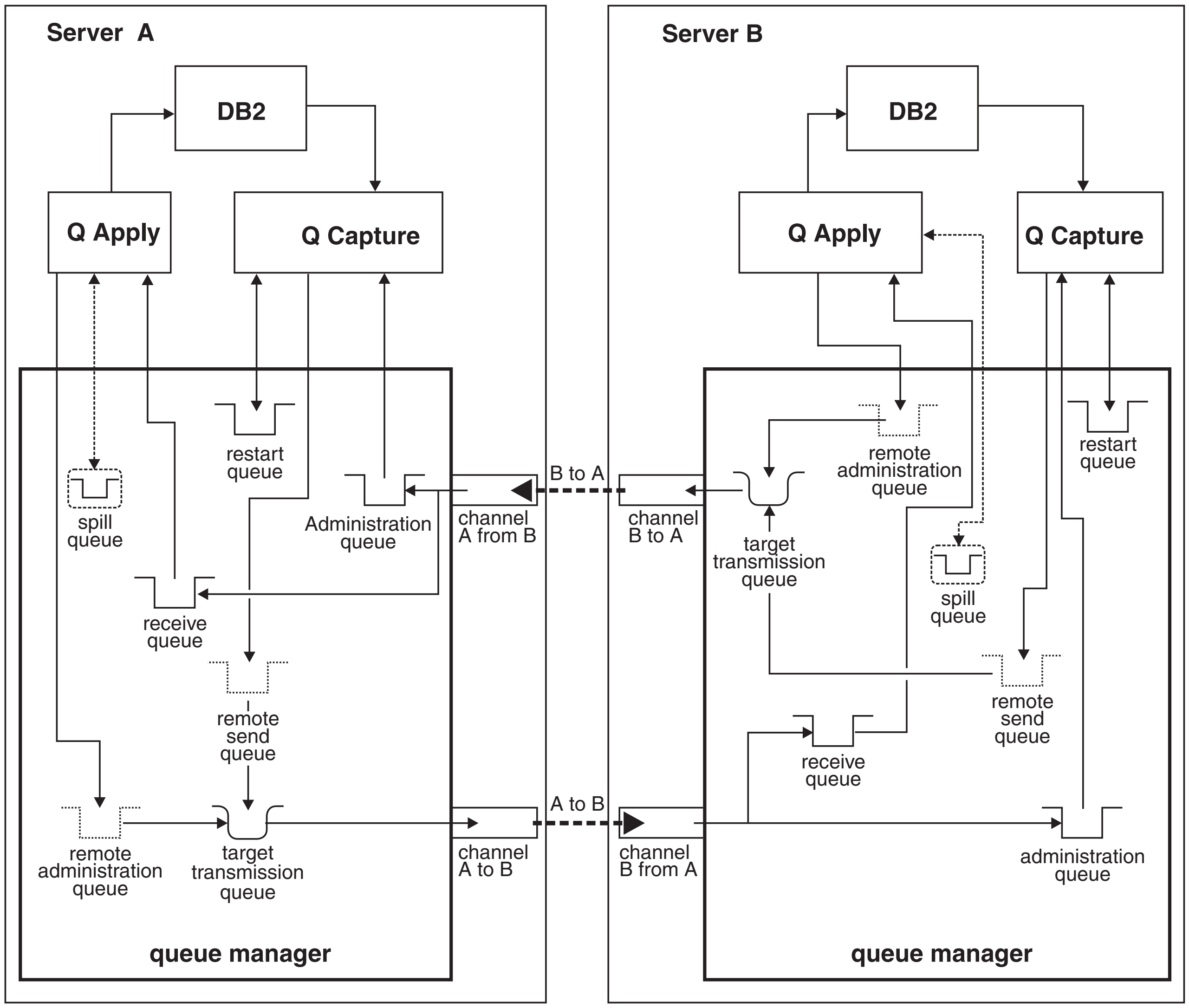


Figure 4. WebSphere MQ objects required for bidirectional or peer-to-peer Q Replication between two remote servers.. You must create two sets of the same WebSphere MQ objects that are required to connect a Q Capture program and a Q Apply program in unidirectional Q Replication. One set of objects handles replication in one direction, and the other set of objects handles replication in the opposite direction. Only one queue manager is required at each system.

WebSphere MQ objects required for peer-to-peer replication (three or more remote servers)

In a peer-to-peer group with three or more remote servers, each server needs one outgoing channel to each additional server in the group. Each server also needs one incoming channel from each additional server in the group.

The Q Apply program at each server requires one remote administration queue per outgoing channel. The Q Capture program requires only one local administration queue because all incoming messages from Q Apply programs are handled by a single queue manager and directed to one queue.

The number of send queues and receive queues depends on the number of servers in the group.

For example, in a group with three remote servers, the Q Capture program at Server A needs two send queues, one for transactions that are going to Server B and one for transactions that are going to Server C. The Q Apply program at Server A needs two receive queues, one for transactions that are coming from Server B and one for transactions that are coming from Server C.

The following lists show the objects that are required at each system in peer-to-peer replication with three or more servers:

Non-channel objects at each system

- One queue manager
- One remote send queue for each outgoing channel
- One local queue to serve as the administration queue for the Q Capture program
- One local queue to serve as the restart queue
- One local receive queue for each incoming channel
- One remote administration queue for the Q Apply program for each outgoing channel
- A model queue definition for any temporary local spill queues that the Q Apply program creates and uses while it loads target tables

Outgoing channel objects at each system

Create these objects for each additional server in the group. For example, in a group with three servers, each server needs two outgoing channels.

- A sender channel that is defined within the local queue manager
- An associated local transmission queue
- A matching receiver channel that is defined within the queue manager on the remote server that this channel connects to

Incoming channel objects at each system

Create these objects for each additional server in the group. For example, in a group with three servers, each server needs two incoming channels.

- A receiver channel that is defined within the local queue manager
- A matching sender channel that is defined within the queue manager on the remote server that this channel connects to

Figure 5 on page 18 shows the WebSphere MQ objects that are required at one server that is involved in peer-to-peer between three remote servers, with one logical table being replicated.

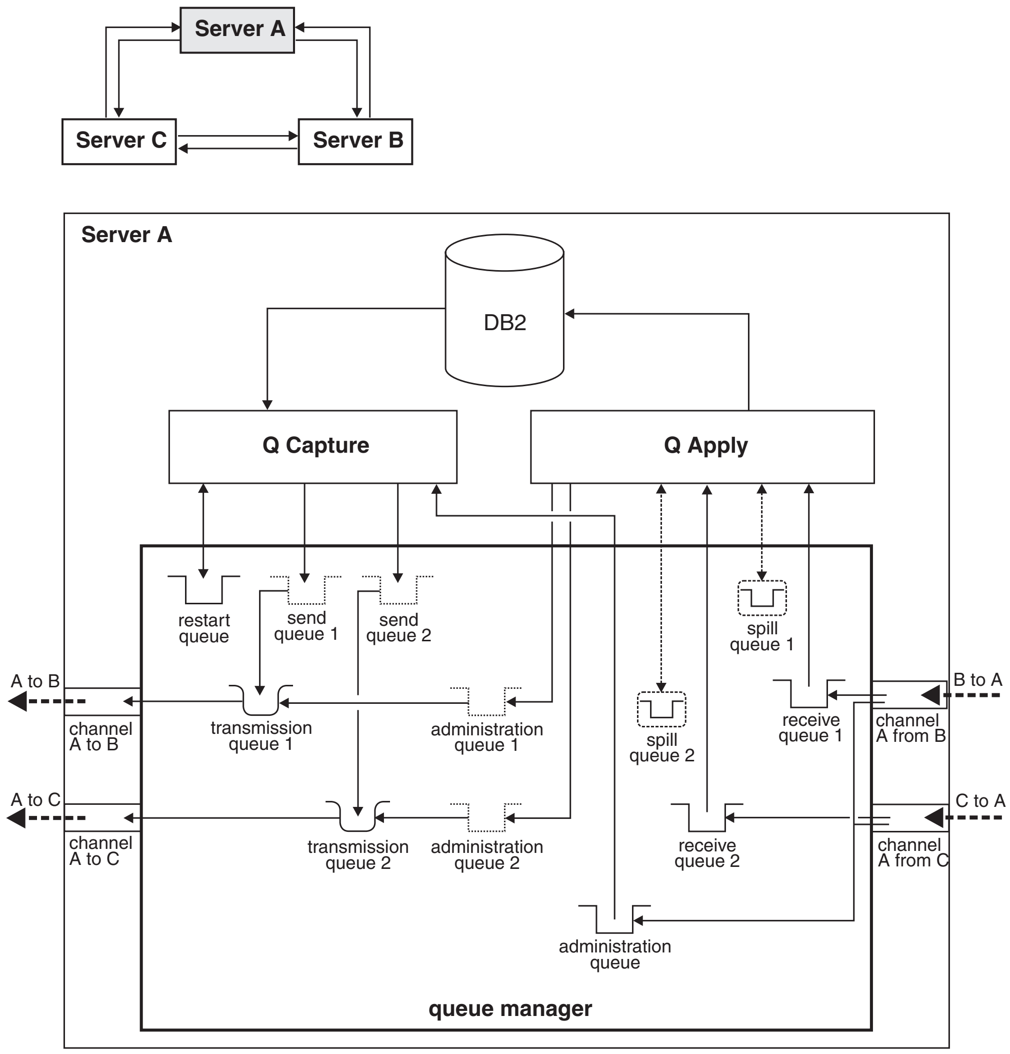


Figure 5. WebSphere MQ objects that are required at one server that is involved in peer-to-peer replication with two other remote servers. At each system, you create one queue manager. The Q Capture program requires one administration queue and one restart queue. You create one remote send queue for each outgoing channel. The Q Apply program requires a remote administration queue for each outgoing channel, and a local receive queue for each incoming channel. You create one outgoing channel and one incoming channel for each additional server in the group.

Required settings for WebSphere MQ objects

The WebSphere MQ objects that are used for Q Replication and Event Publishing must have specific properties.

This topic describes required settings for WebSphere MQ objects that are used in various scenarios, and has the following sections:

- "WebSphere MQ objects at the source" on page 19
- "WebSphere MQ objects at the target" on page 21

Recommendation: Create the queue managers, queues, and other objects for the messaging infrastructure before you create replication objects such as control tables and Q subscriptions, and before you start the Q Capture and Q Apply programs. The following tools and samples are available to help:

- MQ Script Generators that you can use by clicking the Script Generator icon on the Replication Center menu bar
- The CREATE MQ SCRIPT command in the ASNCLP command-line program
- z/OS The ASNQDEFQ sample job in the SASNSAMP data set
- Linux UNIX Windows The asnqdefq sample script in the `SQLLIB/samples/repl/q` directory

Note about persistent messages: Starting with Version 9.7 (or the PTF for APAR level PK85947 or higher on DB2 for z/OS Version 9.1), Q Replication and Event Publishing no longer require persistent WebSphere MQ messages. You can choose to use nonpersistent messages by specifying **`message_persistence`**=n when you start the Q Capture program or by changing the saved value of the MESSAGE_PERSISTENCE column in the IBMQREP_CAPPARMS table. Nonpersistent messages are not logged and cannot be recovered. The Q Capture program always sends persistent messages to its restart queue and the Q Apply program always sends persistent messages to the administration queue, regardless of the setting for **`message_persistence`**. If you create nonpersistent queues with DEFPSIST(N), these persistent messages override the setting for the queue.

For more detail about configuring WebSphere MQ objects, see the WebSphere MQ Information Center at http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp.

WebSphere MQ objects at the source

Table 2 provides required values for selected parameters for WebSphere MQ objects at the source.

Table 2. Required parameter values for WebSphere MQ objects at the source

Object name	Required settings
Queue manager	**MAXMSGL** (The maximum size of messages allowed on queues for this queue manager.) This value must be at least as large as the **`max_message_size`** that you define when you create a replication queue map or publishing queue map. The **`max_message_size`** defines the message buffer that is allocated for each send queue. The value of MAXMSGL should also be at least as large as the MAXMSGL that is defined for each send queue, transmission queue, and the administration queue.
Send queue	**PUT (ENABLED)** Allows the Q Capture program to put data messages and informational messages on the queue.

Table 2. Required parameter values for WebSphere MQ objects at the source (continued)

Object name	Required settings
Administration queue	**GET (ENABLED)** Allows the Q Capture program to get messages from the queue. **PUT(ENABLED)** Allows the Q Apply program to put informational messages on the queue. **SHARE** Enables more than one application instance to get messages from the queue.
Restart queue	**PUT (ENABLED)** Allows the Q Capture programto put a restart message on the queue. **GET (ENABLED)** Allows the Q Capture program to get the restart messages from the queue.
Transmission queue	**USAGE(XMITQ)** Transmission queue. **MAXDEPTH(500000)** The maximum number of messages that are allowed on the transmission queue is 999999. It is unlikely that this maximum would be needed because of the speed at which the Q Apply program keeps up with Q Capture. Use a setting that reflects the transaction workload that you expect. **MAXMSGL** Ensure that the maximum size of messages for the queue is not less than the MAXMSGL defined for the receive queue on the target system, and the value of **`max_message_size`** that you set when you create a replication queue map. **Recommendation:** Use one transmission queue for each send queue-receive queue pair.
Sender channel	**CHLTYPE(SDR)** A sender channel. **DISCINT(0)** Ensure that the disconnect interval is large enough to keep this channel from timing out during periods when there are no transactions to replicate. **CONVERT(NO)** Specify that the sending message channel agent should not attempt conversion of messages if the receiving message channel agent cannot perform this conversion. **HBINT** Coordinate this value with the **`heartbeat_interval`** parameter of the replication queue map or publishing queue map. If you use the HBINT parameter to send heartbeat flows, consider setting **`heartbeat_interval`** to 0 to eliminate heartbeat messages.
Receiver channel	**CHLTYPE(RCVR)** A receiver channel.

WebSphere MQ objects at the target

Table 3 provides required values for selected parameters for WebSphere MQ objects at the target.

Table 3. Required parameter values for WebSphere MQ objects at the target

Object name	Required settings
Queue manager	**MAXMSGL** (The maximum size of messages allowed on queues for this queue manager.) This value must be at least as large as the **`max_message_size`** that you define when you create a replication queue map. The **`max_message_size`** defines the message buffer that is allocated for each send queue.
Receive queue	**GET(ENABLED)** Allows the Q Apply program to get messages from the queue. **PUT(ENABLED)** Allows the Q Capture program to put data and informational messages on the queue. **MAXMSGL** Ensure that the maximum size of messages for the queue is at least as large as the MAXMSGL that is defined for the transmission queue on the source system, and the **`max_message_size`** and MEMORY_LIMIT that you set when you create a replication queue map. **MAXDEPTH(500000)** Set the maximum number of messages that are allowed on the receive queue to a number that reflects your replication workload. **DEFSOPT(SHARED)** Allows multiple Q Apply threads to work with this queue. z/OS **INDXTYPE(MSGID)** Specifies that the queue manager maintain an index of messages based on the message identifier to expedite MQGET operations on the queue.
Administration queue	**PUT(ENABLED)** Allows the Q Apply program or user application to put control messages on the queue.

Table 3. Required parameter values for WebSphere MQ objects at the target (continued)

Object name	Required settings
Model (spill) queue	**Queue name** By default, the Q Apply program looks for a model queue named IBMQREP.SPILL.MODELQ. You can specify a different name for a model queue to be used for a Q subscription when you create or change the Q subscription. **DEFTYPE(PERMDYN)** Specifies that spill queues are permanent dynamic queues. They are created and deleted at the request of the Q Apply program, but they will not be lost if you restart the queue manager. Messages are logged and can be recovered. **DEFSOPT(SHARED)** Allows more than one thread (different agent threads and the spill agent thread) to access messages on the spill queue at the same time. **MAXDEPTH(500000)** This is a recommended upper limit for the number of messages on the spill queue. Adjust this number based on the number of changes that are expected at the source table while the target table is being loaded. **MSGDLVSQ(FIFO)** Specifies that messages on the spill queue are delivered in first-in, first-out order. **Note:** If you have multiple Q Apply programs that share the same queue manager, each Q Apply program must have its own model queue with a name that is unique under that queue manager.
Transmission queue	**USAGE(XMITQ)** Transmission queue.

Sender channel	**CHLTYPE(SDR)**
	A sender channel.
	DISCINT(0)
	Ensure that the disconnect interval is large enough to keep this channel from timing out during periods of inactivity when you expect few control messages to be sent by the Q Apply program or user application.
	CONVERT(NO)
	Specify that the sending message channel agent should not attempt conversion of messages if the receiving message channel agent cannot perform this conversion.

Receiver channel	**CHLTYPE(RCVR)** A receiver channel.

Sample commands for creating WebSphere MQ objects for Q Replication and Event Publishing

You can use WebSphere MQ script (MQSC) commands and system commands to create the WebSphere MQ objects that are required for Q Replication and Event Publishing.

To create queue managers, you can use the following system commands:

Create queue manager

```
crtmqm -lp 50 -ls 10 queue_manager_name
```

This command creates a queue manager and specifies that it use 50 primary log files and 10 secondary log files.

Start queue manager

```
strmqm queue_manager_name
```

After you create and start a queue manager, you can use the MQSC commands in Table 4 and Table 5 on page 24 to create the objects.

The tables contain WebSphere MQ objects that are needed to set up unidirectional replication. You can use the same commands with minor modifications to create objects for multidirectional replication or publishing.

Use the `runmqsc` *`queue_manager_name`* system command to begin an MQSC session, and then issue the MQSC commands in the table interactively or by creating scripts that run at each server.

You can also use the MQ Script Generator tools in the Replication Center to create WebSphere MQ objects at each server.

Objects used by Q Capture program

Table 4. Sample MQSC commands for WebSphere MQ objects that are used by the Q Capture program (assumes a source queue manager named CSQ1 and target queue manager named CSQ2)

Object	Sample command

Send queue
A queue that directs data messages from a Q Capture program. to a Q Apply program or user application. In remote configurations, this is the local definition on the source system of the receive queue on the target system. Each send queue should be used by only one Q Capture program.
	`DEFINE QREMOTE('ASN.SAMPLE_TO_TARGET.DATA')`
	`RNAME('ASN.SAMPLE_TO_TARGET.DATA')`
	`RQMNAME('CSQ2')`
	`XMITQ('CSQ2')`
	`PUT(ENABLED)`

Administration queue
A local queue that receives control messages from a Q Apply program or a user application to the Q Capture program. Each administration queue should be read by only one Q Capture program.
	`DEFINE QLOCAL('ASN.ADMINQ')`
	`PUT(ENABLED)`
	`GET(ENABLED)`
	`SHARE`

Table 4. Sample MQSC commands for WebSphere MQ objects that are used by the Q Capture program (assumes a source queue manager named CSQ1 and target queue manager named CSQ2) (continued)

Object	Sample command

Restart queue
A local queue that holds a single message that tells the Q Capture program where to start reading in the DB2 recovery log for each send queue after a restart. Each Q Capture program must have its own restart queue.
	`DEFINE QLOCAL('ASN.RESTARTQ')`
	`PUT(ENABLED)`
	`GET(ENABLED)`

Transmission queue
A local queue that holds messages that are waiting to go across a channel. This queue can be named for the destination queue manager as a reminder about where its messages go.
Recommendation: Use one transmission queue for each send queue-receive queue pair.
	`DEFINE QLOCAL('CSQ2')`
	`USAGE(XMITQ)`
	`MAXDEPTH(500000)`

Sender channel
The sending end of the channel from the source system to the target system.
	`DEFINE CHANNEL('CSQ1.TO.CSQ2')`
	`CHLTYPE(SDR)`
	`CONNAME('`*`IP_address`* `(`*`port`*`)')`
	`TRPTYPE(TCP)`
	`XMITQ('CSQ2')`
	`DISCINT(0)`
	`CONVERT(NO)`
	
	Where *IP_address* is the IP address of the target system, and *port* is an optional parameter that specifies an unused port on the target system. The default port for WebSphere MQ is 1414.

Receiver channel
The receiving end of the channel from the target system to the source system.
	`DEFINE CHANNEL('CSQ2.TO.CSQ1')`
	`CHLTYPE(RCVR)`
	`TRPTYPE(TCP)`

Objects used by Q Apply program

Table 5. Sample MQSC commands for WebSphere MQ objects that are used by the Q Apply program

Object	Sample command

Receive queue
A queue that receives data and informational messages from a Q Capture program to a Q Apply program or user application. This is a local queue on the target system.
	`DEFINE QLOCAL('ASN.SAMPLE_TO_TARGET.DATA')`
	`GET(ENABLED)`
	`PUT(ENABLED)`
	`DEFSOPT(SHARED)`
	`MAXDEPTH(500000)`

Administration queue
A queue that directs control messages from the Q Apply program or user application to a Q Capture program. In remote configurations, this queue is the local definition on the target system of the administration queue on the source system.
	`DEFINE QREMOTE('ASN.ADMINQ')`
	`RNAME('ASN.ADMINQ')`
	`RQMNAME('CSQ1')`
	`XMITQ('CSQ1')`
	`PUT(ENABLED)`

Table 5. Sample MQSC commands for WebSphere MQ objects that are used by the Q Apply program (continued)

Object	Sample command

Spill queue
A model queue that you define on the target system to hold transaction messages from a Q Capture program while a target table is being loaded. The Q Apply program creates spill queues dynamically during the loading process based on your model queue definition, and then deletes them. The default name for the model queue is IBMQREP.SPILL.MODELQ. When you create or change a Q subscription, you can specify that you want to use a model queue with a different name, and you can specify a different model queue for each Q subscription.
	`DEFINE QMODEL('IBMQREP.SPILL.MODELQ')`
	`DEFSOPT(SHARED)`
	`MAXDEPTH(500000)`
	`MSGDLVSQ(FIFO)`
	`DEFTYPE(PERMDYN)`

Transmission queue
A local queue that holds messages that are waiting to go across a channel. This queue can be named for the destination queue manager as a reminder about where its messages go.
	`DEFINE QLOCAL('CSQ1')`
	`USAGE(XMITQ)`

Sender channel
The sending end of the channel from the target system to the source system.
	`DEFINE CHANNEL('CSQ2.TO.CSQ1')`
	`CHLTYPE(SDR)`
	`CONNAME('`*`IP_address (port)`*`')`
	`TRPTYPE(TCP)`
	`XMITQ('CSQ1')`
	`DISCINT(0)`
	`CONVERT(NO)`
	
	Where *IP_address* is the IP address of the source system and *port* is an optional parameter that specifies an unused port on the source system. The default port for WebSphere MQ is 1414.

Receiver channel
The receiving end of the channel from the source system to the target system.
	`DEFINE CHANNEL('CSQ1.TO.CSQ2')`
	`CHLTYPE(RCVR)`
	`TRPTYPE(TCP)`

Running the replication programs on a WebSphere MQ client

You can run the Q Capture, Q Apply, or Replication Alert Monitor programs on a system that uses a WebSphere MQ client to connect to the queue manager that the replication program works with.

Before you begin

- The user ID for the WebSphere MQ Message Channel Agent (MCA) should be the same user ID that runs the replication programs on the client system. The user ID for the MCA is set by using the MCAUSER parameter in the server-connection channel definition. You can also specify MCAUSER(' '). This is the default for z/OS SVRCONN2.
- Linux UNIX Windows If you changed the default installation path of WebSphere MQ, you must modify the environment variable that points to the WebSphere MQ runtime libraries. See Implications of a 64-bit queue manager in the WebSphere MQ information center for best practices on setting the library path.

- You must define a SVRCONN channel on the system where the queue manager runs that will accept client connections from the system where the Q Capture or Q Apply program runs.
- When you migrate a queue manager to version 7.1, you might need to take additional steps to enable channel authentication. See Channel authentication in the WebSphere MQ information center for details.

Restrictions

- z/OS A WebSphere MQ for z/OS subsystem cannot be a client.
- You cannot use the Replication Center or ASNCLP program to list default queue managers for Q Capture or Q Apply when these programs run on a client.

z/OS

About this task

When you configure a WebSphere MQ client, you set environment variables to point to the system where the queue manager runs, or to a client channel definition table that contains similar information. You also set an environment variable to notify the replication programs that the queue manager is on a server. When these variables are set, the Q Capture, Q Apply, or Replication Alert Monitor program dynamically loads the WebSphere MQ client libraries.

For more information about setting up a WebSphere MQ client-server configuration, see How do I set up a WebSphere MQ client? in the WebSphere MQ information center.

Recommendation: For optimal performance, run the Q Capture and Q Apply programs on the same system as the queue manager that they work with.

Procedure

To run a Q Replication program on a WebSphere MQ client:

1. On the client system, set the replication environment variable *ASNUSEMQCLIENT*=`true`.
2. Define the WebSphere MQ client connectivity using one of the following methods:

Method	Description
***MQSERVER* environment variable**	Set the *MQSERVER* system environment variable to point to the queue manager that the replication program works with. You can set this variable in two different ways, depending on whether you are using a client channel definition table: **No client channel definition table: Point to listener for SVRCONN channel** You specify the listener on the WebSphere MQ server that monitors for incoming client requests in the following format: *channel_name*/*transport_type*/*host*(*port*). For example, to set the *MQSERVER* variable to point to a remote queue manager RQMGR2 with a SVRCONN channel SYSTEM.DEF.SVRCONN on host MQHOST with port 1414: `MQSERVER="SYSTEM.DEF.SVRCONN/TCP/MQHOST(1414)"` **Client channel definition table; use queue manager name only** If you are using a client channel definition table, you need only specify the queue manager name. For example, if the replication program runs on a WebSphere MQ client system and works with the remote queue manager RQMGR2, set `MQSERVER=RQMGR2` on the client system.
***MQCHLLIB* or *MQCHLTAB* environment variables**	These variables are only required when you are using a client channel definition table. Set the *MQCHLLIB* or *MQCHLTAB* environment variables to specify the path to the file containing the client channel definition table. For more details, see "MQCHLLIB" and "MQCHLTAB" in the WebSphere MQ information center. **Note:** Because *MQCHLLIB* and *MQCHLTAB* have platform-specific default values, you can omit setting these environment variables if the client channel definition table is located on the client system in the directory that the MQCHLLIB default value specifies (for example, `/var/mqm/` on Linux or UNIX), and under the file name that the *MQCHLTAB* default value specifies (for example, AMQCLCHL.TAB on Linux or UNIX).

Validating WebSphere MQ objects for Q replication and publishing

You can use the Replication Center to view, select, and validate the settings of the WebSphere MQ queue managers and queues in your configuration. The ASNCLP command-line program can also perform the validation checks. Both tools can send test messages to validate the objects in a replication queue map.

Before you begin

- The queue manager where the objects are defined must be started.
- z/OS To set up administrative access to WebSphere MQ, run the ASNADMSP sample job. For details, see Enabling the replication administration tools to work with WebSphere MQ.
- Linux UNIX On Linux and UNIX, add the path to the WebSphere MQ libraries to the DB2LIBPATH environment variable and stop and start DB2 before you start the Replication Center. Not setting the path can result in unexpected behavior; for example, the number of asnadmt processes might grow without bound.
- Linux UNIX Windows The replication administration stored procedure that is installed to support this function needs a user-defined temporary table space. If no user-defined temporary table space is found with USE permission to the

group PUBLIC or the current user, then a new table space called ASNADMINIFSPACE is created. The user that is connected to the database must have authority to create the stored procedure. If a new table space is created, the user must have authority to create the temporary table space and to grant usage of the table space to ALL.
- If the replication administration tools are running on a WebSphere MQ client system, you must make the client-server environment variables available to DB2 so that the replication administration stored procedure can access them. For details, see "Running the replication programs on a WebSphere MQ client" on page 25.

About this task

If you use the message testing function, the Replication Center or ASNCLP tries to put a test message on the send queue and get the message from the receive queue. The tool also tries to put a test message on the Q Apply administration queue and get the message from the Q Capture administration queue.

Restriction: The message test fails if the Q Capture or Q Apply programs are running.

Procedure

1. To validate WebSphere MQ objects for Q replication and publishing, use one of the following methods:

Method	Description
Replication Center	1. After you specify WebSphere MQ objects on a window or wizard page, click **Validate WebSphere MQ objects** to check that the queue managers and queues have the correct settings. The Replication Center checks the objects and provides messages that describe settings that need to be changed. 2. If you are creating or changing a replication queue map, you can send test messages. In the Validate WebSphere MQ Queues window, select **Send test messages**, and then click **Start**. Messages that show the results of the tests are displayed in the message area of the window.

Method	Description

ASNCLP command-line program	1. Use the **`VALIDATE WSMQ ENVIRONMENT FOR`** command. Messages that show the results of the tests are sent to the standard output (stdout).
	For example, to validate the send queue, receive queue, and Q Apply administration queue that are specified for a replication queue map named SAMPLE_ASN_TO_TARGET_ASN:
	`VALIDATE WSMQ ENVIRONMENT FOR REPLQMAP SAMPLE_ASN_TO_TARGET_ASN`
	2. If you are creating or changing a replication queue map, use the **`VALIDATE WSMQ MESSAGE FLOW FOR REPLQMAP`** command. Messages that show the results of the tests are sent to the standard output (stdout).
	For example, to send test messages between the queues that are specified for a replication queue map SAMPLE_ASN_TO_TARGET_ASN:
	`VALIDATE WSMQ MESSAGE FLOW FOR REPLQMAP SAMPLE_ASN_TO_TARGET_ASN`

2. You can modify the queue manager or queues and use the validation function again to verify your changes.

For more information about WebSphere MQ reason codes, see the WebSphere MQ information center at http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp.

WebSphere MQ validation checks performed by replication tools

When you use the Replication Center or ASNCLP command-line program to validate your WebSphere MQ setup, the tools can check whether the queues have the correct properties and also send test messages to make sure replicated or published data can be transmitted.

The tools use a stored procedure to connect to WebSphere MQ and perform the validation tests. On z/OS, you must install the stored procedure by using the ASNADMSP sample job in the SASNSAMP data set. On Linux, UNIX, and Windows, the stored procedure is installed automatically, but there are prerequisites for using it. See "Validating WebSphere MQ objects for Q replication and publishing" on page 27 for details.

Table 6 describes the validation checks that the tools perform on the WebSphere MQ objects themselves, and Table 7 on page 30 describes the validations checks that are performed to make sure the objects are defined correctly in the control tables.

Table 6. WebSphere MQ validation checks for correct queue properties

ASN message	WebSphere MQ reason code	Validation check
ASN2262E	2058	Does the specified queue manager exist?
ASN2263E	2059	Is the queue manager available for connection?

Table 6. WebSphere MQ validation checks for correct queue properties (continued)

ASN message	WebSphere MQ reason code	Validation check
ASN2264E	2322 (See also Using the command server.)	Is the command server for the queue manager running?
ASN2267E	2085	Does the queue exist in the specified queue manager?
ASN2268E	All reason codes not listed.	Did WebSphere MQ return any of these reason codes?
ASN2272W	2001, 2011, 2042, 2043, 2045, 2046, 2052, 2057, 2091, 2092, 2101, 2152, 2184, 2194, 2196, 2198, 2199, 2201	
ASN2273W	2082	Does the specified alias queue point to the correct base queue?
ASN2274W	None. See Creating a transmission queue.	Does the transmission queue exist for the remote send queue or Q Apply administration queue?
ASN2275W	None. See Altering queue manager attributes or Changing local queue attributes.	Is the maximum message size (MAXMSGL) of the queue larger than the maximum message size (MAXMSGL) of its queue manager?

Table 7. WebSphere MQ validation checks for correct replication setup

ASN message	Validation check
ASN2276W	Is the maximum message size (MAXMSGL) of the send queue smaller than or equal to the MAXMSGL of the receive queue?
ASN2277W	Are the Q Capture administration and restart queues either local queues or alias queues that refer to local queues?
ASN2278W	Is the send queue a local queue, remote queue, or an alias queue that refers to a local or remote queue?
ASN2279W	Is the value of **`max_message_size`** for the queue map smaller than or equal to the maximum message size (MAXMSGL) of the send queue?
ASN2281W	If Q Capture and Q Apply share a queue manager, are the send queue and Q Apply administration queue defined as local queues or alias queues that reference local queues?
ASN2282W	If Q Capture and Q Apply share a queue manager, are the send and receive queue the same local queue or alias queues that refer to the same local queue? Are the administration queues the same local queue or alias queues that refer to the same local queue?

Table 7. WebSphere MQ validation checks for correct replication setup (continued)

ASN message	Validation check
ASN2284W	If Q Capture and Q Apply use different queue managers, are the send queue and Q Apply administration queue either remote queues or alias queues that refer to a remote queue?
ASN2285W	If you specified a load phase for a Q subscription, does a model queue exist for creating spill queues?
ASN2286W	Is the model queue correctly defined for replication?
ASN2287W	Do the send queue, receive queue, and administration queues have the correct settings for replication?

Connectivity and authorization requirements for WebSphere MQ objects

Before you can replicate or publish data, you must configure connections between queue managers on the systems where the Q Replication and Event Publishing programs run. Also, ensure that user IDs that run the replication and publishing programs are authorized to perform required actions on WebSphere MQ objects. This topic describes the connectivity requirements and authorization requirements.

Connectivity requirements

Queue managers on each system that is involved in replication or publishing must be able to connect to each other. In distributed environments, the Q Capture program, Q Apply program, and user applications communicate by connecting to queue managers and sending messages through remote queue definitions, transmission queues, and channels.

Q Replication and Event Publishing also support clustering, where a group of queue managers communicate directly over the network without the use of remote queue definitions, transmission queues, and channels.

Client-server connections, where the queue manager runs on a different system than the Q Capture program or Q Apply program for which it is managing queues, are also supported.

For details about various queue manager configurations and how to set up connections for each, see *WebSphere MQ Intercommunication*.

Authorization requirements

WebSphere MQ queues are the primary means of data exchange and communication that is used by the Q Capture and Q Apply programs, and these programs must be able to access data on the queues.

When you create WebSphere MQ objects, ensure that the user IDs that operate the replication programs have the authority to perform required actions on these objects. The following list summarizes these requirements for the Q Capture program, Q Apply program, and Replication Alert Monitor.

Note: Security administrators add users who need to administer WebSphere MQ to the mqm group. This includes the root user on Linux and UNIX systems. Any changes that you make to the authorities or membership of the mqm group are not recognized until the queue manager is restarted, unless you issue the MQSC command **REFRESH SECURITY**, or the programmable command format (PCF) equivalent.

Authorization requirements for the Q Capture program

User IDs that run a Q Capture program must have authority to:

- Connect to the queue manager (MQCONN or MQCONNX) on the system where the Q Capture program runs.
- Perform the following actions on the send queue: open (MQOPEN), inquire about attributes (MQINQ), put messages (MQPUT), commit messages (MQCMIT), and roll back messages (MQBACK).
- Perform the following actions on the Q Capture administration queue: open (MQOPEN), inquire about attributes (MQINQ), and get messages (MQGET).
- Perform the following actions on the restart queue: open (MQOPEN), inquire about attributes (MQINQ), put messages (MQPUT), and get messages (MQGET).
- Perform the following actions on the transmission queue: open (MQOPEN), put messages (MQPUT), inquire about attributes (MQINQ).

Authorization requirements for the Q Apply program

User IDs that run a Q Apply program must have authority to:

- Connect to the queue manager (MQCONN or MQCONNX) on the system where the Q Apply program runs.
- Perform the following actions on the receive queue: open (MQOPEN), inquire about attributes (MQINQ), and get messages (MQGET).
- Perform the following actions on the Q Apply administration queue: open (MQOPEN), inquire about attributes (MQINQ), and put messages (MQPUT).
- Perform the following actions on temporary spill queues: open (MQOPEN), put messages (MQPUT), get messages (MQGET), delete (dlt), change (chg), and clear (clr).

Authorization requirements for the Replication Alert Monitor

If a Replication Alert Monitor is used to monitor the number of messages on the receive queue (QAPPLY_QDEPTH alert condition) or the number of messages on the spill queue (QAPPLY_SPILLQDEPTH alert condition), the user ID that runs the monitor must have authority to connect to the queue manager on the system where the Q Apply program runs.

For both the Q Capture program and Q Apply program, the user ID that is associated with Message Channel Agents (MCAs) must have the authority to:

- Connect to the local queue manager.
- Perform the following actions on the local transmission queue: open (MQOPEN) and put messages (MQPUT).

For information about WebSphere MQ authorization and privileges, see *WebSphere MQ Security*.

Storage requirements for WebSphere MQ for Q Replication and Event Publishing

Plan the WebSphere MQ resources to achieve the desired level of resilience to network outages, target outages, or both. If messages cannot be transported, more resource is used at the source. If messages cannot be applied, more resource is used at the target.

By default, messages that are used in Q Replication and Event Publishing are persistent. WebSphere MQ writes all persistent messages to logs. If you restart a queue manager after a failure, the queue manager retrieves all of the logged messages as necessary. More storage is required for persistent messages. You can also choose to use nonpersistent messages by specifying **`message_persistence`**=n when you start the Q Capture program.

For more information about WebSphere MQ log files, see the WebSphere MQ Information Center at http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp.

WebSphere MQ message size

You can limit the size of WebSphere MQ messages when you create queues and queue managers, and also when you set up replication or publishing. You must coordinate the message size limits between WebSphere MQ and Q Replication and Event Publishing.

In WebSphere MQ, you define the MAXMSGL (maximum message length) to limit the size of messages.

The following list describes how MAXMSGL relates to memory limits for the Q Capture program and the Q Apply program.

Q Capture program
: You can limit the amount of memory that a Q Capture program uses to buffer each message before putting it on a send queue. You define this MAX_MESSAGE_SIZE when you create a replication queue map or publishing queue map. The default is 64 KB.

  Important: If you allow a larger message buffer for the Q Capture program than the queues are set up to handle, replication or publishing cannot occur. If the send queue is remote from the receive queue, the value of MAX_MESSAGE_SIZE that is specified for the replication queue map or publishing queue map and stored in the IBMQREP_SENDQUEUES table must be at least 4 KB smaller than the MAXMSGL attribute of both the transmission queue and the queue manager. This 4 KB difference accounts for the extra information that is carried in the message header while the message is on the transmission queue. If the send and receive queues are defined within the same queue manager, you can use the same value for MAX_MESSAGE_SIZE as the value for MAXMSGL for the queues.

Q Apply program
: You can limit the amount of memory that a Q Apply program uses to buffer multiple messages that it gets from a receive queue before agent threads reassemble the messages into transactions. You set the MEMORY_LIMIT option when you create a replication queue map. The default is 2 MB.

Attention: Ensure that the MAXMSGL for the local queue that will serve as a receive queue is not larger than the MEMORY_LIMIT for the replication queue map that contains the receive queue.

Message segmentation

The Q Capture program automatically divides transactions that exceed the MAX_MESSAGE_SIZE into multiple messages by breaking up the transaction at a row boundary. If you are replicating or publishing data from large object (LOB) columns in a source table, the Q Capture program automatically divides the LOB data into multiple messages. This ensures that the messages do not exceed the MAX_MESSAGE_SIZE that is defined for the replication queue map or publishing queue map that contains each send queue.

On some operating systems, WebSphere MQ allows you to define message segmentation so that messages that are too large for queues or channels are automatically divided. Q Replication and Event Publishing do not use this message segmentation feature. If you set up WebSphere MQ objects to use message segmentation, you must still ensure that the MAXMSGL for queues is equal to or larger than the MAX_MESSAGE_SIZE for the replication queue map or publishing queue map.

For more information about message size, see the *WebSphere MQ System Administration Guide* for your platform.

Queue depth considerations for large object (LOB) values

Large object (LOB) values from a source table are likely to exceed the maximum amount of memory that a Q Capture program allocates as a message buffer for each send queue.

The default MAX_MESSAGE_SIZE (message buffer) for a send queue is 64 kilobytes. In DB2, LOB values can be up to 2 gigabytes so LOB values will frequently be divided into multiple messages.

If you plan to replicate LOB data, ensure that the MAXDEPTH value for the transmission queue and administration queue on the source system, and the receive queue on the target system, is large enough to account for divided LOB messages. You can reduce the number of messages that are required to send LOB data by increasing the MAX_MESSAGE_SIZE for the send queue when you create a replication queue map or publishing queue map.

A large LOB value that is split based on a relatively small message buffer will create a very large number of LOB messages that can exceed the maximum amount of messages (MAXDEPTH) that you set for a transmission queue or receive queue. This will prompt a queue error. When a remote receive queue is in error, the message channel agent on the target system sends a WebSphere MQ exception report for each message that it is unable to deliver. These exception reports can fill the Q Capture program's administration queue.

Queue manager clustering in Q Replication and Event Publishing

Q Replication will work in a queue manager cluster environment. Clustering allows a group of queue managers to communicate directly over the network, without the need for remote queue definitions, transmission queues, and channels.

A cluster configuration allows you to create multiple instances of a queue with the same name on multiple queue managers in the same cluster. However, in Q Replication, the receive queue on the target system must be defined only once within a cluster. The Q Capture and Q Apply programs use a dense numbering system to identify and retrieve missing messages. (Each message is assigned a positive integer with no gaps between numbers.) Receive queue names must be unique for a given pair of Q Capture and Q Apply programs. If two receive queues have the same name, the dense numbering system will not work.

Q Replication will not work in conjunction with cluster distribution lists, which use a single MQPUT command to send the same message to multiple destinations.

For more information, see *WebSphere MQ Queue Manager Clusters*.

Chapter 4. Configuring databases for Q Replication and Event Publishing (Linux, UNIX, Windows)

Before you can replicate data, you must set environment variables and configure the source and target databases. For event publishing, you need to configure only the source database.

Required: Setting DATA CAPTURE CHANGES on DB2 source tables and DB2 for z/OS system tables

You must set the DATA CAPTURE CHANGES attribute on any DB2 table that you want to replicate. Also, on DB2 for z/OS Version 9 and later, you must set DATA CAPTURE CHANGES on the SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, and SYSIBM.SYSTABLEPART system catalog tables.

About this task

Setting DATA CAPTURE CHANGES on source tables prompts DB2 to log SQL changes in an expanded format that is required for replication. The replication administration tools will generate the DDL to alter the table if this option is not set. However, you can set it when creating tables or alter the table yourself.

For DB2 for z/OS system tables, setting DATA CAPTURE CHANGES enables detection and replication of changes to the structure of source tables such as addition of new columns or changes in column data types.

Note: If the replication source is DB2 for z/OS Version 9 or later, the Q Capture program stops if DATA CAPTURE CHANGES is not set on the SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, and SYSIBM.SYSTABLEPART system catalog tables.

Turning on DATA CAPTURE CHANGES for a table introduces a small amount of extra logging. When this option is set for a table, DB2 logs both the full before and after images of the row for each update. When the option is not set, DB2 logs only the columns that changed because this amount of logging is all that is needed for DB2 recovery. The amount of additional logging is proportional to row size, but only for those tables for which DATA CAPTURE CHANGES is on.

Restrictions

z/OS If the table was altered, you cannot set DATA CAPTURE CHANGES on that table or any other table in the same table space until the table space is reorganized.

Procedure

Use the CREATE TABLE or ALTER TABLE statement to set DATA CAPTURE CHANGES on replication source tables. Use the ALTER TABLE statement to set DATA CAPTURE CHANGES on DB2 for z/OS system catalog tables.
When you create control tables with the replication administration tools at Version 10 on z/OS, the tools check for the DATA CAPTURE CHANGES attribute on the

required system tables. If the attribute is not set, the generated SQL for creating control tables includes the following ALTER TABLE statements:

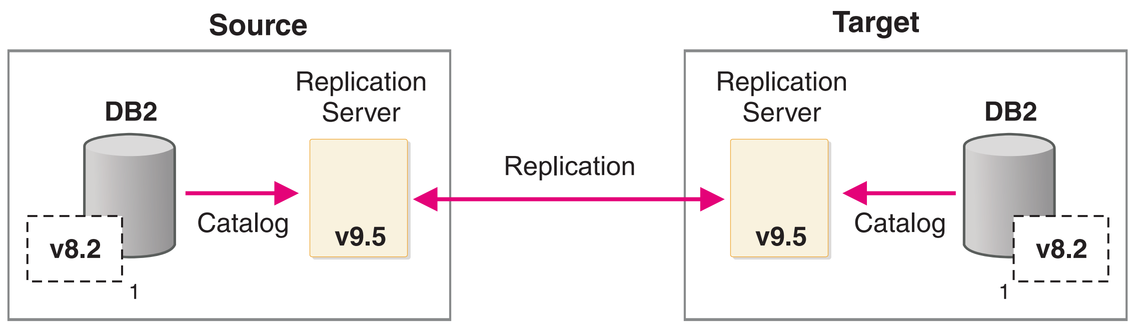
```
ALTER TABLE SYSIBM.SYSTABLES DATA CAPTURE CHANGES
ALTER TABLE SYSIBM.SYSCOLUMNS DATA CAPTURE CHANGES
ALTER TABLE SYSIBM.SYSTABLEPART DATA CAPTURE CHANGES
```

If you are not creating control tables, you can use these same statements to set this required attribute.

Configuring for older or remote DB2 databases

You can configure InfoSphere® Replication Server or Data Event Publisher with some older DB2 databases so that you can use the most recent replication features without migrating these existing databases. You can also configure InfoSphere Replication Server or Data Event Publisher to run on a different server than the servers that host your DB2 databases. Both configurations use similar steps.

The following figure shows the basic configuration for using InfoSphere Replication Server with an older version of DB2, in this example, InfoSphere Replication Server Version 9.5 with DB2 Version 8.2.



(1)This configuration is currently valid with DB2 versions down to v8.2

Figure 6. A configuration of Replication Server with older DB2 databases

The following figure shows the basic configuration for using remote DB2 databases.

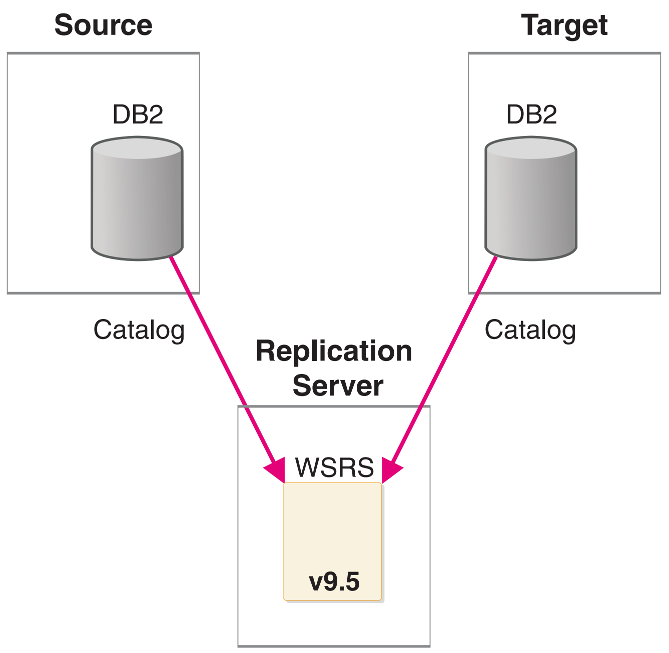


Figure 7. A configuration with remote DB2 databases

Before you begin

If you plan to use a newer version of Replication Server or Data Event Publisher on the same system as an older DB2 database, see Installation scenarios for important considerations during installation.

If you plan to use a Q Capture that is a different version or release than your Q Apply, see Coexistence support in Version 9.7 Q Replication and Event Publishing

About this task

Installing Replication Server or Data Event Publisher does not upgrade any existing DB2 databases, even if an existing DB2 is an older version. After InfoSphere Replication Server or Data Event Publisher is installed and licensed, the replication programs can work with any DB2 database that is at the same release or lower (down to Version 8.2), regardless whether that database resides in another DB2 instance or on another system. A new DB2 instance is created as part of the installation, however the new DB2 instance never needs to be started and no database needs to be created in it.

The following extra steps are required in a mixed-version or remote replication configuration:

- The older or remote DB2 databases and nodes must be cataloged at the instance that contains the replication programs.
- In mixed-version configurations, the replication control tables must be created to match the higher-level version of the replication programs.
- In mixed-version configurations, the instance owner of the higher-level DB2 that is installed with the latest replication programs must have permission to read the DB2 logs of the older version DB2.

Restrictions

The replication administration tools will not create pre-Version 10.1 control tables for a source or target database on DB2 for Linux, UNIX, and Windows Version 10.1.

For Q Capture or Q Apply that are local to DB2

- Replication from compressed or partitioned tables on DB2 for Linux, UNIX, and Windows is only supported for DB2 Version 9.7 or later. DB2 uncompresses log records before passing them to the Q Capture program. The data is passed uncompressed from source to target and the target does not need to have compression set.
- To replicate load operations, the source database and Q Capture must be at APAR PK78558 or newer on z/OS, or at Version 9.7 or newer on Linux, UNIX, and Windows.

For Q Capture or Q Apply that are remote from DB2

- The Q Capture program, Q Apply program, and the source and target databases must be on the same Linux, UNIX, or Windows platform and must all have the same bit architecture, for example 32/32 bit or 64/64 bit.
- You cannot use the replication administration tools to list or validate queues.
- The endianness and code page of the system where the remote Q Capture program runs must match the endianness and code page of the source database and operating system.

Procedure

1. Install Replication Server or Data Event Publisher and create an instance as prompted by the installer. You can install the products on the same system as an older DB2 version (down to Version j8.2) or you can install on a remote system.
2. Catalog the DB2 node and database that is your source or target at the newly installed instance by using the **`CATALOG NODE`** and **`CATALOG DATABASE`** commands. Open a DB2 command window from the DB2 instance that is used by replication or event publishing, and catalog the node and database of the older or remote database.
3. Create an `asnpwd` password file to be used by the replication and publishing programs.
4. If the source or target database is a lower version than Q Capture or Q Apply, use the ASNCLP or Replication Center to create replication control tables that match the version of the replication programs.
5. Use the Replication Center or ASNCLP to create the queue maps and Q subscriptions.
6. If the database is to be a replication source, grant SYSADM or DBADM permission to the user ID with which the Q Capture program connects to the source. This gives the Q Capture program permission to retrieve DB2 log records.
7. Start Q Capture and Q Apply from the replication instance, whether it is a new version or the same version on a remote system. For example, start replication from the newly installed instance by providing the alias for the older or remote DB2 database that you specified in the **`CATALOG DATABASE`** command.

Configuring for GDPS Active/Active (z/OS)

Q Replication provides data synchronization for GDPS® Active/Active Sites and a Network Management Interface (NMI) for monitoring the replication environment from IBM® Tivoli® NetView® for z/OS.

The following sections provide detail about configuration and security for Q Replication with GDPS Active/Active Sites.

Workload name

For Active/Active Sites, the workload name corresponds to the name of the replication queue map that identifies the queues for sending and receiving messages between sites. One replication queue map is required between the paired Q Capture and Q Apply programs at each site. The two workloads that make up the two directions of the bidirectional configuration are required to have the same workload name and therefore the same queue map name. This requirement enables NetView to identify the workload pair.

The replication queue map name is restricted to not exceed 64 characters in length because of the limit for the workload name. The queue map name must start with an alphabetical character, and the remaining characters can be alphanumeric. The only special character that is supported is the underscore (_).

Q replication reports workload metrics only for workloads that are active at the time of request for metrics through the Network Management Interface. Nothing will be reported for workloads that are inactive or those that were stopped since the last request. For metrics that are reported as incremental counters in the common NetView workspaces, Q Replication reports both the incremental counters from the sampling interval between successive NMI requests and the cumulative totals for the counter since the workload was started.

Network Management Interface for monitoring replication

You can use Tivoli NetView for z/OS to monitor the health of the Q Replication environment between sites in a GDPS Active/Active Sites solution. Tivoli monitoring provides a unified and high-level view of the performance and status of the different replication products that are part of the GDPS A/A solution (IMS™ Replication, Q Replication, Load Balancer Advisor). For more detailed replication performance information, tuning, and troubleshooting, you would typically use the Q Replication Dashboard.

Monitoring from Tivoli is performed using a new interface, Network Management Interface, which is a request-reply protocol that uses an AF_UNIX socket. GDPS A/A does not query the Q Capture or Q Apply monitor tables; instead, it uses this new interface to issue NMI requests to the Q Capture and Q Apply programs to obtain performance metrics.

The Network Management Interface is implemented by the Q Apply and Q Capture programs, through which NMI client applications can request operational metrics that include status and performance information, for example replication latency. To enable this interface, you must set two parameters for both the Q Capture program and Q Apply program, **nmi_enable** and **nmi_socket_name**.

nmi_enable
The **nmi_enable** parameter specifies whether the Q Capture or Q Apply

program is enabled to provide a Network Management Interface for monitoring Q Replication statistics from Tivoli NetView. The NMI client application must be on the same z/OS system as the Q Capture or Q Apply program.

nmi_socket_name
The **nmi_socket_name** parameter specifies the name of the AF_UNIX socket where the Q Capture or Q Apply program listens for requests for statistical information from NMI client applications. You can specify this parameter to change the socket name that the program automatically generates.

For more information on these two parameters, see the following topics:
- "Descriptions of asnqcap parameters" on page 351
- "Descriptions of asnqapp parameters" on page 391

When you start the Q Capture or Q Apply programs with the **nmi_enable** or **nmi_socket_name** parameters, the values are used during the current session for these programs. To save the parameter values between sessions, you must update the respective parameter tables, IBMQREP_CAPPARMS for the Q Capture program and IBMQREP_APPLYPARMS for the Q Apply program. Each of these tables includes an NMI_ENABLE column and NMI_SOCKET_NAME column, which are described in the following topics:
- "IBMQREP_CAPPARMS table" on page 464
- "IBMQREP_APPLYPARMS table" on page 504

Use SQL to update the value of these parameters in the control tables. For example:

```
UPDATE schema.IBMQREP_CAPPARMS SET NMI_ENABLE=Y
```

Where *schema* is the schema name of the control tables for this Q Capture instance.

Security

When a NetView client connects, the Q Capture or Q Apply program authenticates the client against the security facility that is installed on the system, for example RACF®, TOP SECRECT, or ACF2. The programs use the following protocol:
- The program obtains the security identifier for the connected client by an ioctl() call with command SECIGET.
- A call to RACROUTE checks for authorization of the client.
- A client is considered authorized when the security identifier has READ access on the following profiles in the SERVERAUTH resource class:
  - `QREP.NETMGMT.`*`sysname`*`.`*`procname`*`.QAPPLY.DISPLAY`
  - `QREP.NETMGMT.`*`sysname`*`.`*`procname`*`.QCAPTURE.DISPLAY`
- If the client cannot access this profile, Q Replication performs another authorization check to see whether the client has superuser authority (effective UID of zero or permission to the BPX.SUPERUSER resource in the FACILITY class). If so, the client is authorized.

Setting environment variables (Linux, UNIX, Windows)

You must set environment variables before you operate the Q Capture program, the Q Apply program, or the Replication Alert Monitor program, before you use the ASNCLP command-line program or Replication Center, or before you issue system commands.

Procedure

To set the environment variables:

1. Set the environment variable for the DB2 instance name (DB2INSTANCE) that contains the Q Capture server, Q Apply server, and Monitor control server.
   - Linux UNIX For Linux and UNIX, use the following command:
     ```
     export DB2INSTANCE=db2_instance_name
     ```
   - Windows For Windows, use the following command:
     ```
     SET DB2INSTANCE=db2_instance_name
     ```
2. If you created the source database with a code page other than the default code page value, set the DB2CODEPAGE environment variable to that code page.
3. Optional: Set environment variable for the default DB2 database (DB2DBDFT) to the Q Capture server or Q Apply server.
4. Linux UNIX Windows Make sure that the system variables include the directory where the Q Replication program libraries and executable files are installed. In the DB2 instance home directory, the default library path is `SQLLIB/lib` and the default executable path is `SQLLIB/bin`. If you moved the libraries or executable files, update your environment variables to include the new path.
5. **AIX® and DB2 Extended Server Edition**: Set the EXTSHM environment variable to ON at the source and target databases on AIX, or at the source database only on DB2 Extended Server Edition (if the Q Capture program must connect to multiple database partitions), by entering the following commands:
   ```
   $ export EXTSHM=ON
   $ db2set DB2ENVLIST=EXTSHM
   ```
   a. Ensure that the EXTSHM environment variable is set each time you start DB2. Do this by editing the`/home/`*`db2inst`*`/sqllib/profile.env` file where *db2inst* is the name of the DB2 instance that contains the target database. In the file, add or change the line: `DB2ENVLIST='EXTSHM'`

   b. Add the following line to the `/home/`*`db2inst`*`/sqllib/userprofile` file: `export EXTSHM=ON`

Setting the TMPDIR environment variable (Linux, UNIX)

You can specify a directory for the Q Capture and Q Apply programs to write temporary files. Writing these files to a specified directory can protect them from accidentally being deleted.

About this task

By default, replication programs use the `/tmp` directory for temporary files. In some cases, these files might be deleted by other programs with root privilege. For example, Linux or UNIX system administrators typically run time-based service jobs to remove files in the `/tmp` directory.

Missing temporary files can prevent programs from communicating. For example, if you issue the **`asnqacmd stop`** command to stop the Q Apply program and a temporary file is missing, the command fails.

To avoid accidental deletion, you can use the *TMPDIR* environment variable to specify a temporary directory.

Note: User IDs that run the replication and publishing programs must have write access to either the `/tmp` directory or the directory specified by the *TMPDIR* variable.

Procedure

To set the TMPDIR environment variable, specify a directory that is accessible to the user ID that runs the replication or publishing programs. Ensure that files cannot be deleted by other user IDs.
For example, the following command specifies the `/home/repldba/tempfiles/` directory:

```
export TMPDIR=/home/repldba/tempfiles/
```

Addressing memory issues for Q Capture and Q Apply (AIX)

On AIX operating systems, you can ensure that the replication programs have enough memory to operate by setting environment variables in addition to having the right settings for Q Replication, WebSphere MQ, and DB2.

About this task

Setting the *EXTSHM* and *LDR_CNTRL* environment variables and using the **`ulimit`** command might help the problems with memory. You can also increase the MAXAGENTS parameter for DB2.

Procedure

To address memory issues for Q Capture and Q Apply:

1. Ensure that the following AIX system environment variables are maximized:

   EXTSHM

   The *EXTSHM* (Extended Shared Memory) variable essentially removes the limitation of only 11 shared memory regions. 64-bit processes are not affected by *EXTSHM*.

   a. Set *EXTSHM* to ON at the source and target databases, or at the source database only on if the Q Capture program must connect to multiple database partitions, by entering the following command:
      ```
      $ export EXTSHM=ON
      $ db2set DB2ENVLIST=EXTSHM
      ```
   b. Ensure that *EXTSHM* is set each time you start DB2. Do this by editing the `/home/`*`db2inst`*`/sqllib/profile.env` file, where *db2inst* is the name of the DB2 instance that contains the target database. In the file, add or change the line: `DB2ENVLIST='EXTSHM'`
   c. Add the following line to the `/home/db2inst/sqllib/userprofile` file:
      ```
      export EXTSHM=ON
      ```

   LDR_CNTRL

   You can use the *LDR_CNTRL* environment variable to configure the Q Apply program to use large pages for its data and heap segments.

   Note: Take precautions before setting this value. If needed, check with AIX support or your system administrator before making this change to the AIX system.

For example, the first command adds four segments and the second command adds three segments:

```
export LDR_CNTRL=MAXDATA=0x40000000

export LDR_CNTRL=MAXDATA=0x30000000
```

`ulimit` command
: The **`ulimit`** command sets or reports user process resource limits, as defined in the `/etc/security/limits` file. The `ulimit -a` value for the DB2 instance owner means that the data and stack are not set to unlimited. For 64-bit instances on AIX servers, there is no constraint on database shared memory.

  Recommendation: Make sure all **`ulimit`** parameters are set to unlimited. Sometimes data/stack/nofiles are limited and cannot be set to unlimited. In this case, consult the system administrator.

2. Ensure that you have the correct WebSphere MQ settings. If WebSphere MQ is a 32-bit client (Version 5.3) , and AIX and DB2 are 64 bit, then the DB2 database that contains the Q Capture or Q Apply control tables should be cataloged using a loopback connection. If WebSphere MQ is V6.0 and DB2 is 32-bit, then you also might need to set up a loopback connection. Loopback is costly (TCP/IP is costly), but if nothing works, try the loopback connection to resolve issues such as SQL1224N.
3. Ensure that you have the correct DB2 settings.
   a. Check **`db2set`** and make sure *EXTSHM* is exported.
   b. Check for errors caused by limits on operating system process, thread, or swap space. These type of errors can be identified by a SQL1225N message. If you receive this message, check the current value of the **`maxuproc`** parameter, which specifies the maximum number of processes per user ID. The following command checks the **`maxuproc`** value:
      ```
      lsattr -E -l sys0
      ```
      Set **`maxuproc`** to a higher value using the following command as user root:
      ```
      chdev -l sys0 -a maxuproc='new_value'
      ```
   c. Increase the setting for the MAXAGENTS parameter in the database manager (DBM) configuration.

      When the number of worker agents reaches the MAXAGENTS value, all subsequent requests that require a new agent are denied until the number of agents falls below the value. This value applies to the total number of agents, including coordinating agents, subagents, inactive agents, and idle agents, that are working on all applications. The value of MAXAGENTS should be at least the sum of the value for the MAXAPPLS parameter in each database that is allowed to be accessed concurrently.
4. Ensure that the **`memory_limit`** parameter for a replication queue map is set to 32MB (the default) or lower. This is the current recommended value for each receive queue.

Configuring the source database to work with the Q Capture program (Linux, UNIX, Windows)

If archive logging is not enabled at the source database, you must enable it to ensure that log entries will not be overwritten before a Q Capture program reads them. For this change to take effect, you must also perform an offline backup of the source database.

About this task

Important: Backing up a large database can take a long time. During this process, applications will be disconnected from the database and new connections will not be allowed.

Procedure

You can configure the source database from the Replication Center or command line:

To configure a DB2 database to run the Q Capture program, use one of the following methods:

Method	Description
Replication Center	Use the Turn On Archive Logging window to enable archive logging. To open the window, right-click the Q Capture server that you want to enable and select **Enable Database for Replication**. **Attention:** If you click the **Use Configure Logging wizard** check box, you must select the **Archive logging** radio button on the Logging type page. The only option on the wizard that affects replication is the option to turn on archive logging.
Command line	1. Check the "Log retain for recovery status" value in the database configuration. If it is set to NO, turn on archival logging by setting the LOGARCHMETH1 database configuration parameter to a value other than OFF. 2. Optional: Use the **`update database configuration`** command to increase values for the source database depending on your replication or publishing needs. The following database configuration values are adequate for many large replication scenarios (if the database is configured to work with other applications, your values might already be larger than these) : APPLHEAPSZ 4096, LOGFILSIZ 2000, LOGPRIMARY 20, LOCKLIST 200, DBHEAP 1000, STMTHEAP 4096, LOGBUFSZ 64, MAXAPPLS 300. You might need to adjust the value of LOCKTIMEOUT if Q Capture replicates large object (LOB) or XML data and you experience lock timeouts. For the LOGSECOND parameter, a value of 20 is adequate for most scenarios. If you expect to replicate long transactions, it is recommended that you set LOGSECOND = -1 on DB2 Version 8.1 or newer to avoid most log space problems. 3. Issue the **`backup database`** command using appropriate parameters for the source database.

Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows)

The Q Apply program is a highly parallel process that you can configure to meet a variety of replication workloads. Depending upon how you configure the Q Apply program, you will need to ensure that the MAXAPPLS (maximum number of active applications) parameter is set appropriately. The Q Apply program uses multiple agents based on a number that you specify to divide the workload of applying transactions to targets. The database treats each agent as a separate application that is trying to connect.

Before you begin

UNIX **For 64-bit environments:** If you are running Q Replication or Event Publishing in a 64-bit environment on the HP-UX or Solaris platforms, catalog the database (the Q Apply server) as a loop back database and create an entry for this database in the password file. If you do not catalog the Q Apply server as a loop back database, you will encounter a Semaphore Wait problem, sqlcode 1224, when the Q Apply program reaches the shared memory limit.

Procedure

To set the MAXAPPLS parameter based on a given replication scenario:

1. Issue the following command: `update database configuration for database using MAXAPPLS n` where `database` is the target database and `n` is the maximum number of applications that will be allowed to run on the target database at the same time. To determine `n`, use the following formula:

   ```
   n >= (number of applications other than Q Apply that can use the database at the
   same time) + (3 * the number of Q Apply programs on the database) +  (number of
   receive queues, each with a browser thread + total number of Q Apply  agents for
   all receive queues)
   ```

   Here is an example of the results for this calculation based on a scenario where two applications other than the Q Apply program can use the database at the same time. There are two Q Apply programs, one that uses 3 agents to process a single receive queue, and one that uses 12 agents to process four receive queues (so a total of five browser threads to process the five receive queues).

   ```
   n >= 2 + (3 * 2) + (5 + 15)
   ```

   In this scenario, you would set MAXAPPLS to at least 28.
2. Optional: If you plan to have the Q Apply program automatically load targets using the LOAD from CURSOR option of the LOAD utility, issue the following command: `update dbm cfg using FEDERATED YES`.
3. Optional: Use the update database configuration command to increase values for the target database depending on your replication needs. The following database configuration values are adequate for many large replication scenarios (if the database is configured to work with other applications, your values might already be larger than these): APPLHEAPSZ 4096, LOGFILSIZ 2000, LOGPRIMARY 20, LOCKLIST 200, DBHEAP 1000, STMTHEAP 4096, LOCKTIMEOUT 30. For the LOGSECOND parameter, a value of 20 is adequate for most scenarios. If you expect to replicate long transactions, it is recommended that you set LOGSECOND = -1 on DB2 Version 8.1 or newer to avoid most log space problems. For LOGBUFSZ, a value between 64 and 512 is recommended.

Optional: Binding the program packages (Linux, UNIX, Windows)

On Linux, UNIX, and Windows, program packages are bound automatically the first time that the Q Capture program, Q Apply program, or Replication Alert Monitor connects to a database. You can bind the packages manually to specify bind options or bind the packages at a time when you expect less contention at the database.

Optional: Binding the Q Capture program packages (Linux, UNIX, Windows)

On Linux, UNIX, and Windows, the Q Capture program packages are bound automatically the first time that the program connects to the Q Capture server. You can choose to specify bind options, or bind the packages manually during a time when you expect less contention at this database. This procedure explains how to manually bind the Q Capture program packages.

Procedure

To bind the Q Capture program packages:

1. Connect to the Q Capture server by entering the following command:
   ```
   db2 connect to database
   ```
   Where *database* is the Q Capture server.
2. Change to the directory where the Q Capture bind files are located.
   - **Linux, UNIX:** `db2homedir/SQLLIB/bnd`, where *db2homedir* is the DB2 instance home directory
   - **Windows:** `DB2_install_drive:\...\SQLLIB\bnd`
3. Create and bind the Q Capture package to the database by entering the following commands:
   ```
   db2 bind @qcapture.lst isolation ur blocking all
   ```
   Where `ur` specifies the list in uncommitted read format for greater performance.

This command creates packages, the names of which are in the file `qcapture.lst`.

Optional: Binding the Q Apply program packages (Linux, UNIX, Windows)

On Linux, UNIX, and Windows, the Q Apply program packages are bound automatically the first time that the program connects to the target database, and to the source database if the Q Apply program is handling the target table loading. You can bind manually to specify bind options or to bind when you expect less contention at these databases.

Procedure

To bind the Q Apply program packages:

1. Change to the directory where the Q Apply bind files are located:
   - Linux UNIX `db2homedir/SQLLIB/bnd`, where *db2homedir* is the DB2 instance home directory
   - Windows `DB2_installation_drive:\...\SQLLIB\bnd`
2. For both the source and target databases, do the following steps:
   a. Connect to the database by entering the following command:
      ```
      db2 connect to database
      ```
      Where *database* is the Q Apply server or Q Capture server.

      If the database is cataloged as a remote database, you might need to specify a user ID and password on the **`db2 connect to`** command.

   b. Create and bind the Q Apply program package to the database by entering the following commands:

      ```
      db2 bind @qapply.lst isolation ur blocking all grant public
      ```

      Where `ur` specifies the list in uncommitted read format.

   c. Optional: If you plan to use the DB2 EXPORT utility to load target tables from a DB2 source that is at Version 9.7 or newer, and the user ID that starts the Q Apply program does not have BINDADD authority, perform the following bind before Q Apply starts:

      ```
      db2 bind @db2ubind.lst CONCURRENTACCESSRESOLUTION WAIT_FOR_OUTCOME
      COLLECTION ASN
      ```

Required for Sybase targets: Manually bind the Q Apply program packages to the federated database using cursor stability (CS) isolation level:

```
db2 bind @qapply.lst isolation CS blocking all grant public
```

Optional: Binding the Replication Alert Monitor packages (Linux, UNIX, Windows)

On Linux, UNIX, and Windows, the Replication Alert Monitor packages are bound automatically the first time that the program connects to the Monitor control server, or to any Q Capture server or Q Apply server that you chose to monitor. You can bind manually to specify bind options or to bind the packages manually during a time when you expect less contention at these databases.

Procedure

To bind the Replication Alert Monitor packages:

1. Change to the directory where the Replication Alert Monitor bind files are located:
   - **Linux, UNIX:** `*db2homedir*/SQLLIB/bnd`, where *db2homedir* is the DB2 instance home directory
   - **Windows:** `*DB2_install_drive*:\...\SQLLIB\bnd`
2. For each Monitor control server, do the following steps:
   a. Connect to the database by entering the following command:

      ```
      db2 connect to database
      ```

      Where *database* is the Q Apply server or Q Capture server.

      If the database is cataloged as a remote database, you might need to specify a user ID and password on the **`db2 connect to`** command.

   b. Create and bind the Replication Alert Monitor package to the database by entering the following commands:

      ```
      db2 bind @asnmoncs.lst isolation cs blocking all grant public
      db2 bind @asnmonur.lst isolation ur blocking all grant public
      ```

      Where `cs` specifies the list in cursor stability format, and `ur` specifies the list in uncommitted read format.

   These commands create packages, the names of which are in the files `asnmoncs.lst` and `asnmonur.lst`.
3. Connect to each server that you are monitoring and to which the Replication Alert Monitor connects, and create and bind the Replication Alert Monitor package to the database by entering the following command:

   ```
   db2 bind @asnmonit.lst isolation ur blocking all grant public
   ```

Where `ur` specifies the list in uncommitted read format.

This command creates packages, the names of which are in the file `asnmonit.lst`.

Creating control tables for the Q Capture and Q Apply programs

Before you can publish or replicate data, you must create *control tables* for a Q Capture program, a Q Apply program, or both. Control tables store information about Q subscriptions and publications, message queues, operational parameters, and user preferences.

Before you begin

- The ASNCLP command-line program or Replication Center must be able to connect to the server where you want to create the control tables:
  - If you are replicating from DB2 to DB2, then the administration tools must be able to connect to the source and target server.
  - If you are replicating from DB2 to a non-DB2 server, then the tools must be able to connect to the source server, the DB2 federated server, and the non-DB2 target server.
- You must have the names of the following WebSphere MQ objects:

  Q Capture control tables
  - A queue manager that the Q Capture program works with.
  - A local queue to serve as the administration queue.
  - A local queue to serve as the restart queue.

  Q Apply control tables
  - A queue manager that the Q Apply program works with.

  Tip: Use the **`VALIDATE WEBSPHERE MQ ENVIRONMENT FOR`** command in the ASNCLP or the validate controls in the Replication Center to ensure that the queue managers and queues that you specify for the control tables exist and have the correct properties.

Restrictions

For partitioned databases, all of the table spaces that are used by the control tables must be in a single-partition table space that is on the catalog partition.

About this task

Each instance of the Q Capture program or Q Apply program has its own set of control tables, identified by the Q Capture schema or Q Apply schema. For example, the control table that stores operational parameters for a Q Capture program with a schema of ASN1 would be named ASN1.IBMQREP_CAPPARMS.

By default, the control tables are placed in two table spaces on z/OS, one for tables that require page-level locking and one for tables that require row-level locking. The control tables are created by default in one table space on Linux, UNIX, and Windows. You can customize where each control table is created, and you can specify existing table spaces or create new table spaces.

Federated targets: If you are replicating to a non-DB2 target, see "Creating Q Apply control tables for federated Q Replication" on page 142.

For peer-to-peer replication, the Q Capture program and Q Apply program run as a pair at each server. Both sets of control tables must have the same schema.

Procedure

To create Q Capture or Q Apply control tables, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE CONTROL TABLES FOR command. For example, the following commands set the environment and create control tables in the SAMPLE database with a Q Capture schema of ASN1:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET QMANAGER "QM1" FOR CAPTURE SCHEMA;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`SET RUN SCRIPT LATER;`
	
	`CREATE CONTROL TABLES FOR CAPTURE SERVER`
	`USING RESTARTQ "ASN1.QM1.RESTARTQ"`
	`ADMINQ "ASN1.QM1.ADMINQ" MEMORY LIMIT 64`
	`MONITOR INTERVAL 600000;`
	
	The CREATE command specifies a restart queue and administration queue, doubles the default amount of memory available to build transactions to 64 MB, and reduces the default interval for recording performance information to 600000 milliseconds (one minute).

Replication Center	Use the Create Control Tables wizard. To open the wizards, right-click the **Q Capture Servers** folder and click **Create Q Capture control tables**, or right-click the **Q Apply Servers** folder and click **Create Q Apply control tables**.

Creating control tables at a different version (Linux, UNIX, Windows)

You can use a version of replication or publishing that is newer than your DB2 version. To do so, you must create the replication control tables to match the version of replication and publishing instead of the version of DB2. For example, if you plan to use replication Version 9.7 with a DB2 Version 9.5 database, your control tables must be at Version 9.7.

Before you begin

To create control tables at a different version, first you must install the replication or publishing product without upgrading DB2.

About this task

The following diagram shows a configuration in which the Replication Center creates Version 9.7 Q Apply control tables on a Version 9.1 DB2. You can also use the ASNCLP command-line program to specify a different version for control tables.

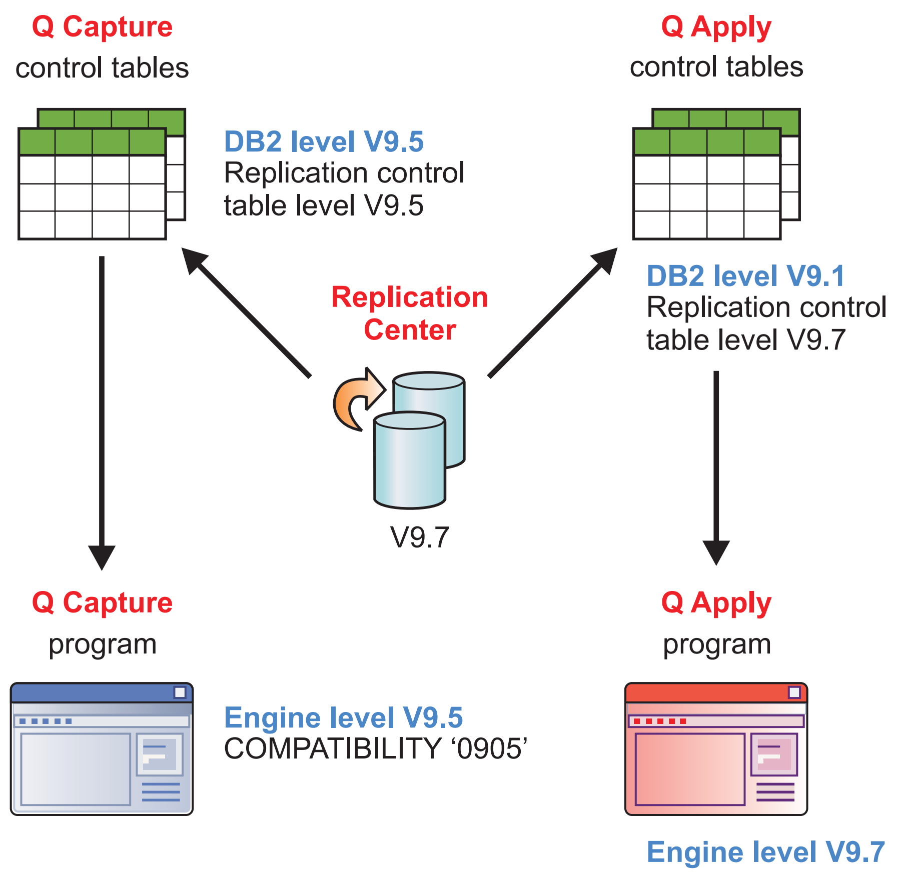


Figure 8. Creating replication control tables with a different version than DB2. A Version 9.7 Replication Center can create control tables at either V9.7, V9.5, or V9.1. In the diagram, the Replication Center creates the default V9.5 control tables on a V9.5 DB2, but creates V9.7 control tables on a V9.1 DB2. The level of the Q Capture program is thus V9.5, and the level of the Q Apply program is V9.7. However, because Q Capture is at a lower version than Q Apply, compatibility for Q Capture must be set to 0905. At this setting, Q Capture sends V9.5 messages to Q Apply and V9.5 functions are supported.

Note: When the Q Capture and Q Apply control tables are at different versions, the value of the COMPATIBILITY column in the IBMQREP_CAPPARMS control table determines the level of messages that are sent by Q Capture. The compatibility level of the Q Capture server must be lower than, or at the same level as, the architecture level of the consuming Q Apply program's control tables. In the diagram, Q Capture compatibility is set to 0905 because the Q Capture program and its control tables are at Version 9.5. In this case, the Q Capture program sends Version 9.5 messages, and even though Q Apply is at Version 9.7 the two programs use Version 9.5 functionality.

Restrictions

- The replication or publishing product must be at a version that is the same as or newer than the DB2 version. The DB2 must be Version 8.2 or higher.
- The version of the control tables must match the version of the replication or publishing product.
- Specifying the version of the control tables is not supported on DB2 for z/OS. On z/OS, the Replication Center always creates control tables that match the version of the DB2 client on which the Replication Center runs. You can also use the sample SQL that comes with the replication and publishing products on z/OS to create control tables.

- Some replication and publishing functions depend on the DB2 level. For example, you can only replicate from compressed tables or range-partitioned tables if the source DB2 is at Version 9.7 or newer. DB2 uncompresses log records before passing them to the Q Capture program. The data is passed uncompressed from source to target and the target does not need to have compression set.

Procedure

To create control tables at a different version, use one of the following methods:

Method	Description
ASNCLP command-line program	In the CREATE CONTROL TABLES FOR command, use the version keyword to specify the version, as in the following example: `CREATE CONTROL TABLES FOR CAPTURE SERVER USING RESTARTQ "ASN1.QM1.RESTARTQ" ADMINQ "ASN1.QM1.ADMINQ" version 9.5`
Replication Center	On the Summary page of the Create Q Capture Control Tables wizard or Create Q Apply Control Tables wizard, click **Change** next to the listing of the Q Capture or Q Apply version. In the Change Control Table version window, specify the version.

Configuring for older or remote DB2 databases

You can configure InfoSphere Replication Server or Data Event Publisher with some older DB2 databases so that you can use the most recent replication features without migrating these existing databases. You can also configure InfoSphere Replication Server or Data Event Publisher to run on a different server than the servers that host your DB2 databases. Both configurations use similar steps.

The following figure shows the basic configuration for using InfoSphere Replication Server with an older version of DB2, in this example, InfoSphere Replication Server Version 9.5 with DB2 Version 8.2.

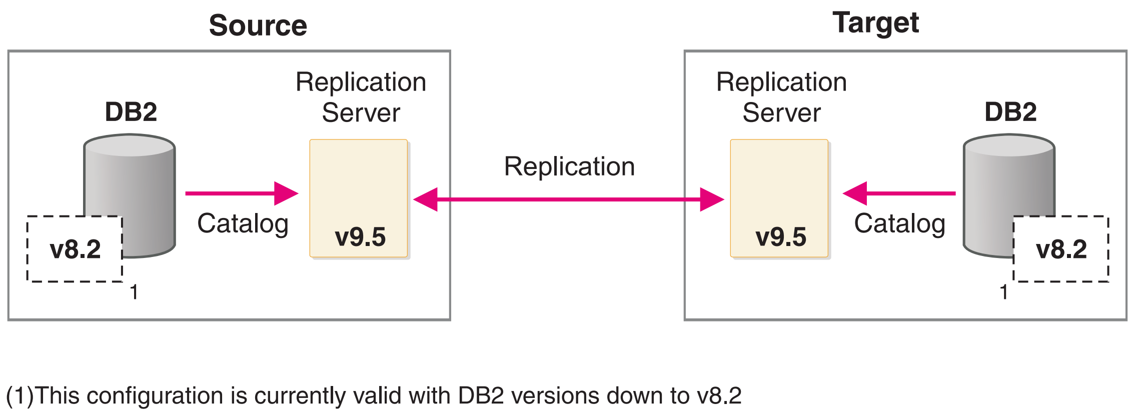


Figure 9. A configuration of Replication Server with older DB2 databases

The following figure shows the basic configuration for using remote DB2 databases.

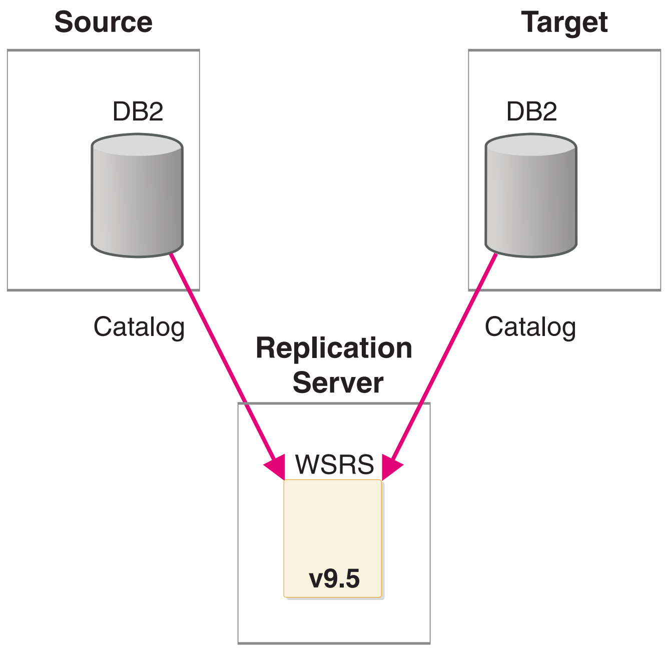


Figure 10. A configuration with remote DB2 databases

Before you begin

If you plan to use a newer version of Replication Server or Data Event Publisher on the same system as an older DB2 database, see Installation scenarios for important considerations during installation.

If you plan to use a Q Capture that is a different version or release than your Q Apply, see Coexistence support in Version 9.7 Q Replication and Event Publishing

About this task

Installing Replication Server or Data Event Publisher does not upgrade any existing DB2 databases, even if an existing DB2 is an older version. After InfoSphere Replication Server or Data Event Publisher is installed and licensed, the replication programs can work with any DB2 database that is at the same release or lower (down to Version 8.2), regardless whether that database resides in another DB2 instance or on another system. A new DB2 instance is created as part of the installation, however the new DB2 instance never needs to be started and no database needs to be created in it.

The following extra steps are required in a mixed-version or remote replication configuration:

- The older or remote DB2 databases and nodes must be cataloged at the instance that contains the replication programs.
- In mixed-version configurations, the replication control tables must be created to match the higher-level version of the replication programs.
- In mixed-version configurations, the instance owner of the higher-level DB2 that is installed with the latest replication programs must have permission to read the DB2 logs of the older version DB2.

Restrictions

The replication administration tools will not create pre-Version 10.1 control tables for a source or target database on DB2 for Linux, UNIX, and Windows Version 10.1.

For Q Capture or Q Apply that are local to DB2

- Replication from compressed or partitioned tables on DB2 for Linux, UNIX, and Windows is only supported for DB2 Version 9.7 or later. DB2 uncompresses log records before passing them to the Q Capture program. The data is passed uncompressed from source to target and the target does not need to have compression set.
- To replicate load operations, the source database and Q Capture must be at APAR PK78558 or newer on z/OS, or at Version 9.7 or newer on Linux, UNIX, and Windows.

For Q Capture or Q Apply that are remote from DB2

- The Q Capture program, Q Apply program, and the source and target databases must be on the same Linux, UNIX, or Windows platform and must all have the same bit architecture, for example 32/32 bit or 64/64 bit.
- You cannot use the replication administration tools to list or validate queues.
- The endianness and code page of the system where the remote Q Capture program runs must match the endianness and code page of the source database and operating system.

Procedure

1. Install Replication Server or Data Event Publisher and create an instance as prompted by the installer. You can install the products on the same system as an older DB2 version (down to Version j8.2) or you can install on a remote system.
2. Catalog the DB2 node and database that is your source or target at the newly installed instance by using the **`CATALOG NODE`** and **`CATALOG DATABASE`** commands. Open a DB2 command window from the DB2 instance that is used by replication or event publishing, and catalog the node and database of the older or remote database.
3. Create an `asnpwd` password file to be used by the replication and publishing programs.
4. If the source or target database is a lower version than Q Capture or Q Apply, use the ASNCLP or Replication Center to create replication control tables that match the version of the replication programs.
5. Use the Replication Center or ASNCLP to create the queue maps and Q subscriptions.
6. If the database is to be a replication source, grant SYSADM or DBADM permission to the user ID with which the Q Capture program connects to the source. This gives the Q Capture program permission to retrieve DB2 log records.
7. Start Q Capture and Q Apply from the replication instance, whether it is a new version or the same version on a remote system. For example, start replication from the newly installed instance by providing the alias for the older or remote DB2 database that you specified in the **`CATALOG DATABASE`** command.

Chapter 5. Setting up unidirectional Q Replication

With unidirectional replication, you can replicate data in one direction from a source table to a target table or manipulate the data at the target using stored procedures. The target for the Q subscription can be either a DB2 server or a non-DB2 server.

The Q Capture program replicates transactions from a source table and puts those transactions on a send queue in compact format; then the Q Apply program gets the compact messages from a receive queue and applies the transactions to a target table or passes them to a stored procedure.

Unidirectional replication

With unidirectional replication, you replicate data from source tables to target tables or stored procedures.

Unidirectional replication is a Q Replication configuration that has the following characteristics:

- Transactions that occur at a source table are replicated over WebSphere MQ queues to a target table or are passed as input parameters to a stored procedure to manipulate the data.
- Transactions that occur at the target table are not replicated back to the source table.
- The target table typically is read only or is not updated by applications other than the Q Apply program.

The Q Capture program replicates transactions from a source table and puts those transactions on a send queue in compact format; then the Q Apply program gets the compact messages from a receive queue and applies the transactions to a target table (either a DB2 table or a nickname on a DB2 federated server), or passes them to a stored procedure.

From any source table, you can replicate either all of the columns and rows or only a subset of the columns and rows. If you want to transform the data, you can specify for the Q Apply program to pass the transactions from a source table as input parameters to a stored procedure that you provide. The stored procedure can update data in either a DB2 or non-DB2 server.

In unidirectional replication, the following objects exist between servers:

Replication queue maps
: You must create at least one replication queue map to transport data from the Q Capture program on the source server to the Q Apply program on the target server (or the DB2 federated server if you are replicating to a non-DB2 target table).

Q subscriptions
: There is one Q subscription for every pair of source and target tables or every pair of source tables and stored procedures. For example, if you have a source table on SERVER_RED, a target table on SERVER_GREEN, and another target table on SERVER_BLUE, there are two Q subscriptions:

- One from the source table on SERVER_RED to the target table on SERVER_GREEN
- One from the source table on SERVER_RED to the target table on SERVER_BLUE

Figure 11 shows what you get by mapping three source tables to three target tables at one time for unidirectional replication. In this example, there are three separate Q subscriptions. Changes from Source A are replicated to Target A, changes from Source B are replicated to Target B, and so on. Changes from Source A cannot be replicated to Target B. These source-and-target pairs use the same replication queue map, Q Capture program, and Q Apply program.

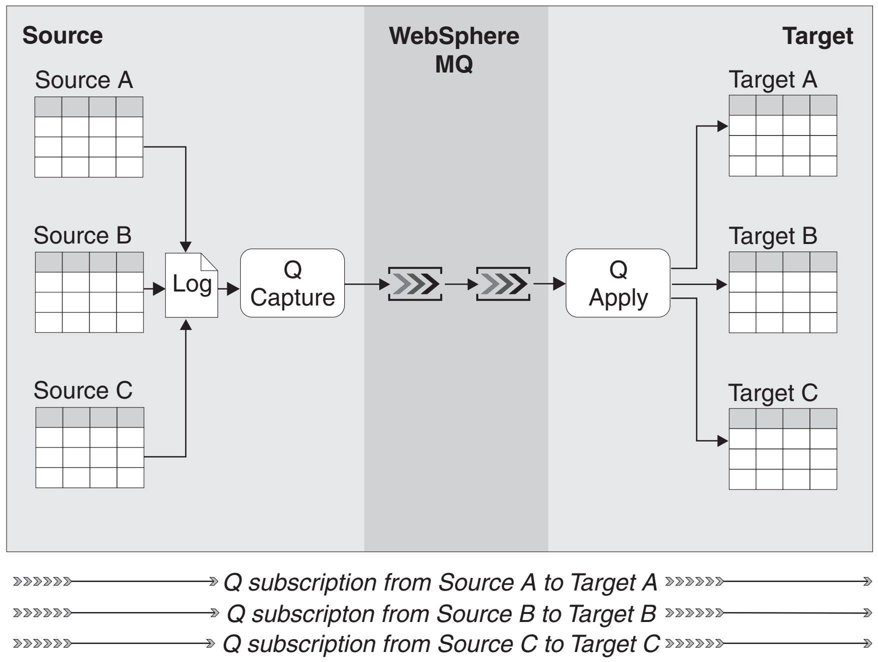

Source
WebSphere MQ
Target
Source A
Source B
Source C
Log
Q Capture
Q Apply
Target A
Target B
Target C
Q subscription from Source A to Target A
Q subscripton from Source B to Target B
Q subscription from Source C to Target C

Figure 11. Multiple Q subscriptions for unidirectional replication. In unidirectional replication, changes from each source table are replicated over WebSphere MQ queues to a particular target table.

Grouping replication queue maps and Q subscriptions

Before you define Q subscriptions and replication queue maps, you should first plan how you want to group Q subscriptions and replication queue maps.

Each Q subscription pairs a single source table with a single target table or stored procedure. When you define a Q subscription, you must also define which replication queue map is used to transport the data from the source table to the target table or stored procedure.

Among other things, each replication queue map identifies the WebSphere MQ queue that the Q Capture program sends changes to and the WebSphere MQ queue that the Q Apply program receives changes from before applying those changes to the target table or passing them to a stored procedure. A single replication queue map can be used to transport data for several Q subscriptions, so you must decide which Q subscriptions use the same replication queue map to transport data.

When you plan how to group Q subscriptions and replication queue maps, follow these rules:

- A WebSphere MQ queue cannot be shared by multiple Q Capture programs or by multiple Q Apply programs.
- A single Q Capture program or Q Apply program can write to or read from multiple queues. For example, a single Q Capture program can write data to many send queues and a single Q Apply program can read and apply data from many receive queues.
- You can create one or more replication queue maps between any pair of Q Capture and Q Apply programs. Each Q Capture and Q Apply program can work with multiple replication queue maps. For example, a single Q Capture program can send messages to multiple send queues, and a Q Apply program can retrieve messages from multiple receive queues.

How the Q Capture program works with the send queue

For a replication queue map, the Q Capture program captures changes for all tables for which there are active Q subscriptions. The Q Capture program stores these changes in memory until it reads the corresponding commit or abort record from the database log. The Q Capture program then sends information about committed transactions to all send queues that were defined for the Q subscriptions.

How the Q Apply program works with the receive queue

The Q Apply program starts a browser thread for every receive queue that was defined for a given Q Apply schema. For each browser, the transaction messages that the Q Apply browser thread reads from the receive queue are applied by one or more Q Apply agents, up to the maximum number of agents that you have defined. Within the context of a receive queue, transactions will be executed serially where dependencies between transactions exist, based on relationships between unique constraints or foreign keys. Where no constraint dependencies exist between transactions, transactions are executed in parallel as much as possible.

Suggestions for grouping similar Q subscriptions with replication queue maps

Generally speaking, for tables that are involved in transactions with one or more applications, you should create Q subscriptions for these tables so that they all share a common replication queue map. Grouping similar Q subscriptions with the same replication queue map assures the transactional consistency of the data when the Q Apply program applies it to the target tables or passes it to stored procedures. Because the Q Apply program is already applying changes for these transactions in parallel, you do not need to create multiple replication queue maps to achieve a high degree of parallelism in the application of the data.

If you define Q subscriptions that are involved in related transactions to send data through independent replication queue maps, the Q Capture program splits the data between the multiple send queues. Multiple Q Apply browsers that are associated with the receive queues apply the data independently.

Q subscriptions that have dependencies must share the same replication queue map. The Q Apply browser at the receive queue detects dependencies between transactions, so all Q subscriptions that involve dependent tables should use the same receive queue. If dependent transactions are sent to different receive queues through different replication queue maps, it is possible that the target database will not be transactionally consistent with the source database.

If multiple applications update the source server but do not update the same tables, and you configure a single pair of Q Capture and Q Apply programs to replicate data from the source server to a target server, then you might consider defining multiple replication queue maps for this pair of Q Capture and Q Apply programs to use. All of the Q subscriptions that are associated in transactions for each application are then replicated over one of these replication queue maps. Such a configuration could provide advantages, such as failure isolation or increased throughput. Still higher throughput and failure isolation might be gained by configuring multiple pairs of Q Capture and Q Apply programs, each with their own replication queue map. However, you must balance these gains against increased CPU consumption and a more complex replication environment.

Creating replication queue maps

When you create Q subscriptions, you specify which WebSphere MQ queues to send the data over by associating each Q subscription with a replication queue map. You can create a replication queue map before you begin creating Q subscriptions or as one of the steps of creating Q subscriptions.

Before you begin

- Plan how you want to group replication queue maps and Q subscriptions.
- On the server that contains the source tables for the Q subscriptions, create the control tables for the Q Capture program.
- On the server that contains the target tables for the Q subscriptions, create the control tables for the Q Apply program.
- Ensure that you have defined the appropriate objects in WebSphere MQ.

Restrictions

The same send queue cannot be used for both Q Replication and Event Publishing because a send queue can transport compact messages (for Q Replication) or XML messages (for Event Publishing), but not both.

Procedure

To create a replication queue map, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE REPLQMAP command. For example, the following commands set the environment and create a replication queue map SAMPLE_ASN1_TO_TARGET_ASN1 :
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`SET SERVER TARGET TO DB TARGET;`
	`SET APPLY SCHEMA ASN1;`
	`SET RUN SCRIPT LATER;`
	
	`CREATE REPLQMAP`
	`SAMPLE_ASN1_TO_TARGET_ASN1 USING`
	`ADMINQ "ASN1.QM1.ADMINQ"`
	`RECVQ "ASN1.QM1_TO_QM2.DATAQ"`
	`SENDQ "ASN1.QM1_TO_QM2.DATAQ"`
	`NUM APPLY AGENTS 8 HEARTBEAT INTERVAL 5;`

Replication Center	Use the Create Replication Queue Map window. You can open the window from the Servers page of the Create Q Subscription wizard, or from the object tree expand the appropriate Q Capture or Q Apply schema, right-click the **Replication Queue Maps** folder, and select **Create**.

Tip: You can use either replication administration tool to validate the send queue, receive queue, and administration queue that you specify for a replication queue map and to send test messages between the queues. To validate, use the VALIDATE WSMQ ENVIRONMENT FOR command in the ASNCLP or click **Validate queues** on the Create Replication Queue Map window in the Replication Center. To send test messages, use the VALIDATE WSMQ MESSAGE FLOW FOR REPLQMAP command in the ASNCLP or the Validate WebSphere MQ Queues window in the Replication Center.

When you create a replication queue map, you can specify the following options:

Send queue
: The WebSphere MQ queue where the Q Capture program sends source transactions and informational messages. When you define a replication queue map, you must select a send queue that is configured to transport compact messages.

Receive queue
: The WebSphere MQ queue from which the Q Apply program receives source transactions and informational messages.

Administration queue
: The queue that the Q Apply program uses to send control messages to the Q Capture program. The messages that the Q Apply program sends on this queue have several purposes, including telling a Q Capture program to start sending messages or initiating the loading process for a target table.

Maximum message length
: The maximum size (in kilobytes) of a message that the Q Capture program can put on this send queue. This maximum message length must be less than or equal to the WebSphere MQ maximum message size attribute (MAXMSGL) that is defined for the queue or queue manager.

Queue error action
: The action that the Q Capture program take when a send queue is no longer accepting messages because of an error, for example when the queue is full:

- Stops running
- Stops putting messages on the queue in error but continues to put messages on other queues

Number of Q Apply agents
The number of threads, or agents, that the Q Apply program uses for concurrently applying transactions from this receive queue. To request that transactions be applied in the order that they were received from the source table, specify only one Q Apply agent. To have changes applied to the target server in parallel, specify more than one Q Apply agent.

Maximum Q Apply memory usage
The maximum amount of memory (in megabytes) that the Q Apply program uses as a buffer for messages from this receive queue.

Heartbeat interval
How often, in seconds, that the Q Capture program sends messages on this queue to tell the Q Apply program that the Q Capture program is still running when there are no transactions to replicate. The heartbeat is sent on the first commit interval after the heartbeat interval expires. A value of 0 tells the Q Capture program not to send heartbeat messages.

This heartbeat interval is different from the WebSphere MQ parameter HBINT (heartbeat interval) that you can define for a WebSphere MQ channel.

Creating Q subscriptions for unidirectional replication

With Q Replication, you can set up replication of data from source tables to target tables or manipulate the data at the target using stored procedures by creating Q subscriptions. You must create a Q subscriptions for each source-to-target pair. The target for the Q subscription can be either a DB2 server or a non-DB2 server.

Each Q subscription is a single object that identifies the following information:
- The source table that you want to replicate changes from
- The target table or stored procedure that you want to replicate changes to
- The columns and rows from the source table that you want to be replicated
- The replication queue map, which names the WebSphere MQ queues that transport information between the source server and target server

You can create one or multiple Q subscriptions at one time.

Attention: Q subscriptions are separate objects from publications. Publications do not publish data to the Q Apply program, but to an application of your choice. Q subscriptions are for replicating data, and publications are for publishing data. If you want to replicate changes from a source table and want the Q Apply program to apply those source changes to a target table or pass them to a stored procedure for data manipulation, define a Q subscription, not a publication.

Creating target object profiles

You can create profiles in the Replication Center or ASNCLP command-line program to specify your own default naming convention for objects that the administration tools create at the target server.

About this task

You can create profiles for the following objects:

Replication Center
: Target tables, table spaces (or their equivalents for non-DB2 targets), indexes, and nicknames

ASNCLP
: Table spaces or indexes

The administration tools use the naming rules that are contained in the target object profile to name objects that they create.

Procedure

To create target object profiles, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the SET PROFILE command to specify custom parameters for table spaces or indexes that are created by the ASNCLP program. For example, the following commands set the environment and create a profile TBSPROFILE that sets the index options for tables that follow the page locking mechanism:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET PROFILE TBSPROFILE FOR OBJECT`
	`PAGE LOCK INDEX OPTIONS ZOS`
	`DB TARGETDB STOGROUP MYSTOGROUPPRIQTY`
	`PERCENT OF SOURCE 70;`
	
	After you issue a SET PROFILE command, you can associate a profile with a task command by specifying the profile's name in the task command.
	
	The scope of the profile lasts only as long as the current session. Once you quit the ASNCLP session, the profile information is not saved for the next session.

Method	Description
Replication Center	Use the Manage Target Object Profiles notebook. To open the notebook, right-click a Q Apply server in the object tree and click **Manage Target Object Profiles**. The Replication Center has a default target object profile that you can modify or override when you create new Q subscriptions. Existing objects are not renamed. The Replication Center stores one target object profile for each target server. When you create a Q subscription, the Replication Center uses the profile for the target server to determine the owner and name of the target table. If a table with that owner and name exists, the Replication Center uses the existing table as the target for the Q subscription. If a table with that owner and name does not exist, the Replication Center creates a table with that owner and name for you. The naming convention for target objects consists of three parts: • A prefix • A base, which is either the name of a related database object or a timestamp • A suffix You can also specify whether to create target tables in new or existing table spaces (or dbspaces for some non-DB2 targets). You can specify operational parameters of the table space, including whether the target-table table space should use the same partitioning as the source-table table space, if the target server is a DB2 subsystem on z/OS. You can also define truncation rules for the names of these objects. If an object name, which is the prefix, base, and suffix, exceeds the maximum length for your operating system, the truncation rules tell the Replication Center to shorten the base of the name from either the left or the right until the name is at the maximum length that your operating system allows.

Creating Q subscriptions for unidirectional replication

By creating Q subscriptions for unidirectional replication, you define how data from source tables is replicated to target tables or is passed to parameters in a stored procedure for data manipulation.

Before you begin

- Plan how you want to group replication queue maps and Q subscriptions.
- Create the control tables for the Q Capture program in the server that contains the source table for the Q subscription.
- Create the control tables for the Q Apply program in the server that contains the target for the Q subscription.
- Specify the queues for replicating and their attributes by creating a replication queue map. (You can do this task before you create a Q subscription or while you create a Q subscription.)
- Prepare the stored procedure if you want the Q Apply program to pass source changes to a stored procedure instead of to a target table.

Restrictions

- Views cannot be sources or targets for Q subscriptions.
- IDENTITY columns in the target table cannot be defined as GENERATED ALWAYS.

z/OS

- Do not select ROWID columns for replication except when the ROWID column is the only unique index that is specified for replication. Replication of ROWID columns is not supported for bidirectional or peer-to-peer replication.

  Recommendation: Use an IDENTITY column rather than a ROWID column as the unique index for replication.
  If you are replicating LOB columns, you must have an unique index besides the ROWID unique index.

Linux UNIX Windows

- Replication is supported from multiple-partition databases. There is no limit to the number of partitions that replication supports.
- Replication is supported from tables that use value compression or row compression. DB2 uncompresses log records before passing them to the Q Capture program. The data is passed uncompressed from source to target and the target does not need to have compression set.
- Replication is supported from materialized query tables (MQTs).

Procedure

To create a Q subscription for unidirectional replication from one source table to one target table or stored procedure, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE QSUB command. For example, the following commands set the environment and create a Q subscription for unidirectional replication, EMPLOYEE0001, with the following characteristics:
	• The replication queue map is SAMPLE_ASN1_TO_TARGETDB_ASN1.
	• The Q Apply program loads the target tables using the EXPORT and IMPORT utilities.
	• The EMPNO column is used as the key column for replication to determine the uniqueness of a row.
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`SET SERVER TARGET TO DB TARGET;`
	`SET APPLY SCHEMA ASN1;`
	`SET RUN SCRIPT LATER;`
	
	`CREATE QSUB USING REPLQMAP`
	`SAMPLE_ASN1_TO_TARGETDB_ASN1`
	`(SUBNAME EMPLOYEE0001 EMPLOYEE OPTIONS`
	`HAS LOAD PHASE I TARGET NAME`
	`TGTEMPLOYEE KEYS (EMPNO) LOAD TYPE 2)`

Method	Description
Replication Center	Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture or Q Apply schema, right click the **Q Subscriptions** folder, and select **Create**. You can create one Q subscription or many by using the wizard. **Rows and columns page** When you create multiple Q subscriptions at one time, the Replication Center assumes that you want to replicate all columns and rows from each source table. On the Review Q Subscriptions page of the wizard you can modify individual Q subscriptions so that only a subset of the source columns and rows are replicated. **Target tables page** When you create more than one Q subscription at a time, the Target Tables page allows you to review the target object profile. Modify the profile if necessary so that the target tables for the Q subscriptions meet your needs. The target object profile determines if an existing target table is used or if a new one is created. The Replication Center looks for an object that matches the naming scheme that is defined in the profile, and, if one does not exist, then the object is created.

Source columns for Q subscriptions (unidirectional replication)

By default when you create Q subscriptions, changes to all columns that are in the source table are replicated to the target table or stored procedure. However, when you create a Q subscription for unidirectional replication, you can replicate a subset of the columns that are in the source table instead of all of the columns.

You might want to replicate a subset of the columns under the following circumstances:

- You do not want to make all of the columns that are in the source table available to the target table or stored procedure.
- The target for the Q subscription does not support all of the data types that are defined for the source table.
- The target table already exists and contains fewer columns than the source table.

Figure 12 on page 67 shows how a subset of columns are replicated.

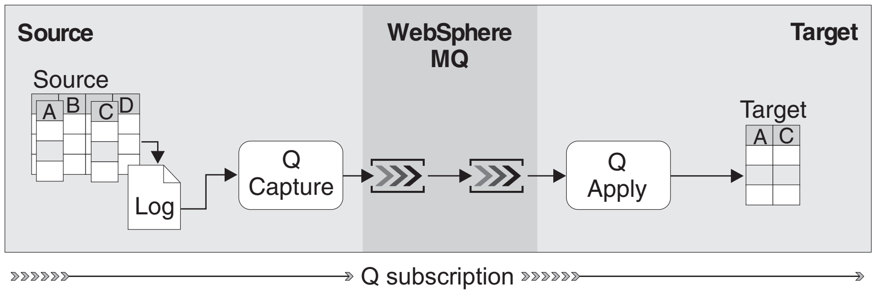


Figure 12. Column subsetting in a Q subscription for unidirectional replication. Only columns A and C are selected for replication. When a row is replicated from the source table to the target table, the Q Capture program replicates only the values from columns A and C.

To replicate a subset of the columns, select only the source columns that you want to be replicated to the target. If you are creating a single Q subscription, the administration tools give you options for how to replicate a subset of the columns from the source table.

In the Replication Center, if you are creating multiple Q subscriptions at one time, on the Review page of the Create Q Subscriptions wizard select the individual Q subscription that you want to subset columns in and edit the properties for that Q subscription.

Important for LOB columns: If you select columns that contain LOB data types for a Q subscription, make sure that the source table enforces at least one unique database constraint (for example, a unique index or primary key). You do not need to select the columns that make up this uniqueness property for the Q subscription.

How often the Q Capture program sends a message (unidirectional replication)

When you create Q subscriptions, you can specify when the Q Capture program sends messages to the Q Apply program. The Q Capture program can send a message either only when the columns change that are part of the Q subscription or every time that a column in the source table changes.

The following sections describe the two different types of events that can cause the Q Capture program to send a message:

- "A message is sent only when columns in the Q subscriptions change"
- "A message is sent every time a change occurs in the source table" on page 68

If you replicate all of the columns that are in the source table, these two options result in the same action.

A message is sent only when columns in the Q subscriptions change

By default, the Q Capture program sends a message only when the change occurs in columns that you selected for the Q subscriptions.

For example, assume that you have 100 columns in your source table and you select 25 of those columns to be replicated in a Q subscription. If you specify that a message is sent only when columns in Q subscriptions change, then any time a change is made to any of the 25 columns that are part of the Q subscription, the Q

Capture program sends a message. Any time a change is made in any of the 75 columns that are not part of the Q subscription, the Q Capture program does not send a message.

Recommendation: Replicate only the changes that occur in columns that are part of Q subscriptions if changes in the source tables frequently occur in columns that are not part of the Q subscriptions. Use this option if you do not want to keep a history of when all changes occur at the source table. This option minimizes the amount of data that is sent across the queues.

A message is sent every time a change occurs in the source table

You can define Q subscriptions so that the Q Capture program sends a message every time a change occurs in the source table. If you are replicating only a subset of the columns in the source table, then the Q Capture program sends a message even if the change occurs in a column that is not part of a Q subscription.

For example, assume that you have 100 columns in your source table and you select 25 of those columns to be replicated in a Q subscription. If you specify that a message is sent every time a change occurs in the source table, then any time that a change is made to any of the of the 100 columns in your source table, the Q Capture program sends a message.

Recommendation: Use this option if you want to keep a history for audit purposes of when all changes occur at the source table.

Search conditions to filter rows (unidirectional replication)

By default when you create Q subscriptions for unidirectional replication, all rows from the source table are replicated to the target table or stored procedure. However, when you create a Q subscription for unidirectional replication, you can specify a WHERE clause with a search condition to identify the rows that you want to be replicated.

When the Q Capture program detects a change in the DB2 recovery log that is associated with a source table, the Q Capture program evaluates the change against the search condition to determine whether to replicate the change to the target table or stored procedure.

If you are creating a single Q subscription, then the Create Q Subscriptions wizard in the Replication Center helps you add a WHERE clause to replicate a subset of the rows from the source table. If you are creating multiple Q subscriptions at one time, then, on the Review page of the Create Q Subscriptions wizard, select the individual Q subscription for which you want to subset rows and edit the properties for that Q subscription to add the WHERE clause.

If you define a Q subscription so that the target table is initially loaded with source data, the search condition for the Q subscription is evaluated when the target table is loaded. Because the row filter is used while loading the target table, the target table initially contains a subset of the rows in the source table.

When you specify a WHERE clause, you can specify whether the column is evaluated with values from the current log record. If you want a column in the WHERE clause to be evaluated with values from the current log record, place a single colon directly in front of the column name.

Example of WHERE clause that evaluates a column with values from the current log record:

```
WHERE :LOCATION = 'EAST' AND :SALES > 100000
```

In the above example, LOCATION and SALES are column names in the source table that are evaluated with values from the current log record. Here, the Q Capture program sends only the changes from the source table that involve sales in the East that exceed $100,000. When you type a column name, the characters fold to uppercase unless you enclose the name in double quotation marks. For example, type "Location" if the column name is mixed case.

If the Q Capture program replicates a column that is part of the WHERE clause, it might need to change the type of operation that needs to be sent to the target table or stored procedure.

Example where the Q Capture program must change the type of operation because of a WHERE clause:

```
WHERE :LOCATION = 'EAST'
AND :SALES > 100000
```

Suppose that the following change occurs at the source table:

```
INSERT VALUES ( 'EAST', 50000 )
UPDATE SET SALES = 200000 WHERE LOCATION = 'EAST'
```

Because the before value does not meet the search condition of the WHERE clause, the Q Capture program sends the operation as an INSERT instead of an UPDATE.

Likewise, if the before value meets the search condition but the after value does not, then the Q Capture program changes the UPDATE to a DELETE. For example, if you have the same WHERE clause as before:

```
WHERE :LOCATION = 'EAST'
AND :SALES > 100000
```

Now suppose that the following change occurs at the source table:

```
INSERT VALUES ( 'EAST', 200000 )
UPDATE SET SALES = 50000 WHERE LOCATION = 'EAST'
```

The first change, the insert, is sent to the target table or stored procedure because it meets the search condition of the WHERE clause (200000 > 100000 is true). However, the second change, the update, does not meet the search condition (50000 > 100000 is false). The Q Capture program sends the change as a DELETE so that the value will be deleted from the target table or stored procedure.

Complex search conditions

Q Replication allows you to specify more complex WHERE clauses. However, complex search conditions might impact performance. For example, you can specify a more complex WHERE clause with a subselect that references other tables or records from either the source table or another table.

Example of WHERE clause with a subselect:

```
WHERE :LOCATION = 'EAST'
AND :SALES > (SELECT SUM(EXPENSE) FROM STORES WHERE STORES.DEPTNO = :DEPTNO)
```

In the above example, the Q Capture program sends only the changes from the East that resulted in a profit, where the value of the sale is greater than the total expense. The subselect references the STORES table and the following columns in the source table: LOCATION, SALES, and DEPTNO.

When you define a Q subscription with a subselect in a WHERE clause, the following problems might occur:

- Performance might be slower because, for each change in the source table, the Q Capture program computes a large select on the STORES table to compute the SUM(EXPENSE) value. Also, this type of select might compete for locks on the tables.
- The subselect might produce unexpected results. For example, because the subselect is evaluated against the current database values, the example above produces a wrong answer if the EXPENSE value changes in the database, whereas columns in the WHERE clause are substituted with the older log record values. If the table name that the subselect references does not change, then the search condition produces the proper results.

Restrictions for search conditions

- Search conditions cannot contain column functions, unless the column function appears within a subselect statement.
- **Invalid WHERE clause with column functions**:

  ```
  #-----------------------------------------------------------------
  #  Incorrect:  Don't do this
  #-----------------------------------------------------------------

  WHERE :LOCATION = 'EAST' AND SUM(:SALES) > 1000000
  ```

  The Replication Center validates search conditions when the Q Capture program evaluates them, not when the Replication Center creates the Q subscription. If a Q subscription contains an invalid search condition, then that Q subscription will fail when the invalid condition is evaluated, and the Q subscription will be deactivated.
- Search conditions cannot contain an ORDER BY or GROUP BY clause unless the clause is within a subselect statement.

  Invalid WHERE clause with GROUP BY:

  ```
  #-----------------------------------------------------------------
  #  Incorrect:  Don't do this
  #-----------------------------------------------------------------

  WHERE :COL1 > 3 GROUP BY COL1, COL2
  ```

  Valid WHERE clause with GROUP BY:

  ```
  WHERE :COL2 = (SELECT COL2 FROM T2 WHERE COL1=1 GROUP BY COL1, COL2)
  ```
- Search conditions cannot reference the actual name of the source table that you are replicating changes from. Do not use the schema.tablename notation in a WHERE clause for the actual name of the source table. However, you can reference another table name in a subselect by using schema.tablename notation.

  Invalid WHERE clause with actual name of source table and column name:

  ```
  #-----------------------------------------------------------------
  #  Incorrect:  Don't do this
  #-----------------------------------------------------------------

  WHERE :ADMINISTRATOR.SALES > 100000
  ```

  In the above WHERE clause, the table that is being replicated is ADMINISTRATOR and the column name is SALES. This invalid WHERE clause is intended to select only the values of the SALES column of the administrator table, for which SALES is greater than 100000.

Valid WHERE clause with column name:

```
WHERE :SALES > 100000
```

In the above WHERE clause, SALES is the column name.

- Search conditions cannot reference values that were in columns before a change occurred; they can reference values only after a change occurred.
- Search conditions cannot contain EXISTS predicates.
- Search conditions cannot contain a quantified predicate, which is a predicate using SOME, ANY, or ALL.
- Search conditions cannot reference LOB values.

Log record variables to filter rows (unidirectional replication)

You can use variables from the database recovery log such as authorization ID or DML operation to filter which rows to replicate for unidirectional Q subscriptions. You can also combine these variables with an SQL WHERE clause to create more extensive row filters.

These variables describe how a change in the table came about, and include information about the operation of the change, the user who owns the transaction, and on z/OS the job name or plan name. You can use the following log record variables:

$OPERATION	The DML operation. Valid values are I (insert), U (update), and D (delete).
$AUTHID	The authorization ID of a transaction.
$AUTHTOKEN	**z/OS:** The authorization token (job name) of a transaction.
$PLANNAME	**z/OS:** The plan name of a transaction.

You create a predicate that uses these variables, and when the Q Capture program reads the log it determines whether or not to replicate transactions based on the predicate. This function enables you to create filters that are based on more than just the data in the base table. For example, you can filter on DML operations that might suit feeding a data warehouse, for example specifying that only insert operations are replicated (`"$OPERATION IN 'I'"`). You can also create ID-based filters at the Q subscription level, for example `"$AUTHID = 'HR'"`, rather than using the Q Capture program's ability to ignore certain transactions, which operates at the Q Capture instance level.

To specify a predicate, you use the ASNCLP command-line program with the CHANGE CONDITION keyword of the CREATE QSUB or ALTER QSUB command. For example:

```
CREATE QSUB USING REPLQMAP DALLAS_ASN_TO_TOKYO_ASN
(SUBNAME SALES0002 OPTIONS CHANGE CONDITION
"$AUTHID = 'HR'")
```

The predicate is stored in the CHANGE_CONDITION column of the IBMQREP_SUBS table.

Combining log record variables with search conditions

You can use log record variables in conjunction with the other type of row filter that Q Capture provides, the search condition that uses an SQL WHERE clause.

Unlike values in the search condition, predicates that are based on log records do not need to be prefixed by the WHERE keyword.

If you specify a search condition with a WHERE clause and a predicate that uses log record variables, Q Capture combines the two predicates by using the AND operator. For example, the following search condition specifies that Q Capture only replicate rows where the value in the STATE column is CA (postal abbreviation for California):

```
WHERE :STATE = 'CA'
```

The following predicate with log record variables specifies that Q Capture replicate transactions with all authorization IDs except HR:

```
$AUTHID <> 'HR'
```

When the Q Capture program detects the WHERE clause in the SEARCH_CONDITION column of the IBMQREP_SUBS table and the log record predicate in the CHANGE_CONDITION column, it combines the two into a single predicate. Q Capture only replicates rows where the STATE is CA and the authorization ID of the transaction is not HR:

```
WHERE :STATE = 'CA' AND $AUTHID <> 'HR'
```

You could specify the entire combined predicate in the CHANGE CONDITION keyword of the CREATE QSUB or ALTER QSUB command and the result would be the same (as long as nothing was specified with the SEARCH CONDITION keyword). However, log record variables can only be used with CHANGE CONDITION.

Note: Although replication allows a CHANGE CONDITION of 2048 characters, DB2 might not be able to process all possible predicates that can be written in 2048 characters. If the statement is too long (combined with SEARCH CONDITION) or too complex, DB2 might return an SQL0101N error ("The statement is too long or too complex. SQLSTATE=54001").

Load considerations

Predicates with log record variables are not used when loading the target table because the variables are based on information that is not part of the source table, from which the load utilities select data. Search conditions are used when loading the target because they specify source table columns from which the load utility can select.

In the example above, a load utility invoked by the Q Apply program would use the following predicate when selecting source data to load into the target:

```
WHERE :STATE = 'CA'
```

If you want to load the target table with a subset of source data in addition to using log record variables, make sure to build your predicates carefully.

Considerations for asntdiff utility

If you use the asntdiff utility to compare data in source and target tables, be aware that many of the insert operations that the utility recommends to synchronize the tables are due to the predicate with log record variables. Do not use the asntrep utility to repair the table if you suspect that this is the case because the repair would undo the effect of filtering at the source.

How source columns map to target columns (unidirectional replication)

For existing target tables, you can specify how you want the data from source columns to map to target columns. If a target table does not exist, the Replication Center or ASNCLP program create the target table (or nickname if you have non-DB2 target tables) with the same columns as the source table.

When you create Q subscriptions with the Replication Center, you can choose from the following options for mapping source columns to target columns:

Map by column name and data type
: Each column in the source table is mapped to the column in the target table (or parameter in the stored procedure) that has an identical name and data type.

Map by column position and data type
: The first column in the source table that you selected for replication is mapped to the first column in the target table or parameter in the stored procedure, the second column in the source table that you selected for replication is mapped to the second target column or parameter, and so on. The columns in the target are mapped from left to right. The data types of each pair of mapped columns must be the same.

If any columns do not map to identical column names and data types, you must manually map those extra columns, which can include defining an SQL expression to map different data types.

Note: Mapping rules that you create in a target object profile do not affect existing Q subscriptions, only future Q subscriptions.

You cannot replicate more columns from the source table than are in the target table. However, the target table can contain more columns than the number of source columns that you selected for replication. If the target table contains extra columns that are not mapped to source columns, those extra target columns cannot be part of the target key and must be either nullable or be not null with a default value.

Non-DB2 target tables

The following considerations pertain to mapping source columns to target tables in non-DB2 relational databases:

- If you choose to have a new target table created for you, the target table will be created with data types that are close to the DB2 source data type. For example, if a DB2 source table has a column with the data type TIME, then an Oracle table will be created with column data type as DATE, and the default mapping for this in the nickname will be TIMESTAMP. For more detail on mappings that are used for federated targets, see Default forward data type mappings in the DB2 Information Center.
- If you choose to replicate to an existing target table, the source columns map to the data types that are defined in the DB2 federated nickname, not to the data types in the non-DB2 target table.
-

Multiple Q subscriptions

If you are creating multiple Q subscriptions at a time, then you can use the Create Q Subscriptions wizard in the Replication Center to specify a rule for how columns in the source table map to columns in the target table or to parameters in the stored procedure. When you create multiple Q subscriptions at a time, the Replication Center assumes that you want to replicate all columns from the source table, and so it applies the column mapping rule to all columns. If any columns do not map according to the rule that you specify, then you must manually map those extra columns.

Stored procedures

For existing stored procedures, you can specify how you want the data from source columns to map to parameters. The source table cannot replicate more columns than parameters in the stored procedure. Stored procedures cannot have extra parameters that do not map to source columns.

Using expressions in Q Replication

You can use expressions to change data while it is being replicated from the source to the target. Q Replication supports both SQL expressions and XML expressions.

SQL expressions in Q Replication

When you create a Q subscription for unidirectional replication, you can use SQL expressions to transform data between the source and target tables, to map multiple source columns to a single target column, or to create other types of computed columns at the target.

Restrictions

The use of SQL expressions is not supported in the following situations:

- Bidirectional or peer-to-peer replication.
- Large object (LOB) columns
- Stored procedure targets
- Before values for CCD tables
- Aggregate functions such as SUM and MAX

Q Replication supports all types of DB2 SQL expressions, including the following examples:

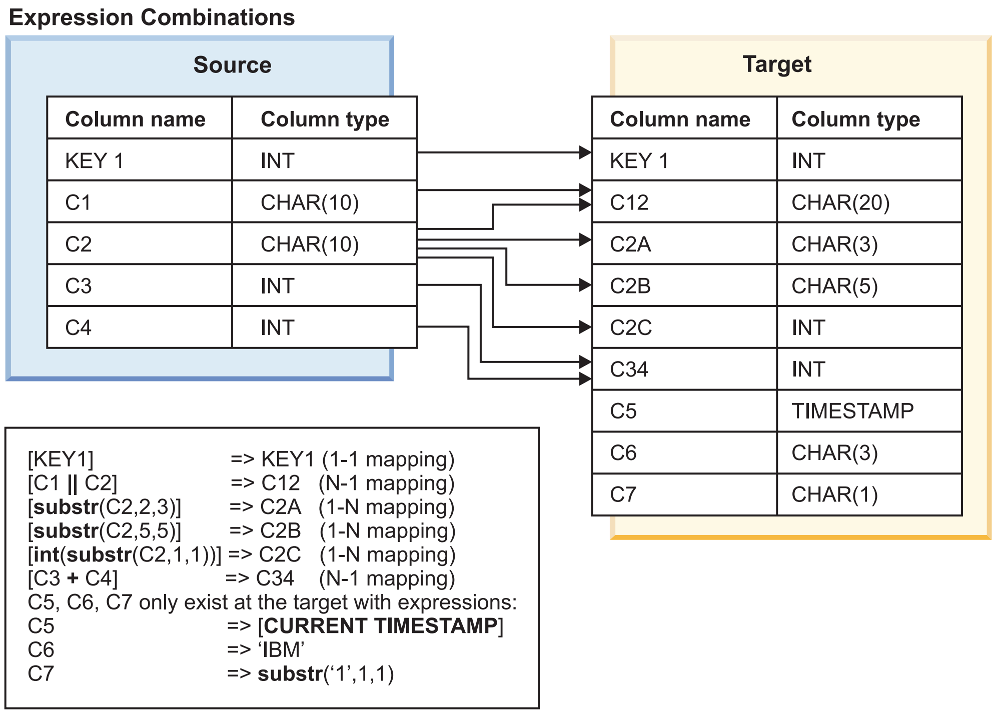
Concatenation
: You can merge the contents of two or more source columns into a single target column. For example, you could map two source columns, COL1 and COL2, to a target column called COLEXP. Rather than specifying a single source column for the Q subscription, you would specify the SQL expression CONCAT(:C1,:C2) and map the expression to the target column COLEXP.

Substring
: You can use the SQL substr() expression to apply only a portion of the data in a source column. For example, this function can be used to replicate data into a target column whose length is shorter than that of the source column, or to extract portions of an address or other character string from a source column.

Constant
You can populate target table columns with data from CONSTANT and DERIVED CONSTANT expressions rather than from data in the source table. For example, you could populate a column with the characters IBM, or the value of a DB2 special register such as CURRENT TIMESTAMP.

Case You can use CASE expressions to achieve more complex data transformations between the source and target tables.

Figure 13 shows how expressions can be used in Q Replication to create computed columns.



Column name	Column type
KEY 1	INT
C1	CHAR(10)
C2	CHAR(10)
C3	INT
C4	INT

Column name	Column type
KEY 1	INT
C12	CHAR(20)
C2A	CHAR(3)
C2B	CHAR(5)
C2C	INT
C34	INT
C5	TIMESTAMP
C6	CHAR(3)
C7	CHAR(1)

```
[KEY1]                 => KEY1 (1-1 mapping)
[C1 || C2]             => C12  (N-1 mapping)
[substr(C2,2,3)]       => C2A  (1-N mapping)
[substr(C2,5,5)]       => C2B  (1-N mapping)
[int(substr(C2,1,1))]  => C2C  (1-N mapping)
[C3 + C4]              => C34  (N-1 mapping)
C5, C6, C7 only exist at the target with expressions:
C5                     => [CURRENT TIMESTAMP]
C6                     => 'IBM'
C7                     => substr('1',1,1)
```

Figure 13. Examples of Q replication support for SQL expressions. This diagram shows how SQL expressions could be used to create a target table with different types of computed columns from a source table. The legend below the source table shows the various expressions that are used, such as concatenation (C1 || C2). In the target table, some of the columns from the source table are mapped to combined columns or changed columns. Other target table columns are created from expressions that are derived from constants rather than from data in the source table.

Any number of source columns can be mapped to one target column, and a single source column can be split and mapped to any number of target columns.

An expression can reference any column from the captured row at the source. You prefix the source column name with a colon (:) and after the column name add a space. For example, `:COL1` .

For Q subscriptions that specify a load phase for the target table, the Q Apply program uses DB2 to evaluate SQL expressions during the load phase, and also while it is applying source changes from the spill queue. If you specify a manual load, you must evaluate the expression during the loading process.

SQL expressions are supported for Classic data sources, federated targets, and CCD target tables. For CCD tables, the Q Apply program evaluates SQL expressions for both after-image and before-image columns.

Expressions in key columns

You can also specify SQL expressions for columns that are used as a key for replication (IS_KEY=Y in the IBMQREP_TRG_COLS table), and for columns that are part of a unique constraint at the target table but are not used as a key for replication.

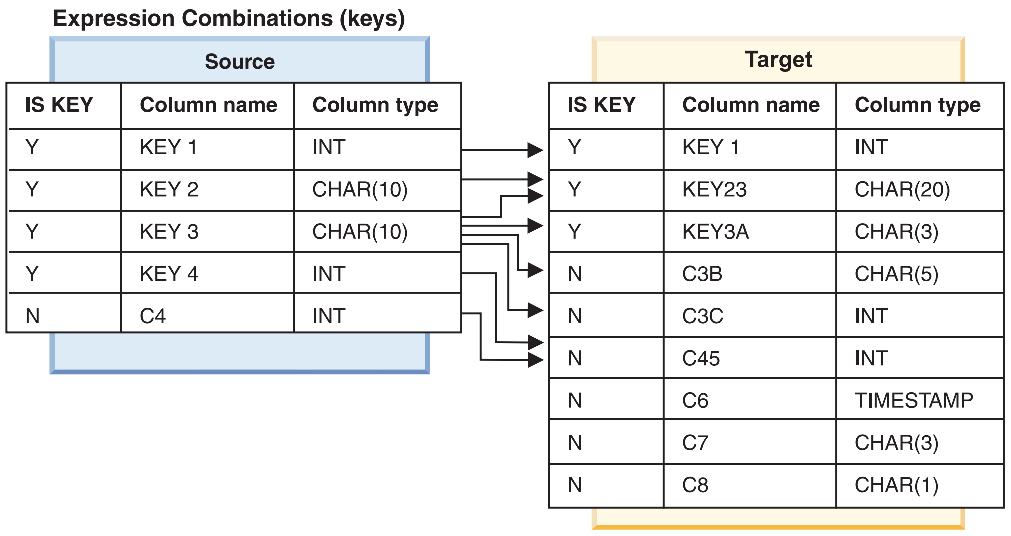
When you map source columns to target columns by using SQL expressions, you can use any of the following combinations:

- One replication source key column that is mapped to one replication target key column
- Any number of replication source key columns that are mapped to one replication target key column
- One replication source key column that is divided and mapped to any number of replication target key and non-key columns
- Replication key and non-key columns at the source that are combined and mapped to one replication target non-key column

Restrictions:

- A combination of replication key and non-key columns at the source that are mapped to one replication target key column is not supported. To resolve conflicts correctly, the Q Apply program requires that data is always sent for any column that is specified as a replication key at the target. If the key column at the target is mapped to non-key columns at the source, the Q Capture program might not always send the necessary data, especially when the value of CHANGED_COLS_ONLY is Y (yes) and not all before values are sent.

Figure 14 on page 77 shows how expressions can be used in Q Replication to create computed columns.



IS KEY	Column name	Column type
Y	KEY 1	INT
Y	KEY 2	CHAR(10)
Y	KEY 3	CHAR(10)
Y	KEY 4	INT
N	C4	INT

IS KEY	Column name	Column type
Y	KEY 1	INT
Y	KEY23	CHAR(20)
Y	KEY3A	CHAR(3)
N	C3B	CHAR(5)
N	C3C	INT
N	C45	INT
N	C6	TIMESTAMP
N	C7	CHAR(3)
N	C8	CHAR(1)

```
[KEY1]                        => KEY1    (1-1 mapping)
[KEY2][KEY3]                  => KEY23  (N-1 mapping)
[substr(KEY3,2,3)]            => KEY3A  (1-N mapping)
[substr(KEY3,5,5)]            => C3B     (1-N mapping)
[init(substr(KEY3,1,1))]      => C3C     (1-N mapping)
[KEY4+C5]                     => C45     (N-1 mapping)
C6, C7, C8 only exist at the target with expressions:
C6                            => [CURRENT TIMESTAMP]
C7                            => 'IBM'
C8                            => substr('1',1,1)
```

Figure 14. Examples of Q Replication support for SQL expressions in key columns. This diagram shows how SQL expressions could be used to create a target table with different types of computed columns from a source table. The legend below the source table shows the various expressions that are used, such as substring [substr(KEY3,2,3)]. In the target table, some of the columns from the source table are mapped to combined columns or changed columns. Other target table columns are created from expressions that are derived from constants rather than from data in the source table.

The index or key that you specify for a Q subscription does not need to match between the source and target. When you pick columns as keys for replication, you must choose all of the columns that match one of the unique constraints at the target. This requirement enables the Q Apply program to correctly detect and resolve conflicts. If no unique constraints exist at the target, then you should choose all columns at the target as part of the key for replication (excluding some column types, such as LOB and LONG).

Expression support in the replication administration tools

If you are using the ASNCLP command-line program to create a Q subscription, you use the EXPRESSION keyword, which is part of the TRGCOLS option. You specify the SQL expression and the name of the target column that the expression maps to.

For example, the following commands set the environment and create a Q subscription with a new target table that includes all of the columns in the source table and specifies an expression that concatenates columns COL1 and COL2 and maps to the target table column CEXP.

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT NOW STOP ON SQL ERROR ON;
SET SERVER CAPTURE TO DB SAMPLE;
SET SERVER TARGET TO DB TARGET;
SET CAPTURE SCHEMA SOURCE ASNCAP1;
SET APPLY SCHEMA ASNAPP1;
CREATE QSUB USING REPLQMAP SAMPLE_ASNCAP1_TO_TARGET_ASNAPP1
(SUBNAME TESTEXP DATA.EMPLOYEE TARGET NAME DATA.TGTEMPLOYEE
TRGCOLS ALL EXPRESSION ("CONCAT(:COL1,:COL2)" TARGET CEXP));
```

You can also specify SQL expressions when you are changing an existing Q subscription with the ALTER QSUB command.

In the Replication Center, you can create computed columns by using the Expression window. You open the Expression window from the Column Mapping window, which is launched from the Rows and Columns page of the Create Q Subscriptions wizard. Click **Validate** to check the syntax of the expression.

Byte-level SQL expressions

If the source and target tables are in different code pages, SQL expressions that evaluate data at the byte level (for example, a HEX function) might produce different results in the target table depending upon whether the data was replicated during an automatic load or during regular Q Apply program operations:

During an automatic load
DB2 evaluates the SQL expression on the source data and then converts the results to the target code page.

When a row is replicated
The Q Apply program converts the source data to the target code page and then DB2 evaluates the SQL expression for the row.

An alternative to using the HEX function is to define the target column as FOR BIT DATA. The Q Apply program then avoids code page conversion, and the source data is saved in the target table in a hexadecimal representation.

XML expressions in Q Replication

When you create a Q subscription for unidirectional replication, you can use XML expressions to transform XML data between the source and target tables.

Q Replication supports the XML data type. Support for expressions on XML columns enables powerful transformations over XML data. For example, you can change the shape of an XML object, replicate only a subset of it, and make it conform to a different schema at the target. Expressions on XML data types might be required in situations where the application at the target needs data in a different form. Examples include application migrations, feeding a warehouse, and application integration.

You can use the Replication Center or ASNCLP command-line program to specify XML expressions when you create a Q subscription in the same way that you specify SQL expressions.

- “Supported DB2 SQL/XML functions ” on page 79
- “Restrictions” on page 79
- “Examples of XML expressions” on page 80

Supported DB2 SQL/XML functions

Q Apply integrates XML expressions into the SQL statements that it uses to update target tables. Q Apply does not parse expressions to determine whether they are syntactically and semantically correct. If the use is incorrect, Q Apply reports the DB2 SQL error or warning in the IBMQREP_EXCEPTIONS table when it tries to run the statement and follows the error action that is defined for the Q subscription.

Recommendation: Test the SQL statement to validate its syntax before adding it to a Q subscription.

Table 8 provides examples of supported XML expressions and lists unsupported expressions.

Table 8. Supported and unsupported XML expressions

Examples of supported XML expressions	Unsupported XML expressions
• XMLATTRIBUTES scalar function • XMLCOMMENT scalar function • XMLCAST specification • XMLCONCAT scalar function • XMLDOCUMENT scalar function • XMLELEMENT scalar function • XMLFOREST scalar function • XMLNAMESPACES declaration • XMLPARSE scalar function • XMLPI scalar function • XMLQUERY scalar function • XMLROW scalar function • XMLSERIALIZE scalar function • XMLTEXT scalar function • XMLVALIDATE scalar function	Some of these functions, for example the XMLAGG aggregate function, are not supported because they use XML values from different rows and Q Apply handles data one row at a time. Other functions can be used in queries but cannot be part of an expression. • PARAMETER scalar function: You can use this function in the db2-fn:sqlquery function; a query is not part of an expression • XMLAGG: aggregate function • XMLEXISTS predicate: Can be used in the WHERE clause of a query • XMLGROUP: aggregate function • XMLTABLE: aggregate function • XMLXSROBJECTID: scalar function with no XML value output • XMLTRANSFORM: scalar function; not supported because other output than an XML value is possible.

Restrictions

Q Replication support for XML expressions has the following restrictions:
- You cannot replicate XML expressions on both XML columns and key columns.
- You cannot map large object (CLOB, BLOB, DBCLOB) columns to XML columns.
- You cannot map XML columns to non-LOB columns or to non-XML columns such as CHAR and VARCHAR.
- Several functions are not supported when they are used across various DB2 platforms:
  - XMLQUERY is not supported when the source is DB2 for z/OS and the target is DB2 for Linux, UNIX, and Windows.
  - XMLQUERY on DB2 for Linux, UNIX, and Windows supports XQuery language expressions while XMLQUERY on DB2 z/OS supports XPath language expressions.

– XMLVALIDATE on DB2 for Linux, UNIX, and Windows has a different syntax than the corresponding DSN_XMLVALIDATE function on DB2 z/OS. In addition, the schema that describes the transformed document must be registered at both the source and target. The LOAD operation extracts transformed data from the source before applying the transformed data at the target. If the schema is not registered for use at the source, the LOAD operation terminates and the Q subscription is disabled.

Examples of XML expressions

The following examples show some of the ways that XML expressions can be used between the source and target:

Concatenation of XML documents and adding a new root node

You can use expressions to merge the data in two XML columns into an XML document that is surrounded by a new root node. The following example combines the data in the xmlcol1 and xmlcol2 columns at the source by using the XMLCONCAT function and surrounds the data with a new root element by using the XMLELEMENT function. In this case you would map the xmlcol1 and xmlcol2 columns to a single target column and specify the expression.

Source data	**xmlcol1:** <book><author>Foo</author><title>bar</title></book> **xmlcol2:** <book><author>Bar</author><title>Foo</title></book>
Expression	XMLELEMENT(NAME "bookstore", XMLCONCAT(:xmlcol1, :xmlcol2))
Resulting data at target	<bookstore> <book><author>Foo</author><title>bar</title></book> <book><author>Bar</author><title>Foo</title></book> </bookstore>

Transforming or subsetting data

You can use XQUERY to modify replicated XML documents. For example, you can delete an element in the XML document or add new elements. The following example deletes the `<title>` element and its contents from the source XML document:

Source data	<book><author>Foo</author><title>bar</title></book>
Expression	XMLQUERY(copy $oldval := $d modify do delete $oldval/book/title return $oldval' PASSING :xmlcol AS "d")
Resulting data at target	<book><author>Foo</author></book>

Validating final XML documents

You might need to validate a changed XML document against a certain schema located at the target database. By using the XMLVALIDATE function in an expression you can perform the validation while the data is replicated. The expression must contain XMLVALIDATE around any other expression that indicates the schema location as a parameter, as in the following example:

Source data	<book><author>Foo</author><title>bar</title></book>

Expression	XMLVALIDATE(XMLQUERY('transform copy $oldval := $d modify do delete $oldval/book/title return $oldval' PASSING :xmlcol AS "d") ACCORDING TO XMLSCHEMA ID modifiedBooks.xsd)
Resulting data at target	<book><author>Foo</author></book>

Mapping CHAR and VARCHAR columns to XML columns

You can migrate CHAR and VARCHAR source data that contains XML documents into the DB2 native XML format. The XMLPARSE function parses the character data and replication stores the data in an XML column at the target, as in the following example:

Source data	<book><author>Foo</author><title>bar</title></book>
Expression	XMLPARSE(DOCUMENT :xmlcol)
Resulting data at target	<book><author>Foo</author><title>bar</title></book>

Index or key columns for targets (unidirectional replication)

The Q Apply program uses a primary key, unique constraint, or unique index at the target to enforce the uniqueness of each row when it applies the row to target tables or parameters in stored procedures.

If you are creating a single Q subscription, then the Create Q Subscriptions wizard in the Replication Center helps you select the columns that uniquely identify rows in the target. If you are creating multiple Q subscriptions at one time, then you can use the Review page of the Create Q Subscriptions wizard to customize which columns are used for uniqueness at the target.

If the Replication Center finds uniqueness in the source table, it recommends a target index or key for you. You can accept the recommendation or specify the columns that you want as the replication key for the target.

Restriction: Large-object (LOB) columns and LONG columns cannot be used as keys (IS_KEY in the IBMQREP_SRC_COLS and IBMQREP_TRG_COLS control tables).

The recommendation that the Replication Center makes depends on whether the target table already exists. The following sections explain the logic that Replication Center uses when recommending a target key or index:
- "Target key or index for new target tables"
- "Target key or index for existing target tables in Q subscriptions" on page 82

Target key or index for new target tables

When the Replication Center recommends a primary key, unique constraint, or unique index for new target tables, it checks the source table for one of the following definitions, in the following order:

1. A primary key
2. A unique constraint
3. A unique index

If the Replication Center finds one of these definitions for the source table and those source columns are selected for replication, then the Replication Center uses the source table's primary key (or unique constraint or index) as the target's key. When the Replication Center generates the SQL to build the new target tables, the Replication Center builds them with the key or index that you specify. You can use the default index name and schema or change the defaults to match your naming conventions.

If the Replication Center cannot find a primary key, unique constraint, or unique index at the source, it will automatically create a unique index for the new target table that is based on all valid, subscribed source columns.

Target key or index for existing target tables in Q subscriptions

When the Replication Center recommends a key or index for target tables that already exist, it first checks whether a primary key, unique constraint, or unique index already exists on the target table. If Replication Center finds uniqueness on the target table, the Replication Center makes sure that those columns are part of the columns that you chose to replicate from the source table. The Replication Center also checks whether the source table uses those columns to enforce uniqueness at the source table. If the source and target tables have at least one exact match for key or index columns, then the Replication Center recommends that those columns be used to establish target row uniqueness.

If no uniqueness exists on the target table, then the Replication Center checks the source table for one of the following definitions, in the following order:
1. A primary key
2. A unique constraint
3. A unique index

If the Replication Center finds one of these definitions for the source table and those source columns are selected for replication, then it recommends the primary key, unique constraint, or unique index from the source table.

Options for unexpected conditions in the target table (unidirectional replication)

The Q Apply program updates the targets with changes that occur at the source table. If other applications are also making changes to the target, then the Q Apply program might encounter rows in the target that are different than expected.

For example, the Q Apply program might try to update a row in the target that another application has already deleted. The option that you choose depends on the level of granularity at which you want to isolate and fix the problem.

In many scenarios, you will likely want the Q Apply program to either force the change or ignore unexpected conditions in target data. However, in some scenarios, you might never expect problems in the target data and, therefore, might choose a different action, depending on the level at which you think you will need to troubleshoot problems.

You can specify that the Q Apply program takes one of the following actions when it encounters unexpected conditions in target data:

- “Force the change”
- “Ignore the unexpected condition”
- “Deactivate the corresponding Q subscription” on page 84
- “Have the Q Apply program stop reading from the corresponding receive queue” on page 84
- “Stop the Q Apply program” on page 85

CCD target tables: If the CCD table is condensed and complete (CONDENSED=Y and COMPLETE=Y), you can choose between forcing the change and ignoring the unexpected condition. For all other CCD table types, the only valid choice is force.

Stored procedures: If the Q Apply program is passing the data to a stored procedure, then the action that you should select depends on the behavior of the stored procedure. In most scenarios where the Q Apply program is passing data to a stored procedure, you often do not want to force changes by transforming the row operation. In most scenarios involving stored procedures that are manipulating the data, you might want to specify for the Q Apply program to ignore unexpected conditions in target data. However, if you have a stored procedure that rebuilds the SQL statements for that row and transforms only some data (for example, it might transform only American dollars to European euros), then you might consider forcing the change.

Regardless of which options you select for unexpected conditions, whenever the Q Apply program encounters a problem when processing a row, the Q Apply program logs the unexpected condition in the IBMQREP_APPLYTRACE table and in its diagnostic log file. Also, a copy of the row in error is inserted into the IBMQREP_EXCEPTIONS table.

Force the change

When the Q Apply program encounters an unexpected condition in the target data, the Q Apply program forces the change from the source table into the target table or the parameters for the stored procedure. The Q Apply program changes the operation (for example, from an insert to an update) so that it can be applied to the target table or passed to the stored procedure parameters. Then it tries to reapply the change.

Recommendation: You might want the Q Apply program to force changes if the data that the Q Apply program receives from the source table is always what you want at the target.

Ignore the unexpected condition

When the Q Apply program encounters an unexpected condition in the target data, the Q Apply program ignores the unexpected condition, does not apply the row, logs the error and any rows that it did not apply, and then completes and commits the rest of the transaction. Whatever data is at the target wins; the target data is not overwritten. However, if you choose for the Q Apply program to ignore the unexpected condition, some data is not applied to the target; all rows that the are not applied are logged in the IBMQREP_EXCEPTIONS table.

Recommendation: You might want the Q Apply program to ignore an unexpected condition in the target data under the following circumstances:

- Convergence of data at the source table and the target table is not important in your scenario.
- All SQL states are expected and can be tolerated.

Deactivate the corresponding Q subscription

When the Q Apply program encounters an unexpected condition in the target data, it deactivates (or stops) only the Q subscription where the unexpected condition occurred but continues to apply changes for the other Q subscriptions. The Q Apply program logs the error and any rows that it did not apply, and then completes and commits the rest of the transaction. The Q Capture program stops capturing changes that occur at the source table for the deactivated Q subscription. This option provides you the finest level of granularity for troubleshooting problems at a particular table. You can then check what went wrong for the Q subscription and then activate the Q subscription when it is fixed. Keep in mind that, when a Q subscription is deactivated, the target of that Q subscription needs to be reloaded when the Q subscription is activated again.

Recommendations:

- You might want to choose to deactivate the corresponding Q subscription when unexpected conditions in target data occur under the following circumstances:
  - You have multiple Q subscriptions that are defined over unrelated tables that are replicated using the same replication queue map.
  - Few changes occur at the table (especially if this table is a parent and changes rarely occur in the parent key columns).
- You might not want to choose to deactivate the corresponding Q subscription when unexpected conditions in target data occur under the following circumstances:
  - The table in the Q subscription has referential integrity with other tables, and, therefore, other tables might be impacted if the one Q subscription is deactivated. If the table is a child table, then deletes or updates that occur at the parent table might fail in the child table because of referential integrity errors (if DELETE RESTRICT is on). All Q subscriptions might end up having errors related to referential integrity and might eventually all get disabled as a result of the errors.
  - You do not want to reload the target when the Q subscription needs to be reloaded when the Q subscription is activated again.
  - Deactivating a Q subscription is too severe of a result in your scenario if an unexpected condition in target data occurs.

Have the Q Apply program stop reading from the corresponding receive queue

When the Q Apply program encounters an unexpected condition in the target data, it stops applying changes for all of the Q subscriptions on the receive queue, not just for the Q subscription that had the error. The Q Apply program logs the error and any rows that it did not apply, but does not complete or commit the rest of the transaction. Any transactions that are affected by the unexpected condition in target data are rolled back.

The Q Apply program continues to read from any other receive queues but does not attempt to read from the deactivated queue. The Q Capture program continues to send data to the deactivated queue, so you must correct the problem and restart the receive queue before it fills up. If the Q Capture program can no longer write to the send queue, then the Q Capture program either deactivates all Q subscriptions that use that send queue or it shuts down, depending on what error option you specified for the replication queue map.

Recommendations:

- You might want to stop the Q Apply program from applying transactions from the receive queue when unexpected conditions in target data occur under the following circumstances:
  - Unexpected conditions in target data are not tolerated in your scenario and you do not expect them to occur often.
  - You have multiple Q subscriptions that are defined over related tables and those Q subscriptions share the same replication queue map. Therefore, when one Q subscription has an unexpected condition in the target data, you want all of the Q subscriptions to stop. You can then check what went wrong with the one Q subscription and then, after the Q subscription is fixed, restart the receive queue. If few changes are being replicated to the target table, then using this option might be a good choice because it helps to preserve transactions and helps related tables maintain referential integrity.
- You might not want to stop the Q Apply program from applying transactions from the receive queue if this result is too severe in your scenario if an unexpected condition in target data occurs.

Stop the Q Apply program

When the Q Apply program encounters an unexpected condition in the target data, it shuts down, but the receive queue continues to receive source data from the Q Capture program. The Q Apply program logs the error and any rows that it did not apply, but does not complete or commit the rest of the transaction. Any transactions that are affected by the unexpected condition in target data are rolled back. If the Q Apply program reads from only one receive queue, then this option has the same result as having the Q Apply program stop applying changes for all the Q subscriptions on the receive queue.

Shutting down is the most severe response that you can set for the Q Apply program if unexpected conditions in target data occur. The Q Capture program continues to send data to the receive queue, so you must correct the problem and restart the receive queue before the receive queue becomes full. If the Q Capture program can no longer write to the send queue, then the Q Capture program either deactivates all Q subscriptions that use that send queue or it shuts down, depending on what error option you specified for the replication queue map.

Recommendations:

- You might want to stop the Q Apply program when unexpected conditions in target data occur under the following circumstances:
  - You have multiple Q subscriptions that are defined over related tables and the data is transported over multiple replication queue maps. When one Q subscription has an unexpected condition in the target data, you want all of the Q subscriptions to stop. You can then check what went wrong with the one Q subscription and then, after the Q subscription is fixed, restart the receive queue. If few changes are being replicated to the target table, then having this option might be a good choice because it helps to preserve transactions and helps related tables maintain referential integrity.
  - You want to easily monitor your configuration. This option of stopping the Q Apply program is similar to stopping the Q Apply program from applying

transactions from the receive queue; however, your environment might be easier to monitor if the Q Apply program shuts down instead of just stopping reading from a particular receive queue. For example, you might want to select this option while you are testing.
- You might not want to stop the Q Apply program if this result is too severe in your scenario if an unexpected condition in target data occurs.

Error options for Q Replication

In Q Replication, you can specify what action the Q Apply program takes when it encounters errors, such as SQL errors, in your environment. The option that you choose depends on the level of granularity at which you want to isolate and fix the problem. The same error options apply for both unidirectional and multidirectional replication.

You can choose for one of the following actions to occur when the Q Apply program encounters errors:
- "Deactivate the corresponding Q subscription"
- "Have the Q Apply program stop reading from the corresponding receive queue" on page 87
- "Stop the Q Apply program" on page 87

Regardless of which error options you select, whenever the Q Apply program encounters an error, the Q Apply program logs the error in the IBMQREP_APPLYTRACE table and in its diagnostic log files, and a copy of any rows that were not applied and the details about the error are inserted into the IBMQREP_EXCEPTIONS table.

Deactivate the corresponding Q subscription

When the Q Apply program encounters an error, it deactivates (or stops) only the Q subscription where the error occurred but continues to apply changes for the other Q subscriptions. The Q Apply program logs the error and any rows that it did not apply, and then completes and commits the rest of the transaction. The Q Capture program stops capturing changes that occur at the source table for the deactivated Q subscription. This option provides you the finest level of granularity for troubleshooting problems at a particular table. You can check what went wrong for the Q subscription and then activate the Q subscription when it is fixed. Keep in mind that, when a Q subscription is deactivated, the target of that Q subscription needs to be reloaded when the Q subscription is activated again.

Recommendations:
- You might want to choose to deactivate the corresponding Q subscription when an error occurs under the following circumstances:
  - You have two Q subscriptions that are defined over unrelated tables that are replicated using the same replication queue map.
  - Few changes occur at the table (especially if this table is a parent and changes rarely occur in the parent key columns).
- You might not want to choose to deactivate the corresponding Q subscription when an error occurs under the following circumstances:
  - The table in the Q subscription has referential integrity with other tables, and, therefore, other tables might be impacted if the one Q subscription is deactivated. If the table is a child table, then deletes or updates that occur at the parent table might fail in the child table because of referential integrity

errors (if DELETE RESTRICT is on). All Q subscriptions might end up having errors related to referential integrity and might eventually all get disabled as a result of the errors.
  - You do not want to reload the target when the Q subscription needs to be reloaded when the Q subscription is activated again.
  - Deactivating a Q subscriptions is too severe of a result in your scenario if an error occurs.

Have the Q Apply program stop reading from the corresponding receive queue

When the Q Apply program encounters an error, it stops applying changes for all of the Q subscriptions on the receive queue, not just for the Q subscription that had the error. The Q Apply program logs the error and any rows that it did not apply, but does not complete or commit the rest of the transaction. Any transactions that are affected by the error are rolled back. If the Q Apply program reads from other receive queues, then it continues to process data from the other receive queues. If the Q Apply program does not read from other receive queues, then it shuts down. The Q Capture program continues to send data to the receive queue, so you must correct the problem and restart the receive queue before the receive queue becomes full. If the Q Capture program can no longer write to the send queue, then the Q Capture program either deactivates all Q subscriptions that use that send queue or it shuts down, depending on what error option you specified for the replication queue map.

Recommendations:

- You might want to stop the Q Apply program from applying transactions from the receive queue when an error occurs under the following circumstances:
  - Errors are not tolerated in your scenario and you do not expect them to occur often.
  - You have multiple Q subscriptions that are defined over related tables and those Q subscriptions share the same replication queue map. Therefore, when one Q subscription has an error, you want all of the Q subscriptions to stop. You can check what went wrong with the one Q subscription and then, after the Q subscription is fixed, restart the receive queue. If few changes are being replicated to the target table, then using this option might be a good choice because it helps to preserve transactions and helps related tables maintain referential integrity.
- You might not want to stop the Q Apply program from applying transactions from the receive queue if this result is too severe in your scenario if an error occurs.

Stop the Q Apply program

When the Q Apply program encounters an error, it shuts down, but the receive queue continues to receive source data from the Q Capture program. The Q Apply program logs the error and any rows that it did not apply, but does not complete or commit the rest of the transaction. Any transactions that are affected by the error are rolled back. If the Q Apply program reads from only one receive queue, then this option has the same result as having the Q Apply program stop applying changes for all the Q subscriptions on the receive queue.

Shutting down is the most severe response that you can set for the Q Apply program when errors occur. The Q Capture program for those Q subscriptions continues to send data to the receive queue, so you must correct the problem and restart the receive queue before the receive queue becomes full. If the Q Capture program can no longer write to the send queue, then the Q Capture program either deactivates all Q subscriptions that use that send queue or it shuts down, depending on what error option you specified for the replication queue map.

Recommendations:

- You might want to stop the Q Apply program from applying transactions from the receive queue when an error occurs under the following circumstances:
  - Errors are not tolerated in your scenario and you do not expect them to occur often.
  - You have multiple Q subscriptions that are defined over related tables and the data is transported over multiple replication queue maps. When one Q subscription has a error, you want all of the Q subscriptions to stop. You can check what went wrong with the one Q subscription and then, after the Q subscription is fixed, restart the receive queue. If few changes are being replicated to the target table, then using this option might be a good choice because it helps to preserve transactions and helps related tables maintain referential integrity.
- You might not want to stop the Q Apply program from applying transactions from the receive queue if this result is too severe in your scenario if an error occurs.

Chapter 6. Setting up multidirectional Q Replication

With Q Replication, you can replicate data back and forth between tables on two or more servers while applications update the identical copies of a table at all servers while keeping all copies of the table synchronized. This type of replication is known as *multidirectional* replication.

Bidirectional replication

In bidirectional replication, Changes that are made to one copy of a table are replicated to a second copy of that table, and changes that are made to the second copy are replicated back to the first copy.

Bidirectional replication has the following characteristics:

- Applications on either server can update the same rows in those tables at the same time. However, there is little or no potential for the same data in the replicated tables to be updated simultaneously by both servers. Either the same row is updated by one server at a time, or one server updates only certain columns or rows of your data, and the other server updates different columns or rows.
- You can choose which copy of the table wins if a conflict occurs.

The collection of both copies of a single table is called a *logical table*. Each server has a copy of the table. Each copy of the table:

- Must have the same number of columns and rows
- Must have identical column names
- Must have compatible data types
- Can have different names and schemas

In this type of replication, you cannot manipulate the data by having the Q Apply program pass the data to a stored procedure. There is at least one Q Capture program and one Q Apply program running on each server that is part of a bidirectional configuration.

Attention: The control tables for the Q Capture and Q Apply programs that are on each individual server must have the same schema name. For example, if you have a server named SERVER_RED and a server named SERVER_GREEN, then the Q Capture and Q Apply programs that are on SERVER_RED must both have the same schema, and the Q Capture and Q Apply programs that are on SERVER_GREEN must both have the same schema.

Replication objects for bidirectional replication

In a bidirectional configuration, you must have the appropriate number of replication queue maps and Q subscriptions:

Number of replication queue maps

Between each pair of servers that participate in bidirectional replication, you need two replication queue maps. For example, if you have two servers named SERVER_RED and SERVER_GREEN, you need two replication queue maps:

- One to identify the WebSphere MQ queues that transport data from SERVER_RED to SERVER_GREEN
- One to identify the WebSphere MQ queues that transport data from SERVER_GREEN to SERVER_RED

Number of Q subscriptions

For every logical table that is replicated in bidirectional replication, you need a pair of Q subscriptions between the two servers. For example, if you have two servers named SERVER_RED and SERVER_GREEN, then two Q subscriptions are built for you:

- One from the source table on SERVER_RED to the target table on SERVER_GREEN
- One from the source table on SERVER_GREEN to the target table SERVER_RED

If you have two logical tables, you need four Q subscriptions; for three logical tables, you need six Q subscriptions, and so on.

Figure 15 shows bidirectional replication of one logical table between two servers. For one logical table, you need two Q subscriptions and two replication queue maps.

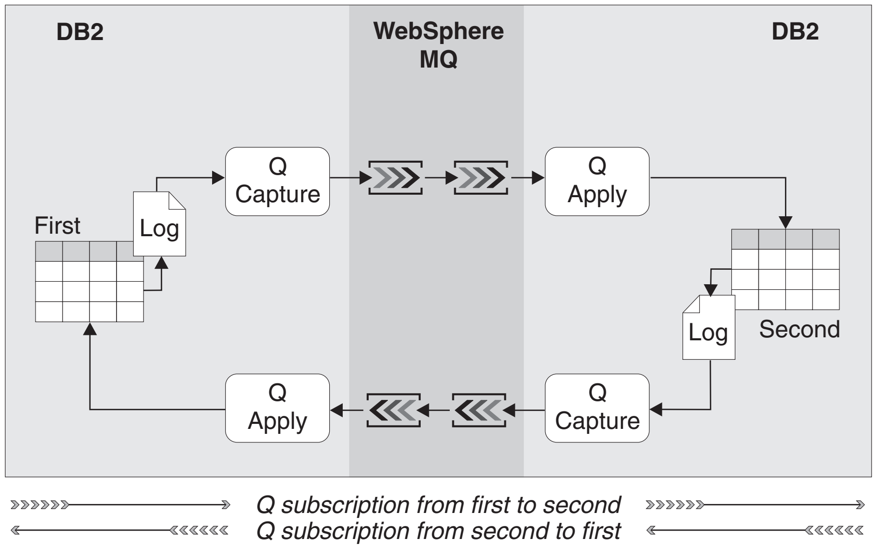

DB2
WebSphere MQ
DB2
Q Capture
Q Apply
First
Log
Log
Second
Q Apply
Q Capture
Q subscription from first to second
Q subscription from second to first

Figure 15. Q subscriptions between two copies of a table for bidirectional replication. Changes are replicated from each copy of the table to the other copy of that table over WebSphere MQ queues.

Conflict detection in bidirectional replication

In bidirectional replication, it is possible for data that is replicated from the source table in one Q subscription to conflict with changes made to the corresponding target table by an application other than the Q Apply program. Bidirectional replication uses data values to detect and resolve conflicts. You can choose which data values are used to detect conflicts. These data values can be key column values only, changed column values, or all column values.

For example, imagine a scenario in which applications on one system make changes to tables in a server (SERVER_RED) and that server replicates those changes to identical tables in a server (SERVER_GREEN) on a standby system. The first system fails, at which time your applications start using the tables on

SERVER_GREEN. When the first system comes back online, you want to replicate changes from SERVER_GREEN to SERVER_RED. However, when the first system shut down, it could have failed to replicate some data to the second system. That data, which is now old, should be replaced by the data replicated from SERVER_GREEN. When you replicate the new data, the Q Apply program for SERVER_RED recognizes the conflicts and forces the changes that come from SERVER_GREEN to SERVER_RED.

You can choose how the Q Apply programs on both servers check for conflicts when they try to apply data to both copies of the table and what actions those programs should take if they detect conflicts. The choices that you make for conflict rules and conflict actions are critical decisions because they affect the behavior of how rows are applied.

Peer-to-peer replication

In peer-to-peer replication (also known as multimaster replication) updates on any one server are replicated to all other associated servers.

Peer-to-peer replication has the following characteristics:

- Replication occurs between tables on two or more servers.
- Applications on any of the servers can update the same rows and columns in those tables at the same time.
- All servers are equal peers with equal ownership of the data; no server is the "master" or source owner of the data.

You replicate copies of tables between multiple servers in peer-to-peer replication. The collection of all copies of a single table is called a logical table. Each server has a copy of the table. Each copy of the table:

- Must have the same number of columns and rows
- Must have identical column names
- Must have compatible data types
- Can have different names and schemas

In peer-to-peer replication, data convergence is assured between all copies of the logical table, meaning that each copy of the table eventually achieves the same state as the other copies and has the most recent committed values. Because peer-to-peer replication is asynchronous, the copies of the tables might not converge until your applications stop making changes to all tables, all changes are replicated, and all messages are processed.

In this type of replication, you cannot manipulate the data by having the Q Apply program pass the data to a stored procedure. There is at least one Q Capture program and one Q Apply program running on each server that is part of a peer-to-peer configuration.

Important: The control tables for the Q Capture and Q Apply programs that are on each individual server must have the same schema name. For example, if you have a server named SERVER_RED and a server named SERVER_GREEN, then the Q Capture and Q Apply programs that are on SERVER_RED must both have the same schema, and the Q Capture and Q Apply programs that are on SERVER_GREEN must both have the same schema.

In a peer-to-peer configuration, conflict detection and resolution are managed automatically by the Q Apply program in a way that assures data convergence; you do not need to configure anything for conflict detection and resolution. Q Replication maintains additional information to track the version of each data change, and the Q Replication system uses this additional versioning information to detect and resolve conflicts.

All tables that are replicated in peer-to-peer replication are altered to include two columns that are used only by Q Replication: a timestamp column and a small integer column. These columns are both maintained by triggers. These extra replication columns and triggers are created when you create the Q subscriptions for peer-to-peer replication. The versioning columns reflect which version of the row is most current. By examining the values of the versioning columns, it is possible to determine at which time the row was last updated, and by which server.

Conflict detection and resolution is based on the contents of these versioning columns. If a conflict is detected, the most recent version of the row is kept, which is the one that contains the most recent timestamp value (after the times are corrected for time zones).

When you create a Q subscription for peer-to-peer replication with the ASNCLP command-line program or Replication Center, the following conflict handling options are set automatically in the IBMQREP_TARGETS table:

Conflict rule
: V (check version): The Q Apply program checks the version column before applying a row.

Conflict action
: F (force): The Q Apply program tries to force the change. This requires that the Q Capture program send all columns, so the CHANGED_COLS_ONLY value must be set to N (no) in the IBMQREP_SUBS table.

Attention: The V conflict rule and F conflict action are required for peer-to-peer replication. Do not change these settings in the control tables.

Because versioning columns are used to detect conflicts in peer-to-peer replication, columns with LOB data types are handled the same as columns with other data types.

The following topics describe the number of replication queue maps and Q subscriptions that are needed for peer-to-peer replication and how peer-to-peer replication handles referential integrity:

- "Replication objects for peer-to-peer replication with two servers"
- "Replication objects for peer-to-peer replication with three or more servers" on page 93
- "Conflict resolution and referential integrity" on page 95

Replication objects for peer-to-peer replication with two servers

In a peer-to-peer configuration with two servers, you must have the appropriate number of replication queue maps and Q subscriptions:

Number of replication queue maps
: Between each pair of servers that participate in peer-to-peer replication,

you need two replication queue maps. For example, if you have two servers named SERVER_RED and SERVER_GREEN, you need two replication queue maps:

- One to identify the WebSphere MQ queues that transport data from SERVER_RED to SERVER_GREEN
- One to identify the WebSphere MQ queues that transport data from SERVER_GREEN to SERVER_RED

Number of Q subscriptions

For every logical table that is replicated in peer-to-peer replication, you need a pair of Q subscriptions between the two servers. For example, if you have two servers named SERVER_RED and SERVER_GREEN, then two Q subscriptions are built for you:

- One from the source table on SERVER_RED to the target table on SERVER_GREEN
- One from the source table on SERVER_GREEN to the target table SERVER_RED

If you have two logical tables, you need four Q subscriptions; for three logical tables, you need six Q subscriptions, and so on.

Figure 16 shows peer-to-peer replication of one logical table between two servers. For one logical table replicated between two servers, you need two Q subscriptions: one to replicate data from peer table A to peer table B, and one to replicate data from peer table B to peer table A. You also need at least two replication queue maps.

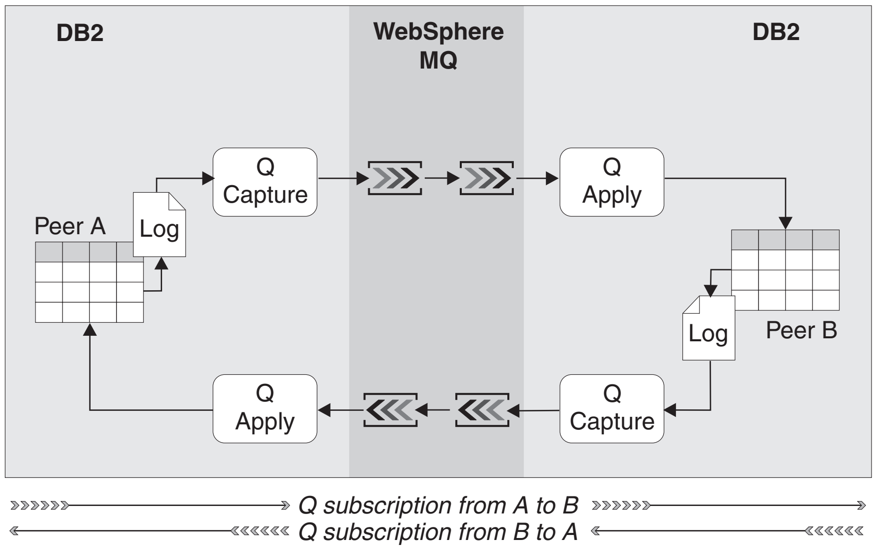


Figure 16. Q subscriptions in peer-to-peer replication with two servers. Changes are replicated from each copy of the table to the other copy of that table over WebSphere MQ queues.

Replication objects for peer-to-peer replication with three or more servers

In a peer-to-peer configuration with three or more servers, you must have the appropriate number of replication queue maps and Q subscriptions:

Number of replication queue maps

Between each pair of servers that participate in peer-to-peer replication, you need two replication queue maps. You can calculate the number of replication queue maps that you need by using the equation *n**(*n*-1), where *n* is the number of servers. For example, if you have three servers named SERVER_RED, SERVER_BLUE, and SERVER_GREEN, you need six replication queue maps:

- One to identify the WebSphere MQ queues that transport data from SERVER_RED to SERVER_GREEN
- One to identify the WebSphere MQ queues that transport data from SERVER_GREEN to SERVER_RED
- One to identify the WebSphere MQ queues that transport data from SERVER_RED to SERVER_BLUE
- One to identify the WebSphere MQ queues that transport data from SERVER_BLUE to SERVER_RED
- One to identify the WebSphere MQ queues that transport data from SERVER_BLUE to SERVER_GREEN
- One to identify the WebSphere MQ queues that transport data from SERVER_GREEN to SERVER_BLUE

Number of Q subscriptions

For every logical table that is replicated in peer-to-peer replication, there is a pair of Q subscriptions between the two servers. You can calculate the number of Q subscriptions that are built for you by using the equation *n**(*n*-1), where *n* is the number of servers. For example, if you have three servers named SERVER_RED, SERVER_GREEN, and SERVER_BLUE, then six Q subscriptions are built for you:

- One from the source table on SERVER_RED to the target table on SERVER_GREEN
- One from the source table on SERVER_GREEN to the target table on SERVER_RED
- One from the source table on SERVER_RED to the target table on SERVER_BLUE
- One from the source table on SERVER_BLUE to the target table on SERVER_RED
- One from the source table on SERVER_BLUE to the target table on SERVER_GREEN
- One from the source table on SERVER_GREEN to the target table on SERVER_BLUE

If you have two logical tables, you need 12 Q subscriptions; for three logical tables, you need 18 Q subscriptions, and so on.

Figure 17 on page 95 shows peer-to-peer replication of one logical table between three servers. In this case, you need six Q subscriptions: two going between each pair of servers. You also need at least six replication queue maps.

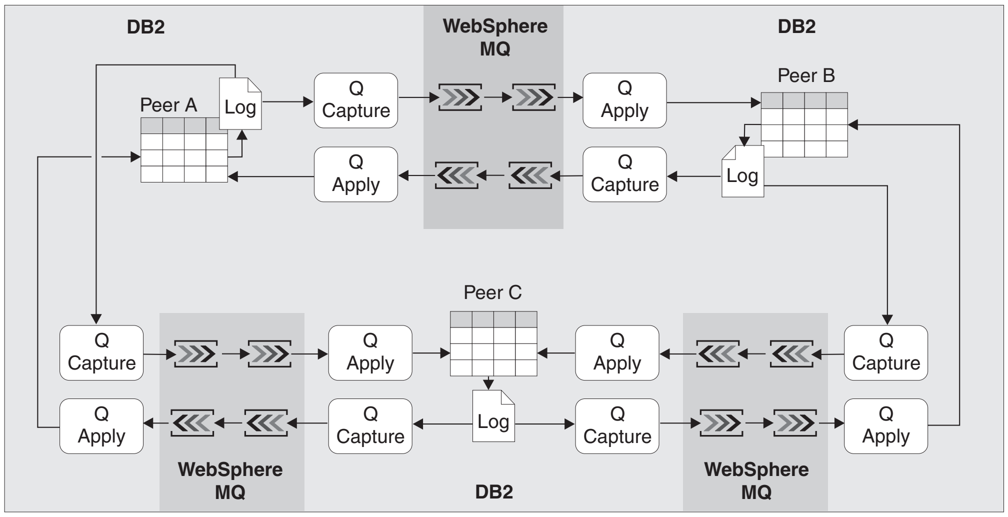


Figure 17. Q subscriptions in peer-to-peer replication with three servers. Changes are replicated from each copy of the table to all other copies of that table over WebSphere MQ queues.

Conflict resolution and referential integrity

In almost all cases, peer-to-peer replication assures that all copies of a replicated table converge to the same state, even when conflicting changes occur at different copies. However, unresolvable conflicts can occur when a conflict stems from duplicate values in unique constraints that are defined on columns other than key columns or from referential constraint violations. When an unresolvable conflict occurs, the conflicting row is recorded in the IBMQREP_EXCEPTIONS table, and the Q Apply program performs the error action that you specified for the Q subscription.

If you want specific, unresolvable conflicts to be tolerated and the Q Apply program not to perform the error action that you specified for the Q subscription, then you can specify acceptable SQLSTATE values by setting the OKSQLSTATES for the Q subscription. Note, however, that even if you specify specific SQL states in the OKSQLSTATES, peer-to-peer replication still does not ensure convergence of all copies of the table for conflicts that result from referential constraint violations or from duplicate values in unique constraints that are defined on non-key columns. You can use the table differencing utility and the table repair utility to find and repair differences that are caused by any unresolvable conflicts that you allow.

Conflicts cannot be resolved when changes occur at different copies of the replicated table that introduce the same value for a unique constraint on columns other than the key columns in the Q subscription. If you specify that SQLSTATE 23505 is allowed by adding the value to the OKSQLSTATES for the Q subscription, then any unresolvable unique key conflicts do not cause the Q Apply program to perform the error action that you specified for the Q subscription.

Conflicts cannot be resolved when changes occur in rows in different copies of tables on which referential constraints are defined. These conflicts might be caused by either delays in the propagation of messages that involve the rows or by true conflicts. An example of a true conflict is when a parent row is deleted in one copy and concurrently a child row is inserted in another copy. When the Q Apply program tries to insert the child row at the copy where the parent row was concurrently deleted, an unresolvable conflict occurs, and the Q Apply program records the child row in the IBMQREP_EXCEPTIONS table with SQLSTATE 23503. When the Q Apply program attempts to delete the parent row at the copy where the child row was concurrently inserted, the delete fails if the referential constraint's delete rule is to restrict deletes (DELETE RESTRICT). The Q Apply program records the parent row in the IBMQREP_EXCEPTIONS table with SQLSTATE 23504 or SQLSTATE 23001.

Another example of a true conflict is when a child row is concurrently inserted and removed by the delete rule (CASCADE DELETE) of the referential integrity when a delete of the parent row is applied. In this case, when the cascade delete of the child row is replicated to the other copies of the table, the other copies might not find that child row, and a SQLSTATE 02000 is reported. The same SQLSTATE 02000 might be caused by delays in the propagation of messages that involve the rows. The insert of a child row at Copy 2 might arrive at Copy 3 before the insert of the parent row at Copy 1 arrives at Copy 3.

Referential integrity for partitioned databases

In a multiple partitioned database environment with tables that have referential integrity relationships, ensure that both the parent and child rows are on the same partition. If the parent and child rows are in a referential integrity relationship and are not on the same partition, the target might have referential integrity problems that result in SQLSTATE 23504, 23001, or 23503 (which correspond to SQLCODE 530 and 532).

Avoiding deadlocks in the IBMQREP_DELTOMB table

The IBMQREP_DELTOMB table is used by the Q Apply program to record conflicting deletes in peer-to-peer replication. If you experience deadlocks in this control table on any of the servers, increase the value of the **`deadlock_retries`** parameter for the Q Apply program. Also, try to reduce delete conflicts in your workload if possible.

Bidirectional replication versus peer-to-peer replication

If you want to replicate data between tables on two servers, you have two choices for multidirectional replication: bidirectional replication or peer-to-peer replication.

The following information will help you decide whether to choose bidirectional or peer-to-peer replication to better meet your business needs. If your configuration requires more than two servers, then peer-to-peer replication is the only choice offered for multidirectional replication.

Scenarios that work best with bidirectional replication

Consider choosing bidirectional replication for the following circumstances:
- You do not expect conflicts to occur in your configuration, and you do not need to check if conflicts do occur. For minimal overhead and network traffic, specify for both servers to ignore conflicts.
- You do not expect conflicts to occur in your configuration, you want to check if conflicts do occur as a safety measure, and it is acceptable to have one server win if an unexpected data collision occurs.
- One server updates only certain columns of your data, and the other server updates the other columns. If you specify that the Q Apply program is to check

both key and changed columns for conflicts, the Q Apply program merges updates that affect different columns in the same row.
- One server updates only certain rows of your data, and the other server updates other rows.

Scenarios that work best with peer-to-peer replication

Consider choosing peer-to-peer replication if conflicts might occur in columns with LOB data types. Because versioning columns are used to detect conflicts in peer-to-peer replication, columns with LOB data types are handled the same as columns with other data types.

Creating Q subscriptions for bidirectional replication

You can create Q subscriptions that specify what changes to capture from either of two tables, what queues to use for exchanging change messages, and how to process the messages. Changes to either of the two tables replicate to the other table.

Before you begin

- Plan how you want to group replication queue maps and Q subscriptions.
- On the server that has the first copy of the table, create the control tables for the Q Capture and Q Apply programs. The control tables for the Q Capture and Q Apply programs that are on each individual server must have the same schema.
- On the server that has the second copy of the table, create the control tables for the Q Capture and Q Apply programs.
- Create the two replication queue maps that will transport data between each server. You need one replication queue map for replicating data from the first copy of the table to the second, and one for replicating data from the second copy of the table back to the first. (You can do this task before you create Q subscriptions or while you create Q subscriptions.)

Restrictions

- Stored procedures cannot participate in bidirectional replication.
- Because before values of LOB columns are not replicated in bidirectional replication, conflicts for LOB columns are not detected.
- IDENTITY columns in the target table cannot be defined as GENERATED ALWAYS.
- Q subscriptions for tables that have referential integrity relationships with each other should be created at the same time (in the same CREATE QSUB command when you are using the ASNCLP command-line program or in the same session with the Create Q Subscriptions wizard in the Replication Center).

About this task

One Q subscription is created to replicate transactions from the first copy of the table to the second copy of the table, and another Q subscription is created to replicate transactions from the second copy of the table back to the first copy. When you create Q subscriptions for bidirectional replication using the ASNCLP command-line program or the Replication Center, the administration tool creates both Q subscriptions at one time.

Procedure

To create Q subscriptions for bidirectional replication, use one of the following methods:

Method	Description
ASNCLP command-line program	Use the CREATE QSUB command for bidirectional replication. For example, the following commands set the environment and create two bidirectional Q subscriptions for the EMPLOYEE table at servers SAMPLE and SAMPLE2: `SET SUBGROUP "bidirgroup";` `SET BIDI NODE 1 SERVER DBALIAS SAMPLE SCHEMA RED;` `SET BIDI NODE 2 SERVER DBALIAS SAMPLE2 SCHEMA BLUE;` `SET CONNECTION SOURCE SAMPLE.RED` `TARGET "SAMPLE2".BLUE REPLQMAP` `"SAMPLE_RED_TO_SAMPLE2_BLUE";` `SET CONNECTION SOURCE SAMPLE2.BLUE` `TARGET SAMPLE.RED REPLQMAP` `"SAMPLE2_BLUE_TO_SAMPLE_RED";` `SET TABLES (SAMPLE.RED.RED.EMPLOYEE);` `CREATE QSUB SUBTYPE B` `FROM NODE SAMPLE.RED SOURCE` `ALL CHANGED ROWS Y HAS LOAD PHASE I` `TARGET CONFLICT RULE C CONFLICT ACTION F` `FROM NODE SAMPLE2.BLUE SOURCE` `ALL CHANGED ROWS N HAS LOAD PHASE E` `TARGET CONFLICT RULE C CONFLICT ACTION I;` The SET CONNECTION statements specify the two replication queue maps that are used. The FROM NODE statements specify options that are unique to each Q subscription.
Replication Center	Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture or Q Apply schema, right-click the **Q Subscriptions** folder, and select **Create**. On the Target Tables page, review the target object profile. Modify the profile if necessary so that the target tables for the Q subscriptions meet your needs. The target object profile determines if an existing target table is used or if a new one is created. The Replication Center looks for an object that matches the naming scheme that is defined in the profile. If a matching object does not exist, then the object is created.

Improving performance in bidirectional replication with the IBMQREP_IGNTRAN table

You can use the Q Capture program's ability to ignore specified transactions to improve performance in a pure, two-server bidirectional configuration.

About this task

To avoid the recapture of transactions, by default the Q Apply program inserts P2PNORECAPTURE signals into the IBMQREP_SIGNAL table for each transaction that it applies. The signals are inserted at the Q Capture server that is shared by the Q Apply program. When Q Capture reads the signals in the log, it ignores these transactions.

When there are many bidirectional Q subscriptions, the number of signal inserts can affect replication performance. To avoid this, you can specify that the programs use the IBMQREP_IGNTRAN table to avoid recapture. This method tells the Q Capture program to automatically ignore any changes that come from the Q Apply program. You also start the Q Apply program with the **insert_bidi_signal**=N startup parameter.

Use the following guidelines to determine which method to use to avoid recapture:

Table 9. Recapture avoidance methods for different bidirectional replication configurations

Configuration	Recapture avoidance method
Multiple bidirectional configurations between two servers, or a combination of bidirectional, unidirectional, and peer-to-peer configurations	**You must accept the default method** The default method of signal inserts ensures that all changes are propagated correctly between servers. If you start with a pure, two-server bidirectional topology but plan to later add branching unidirectional or peer-to-peer configurations, you should also accept the default method of recapture avoidance.
Pure, two-server bidirectional configuration	**Performance can be improved by using the IBMQREP_IGNTRAN table to avoid recapture** If you use the IBMQREP_IGNTRAN table method, do not later add branching unidirectional or peer-to-peer configurations to the bidirectional configuration.

Procedure

To use the IBMQREP_IGNTRAN table to avoid recapture in bidirectional replication:

1. Insert an identifier for the Q Apply program into the IBMQREP_IGNTRAN table at the server that is shared by the Q Apply program and Q Capture program in the bidirectional configuration. Use the following SQL depending on your operating system:

   z/OS

   ```
   insert into schema.IBMQREP_IGNTRAN (
       PLANNAME,
       IGNTRANTRC
   ) values (
       'qapply_plan',
       'N' );
   ```

   Where *schema* is the schema that is shared by the Q Apply program and Q Capture program at the server, and *qapply_plan* is the plan name for the Q Apply program, for example ASNQA910 for a Version 9.1 Q Apply program.

Linux UNIX Windows

```
insert into schema.IBMQREP_IGNTRAN (
    AUTHID,
    IGNTRANTRC
) values (
    'qapply_authid',
    'N' );
```

Where *schema* is the schema shared by the Q Apply program and Q Capture program at the server, and *qapply_authid* is the authorization ID that started the Q Apply program.

To use the IBMQREP_IGNTRAN table option, the Q Apply program's authorization ID must be unique and not shared by other applications, except for the Q Capture program. Otherwise, the Q Capture program will incorrectly ignore transactions from these other applications as well. On z/OS, this situation is rare because plan names are unique. However, on Linux, UNIX, and Windows it is not unusual to run the Q Apply program under the same authorization ID as other applications.

If Q Apply was migrated to Version 10.1 on z/OS: The Q Apply plan name for Version 10.1 on z/OS changed to ASNQA101 from ASNQA910. To ensure that transactions from the Version 10.1 Q Apply program are ignored, after Q Apply is migrated to Version 10.1 you must enter the new ASNQA101 plan name in the IBMQREP_IGNTRAN table and then either reinitialize the Q Capture program or stop and start Q Capture so that it reads the new plan name. As a safeguard, you can put both the ASNQA101 and ASNQA910 plan names in IBMQREP_IGNTRAN.

The Q Capture program ignores all replicated transactions that come from the specified Q Apply plan name or authorization ID, but continues to read and process signals from Q Apply in the IBMQREP_SIGNAL table.

2. When you start the Q Apply program, specify the **insert_bidi_signal**=n startup parameter. The short form syntax is INS=N.
3. Optional: For improved performance when you use **insert_bidi_signal**=n, ensure that the value of the IGNTRANTRC column in the IBMQREP_IGNTRAN table is set to N (no tracing). A value of N, the default, prevents the Q Capture program from inserting a row into the IBMQREP_IGNTRANTRC table for each transaction that it does not recapture and reduces maintenance overhead on the table.

Options for conflict detection (bidirectional replication)

The choices that you make for conflict rules and conflict actions affect the behavior of how rows are applied. The conflict rules determine how much of the data is checked to detect a conflict and the types of conflicts that are detected. When you choose to have more data checked for conflicts, then the Q Capture program must send more data to the Q Apply program to make that data available to be checked, which might influence performance and network traffic.

For conflict detection in bidirectional replication, before values at the source server are compared against the current values at the target server. Based on the level of conflict detection that you choose, the Q Capture program sends a different combination of before or after values to the Q Apply program. The information here is provided to help you make a more informed decision about the level of conflict detection.

Note: Because before values are used to detect conflicts in bidirectional replication and Q Replication does not replicate before values for LOB data types, conflicts in LOB columns are not detected.

The following sections describe your options for conflict detection in bidirectional replication and the results of different combinations of conflict options:

- "How the Q Apply program checks for conflicts"
- "How conflicts are resolved at each server"
- "Outcomes of different choices for checking and resolving conflicts" on page 102

How the Q Apply program checks for conflicts

You can choose for the Q Apply program to check one of the following groups of columns when determining conflicts:

- Only key columns
- Key columns and changed columns
- All columns

Only key columns
: The Q Apply program attempts to update or delete the target row by checking the values in the key columns. The Q Apply program detects the following conflicts:
  - A row is not found in the target table.
  - A row is a duplicate of a row that already exists in the target table.

  With this conflict rule, the Q Capture program sends the least amount of data to the Q Apply program for conflict checking. No before values are sent, and only the after values for any changed columns are sent.

Key and changed columns
: The Q Apply program attempts to update or delete the target row by checking the key columns and the columns that changed in the update. The Q Apply program detects the following conflicts:
  - A row is not found in the target table.
  - A row is a duplicate of a row that already exists in the target table.
  - A row is updated at both servers simultaneously and the same column values changed.

  If a row is updated at both servers simultaneously and the different column values changed, then there is no conflict. With this conflict rule, the Q Apply program merges updates that affect different columns into the same row. Because the Q Apply program requires the before values for changed columns for this conflict action, the Q Capture program sends the before values of changed columns.

All columns
: The Q Apply program attempts to update or delete the target row by checking all columns that are in the target table. With this conflict rule, the Q Capture program sends the greatest amount of data to the Q Apply program for conflict checking.

How conflicts are resolved at each server

For each server, you can choose what action each server takes when a conflict occurs. Each server can either force the conflicting row into its target table or ignore the conflict. These options of force and ignore can be paired in two different ways to provide different behaviors for the Q Apply program.

One server forces conflicts, the other server ignores conflicts
One server (the one with the conflict action of ignore) wins if a conflict occurs; this server is the "master" or source owner of the data. If a row is updated at both servers simultaneously and the same column values changed, then the master server (the one with the conflict action of ignore) ignores the conflict, and the row from the master server is forced in the target table on the other server (the one with the conflict action of force). For this conflict action, the Q Capture program sends the before values of all columns to the Q Apply program. The Q Apply program logs all conflicts in the IBMQREP_EXCEPTIONS table.

Both servers ignore conflicts
Any time a conflict occurs because a row is not found or a row is a duplicate of a row that already exists in the target table, the Q Apply program logs the conflict in the IBMQREP_EXCEPTIONS table, but otherwise ignores the conflict. This conflict action specifies that the Q Capture program does not send before values to the Q Apply program for conflict checking. Only the after values for any changed columns are sent.

Recommendation: Set both servers to ignore conflicts if you do not expect any conflicts to occur between the two servers and you want the least overhead to be used for conflict detection by the Q Capture and Q Apply programs.

Outcomes of different choices for checking and resolving conflicts

Table 10 on page 103 describes the outcomes for different combinations of options that you can choose from for conflict detection. In all cases, the first server is the server that you have opened the wizard from.

Table 10. Outcomes of different combinations of options for conflict detection

How to check for conflicts	How to resolve conflicts	Outcome
Check all columns for conflicts.	The first server takes precedence.	For the Q subscription from the first server to the second server: If any change made to the source table at the first server conflicts with data in the target table at the second server, the Q Apply program applies the source change to the target table. The Q Apply program logs the conflict in the IBMQREP_EXCEPTIONS table, deletes the conflicting row in the target table, and inserts the row from the source table.
		For the Q subscription from the second server to the first server: If any change made to the source table conflicts with data in the target table, the Q Apply program logs the conflict but does not force the change into the target table.
	The second server takes precedence.	For the Q subscription from the first server to the second server: If any change made to the source table conflicts with data in the target table, the Q Apply program logs the conflict but does not force the change into the target table.
		For the Q subscription from the second server to the first server: If any change made to the source table conflicts with data in the target table, the Q Apply program applies the source change to the target table. The Q Apply program deletes the conflicting row in the target table and inserts the row from the source table. If there is a conflicting delete and the row is not found at the target table, the Q Apply program ignores the delete from the source.
	Neither server takes precedence.	The Q Apply program logs all conflicts in the IBMQREP_EXCEPTIONS table and continues processing. Over time, the two copies of a logical table will diverge.

Table 10. Outcomes of different combinations of options for conflict detection (continued)

How to check for conflicts	How to resolve conflicts	Outcome
Check only changed non-key columns for conflicts.	The first server takes precedence.	For the Q subscription from the first server to the second server: If a change to a non-key column in the source table conflicts with a change made to the corresponding column in the target table, the Q Apply program applies the source change to the target table anyway. The Q Apply program deletes the conflicting row in the target table and inserts the row from the source table. If there is a conflicting delete and the row is not found at the target table, the Q Apply program ignores the delete from the source.
		For the Q subscription from the second server to the first server: If a change to a non-key column in the source table conflicts with a change made to the corresponding column in the target table, the Q Apply program logs the conflict but does not force the change into the target table.
	The second server takes precedence.	For the Q subscription from the first server to the second server: If a change to a non-key column in the source table conflicts with a change made to the corresponding column in the target table, the Q Apply program logs the conflict but does not force the change into the target table.
		For the Q subscription from the second server to the first server: If a change to a non-key column in the source table conflicts with a change made to the corresponding column in the target table, the Q Apply program applies the source change to the target table anyway. The Q Apply program deletes the conflicting row in the target table and inserts the row from the source table. If there is a conflicting delete and the row is not found at the target table, the Q Apply program ignores the delete from the source.
	Neither server takes precedence.	The Q Apply program logs all conflicts in the IBMQREP_EXCEPTIONS table and continues processing. Over time, the two copies of a logical table will diverge.

Table 10. Outcomes of different combinations of options for conflict detection (continued)

How to check for conflicts	How to resolve conflicts	Outcome
Check only key columns for conflicts.	The first server takes precedence.	For the Q subscription from the first server to the second server: If a change to the key at the source table conflicts with the key at the target table, the Q Apply program applies the source change to the target table. The Q Apply program deletes the conflicting row in the target table and inserts the row from the source table. If there is a conflicting delete and the row is not found at the target table, the Q Apply program ignores the delete from the source.
		For the Q subscription from the second server to the first server: If a change to the key at the source table conflicts with the key at the target table, the Q Apply program logs the conflict but does not force the change into the target table.
	The second server takes precedence.	For the Q subscription from the first server to the second server: If a change to the key at the source table conflicts with the key at the target table, the Q Apply program logs the conflict but does not force the change into the target table.
		For the Q subscription from the second server to the first server: If a change to the key at the source table conflicts with the key at the target table, the Q Apply program applies the source change to the target table. The Q Apply program deletes the conflicting row in the target table and inserts the row from the source table. If there is a conflicting delete and the row is not found at the target table, the Q Apply program ignores the delete from the source.
	Neither server takes precedence.	The Q Apply program logs all conflicts in the IBMQREP_EXCEPTIONS table and continues processing. Over time, the two copies of a logical table will diverge.

Creating Q subscriptions for peer-to-peer replication

You can create Q subscriptions to map peer tables to one another so that changes are replicated back and forth from each table. This task is part of the larger task of setting up replication from sources to targets (multidirectional).

Creating Q subscriptions for peer-to-peer replication with two servers

In peer-to-peer replication with two servers, changes that are made to one copy of a table are replicated to a second copy of that table, and changes from the second copy are replicated back to the first copy. In peer-to-peer replication, all rows and columns are replicated, and the column names in each copy of the table must match.

Before you begin

Before you create Q subscriptions for peer-to-peer replication, you must perform the following actions:

- On each server that will participate in peer-to-peer replication, create the control tables for the Q Capture and Q Apply programs. The control tables for the Q Capture and Q Apply programs that are on each individual server must have the same schema.
- Create the two replication queue maps that will transport data between the pair of servers. You need one replication queue map for replicating data from the first copy of the table to the second, and one for replicating data from the second copy of the table back to the first. (You can do this task before you create Q subscriptions or while you create Q subscriptions.)

Restrictions

- Stored procedures cannot participate in peer-to-peer replication.
- z/OS If the source table includes a LONG VARCHAR column type, that table cannot participate in a peer-to-peer replication. For peer-to-peer replication, the Replication Center or ASNCLP must add two columns to the source table. DB2 for z/OS does not allow the replication administration tools to add columns to a table that includes a LONG VARCHAR column.
- For peer-to-peer and bidirectional configurations, do not use the IMPORT utility. The IMPORT utility logs the inserts and, therefore, the inserts will be recaptured.
- In peer-to-peer and bidirectional replication, you must use the same constraints on both the source and target.
- IDENTITY columns in tables that are in peer-to-peer configurations must be defined as GENERATED BY DEFAULT.
- Peer-to-peer replication is not supported on systems that use IBM HourGlass to alter the date and time that is returned when a time request is made. This software alters the version columns that are required for peer-to-peer.

About this task

One Q subscription is created from the first peer table to the second, and another Q subscription is created from the second peer table back to the first. When you create Q subscriptions for peer-to-peer replication using the ASNCLP command-line program or Replication Center, the administration tool creates both Q subscriptions at one time.

Procedure

To create Q subscriptions for peer-to-peer replication with two servers, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE QSUB command for peer-to-peer replication. For example, the following commands set the environment and create two peer-to-peer Q subscriptions for the EMPLOYEE table at servers SAMPLE and SAMPLPEER:
	`SET SUBGROUP "p2p2group";`
	
	`SET PEER NODE 1 SERVER DBALIAS SAMPLE`
	`SCHEMA GREEN;`
	`SET PEER NODE 2 SERVER DBALIAS SAMPLPEER`
	`SCHEMA MAGENTA;`
	
	`SET CONNECTION SOURCE SAMPLE.GREEN`
	`TARGET SAMPLPEER.MAGENTA REPLQMAP`
	`"SAMPLE_GREEN_TO_SAMPLPEER_MAGENTA";`
	`SET CONNECTION SOURCE SAMPLPEER.MAGENTA`
	`TARGET SAMPLE.GREEN REPLQMAP`
	`"SAMPLPEER_MAGENTA_TO_SAMPLE_GREEN";`
	
	`SET TABLES (SAMPLE.GREEN.GREEN.EMPLOYEE);`
	
	`CREATE QSUB SUBTYPE P;`
	
	The SET CONNECTION commands specify the two replication queue maps that are used. The SET TABLES command specifies the EMPLOYEE table at the SAMPLE server. A matching copy of the table will be created at the SAMPLPEER server.

Replication Center	Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture or Q Apply schema, right-click the **Q Subscriptions** folder, and select **Create**. **Target Tables page** Review the target object profile. Modify the profile if necessary so that the target tables for the Q subscriptions meet your needs. The target object profile determines if an existing target table is used or if a new one is created. The Replication Center looks for an object that matches the naming scheme that is defined in the profile. If a matching object does not exist, then the object is created.

Creating Q subscriptions for peer-to-peer replication with three or more servers

In peer-to-peer replication with three or more servers, changes that are made to each copy of a table are replicated to all other copies of that table. All rows and columns are replicated, and the column names in each copy of the table must match.

Before you begin

Before you create Q subscriptions for peer-to-peer replication, you must perform the following actions:

- On each server that will participate in peer-to-peer replication, create the control tables for the Q Capture and Q Apply programs. The control tables for the Q Capture and Q Apply programs that are on each individual server must have the same schema.

- Create the replication queue maps that will transport data between each pair of servers. You need one replication queue map for each source-to-target pair. If you have one table that you want to replicate between three servers, you need six replication queue maps. If you have one table that you want to replicate between four servers, you need twelve replication queue maps. (You can do this task before you create Q subscriptions or while you create Q subscriptions.)

Restrictions

- Views and stored procedures cannot participate in peer-to-peer replication.
- z/OS If the source table includes a LONG VARCHAR column type, that table cannot participate in a peer-to-peer replication. For peer-to-peer replication, the Replication Center or ASNCLP must add two columns to the source table. DB2 for z/OS does not allow the replication administration tools to add columns to a table that includes a LONG VARCHAR column.
- For peer-to-peer and bidirectional configurations, do not use the IMPORT utility. The IMPORT utility logs the inserts and, therefore, the inserts will be recaptured.
- In peer-to-peer and bidirectional replication, you must use the same constraints on both the source and target.
- IDENTITY columns in tables that are in peer-to-peer configurations must be defined as GENERATED BY DEFAULT.
- Peer-to-peer replication is not supported on systems that use IBM HourGlass to alter the date and time that is returned when a time request is made. This software alters the version columns that are required for peer-to-peer.

About this task

One Q subscription is created for each source-to-target pair. If you have one table that you want to replicate to and from three servers, six Q subscriptions are created. If you have one table that you want to replicate to and from four servers, twelve Q subscriptions are created. The formula for determining the number of Q subscriptions that are created is *n*(n-1)*, where *n* is the number of servers. When you create Q subscriptions for peer-to-peer replication by using the ASNCLP command-line program or Replication Center, the administration tool creates all necessary Q subscriptions at one time.

Procedure

To create Q subscriptions for peer-to-peer replication with three or more servers, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE QSUB command for peer-to-peer replication. For example, the following commands set the environment and create six peer-to-peer Q subscriptions for the EMPLOYEE table at servers SAMPLE, SAMPLE2, and SAMPLE3:
	`SET SUBGROUP "p2p3group";`
	
	`SET PEER NODE 1 SERVER DBALIAS SAMPLE`
	`SCHEMA GRAY;`
	`SET PEER NODE 2 SERVER DBALIAS SAMPLE2`
	`SCHEMA BROWN;`
	`SET PEER NODE 3 SERVER DBALIAS SAMPLE3`
	`SCHEMA YELLOW;`
	
	`SET CONNECTION SOURCE SAMPLE.GRAY`
	`TARGET SAMPLE2.BROWN REPLQMAP`
	`"SAMPLE_GRAY_TO_SAMPLE2_BROWN";`
	`SET CONNECTION SOURCE SAMPLE.GRAY`
	`TARGET SAMPLE3.YELLOW REPLQMAP`
	`"SAMPLE_GRAY_TO_SAMPLE3_YELLOW";`
	`SET CONNECTION SOURCE SAMPLE2.BROWN`
	`TARGET SAMPLE.GRAY REPLQMAP`
	`"SAMPLE2_BROWN_TO_SAMPLE_GRAY";`
	`SET CONNECTION SOURCE SAMPLE2.BROWN`
	`TARGET SAMPLE3.YELLOW REPLQMAP`
	`"SAMPLE2_BROWN_TO_SAMPLE3_YELLOW";`
	`SET CONNECTION SOURCE SAMPLE3.YELLOW`
	`TARGET SAMPLE.GRAY REPLQMAP`
	`"SAMPLE3_YELLOW_TO_SAMPLE_GRAY";`
	`SET CONNECTION SOURCE SAMPLE3.YELLOW`
	`TARGET SAMPLE2.BROWN REPLQMAP`
	`"SAMPLE3_YELLOW_TO_SAMPLE2_BROWN";`
	
	`SET TABLES (SAMPLE.GRAY.GRAY.STAFF);`
	
	`CREATE QSUB SUBTYPE P;`
	
	The SET CONNECTION commands specify the six replication queue maps that are used. The SET TABLES command specifies the EMPLOYEE table at the SAMPLE server. Matching copies of the table will be created at the SAMPLE2 and SAMPLE3 servers.

Method	Description
Replication Center	Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture or Q Apply schema, right-click the **Q Subscriptions** folder, and select **Create**. **Servers page** Each time that you specify a server to participate in peer-to-peer replication, you must specify the replication queue map from that new server to each of the other servers that you already selected. You must also specify the replication queue map from each existing server back to the new server. **Target Tables page** Review the target object profile for each server other than the server that contains the existing (base) tables. Modify the profile if necessary so that the target tables for the Q subscriptions meet your needs. The target object profile determines if an existing target table is used or if a new one is created. The Replication Center looks for an object that matches the naming scheme that is defined in the profile. If a matching object does not exist, then the object is created. **Loading Target Table page** When you create Q subscriptions for peer-to-peer replication with three or more servers, the Q subscriptions are always created in an inactive state, and you must activate them.

Starting bidirectional or peer-to-peer replication with two servers

To start bidirectional or peer-to-peer replication with two servers, you start the Q Capture and Q Apply programs at both servers, and then make sure that Q subscriptions are activated.

Before you begin

Linux UNIX Windows If a table that participates in a peer-to-peer Q subscription contains data before the Q subscription is created, you must run the REORG command against the table after you create the Q subscription and before you start the Q Capture programs at the various peer servers.

About this task

By default, when you create Q subscriptions for bidirectional or peer-to-peer replication with two servers, the new Q subscriptions are automatically started when you start or reinitialize the Q Capture program. However, Q subscriptions are only automatically started when they are new. If you stop replication of a logical table and want to start replication again, you must follow this procedure to manually start replication.

Important note about Version 9.7 changes: The initialization protocol for multidirectional replication changed with Version 9.7 so that replication does not pause while all Q subscriptions are being activated. Because of this change, if any of the Q Capture or Q Apply programs in the configuration are migrated to Version 9.7 and you need to add a new Q subscription or activate a disabled Q subscription, all of the Q Capture and Q Apply programs in the configuration must be at Version 9.7. If a Q Capture program is participating in both unidirectional and bidirectional or peer-to-peer configurations and any of the servers are migrated to V9.7, all components that are involved in both unidirectional and multidirectional configurations must be migrated to V9.7.

Procedure

To start replication for a logical table in bidirectional or peer-to-peer replication with two servers

1. Start the Q Capture and Q Apply programs at both servers. You can cold start or warm start the Q Capture programs:

   cold start
   If you use a cold start, you must start the Q Capture programs at each server before you start the Q Apply programs. At each server, make sure the Q Capture program is started before you start the Q Apply program by using the Check Status window in the Replication Center or by checking the Q Capture log for an initialization message.

   warm start
   If you use a warm start, you can start the Q Capture and Q Apply program at each server in any order.

   If you created the Q subscriptions to be automatically started, replication begins when you start the Q Capture and Q Apply programs.
2. If you created the Q subscriptions without automatic start or if you are restarting replication, start one of the two Q subscriptions for the logical table. The Q Capture and Q Apply programs automatically start the other Q subscription.

If the Q subscriptions specify a load phase, the source table for the Q subscription that you start is used to load the target table at the other server.

Starting peer-to-peer replication with three or more servers

After you create the Q subscriptions for a logical table in peer-to-peer replication with three or more servers, you must start the group of Q subscriptions to start replication.

Before you begin

- The Q Capture and Q Apply programs must be running at all servers in the group. You can cold start or warm start the Q Capture programs:

  cold start
  If you use a cold start, you must start the Q Capture programs at each server before you start the Q Apply programs. At each server, make sure the Q Capture program is started before you start the Q Apply program by using the Check Status window in the Replication Center or by checking the Q Capture log for an initialization message.

  warm start
  If you use a warm start, you can start the Q Capture and Q Apply program at each server in any order.
- Linux UNIX Windows If a table that participates in a peer-to-peer Q subscription contains data before the Q subscription is created, you must run the REORG

command against the table after you create the Q subscription and before you start the Q Capture programs at the various peer servers.

About this task

You also must start the group if you stopped all of the Q subscriptions for a logical table and you want replication to start again.

Starting a peer-to-peer group with three or more servers is a staged process. First you start the Q subscriptions for a logical table between two servers, and then you add new servers one at a time until all the servers are actively replicating the logical table.

You can only add one new server at a time to the group. Begin the process of adding a new server only after the other servers are actively replicating the logical table.

Important note about Version 9.7 changes: The initialization protocol for multidirectional replication changed with Version 9.7 so that replication does not pause while all Q subscriptions are being activated. Because of this change, if any of the Q Capture or Q Apply programs in the configuration are migrated to Version 9.7 and you need to add a new Q subscription or activate a disabled Q subscription, all of the Q Capture and Q Apply programs in the configuration must be at Version 9.7. If a Q Capture program is participating in both unidirectional and bidirectional or peer-to-peer configurations and any of the servers are migrated to V9.7, all components that are involved in both unidirectional and multidirectional configurations must be migrated to V9.7.

Procedure

To start replication in a peer-to-peer group with three or more servers:

1. Choose two servers in the group to begin the activation process.
2. Start one of the two Q subscriptions for a logical table between the two servers.

   The Q Capture and Q Apply programs automatically start the other Q subscription for this logical table between the two servers. If the Q subscriptions specify a load phase, the Q subscription that you start must be the Q subscription with the source table that you want to load from. This table is used to load the table at the other server.
3. After both Q subscriptions are active, choose a new server to bring into the group.
4. Choose one of the servers that is actively replicating the logical table.
5. Start the Q subscription for the logical table that specifies the server that you chose in Step 4 as its source, and the new server as its target.

   The Q Capture and Q Apply programs start the other Q subscription for the logical table between the new server and the server that is actively replicating. The Q subscriptions for the logical table between the new server and the other server that is actively replicating are also started.

   At this point, replication begins in all directions between all servers.
6. Follow steps 3, 4, and 5 until all of the Q subscriptions in the group are active.

Stopping bidirectional or peer-to-peer replication with two servers

In bidirectional or peer-to-peer replication with two servers, you can stop replication of a logical table without stopping the Q Capture or Q Apply programs. To do so, you deactivate the Q subscriptions for the logical table. Replication of other logical tables continues between the two servers.

Before you begin

The Q subscriptions for the logical table must be in A (active) state.

Restrictions

You cannot stop only one of the two Q subscriptions for a logical table. When you stop one Q subscription, the other is automatically stopped.

Procedure

To stop replication of a logical table in bidirectional or peer-to-peer replication with two servers, stop one of the two Q subscriptions.

Stopping peer-to-peer replication with three or more servers

In peer-to-peer replication with three or more servers, you can stop replication of a logical table without stopping the Q Capture or Q Apply programs.

Before you begin

The Q subscriptions for the logical table must be in A (active) state.

About this task

You can stop replication of the logical table at one server, or at all servers in the group.

To stop replication of a logical table at all servers in a group, follow this same procedure, one server at a time, until all Q subscriptions for the logical table are stopped.

Procedure

To stop replication of a logical table:

1. Choose a server in the group that is actively replicating the logical table.
2. Stop the Q subscription that specifies this server as its source.

Chapter 7. Replicating Data Definition Language (DDL) operations

You can speed up and simplify the configuration and maintenance of DB2 continuous availability and disaster recovery solutions by taking advantage of the Data Definition Language (DDL) replication features in InfoSphere Data Replication Version 10.1.3 and later.

Automatic replication of Data Definition Language (DDL) operations eliminates or greatly reduces the need for administrator interventions when tables are added or changed at the primary database. When you take advantage of this feature, the replication programs automatically subscribe newly created tables for replication and create or drop these tables at targets.

DDL replication helps ensure that the failover database truly reflects the standby database. The following topics provide more detail about DDL replication support.

Schema-level subscriptions and DDL replication

The mechanism for replicating Data Definition Language (DDL) changes such as CREATE TABLE or DROP TABLE between databases is a *schema-level subscription*.

A schema-level subscription is a pattern based on "*schema_name.object_name*" that indicates which schema-level statements are replicated. It tells the replication programs to respond to DDL changes within certain database schemas – and certain tables within these schemas – by capturing the DDL statements, publishing them as WebSphere MQ messages, and replaying the DDL at a target database. A schema-level subscription is different from a *table-level Q subscription* which is associated with moving data from a particular source to a particular target.

Each part of the pattern can contain a wild card suffix that is represented by the percentage sign character (%).

Another important component of schema-level subscriptions is a profile that specifies how table-level Q subscriptions should be created. The profile is used by both the ASNCLP command-line program and the replication programs to create Q subscriptions for tables that match the pattern. You create both the schema-level subscription and the profile with the ASNCLP program.

The CREATE SCHEMASUB command performs the following tasks:

- Creates a schema-level subscription that includes the schemas that you specify.
- Creates table-level Q subscriptions for all tables within the schemas.
- Saves the schema pattern and the profile so that the replication programs automatically create table-level Q subscriptions with your specified options for any tables that are added within the schema.

Schema-level subscription information is stored in the IBMQREP_SCHEMASUBS table.

The following sections provide more detail about schema-level subscriptions and DDL replication:

- "Using wild cards to specify the included schemas and tables" on page 116

- “Supported DDL operations for replication”
- “Relationship between schema-level subscriptions and queue maps” on page 117
- “Profiles for creating table-level Q subscriptions” on page 117
- “Creation of new target tables” on page 118

Using wild cards to specify the included schemas and tables

A schema-level subscription can include one or many schemas, and one or many tables within those schemas. To specify multiple schemas and tables, you use the percentage sign character (%) as a wild card. The wild card must be used as a suffix; for example given a schema-level subscription for "ANU%.%", any table that is created with a schema name that starts with the string "ANU" is automatically replicated.

Wild cards greatly simplify the task of subscribing an entire database, and you can also use wild cards to exclude certain schemas or tables from the schema-level subscription.

Replicating an entire database

If you want to replicate all CREATE TABLE and DROP TABLE operations for all tables in all schemas, you can do so by creating a schema-level subscription for the schema-object pattern: "%. %". In the ASNCLP program’s CREATE SCHEMASUB command, you can accomplish this by simply using the ALL keyword, as in the following example:

```
CREATE SCHEMASUB SUBTYPE B FOR TABLES NODE 1 ALL OPTIONS options1;
```

This command says, "Create two schema-level Q subscriptions, one in each direction, for a bidirectional replication configuration; replicate CREATE TABLE and DROP TABLE for all tables in all schemas; and use the options1 profile to create the necessary table-level Q subscriptions."

Excluding schemas and tables from replication

When you specify schema names and objects to exclude from the schema-level subscription, the exclusions are stored in the IBMQREP_EXCLSCHEMA control table. For example, given a schema-level subscription for "%.%" but an entry in the IBMQREP_EXCLSCHEMA table for "BOB," the statement CREATE TABLE BOB.T1 would not be replicated even with a schema-level subscription active for "%.%".

IBMQREP_EXCLSCHEMA table: Specifies schemas and tables to exclude from replication.

SCHEMA_NAME	OBJECT_NAME	QMAP
BOB	T%	SITEA2B

Wild cards are not allowed when you are excluding schemas but can be used to exclude tables. The exclusion only pertains to the schema-level subscription that uses a specified replication queue map.

By default, DB2 and replication catalog tables are already excluded even if you use a pattern of "%.%".

Supported DDL operations for replication

For the first release of schema-level subscriptions, the CREATE TABLE and DROP TABLE operations are supported.

The ALTER TABLE ADD COLUMN and ALTER TABLE ALTER COLUMN SET DATA TYPE operations can be automatically replicated with replication Version 10.1 on both z/OS and Linux, UNIX, and Windows. These operations are carried out at the table level and are used for newly added columns; the operations are not part of the schema-level subscription.

To add existing source table columns to Q subscriptions, you can use the Q Replication Dashboard, Replication Center, or ASNCLP program. The replication programs add columns to the Q subscription and can also add the columns to the target table. See “Adding existing columns to a Q subscription (unidirectional replication)” on page 131 and “Adding existing columns to a Q subscription (bidirectional or peer-to-peer replication)” on page 133 for details.

You can also specify that LOAD operations at source tables are replicated. See “Replicating load operations at the source table” on page 190 for details.

RENAME TABLE is not supported. If you rename a table, you must stop the table-level Q subscription, update the Q subscription properties with the new table name, and reinitialize the Q Capture program.

Relationship between schema-level subscriptions and queue maps

You cannot have multiple table-level Q subscriptions for the same source table that specify the same replication queue map, and schema-level subscription patterns for the same queue map must not overlap. For example, the patterns, "SERGE.%" and "%.T1" both match the table SERGE.T1. This type of overlap is not allowed, and the ASNCLP program enforces the restriction when creating schema-level subscriptions.

For example, if a schema subscription with the pattern "ANUPAMA.%" already exists, you cannot add a schema-level subscription with the pattern "%.%", because tables could match both patterns. The CREATE SCHEMASUB command would fail.

You can have multiple schema-level subscriptions for the same set of tables, but to different queue maps. For example:

- %.% in bidirectional replication on QUEUEMAP1, for disaster recovery
- ANUPAMA.% in unidirectional replication on QUEUEMAP2, for feeding a warehouse

Profiles for creating table-level Q subscriptions

The profile that you create with the ASNCLP’s CREATE SUBSCRIPTION OPTIONS command is used for two purposes:

- By the ASNCLP program to create table-level Q subscriptions for existing tables that match the schema-level subscription pattern.
- By the replication programs to create table-level Q subscriptions in response to a CREATE TABLE operation that matches the pattern.

Creating the profile is one of the first things that you do when setting up DDL replication because you need to specify the profile in the CREATE SCHEMASUB command. If you do not create a profile, the ASNCLP program and replication programs use default profiles for unidirectional and bidirectional replication.

Profiles are stored in the IBMQREP_SUBS_PROF table on the Q Capture server.

For more detail, see "Creating profiles for table-level Q subscriptions" on page 123.

Creation of new target tables

When it creates new target tables after a CREATE TABLE operation at the source, the Q Apply program creates the target table with all indexes and constraints that existed at the source, even if the indexes and constraints were created with several ALTER statements and in multiple transactions.

Q Apply replays the DDL statements as they were executed at the source, except that it issues a SET CURRENT DEFAULT SCHEMA statement before it creates each table and then changes its ownership to be the same as the user who created the object at the source database (otherwise the creator for the object would be the user ID that started the Q Apply program).

For existing source tables, the ASNCLP program uses the options that are specified in the SET PROFILE command to create new target tables when it processes a CREATE SCHEMASUB command.

How Q Capture handles DDL operations at the source database

The Q Capture program automatically replicates some Data Definition Language (DDL) operations at the source database. Other DDL changes require you to take action at the target database.

The following table describes how Q Capture handles different types of DDL operations and what you need to do for any affected Q subscriptions.

Table 11. How Q Capture handles DDL changes to the source database and what you need to do

DDL operation	How it is handled	What you need to do
CREATE TABLE	**Version 10.1 on Linux, UNIX, and Windows** Automatically replicated if the new source table matches the schema- and table-naming pattern in a schema-level subscription. When the Q Capture program detects a CREATE TABLE operation in the log that matches a schema-level subscription, it informs the Q Apply program to create a matching target table. A table-level Q subscription is also created that maps the new source and target tables. **z/OS or earlier versions on Linux, UNIX, and Windows** No automatic replication of CREATE TABLE operations.	**Version 10.1 on Linux, UNIX, and Windows** Ensure that newly created source table matches the schema- and table-naming pattern in a schema-level subscription. **z/OS or earlier versions on Linux, UNIX, and Windows** Create a table-level Q subscription for the new source table and use the replication administration tools to create a matching target table, or use an existing target table.

Table 11. How Q Capture handles DDL changes to the source database and what you need to do (continued)

DDL operation	How it is handled	What you need to do
DROP TABLE	**Version 10.1 on Linux, UNIX, and Windows** Automatically replicated if the source table is part of a schema-level subscription. When the Q Capture program detects a DROP TABLE operation in the log that matches a schema-level subscription, the associated table-level Q subscriptions for all queues are also dropped. **z/OS or earlier versions on Linux, UNIX, and Windows** Q Capture leaves the Q subscription active, but there are no log records to read for the source table. On z/OS, the ASN0197W warning message is issued.	**Version 10.1 on Linux, UNIX, and Windows** Ensure that source table is included in a schema-level subscription. **z/OS or earlier versions on Linux, UNIX, and Windows** When you drop a table the Q subscription for the table still exists. To remove, stop the Q subscription and then delete the Q subscription.

Table 11. How Q Capture handles DDL changes to the source database and what you need to do (continued)

DDL operation	How it is handled	What you need to do
ALTER TABLE ADD (COLUMN)	**Version 10.1 on z/OS and Linux, UNIX, or Windows or later** If you set the value of the REPL_ADDCOL column in the IBMQREP_SUBS table to Y (yes), when you add new columns to a table the columns are automatically added to the Q subscription and added to the target table if they do not already exist. **Earlier versions** Q Capture leaves the Q subscription active, but does not replicate the added column until it receives an ADDCOL signal.	**Version 10.1 on z/OS and Linux, UNIX, or Windows or later** Specify REPLICATE ADD COLUMN YES in the ASNCLP CREATE QSUB command or click the **Automatically replicate new columns added to the source table** check box when you are creating or changing the properties of a Q subscriptions in the Replication Center. For more detail, see these topics: • z/OS Enabling replication of ADD COLUMN and SET DATA TYPE operations • "Enabling automatic replication of newly added columns from the source table" on page 130 **Earlier versions** Use the Q Replication Dashboard or ASNCLP ALTER ADD COLUMN command, or manually insert an ADDCOL signal to indicate that you want to replicate the new column.
TRUNCATE TABLE	z/OS A TRUNCATE operation is logged similarly to a mass delete, so the operation is replicated as a series of single row deletes. Linux UNIX Windows Replication of TRUNCATE operations is not supported.	z/OS No action is required. If the target table has rows that are not in the source table, those rows are not deleted. Linux UNIX Windows If you need to perform TRUNCATE on a target table in addition to its source, you must issue the TRUNCATE statement directly against the target table.
ALTER TABLE ALTER COLUMN SET DATA TYPE	Automatically replicated for Version 10.1 and later. The data type of the corresponding target table column is changed and replication continues normally.	z/OS See Enabling replication of ADD COLUMN and SET DATA TYPE operations. Linux UNIX Windows See "Automatic replication of ALTER TABLE ALTER COLUMN SET DATA TYPE operations" on page 135.
Other DDL that alters the structure of a table	Q Capture leaves the Q subscription unchanged.	1. Stop the Q subscription. 2. Alter the source and target tables. 3. Start the Q subscription.

Table 11. How Q Capture handles DDL changes to the source database and what you need to do (continued)

DDL operation	How it is handled	What you need to do
DDL that does not alter the table structure Examples: • CREATE INDEX • ALTER FOREIGN KEY • ADD CONSTRAINT	Q Capture leaves the Q subscription active.	Ensure that unique constraints, primary keys, and referential integrity constraints match between the source and target tables. If you change any of these properties at the source, make a corresponding change to the target to avoid unexpected behavior. Also, restart the Q Apply program so that it recognizes the change.

Creating schema-level subscriptions

You create a schema-level subscription to replicate supported DDL operations such as CREATE TABLE and DROP TABLE for one or more schemas in a database.

Before you begin

The user ID that starts the Q Apply program must have authority to create and drop tables at the target database.

About this task

When you create this type of subscription, the replication programs automatically create table-level Q subscriptions when they process log records for CREATE TABLE operations that match the schema and table naming pattern for the subscription. Other supported DDL operations within the schema are also replicated.

For bidirectional replication, a schema-level subscription is required for each replication direction. When you use the ASNCLP command-line program, it creates the paired set of schema-level subscriptions with a single command.

The ASNCLP command can also create the required table-level Q subscriptions for existing tables that meet the table naming pattern.

Restrictions

The following configurations are not supported:
- Peer-to-peer replication
- CCD targets
- Federated targets
- Stored procedure targets
- Replication between DB2 for z/OS and DB2 for Linux, UNIX, and Windows

The following types of DDL are not replicated:
- ADD, ATTACH, and DETACH operations for partitioned tables
- Database sequences.
- CREATE or DROP of triggers or Materialized Query Tables (MQT)
- Linux UNIX Windows CREATE TABLESPACE

Other restrictions:

- Table-level Q subscriptions that are part of a schema-level subscription cannot use Q Capture search conditions or Q Apply SQL expressions.
- For bidirectional replication, a table must be used for DML or LOAD operations on only one site until all automatically created table-level Q subscriptions are created and active at both sites. For example, in a bidirectional configuration if you create a table at site 1 and site 2, you can start loading the table at site 1, but must not do any DML or LOAD at site 2 until all Q subscriptions are created at site 2 and are in active (A) state.

Note: The list of unsupported operations does not include every possible DDL replication limitation, but does list noteworthy exclusions. If something is not listed as supported, it is not supported.

Procedure

To create a schema-level subscription, use the CREATE SCHEMASUB command in the ASNCLP program.
The command creates table-level Q subscriptions for all tables within the schema that meet the naming pattern that you specify. It also saves the schema pattern so that the replication programs automatically create Q subscriptions for any tables that are added within the schema.
You can create a schema-level subscription that includes multiple schemas by using the percentage sign (%) as a wild card. To replicate all CREATE TABLE and DROP TABLE operations within all schemas in the database, specify the ALL keyword (which is equivalent to OWNER LIKE % NAME LIKE %, and is stored as %.%).
You can optionally include the name of a profile for creating table-level Q subscriptions, and the ASNCLP and replication programs use the options that you specify in the profile.
The following example creates a schema-level subscription called sampschemasub1 for bidirectional replication. The subscription includes all schemas and tables on the SAMPLE1 database and uses the saved profile options1:

```
SET BIDI NODE 1 SERVER DBALIAS SAMPLE1;
SET BIDI NODE 2 SERVER DBALIAS SAMPLE2;

CREATE SCHEMASUB "sampschemasub1" SUBTYPE B FOR TABLES NODE 1 ALL OPTIONS options1;
```

Creating profiles for table-level Q subscriptions

You can create a profile that specifies what options the replication programs should use when they automatically create table-level Q subscriptions for a schema-level subscription.

About this task

The ASNCLP program also uses the profile when it automatically creates table-level Q subscriptions for all tables within one or more specified schemas in response to a CREATE SCHEMASUB command.

The profile provides values for establishing the Q subscription on both the Q Capture and Q Apply servers. The replication programs use values from the profile for populating the IBMQREP_SUBS table at the Q Capture server and the IBMQREP_TARGETS table at the Q Apply server.

Q Capture reads the profile when the schema-level subscription is first started or when Q Capture is reinitialized. Changing a profile does not change existing table-level Q subscription. To modify existing Q subscriptions, you use the replication administration tools to change the Q subscription options and then issue a **reinit** command so that Q Capture can read your changes.

If you do not create a profile, the replication programs use default profiles for creating table-level Q subscriptions. These profiles differ depending on the type of Q subscription (unidirectional or bidirectional) and are named ASNBIDI and ASNUNI.

Important: Do not create a profile that would result in the creation of table-level Q subscriptions that differ from existing table-level Q subscriptions within the same schema-level subscription. All table-level subscriptions for a common schema-level subscription must be consistent.

Procedure

To create a profile for table-level Q subscriptions, use the CREATE SUBSCRIPTION OPTIONS command in the ASNCLP command-line program.
In the command, you specify options much as you would for creating an individual table-level Q subscription. For example, the following command for bidirectional replication specifies that in case of row conflicts, the row from the source table is used (CONFLICT ACTION F), that load operations at source tables are replicated (CAPTURE LOAD R) and that cascaded delete operations are not replicated (IGNORE CASCADE DELETES):

```
SET BIDI NODE 1 SERVER SAMPLE;
SET BIDI NODE 2 SERVER SAMPLE2;

CREATE SUBSCRIPTION OPTIONS bidioptions
   SUBTYPE B, CONFLICT ACTION F, CAPTURE LOAD W, IGNORE CASCADE DELETES;
```

If you specify this profile in the CREATE SCHEMASUB command, the ASNCLP program uses the profile options when it creates table-level Q subscriptions for all of the specified schemas and tables, and the Q Capture and Q Apply programs use the profile when they create Q subscriptions for newly created source tables in response to a CREATE TABLE operations.

DATA CAPTURE CHANGES and schema-level subscriptions

To enable replication of CREATE TABLE operations, newly created tables that are part of a schema-level subscription must have the DATA CAPTURE CHANGES attribute set.

DATA CAPTURE CHANGES ensures that DB2 provides additional information in its recovery log regarding SQL changes to the table. You can enable DATA CAPTURE CHANGES for newly created tables by using one of several methods:

- Explicitly specify DATA CAPTURE CHANGES in the CREATE TABLE statement.
- Linux UNIX Windows Use the DATA CAPTURE CHANGES attribute with the CREATE SCHEMA statement. With this option, all tables that are created within the specified schema have DATA CAPTURE CHANGES set by default.
- Linux UNIX Windows Set the database-level configuration parameter **dft_schemas_dcc** to YES. By default, **dft_schemas_dcc** is set to NO. When set to YES, all newly created schemas by default have the DATA CAPTURE CHANGES clause.

If the CREATE TABLE command explicitly specifies DATA CAPTURE NONE, the replication programs assume that you do not want to replicate this table and issue a warning message so that you can change the DATA CAPTURE attribute if desired.

Starting schema-level subscriptions

You start a schema-level subscription to instruct the Q Capture program to begin capturing SQL operations such as CREATE TABLE and DROP TABLE for tables that are part of the subscription. You can also optionally start all of the included table-level Q subscriptions.

About this task

Newly created schema-level subscriptions are in N (new) state and are automatically started when the Q Capture program is started or reinitialized. You might need to use this procedure if you stopped the subscription or it was stopped because of an error.

Starting a schema-level subscription entails inserting a START SCHEMASUB signal into the IBMQREP_SIGNAL table at one or more Q Capture servers. When Q Capture processes the signal, the state of the corresponding schema-level subscription changes to A (active) in the IBMQREP_SCHEMASUBS table.

When you use the ASNCLP program to start schema-level subscriptions, it performs the signal inserts. If you use SQL to perform the inserts yourself, the START SCHEMASUB signal must be inserted into the IBMQREP_SIGNAL table at all Q Capture servers in bidirectional or peer-to-peer configurations.

Procedure

To start a schema-level subscription, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the START SCHEMASUB command. You have two options:
	
	Start only the schema-level subscription
	This option instructs the Q Capture program to begin replicating CREATE TABLE and DROP TABLE operations for all tables that are part of the schema-level subscription. The following example accomplishes this task for schemasub1 between the bidirectional replication servers SAMPLE1 and SAMPLE2:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	
	`SET BIDI NODE 1 SERVER DBALIAS SAMPLE1;`
	`SET BIDI NODE 2 SERVER DBALIAS SAMPLE2;`
	
	`SET RUN SCRIPT NOW STOP ON SQL ERROR ON;`
	
	`START SCHEMASUB schemasub1 NEW ONLY;`
	
	Start the schema-level subscription and all of the table-level Q subscriptions that belong to it
	This option instructs the Q Capture program to begin replicating supported DDL operations for all tables that are part of the schema-level subscription, and to start replicating row changes for all of the table-level Q subscriptions that are part of the schema-level subscription.
	
	In the following example, the other commands that are needed in the script are the same as the previous example and are not shown:
	`START SCHEMASUB schemasub1 ALL;`

SQL	Insert a START SCHEMASUB signal into the IBMQREP_SIGNAL table at one or more Q Capture servers:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'START SCHEMASUB',`
	`    'schema_subname',`
	`    'P' );`
	
	Where *schema* identifies a Q Capture program, and *schema_subname* is the name of the schema-level subscription that you want to start.
	
	For bidirectional or peer-to-peer replication, you must insert the signal into all Q Capture servers in the configuration.

Changing table-level options used by schema-level subscriptions

You cannot update the profile that is used to create table-level Q subscriptions, but you can create a new profile with the changes that you require and then point the schema-level subscription to start using the new profile.

About this task

The reason that profiles cannot be updated is that other schema-level subscriptions might be using the profile that you want to change. To start using a new profile, you use ASNCLP commands and a SQL update statement.

Changing to a new profile only affects future Q subscriptions that are created for new tables. To update existing table-level Q subscriptions you need to issue another SQL statement.

Procedure

1. Create a new profile for table-level Q subscriptions that contains the new options. Use the CREATE SUBSCRIPTION OPTIONS command in the ASNCLP.

   The following example creates a new profile called bidilist_new for a bidirectional configuration. The new profile specifies an error action of S (Q Apply stops), changing from the default error action of Q (Q Apply stops reading from the receive queue).

   ```
   SET BIDI NODE 1 SERVER SAMPLE1;
   SET BIDI NODE 2 SERVER SAMPLE2;
   CREATE SUBSCRIPTION OPTIONS bidilist_new
    SUBTYPE B
     ERROR ACTION S;
   ```
2. Update the IBMQREP_SCHEMASUBS table to point to the new options list. For bidirectional replication, you update the IBMQREP_SCHEMASUBS table on both servers.

   Using the bidirectional configuration as an example, you would issue the following update statement on both SAMPLE1 and SAMPLE2 servers:

   ```
   UPDATE ASN.IBMQREP_SCHEMASUBS SET SUBPROFNAME = 'bidilist_new'
   WHERE SCHEMA_SUBNAME = 'schemasub1';
   ```
3. Update the existing table-level Q subscriptions to change the error action to S by issuing the following SQL statement for the IBMQREP_TARGETS tables on both servers.

   ```
   UPDATE ASN.IBMQREP_TARGETS SET ERROR_ACTION = 'S'
   nWHERE SCHEMA_SUBNAME = 'schemasub1';
   ```
4. Reinitialize the schema-level subscription and all of its table-level Q subscription by using the REINIT SCHEMASUB command with the ALL keyword in the ASNCLP. This command prompts the Q Capture program to read the changes that you made to the control tables.

   ```
   SET BIDI NODE 1 SERVER SAMPLE1;
   SET BIDI NODE 2 SERVER SAMPLE2;

   REINIT SCHEMASUB schemasub1 ALL;
   ```

Stopping schema-level subscriptions

You stop a schema-level subscription to instruct the Q Capture program to quit capturing SQL operations such as CREATE TABLE and DROP TABLE for tables that are part of the subscription. You can also optionally stop all of the included table-level Q subscriptions.

About this task

Stopping a schema-level subscription entails inserting a STOP SCHEMASUB signal into the IBMQREP_SIGNAL table at one or more Q Capture servers. When you use the ASNCLP program to stop schema-level subscriptions, it performs these inserts.

When Q Capture processes the signal, the state of the corresponding schema-level subscription changes to I (inactive) in the IBMQREP_SCHEMASUBS table. The state of table-level Q subscriptions remains unchanged unless you explicitly stop these subscriptions, which can be done in a single command by the ASNCLP program.

If you use SQL to perform the inserts yourself, the STOP SCHEMASUB signal must be inserted into the IBMQREP_SIGNAL table at all Q Capture servers in bidirectional or peer-to-peer configurations.

Procedure

To stop a schema-level subscription, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the STOP SCHEMASUB command. You have two options:
	Stop only the schema-level subscription
	This option instructs the Q Capture program to stop replicating CREATE TABLE and DROP TABLE operations for all tables that are part of the schema-level subscription. The following example accomplishes this task for schemasub1 between the bidirectional replication servers SAMPLE1 and SAMPLE2:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	
	`SET BIDI NODE 1 SERVER DBALIAS SAMPLE1;`
	`SET BIDI NODE 2 SERVER DBALIAS SAMPLE2;`
	
	`SET RUN SCRIPT NOW STOP ON SQL ERROR ON;`
	
	`STOP SCHEMASUB schemasub1 NEW ONLY;`
	Stop the schema-level subscription and all of the table-level Q subscriptions that belong to it
	This option instructs the Q Capture program to stop replicating supported DDL operations for all tables that are part of the schema-level subscription, and to stop replicating row changes for all of the table-level Q subscriptions that are part of the schema-level subscription.
	In the following example, the other commands that are needed in the script are the same as the previous example and are not shown:
	`STOP SCHEMASUB schemasub1 ALL;`

Method	Description

SQL	Insert a STOP SCHEMASUB signal into the IBMQREP_SIGNAL table at one or more Q Capture servers:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'STOP SCHEMASUB',`
	`    'schema_subname',`
	`    'P' );`
	
	Where *schema* identifies a Q Capture program, and *schema_subname* is the name of the schema-level subscription that you want to stop.
	
	For bidirectional or peer-to-peer replication, you must insert the signal into all Q Capture servers in the configuration.

Deleting schema-level subscriptions

You can delete schema-level subscriptions that are not being actively processed and optionally specify to delete all associated table-level Q subscriptions.

About this task

The DROP SCHEMASUB command in the ASNCLP program enables you to delete just the schema-level subscription (use the NEW ONLY keywords), or delete all of the table-level Q subscriptions (use the ALL keyword).

Procedure

1. Use the Q Replication Dashboard, Replication Center, or SQL to ensure that the schema-level subscription is in inactive (I) state.
2. Use one of these methods to ensure that all of the associated table-level Q subscriptions are inactive.
3. Issue the DROP SCHEMASUB command.

   The following example deletes the schema-level subscription JASLY10 in a bidirectional configuration and deletes all of the table-level Q subscription that belong to it:

   ```
   SET BIDI NODE 1 SERVER SAMPLE;
   SET BIDI NODE 2 SERVER SAMPLE2;

   DROP SCHEMASUB JASLY10 ALL;
   ```

   Subscription-related entries are deleted from the Q Capture and Q Apply control tables on both servers in the bidirectional configuration.

Enabling automatic replication of newly added columns from the source table

You can set up your Q subscriptions so that when new columns are added to the source table, they are automatically added to the target table and replicated.

Before you begin

- The Q Capture and Q Apply servers must be at Version 10.1 or newer on both z/OS and Linux, UNIX, and Windows.
- z/OS Some configurations steps are required on z/OS. See Enabling replication of ADD COLUMN and SET DATA TYPE operations.
- Ensure that the user ID that runs the Q Apply program has ALTER privilege on any target tables to which you want new columns to be added.

Restrictions

This function is not supported in the following situations:

- You are adding existing columns to a Q subscription
- You want to specify a before-image column name for a target consistent-change data (CCD) table
- You want to specify a different name for the target column

In these cases, use the ADDCOL signal that is described in "Adding existing columns to a Q subscription (unidirectional replication)" on page 131 and "Adding existing columns to a Q subscription (bidirectional or peer-to-peer replication)" on page 133.

Procedure

To enable automatic replication of newly added columns, use one of the following methods:

Method	Description

ASNCLP command-line program	In the CREATE QSUB command, specify REPLICATE ADD COLUMN YES. The following example shows the setup commands and then a CREATE QSUB command with this option:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DBALIAS SAMPLE;`
	`SET SERVER TARGET TO DBALIAS TARGETDB;`
	`SET RUN SCRIPT NOW STOP ON SQL ERROR ON;`
	
	`CREATE QSUB USING REPLQMAP SAMPLE_ASN_TO_TARGETDB_ASN`
	`(SUBNAME EMPLOYEE0001 db2admin.EMPLOYEE`
	`OPTIONS HAS LOAD PHASE I REPLICATE ADD COLUMN YES`
	`KEYS (EMPNO) LOAD TYPE 1);`

Replication Center	Click the **Automatically replicate new columns added to the source table** check box on one of these windows or wizard pages: • Source Table Columns (single unidirectional Q subscription) • Which source columns map to which target columns? (multiple unidirectional Q subscriptions) • Which profile settings for target tables? (bidirectional replication)

Adding existing columns to a Q subscription (unidirectional replication)

You can add existing columns from the source table to a unidirectional Q subscription while the replication programs are running. If the columns are not existing and your servers are at Version 10.1 or later, you use a different procedure.

Before you begin

- The Q subscription that the columns are being added to must be in A (active) state.
- Linux UNIX Windows If the data type of the column is LONG VARCHAR or GRAPHIC, the source database or subsystem must be configured with DATA CAPTURE CHANGES INCLUDE VARCHAR COLUMNS.

Restrictions

- The columns that you are adding must be nullable, or defined as NOT NULL WITH DEFAULT.
- If you add columns with default values, you must run the REORG utility on the source table before you begin replicating the new column. For more detail, see Avoiding CHECKFAILED errors when adding columns to DB2 for z/OS target tables.
- You cannot add more than 20 columns within one Q Capture commit interval as specified by the `commit_interval` parameter.
- **Federated targets:** To add columns to an existing Q subscription, you can use the ADDCOL signal but you must drop the Q subscription and recreate it after you alter the target table because you cannot add columns to a nickname.

About this task

For Version 10.1 or later: You do not need to use this procedure when you are adding new columns to the source table if the participating servers are at Version 10.1 or later on both z/OS and Linux, UNIX, and Windows. If you set the value of the REPL_ADDCOL column in the IBMQREP_SUBS table to Y (yes), when you add new columns to a table, the columns are automatically added to the Q subscription, and added to the target table if they do not already exist. To make this setting, specify REPLICATE ADD COLUMN YES in the ASNCLP CREATE QSUB command or click the **Automatically replicate new columns added to the source table** check box when you are creating or changing the properties of a Q subscriptions in the Replication Center. For more detail, see these topics:

- z/OS Enabling replication of ADD COLUMN and SET DATA TYPE operations
- "Enabling automatic replication of newly added columns from the source table" on page 130

When you insert the signal at the Q Capture server, the column is automatically added to the target table if you did not already add it. If you want to add multiple columns to a Q subscription, you insert one signal for each new column. You can add multiple columns in a single transaction. The Q Capture program can be stopped when you insert the signals and it will read them from the log when it restarts.

If you let the replication programs automatically add new columns to the target table it helps ensure that they match the columns at the source. Columns are added to the target table with the same data type, null characteristic, and default value as the matching columns in the source table. You can specify a different name for the target column if you use the ALTER ADD COLUMN command in the ASNCLP command-line program or an ADDCOL signal.

Procedure

To add columns to a unidirectional Q subscription, use one of the following methods:

Method	Description

ASNCLP command-line program.	Use the ALTER ADD COLUMN command. For example, the following command adds the column BONUS to the DEPARTMENT0001 Q subscription:
	`ALTER ADD COLUMN USING SIGNAL (BONUS)`
	`QSUB DEPARTMENT0001`
	`USING REPQMAP SAMPLE_ASN_TO_TARGET_ASN;`

Q Replication Dashboard	On the Subscriptions tab, select a Q subscriptions from the table and click **Actions** > **Add Columns**.

SQL	Use a command prompt or one of the DB2 command-line tools to insert an ADDCOL signal into the IBMQREP_SIGNAL table at the Q Capture server. For example:
	`insert into schema.IBMQREP_SIGNAL(`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` ) values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'ADDCOL',`
	`     'subname;column_name;before_column_name;`
	`target_column_name',`
	`     'P' );`
	schema
	Identifies the Q Capture program that is processing the Q subscription that you are adding a column to.
	subname;column_name;before_column_name;target_column_name
	The name of the Q subscription that you want to add the column to and the name of the column that you are adding, separated by a semicolon. These names are case-sensitive and do not require double quotation marks to preserve case. Follow these examples:
	Add column in source table to Q subscription and to target table `QSUB1;COL10`
	Add column and before image of the column (for CCD target tables)
	`QSUB1;COL10;XCOL10`
	Add column without before image but with different target column name
	`QSUB1;COL10;;TRGCOL10` (Use the double semicolon (;;) to indicate that you are omitting the before-image column.)

After processing the signal, the Q Capture program begins capturing changes to the new column when it reads log data that includes the column. Changes to the column that are committed after the commit of the ADDCOL signal insert will be replicated to the new column in the target table. Rows that existed in the target table before the new column is added will have a NULL or default value for the new column.

Adding existing columns to a Q subscription (bidirectional or peer-to-peer replication)

You can add existing columns from the source table to a bidirectional or peer-to-peer Q subscription while the replication programs are running. If the columns are not existing and your servers are at Version 10.1 or later, you use a different procedure.

Before you begin

- The Q subscriptions that specify the table must be in A (active) state at all servers.
- Linux UNIX Windows If the data type of the column is LONG VARCHAR or GRAPHIC, the source database or subsystem must be configured with DATA CAPTURE CHANGES INCLUDE VARCHAR COLUMNS.

Restrictions

- Any columns that you add must be nullable, or defined as NOT NULL WITH DEFAULT.
- If you add columns with default values, you must run the REORG utility on the source table before you begin replicating the new column. For more detail, see Avoiding CHECKFAILED errors when adding columns to DB2 for z/OS target tables.
- You cannot alter the default value of a newly added column until the ADDCOL signal for that column is processed.
- You cannot add more than 20 columns within one Q Capture commit interval as specified by the **`commit_interval`** parameter.

About this task

For Version 10.1 or later: You do not need to use this procedure when you are adding new columns to the source table if the participating servers are at Version 10.1 or later on both z/OS and Linux, UNIX, and Windows. If you set the value of the REPL_ADDCOL column in the IBMQREP_SUBS table to Y (yes), when you add new columns to a table, the columns are automatically added to the Q subscription, and added to the target table if they do not already exist. To make this setting, specify REPLICATE ADD COLUMN YES in the ASNCLP CREATE QSUB command or click the **Automatically replicate new columns added to the source table** check box when you are creating or changing the properties of a Q subscriptions in the Replication Center. For more detail, see these topics:

- z/OS Enabling replication of ADD COLUMN and SET DATA TYPE operations
- "Enabling automatic replication of newly added columns from the source table" on page 130

To use this procedure, first you alter a table at one server to add a column. Then you insert an SQL signal at the server. When the signal is processed, the versions of the table at the other servers are automatically altered to add the column, unless you added it manually. The signal also adds the column to the Q subscription definitions at all servers.

You can add any number of columns to the source table at a time. You can do this while the Q Capture and Q Apply programs are running or stopped.

Recommendation: Insert one ADDCOL signal at a time and issue a COMMIT before inserting the next ADDCOL signal or doing any other transactions.

Procedure

To add columns to replicate in bidirectional or peer-to-peer replication:

1. Alter the logical table at one of the servers to add the column.

   If the ALTER TABLE operation that adds the column to the source table fails, all the Q subscriptions in the peer-to-peer group will be deactivated.
2. Use one of the following methods to signal the Q Capture program that you want to add the column to the Q subscription for the source table.

Method	Description

ASNCLP command-line program	Use the ALTER ADD COLUMN command. For example, the following command adds the column BONUS to the DEPARTMENT0001 Q subscription:
	`ALTER ADD COLUMN USING SIGNAL (BONUS)`
	`QSUB DEPARTMENT0001`
	`USING REPQMAP SAMPLE_ASN_TO_TARGET_ASN;`

Q Replication Dashboard	On the Subscriptions tab, select a Q subscriptions from the table and click **Actions** > **Add Columns**.

Method	Description
SQL	Use a command prompt or one of the DB2 command-line tools to insert an ADDCOL signal into the IBMQREP_SIGNAL table at the Q Capture server. For example: `insert into schema.IBMQREP_SIGNAL( SIGNAL_TIME, SIGNAL_TYPE, SIGNAL_SUBTYPE, SIGNAL_INPUT_IN, SIGNAL_STATE ) values ( CURRENT TIMESTAMP, 'CMD', 'ADDCOL', 'subname;column_name', 'P' );` *schema* Identifies the Q Capture program at the server where you altered the table. *subname;column_name* The name of a Q subscription that originates at the Q Capture server where you altered the table, and the name of the column that you are adding, separated by a semicolon. These names are case-sensitive and do not require double quotation marks to preserve case.

Consider the following example. A peer-to-peer configuration has three servers: ServerA, ServerB, and ServerC, and six Q subscriptions: subA2B, subB2A, subA2C, subC2A, subB2C, and subC2B for the EMPLOYEE table.

You add a column, ADDRESS, to the EMPLOYEE table on ServerA. Then you insert an ADDCOL signal for the Q subscription that handles transactions from ServerA to ServerB, and specify `subA2B;ADDRESS` for the Q subscription name and column name. Only one ADDCOL signal is required. The replication programs automatically add the ADDRESS column to the EMPLOYEE tables at ServerB and ServerC, and add the column definition to all six Q subscriptions.

Automatic replication of ALTER TABLE ALTER COLUMN SET DATA TYPE operations

Starting with Version 10.1 on z/OS and Linux, UNIX, and Windows, ALTER TABLE ALTER COLUMN SET DATA TYPE operations at the source table are automatically replicated. The data type of the corresponding target table column is changed and replication continues normally.

Prerequisites:

- The Q Capture and Q Apply servers must be at Version 10.1 or newer on both z/OS and Linux, UNIX, and Windows. The exception is extending a VARCHAR/VARGRAPHIC column. This operation is supported on older versions.

- z/OS Some configurations steps are required on z/OS. See Enabling replication of ADD COLUMN and SET DATA TYPE operations.
- Ensure that the user ID that runs the Q Apply program has ALTER privilege on any target tables to be changed.

In the following situations, when you change the data type of a source column you must manually alter the target column:

- The Q Apply program is at Version 9.7 or earlier. You must alter the target column data type before you alter the source table data type.
- The source column is mapped to an expression at the target. You must make any required changes to the expression or the underlying target table column.
- The altered column is a string that is mapped to a longer string at the target for code page conversion concerns. For example, consider a situation where the data type of the source column is CHAR(10). The target column is set up as CHAR(40) to handle code page conversion. If the source column is altered to CHAR(20), the change is replicated. However, string data types cannot be altered to a smaller size and the change from CHAR(40) to CHAR(20) would fail. To avoid this situation, you must manually alter the target column to CHAR(80) to prevent any row errors after code page conversion.

After the target column is altered, DB2 puts the table into REORG PENDING state, which requires a REORG operation before the table can be used. The Q Apply program calls the DB2 stored procedure ADMIN_REVALIDATE_DB_OBJECTS to remove the table from REORG PENDING state. The exceptions to this REORG requirement are extending the length of VARCHAR or VARGRAPHIC columns.

Chapter 8. Q Replication to federated targets

Q Replication to targets such as Oracle or Microsoft SQL Server works much like a scenario where both source and target are DB2 servers.

WebSphere MQ is configured for a typical unidirectional replication scenario between two DB2 databases. Transactions are replicated using the existing Q Capture and Q Apply programs.

The primary difference is on the Q Apply side. With a target other than DB2, the Q Apply program runs on a federated server, retrieving captured changes from queues and writing them to target tables in the non-DB2 relational database by using federated server nicknames.

When you configure Q Replication for federated targets, several Q Apply control tables are created on the target system and accessed through nicknames just as target tables are. Q Apply writes to these tables during the same unit of work as the changes to target tables. One set of control tables is needed for each federated target.

Figure 18 shows the relationships between the replication programs, DB2 servers, and non-DB2 server in federated Q Replication.

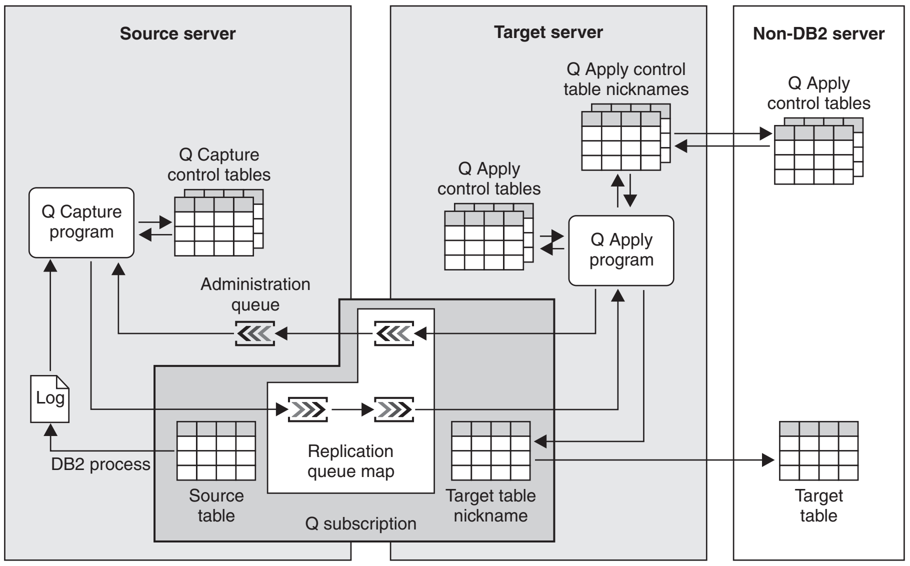


Figure 18. Q Replication to a non-DB2 server through a federated server

The following federated Q Replication support is available:

Sources and targets
: Unidirectional replication is supported from DB2 for z/OS, Linux, UNIX, and Windows to target tables in Oracle, Sybase, Microsoft SQL Server, Informix®, or Teradata relational databases that use the federated wrappers that are defined for these targets. Transforming data by replicating to DB2 stored procedures that write to nicknames is supported. The Q Apply program can load source data into one or more federated target tables in parallel by using the EXPORT and IMPORT utilities. You can also specify

that Q Apply use an ODBC select to fetch data from the source for loads, and for Oracle targets, you can specify that Q Apply call the SQL*Plus utility.

Q Replication also supports the use of SQL expressions to transform data between sources table and non-DB2 target tables. The Q Apply program performs data transformation before applying the data to the non-DB2 target table through the nicknames on the federated server.

Application of data to targets
Q Replication requires columns of like attributes (for example, INTEGER at the source table to INTEGER at the nickname). Q Replication can replicate source columns to nickname columns with different but compatible attributes such as replication from small column data lengths at the source to larger data lengths at the target (for example, replication from CHAR(4) at the source to CHAR(8) at the target).

Utilities
The Replication Alert Monitor and table differencing and reconciliation utilities (asntdiff and asntrep) are supported for federated targets. The asntdiff utility compares the source table with the nickname, and the asntrep utility updates the nickname to bring the source and target into synch.

Note about terminology

In topics about federated Q Replication, the term "federated target" refers to the Oracle, Sybase, Microsoft SQL Server, Informix, or Teradata database that contains the target tables. In other DB2 federated topics, these targets are referred to as "data sources." In other Q Replication topics, the term "target" refers to the DB2 database where the Q Apply program runs and the target tables are located. This database contains the nicknames in federated Q Replication. Figure 19 depicts these relationships.

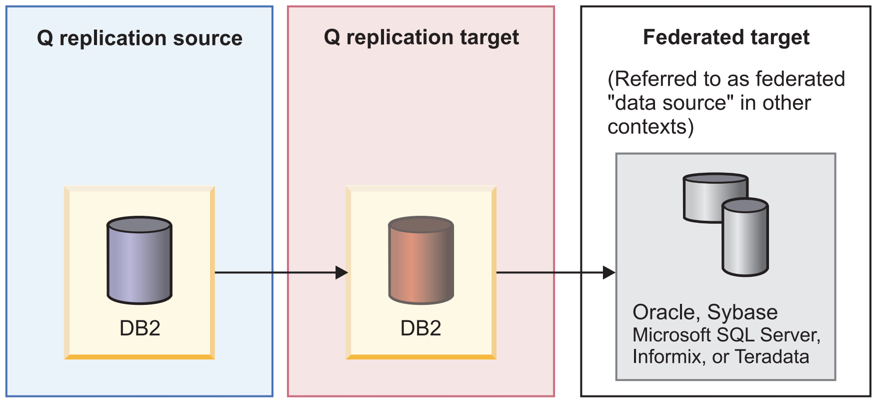


Figure 19. Sources and targets in federated Q Replication

Setting up Q Replication to federated targets

Setting up Q Replication to federated targets involves configuring the federated server where the Q Apply program runs, creating WebSphere MQ objects, creating federated objects, and creating Q Replication objects.

General restrictions

- The asntdiff and asntrep utilities require the data types to be the same at the DB2 source table and at the nickname at the federated server where the Q Apply program runs.
- If you use the ADDCOL signal to add a column to an existing Q subscription, the new column must already exist in the target table and the corresponding nickname (you cannot add a column to a nickname).
- Ensure that the code pages of the source database, federated DB2 database, and non-DB2 target database match.

Data type restrictions

- Replication of large object (LOB) values is supported for Oracle targets only, and these targets must use the NET8 wrapper.
- To replicate GRAPHIC, VARGRAPHIC, or DBCLOB data types, your Oracle server and client must be Version 9 or newer. Your server mapping must also be Version 9 or newer.
- Replication of LONG VARGRAPHIC data types is not supported.
- For Sybase, Microsoft SQL Server, and Informix, if the source table has a column data type of LONG VARCHAR, the nickname is created with a data type of VARCHAR(32672). The length allowed for a LONG VARCHAR is greater than 32672 and this could result in truncation of data.

Procedure

To set up Q Replication to federated targets:

1. Configure the federated server.
2. Create federated objects.
3. Create WebSphere MQ objects.
4. Create control tables.
5. Create Q subscriptions.

Figure 20 on page 140 shows the steps involved in setting up federated Q Replication.

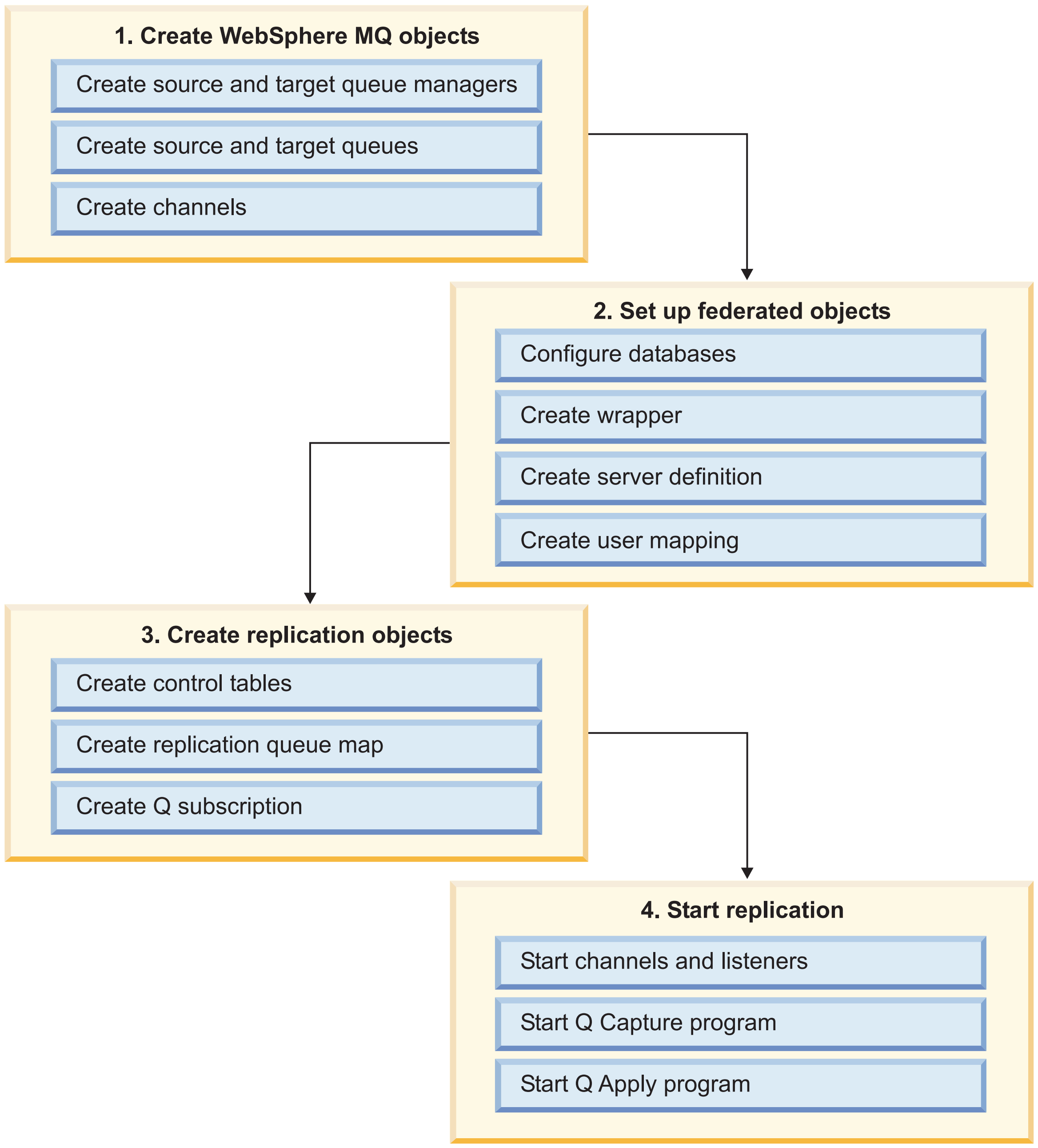

1. Create WebSphere MQ objects
Create source and target queue managers
Create source and target queues
Create channels
2. Set up federated objects
Configure databases
Create wrapper
Create server definition
Create user mapping
3. Create replication objects
Create control tables
Create replication queue map
Create Q subscription
4. Start replication
Start channels and listeners
Start Q Capture program
Start Q Apply program

Figure 20. Overview of steps to set up federated Q Replication

Configuring the federated server for Q Replication

Before you can replicate to a federated target, you need to configure the DB2 instance and database on the system where the Q Apply program runs.

Before you begin

Ensure that the code pages of the source database, federated DB2 database, and non-DB2 database database match.

Procedure

To configure the federated server for Q Replication:

1. Enable federated support in the DB2 instance where the Q Apply program runs by using one of the following methods:

Method	Description
Control Center	Use the DBM Configuration window. To open the window, in the object tree right-click the DB2 instance that contains the Q Apply database and click **Configure Parameters**. Under the **Environment** heading, click **Federated** and click the ellipsis button to turn on federated support.
`UPDATE DBM CFG` command	Ensure that you are attached to the instance that contains the Q Apply server, and issue the following command: `UPDATE DBM CFG USING FEDERATED YES`

   You must stop and restart the instance for the change to take effect.
2. Set up and test the Oracle, Sybase, Microsoft SQL Server, or Informix client configuration file on the system where Q Apply runs. See one of the following topics for details:
   - Setting up and testing the Oracle client configuration file
   - Setting up and testing the Sybase client configuration file
   - Preparing the federated server to access Microsoft SQL Server data sources
   - Setting up and testing the Informix client configuration file
3. Set the environment variables for connecting to the Oracle, Sybase, Microsoft SQL Server, or Informix server in the `db2dj.ini` file on the federated server where the Q Apply program runs. See one of the following topics for details:
   - Setting the Oracle environment variables
   - Setting the Sybase environment variables
   - Setting the Microsoft SQL Server environment variables
   - Setting the Informix environment variables
   - Setting the Teradata environment variables
4. **Required for Sybase targets:** Bind the Q Apply packages manually with isolation level CS.

Creating federated objects for Q Replication

You must create or register wrappers, server definitions, and user mappings on the federated server before the Q Apply program can communicate and exchange data with the federated target.

Before you begin

- The DB2 instance that contains the Q Apply server must be configured for federated support.
- Wrappers that are used for replication to federated targets must allow write access to the federated target.

Restrictions

Q replication does not support replication to federated targets through a generic wrapper such as the ODBC wrapper.

Procedure

To create federated objects for Q Replication:

1. Register the appropriate wrapper for your Oracle, Sybase, Microsoft SQL Server, or Informix database. See one of the following topics:
   - Registering the Oracle wrapper
   - Registering the Sybase wrapper
   - Registering the Microsoft SQL Server wrapper
   - Registering the Informix wrapper
   - Registering the Teradata wrapper
2. Register the server definition for the Oracle, Sybase, Microsoft SQL Server, or Informix target. See one of the following topics:
   - Registering the server definitions for an Oracle data source
   - Registering the server definitions for a Sybase data source
   - Registering the server definitions for a Microsoft SQL Server data source
   - Registering the server definitions for an Informix data source
   - Registering the server definitions for a Teradata data source
3. Create a user mapping between the user ID on the federated server where the Q Apply program runs and the user ID on the Oracle, Sybase, Microsoft SQL Server, or Informix server. See one of the following topics:
   - Creating the user mappings for an Oracle data source
   - Creating a user mapping for a Sybase data source
   - Creating a user mapping for a Microsoft SQL Server data source
   - Creating the user mapping for an Informix data source
   - Creating the user mapping for a Teradata data source

   The user ID in the mapping must have the following authorities:
   - Create tables in the target database (control tables and target table if you choose).
   - SELECT, INSERT, UPDATE, and DELETE authority for the control tables and target table.

   Restriction: The Replication Center and ASNCLP command-line program do not support creating control tables or target tables in Oracle databases if the server mapping has two-phase commit enabled.

Creating Q Apply control tables for federated Q Replication

Before you can replicate data to the federated target, you need to create control tables to store information about Q subscriptions, message queues, operational parameters, and user preferences.

Before you begin

- You must set up federated access by creating wrappers, server definitions, and user mappings. The replication administration tools will work with the Oracle, Sybase, Microsoft SQL Server, Informix, or Teradata target by using wrappers.
- By default, the remote authorization ID is used as the schema for the Q Apply control tables that are created in the Oracle, Sybase, Microsoft SQL Server, Informix, or Teradata target database. The user ID must have the authority to create objects in this schema.

About this task

For federated targets, some Q Apply control tables are created in the target database and accessed through nicknames just as target tables are. The rest of the control tables are created in the federated server where the Q Apply program runs. Table 12 shows the location of the control tables.

Table 12. Location of control tables for federated Q Replication

Tables in the federated server	Tables in non-DB2 target server
• IBMQREP_APPLYENQ • IBMQREP_APPLYTRACE • IBMQREP_APPLYMON • IBMQREP_APPLYPARMS	• IBMQREP_DONEMSG • IBMQREP_EXCEPTIONS • IBMQREP_RECVQUEUES • IBMQREP_SAVERI • IBMQREP_SPILLEDROW • IBMQREP_SPILLQS • IBMQREP_TRG_COLS • IBMQREP_TARGETS

Informix: The data type of the MQMSGID column in the IBMQREP_DONEMSG and IBMQREP_SPILLEDROW control tables is changed to BYTE on Informix to stop the federated server from truncating binary data. In addition, no primary key is defined for this table because Informix does not allow indexes to be created on binary data.

Procedure

To create Q Apply control tables for federated Q Replication:

Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **`CREATE CONTROL TABLES FOR`** command. Specify the FEDERATED keyword.
	
	You can optionally use the RMT SCHEMA keyword to specify a schema for the control tables on the federated target. The default is the remote authorization ID. You can also optionally specify the table space (Oracle), segment (Sybase), filegroup (SQL Server), or dbspace (Informix) where these remote control tables will be created.
	
	For example, the following commands set the environment and create Q Apply control tables for replication to an Oracle target ORACLE_TARGET using a federated server FED_DB and a remote authorization ID of FED_ASN:
	
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER TARGET TO DB FED_DB`
	`NONIBM SERVER ORACLE_TARGET;`
	`SET QMANAGER QM2 FOR APPLY SCHEMA;`
	`SET APPLY SCHEMA ASN;`
	`CREATE CONTROL TABLES FOR APPLY SERVER`
	`IN FEDERATED RMT SCHEMA FED_ASN;`

Method	Description
Replication Center	Use the Create Q Apply Control Tables wizard. To open the wizard, right-click the **Q Apply Servers** folder and click **Create Q Apply Control Tables**. **Start page** To specify the location of the control tables on the Q Apply server or non-DB2 server, click **Custom**. • For the Q Apply server, the control tables are created in one table space. You can specify an existing table space or create a new table space. • At the federated target, the control tables are created in the default table space (Oracle), segment (Sybase), filegroup (Microsoft SQL Server), or dbspace (Informix), or you can specify an existing table space, segment, filegroup, or dbspace. **Server page** Select a federated server. **Target Tables page** Specify that the target tables are in a non-DB2 relational database that is mapped to the Q Apply server, and verify: • The server name (the server definition for the non-DB2 relational server that is mapped to the DB2 federated server). • The remote schema that the Q Apply control tables will be created under in the Oracle, Sybase, Microsoft SQL Server, Informix, or Teradata database (the **Schema** field shows the remote authorization ID for these targets, or you can change the value).

Creating Q subscriptions for federated Q Replication

A Q subscription for federated Q Replication maps the DB2 table that contains your source data to a copy of that table at the federated target. When you create a Q subscription, you specify a queue map, target table options, and other preferences. You create one Q subscription for each table that you want to replicate.

Before you begin

To replicate to Informix targets, you must enable transaction logging with the `buffered_log` option.

Restrictions

- Multidirectional replication is not supported for federated targets.
- Only DB2 stored procedures that write to nicknames are supported. The stored procedure cannot write to both local DB2 tables and nicknames at the same time because two-phase commit is not supported.
- Views or stored procedures in an Oracle, Sybase, Microsoft SQL Server, Informix, or Teradata database are not supported as targets.
- If you want the Q Apply program to load the target table, you must specify the EXPORT and IMPORT utilities for all targets except Oracle, for which you can

also use SQL*Plus. Before you use the loading function, nicknames that refer to Teradata target tables must be empty. Nicknames that refer to tables in Oracle, Microsoft SQL Server, Informix, and Sybase database versions that support TRUNCATE operations do not have to be empty because Q Apply initiates a TRUNCATE operation on the target table. Truncate operations are not supported if the target table has referential integrity constraints.

Oracle targets: To use SQL*Plus, you must create a password file by using the **`asnpwd`** command in the directory that is specified by the **`apply_path`** parameter or the directory from which Q Apply is invoked with the following values for these keywords:

– **`alias`**: The Oracle `tnsnames.ora` entry that refers to the Oracle server (the same name that is used for the NODE option of the CREATE SERVER command for setting up federation).
– **`id`**: The remote user ID for connecting to Oracle.
– **`password`**: The password for connecting to Oracle.

The file must have the default name `asnpwd.aut`. Before starting the Q subscription, you should test connectivity with this command: `$> sqlplus` *`id/password@alias`*.

- Replication to federated target tables with referential integrity constraints is supported only if you manually define the constraints on the corresponding nicknames. The ASNCLP or Replication Center do not automatically create these constraints when the tools create nicknames. Also, the Q Apply program does not drop referential integrity constraints on nicknames during the loading process and then restore them.

  Recommendation: Use the "no load" option for nicknames with referential integrity constraints and load the target table outside of the replication administration tools.
- For target nicknames with multiple indexes, the BEFORE_VALUES attribute for the Q subscription must be Y and the CHANGED_COLS_ONLY value must be N in the IBMQREP_SUBS table.

Procedure

To create Q subscriptions for federated Q Replication:

Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE QSUB command for unidirectional replication. For TARGET NAME specify the non-DB2 target table. Specify the FEDERATED keyword. You can optionally specify a nickname name and owner if you want to change the default. For example, the following series of commands creates a Q subscription with these characteristics:
	• The source server is SAMPLE.
	• The federated server where the Q Apply program runs (TARGET keyword) is FED_DB.
	• The non-DB2 target server (NONIBM SERVER keyword) is ORACLE_TARGET.
	• The replication queue map (REPLQMAP keyword) is SAMPLE_ASN_TO_FED_DB_ASN.
	• The Q subscription name (SUBNAME keyword) is FEDSUB.
	• The target table on the Oracle database is EMPLOYEE.
	• The nickname on the federated Q Apply server that points to the EMPLOYEE table (NICKNAME keyword) is EMPNICKNAME
	• The Q subscription specifies a manual (E) load phase (HAS LOAD PHASE keyword)
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET SERVER TARGET TO DB FED_DB`
	`NONIBM SERVER ORACLE_TARGET;`
	`SET CAPTURE SCHEMA SOURCE ASN;`
	`SET APPLY SCHEMA ASN;`
	`SET QMANAGER QM1 FOR CAPTURE SCHEMA;`
	`SET QMANAGER QM2 FOR APPLY SCHEMA;`
	`CREATE QSUB USING REPLQMAP`
	`SAMPLE_ASN_TO_FED_DB_ASN`
	`(SUBNAME FEDSUB TARGET NAME EMPLOYEE`
	`FEDERATED NICKNAME EMPNICKNAME OPTIONS`
	`HAS LOAD PHASE E);`

Method	Description
Replication Center	Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture schema or Q Apply schema in the object tree, right-click the **Q Subscriptions** folder, and click **Create**. **Servers page** Specify the federated server where the Q Apply program runs as the target server. The Q Apply program updates a nickname at this server that is mapped to a table at the non-DB2 relational database. **Target page** Specify the type of target that you want to replicate to: **Table in the non-DB2 database** The table will be updated by using a nickname. You can let the Replication Center create a new table, or specify an existing table. **DB2 stored procedure** A stored procedure allows you to manipulate source data before it is applied to the nickname. The stored procedure must already exist on the Q Apply server and must only write to nicknames. **For multiple Q subscriptions:** If you specified more than one source table, the Target Tables page shows the profile to be used for non-DB2 target tables, indexes, and table spaces or segments, and for target nicknames at the federated server. Click **Change** to open the Manage Target Object Profile window and change the names. **Rows and Columns page** Use the **Column mapping** field if you need to change the default mapping of source columns to columns in the non-DB2 target table. The Column Mapping window shows the default data type mappings between the nickname at the federated server and the non-DB2 target table and validates whether a source column can be mapped to a target column.

Chapter 9. Q Replication from Oracle sources

Q Replication provides log-based capture from Oracle source tables combined with the high throughput and low latency of WebSphere MQ messaging and a highly parallel Q Apply program.

A typical use for this solution is replicating data from an online transaction processing database (OLTP) on Oracle to a read-only server. This approach allows real-time queries of live business data for generating reports, while minimizing overhead and avoiding contention at the OLTP server.

The following figure shows the Oracle LogMiner utility translating log records from source tables into SQL statements, which the Q Capture program turns into WebSphere MQ messages for delivery to the Q Apply program. A replication queue map specifies which messages queues are used to deliver transactions, and Q subscriptions map the Oracle source tables to DB2 target tables.

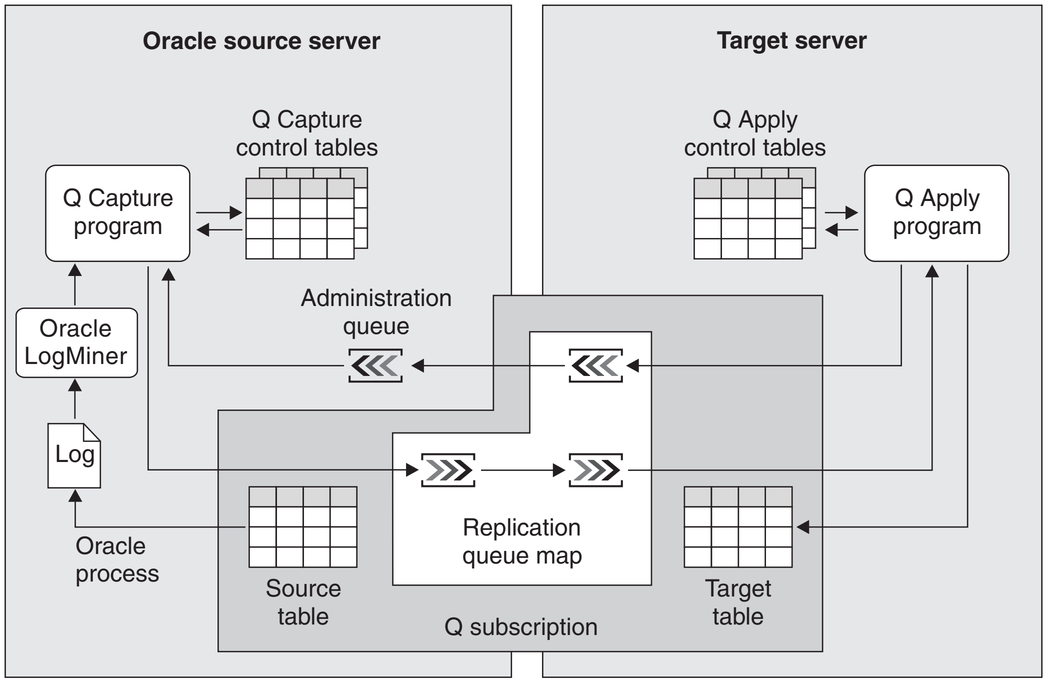


Figure 21. High-level overview of Q Replication from an Oracle source

Setting up Q Replication from Oracle sources

Setting up Q Replication from Oracle sources involves configuring the Oracle source database, creating WebSphere MQ objects, configuring the Q Capture and Q Apply servers, and creating Q Replication objects.

Procedure

To set up Q Replication from Oracle sources:

1. Configure the Oracle source.
2. Create WebSphere MQ objects.

3. Create control tables.
4. Set up for target table loading.
5. Create Q subscriptions.
6. Start the Q Capture program
7. Start the Q Apply program

Figure 22 on page 151 shows the steps involved in setting up Q Replication from Oracle sources.

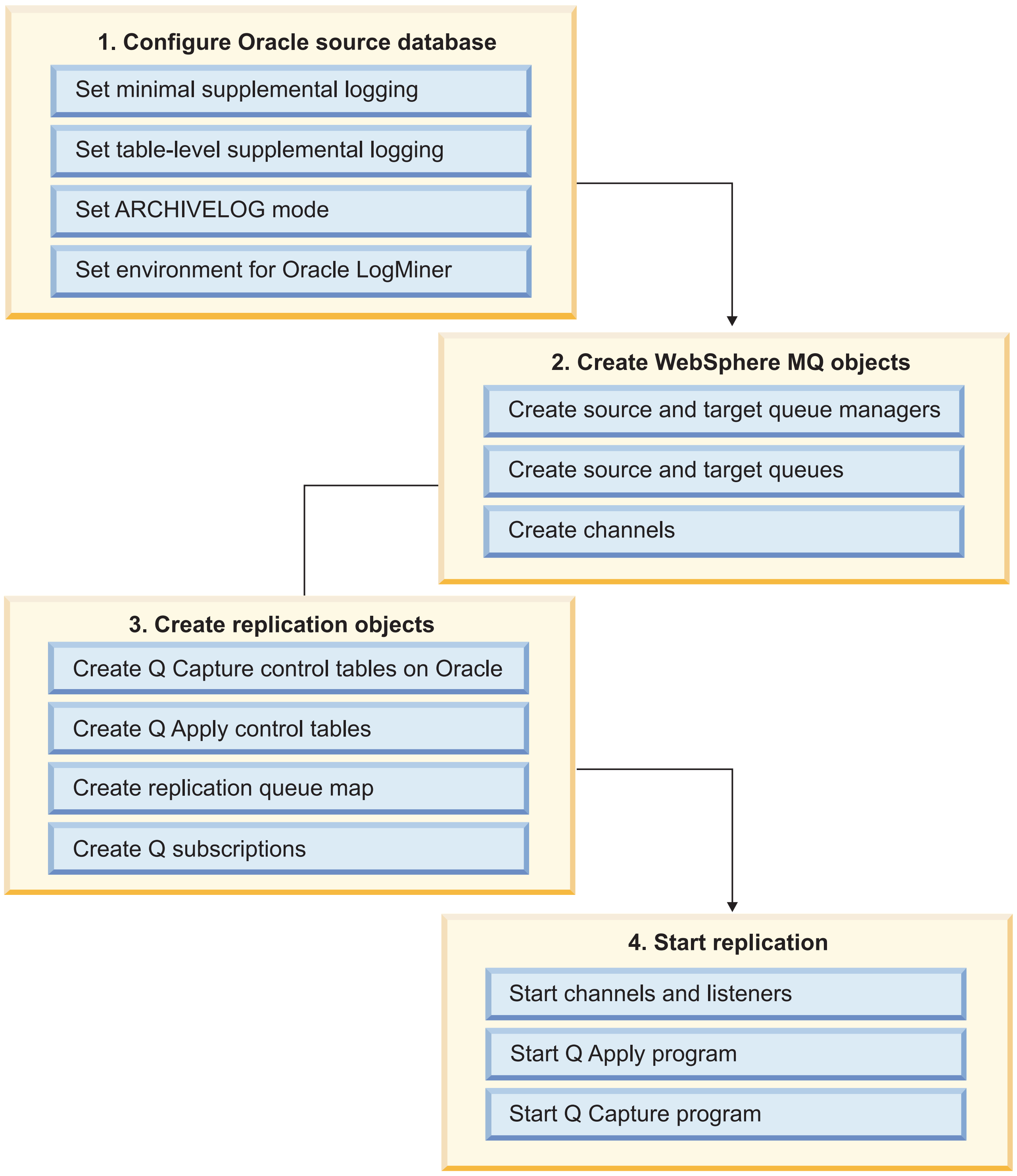

1. Configure Oracle source database
Set minimal supplemental logging
Set table-level supplemental logging
Set ARCHIVELOG mode
Set environment for Oracle LogMiner
2. Create WebSphere MQ objects
Create source and target queue managers
Create source and target queues
Create channels
3. Create replication objects
Create Q Capture control tables on Oracle
Create Q Apply control tables
Create replication queue map
Create Q subscriptions
4. Start replication
Start channels and listeners
Start Q Apply program
Start Q Capture program

Figure 22. Overview of steps to set up Q Replication from Oracle

Oracle client support for Q Replication

If you are running Q Capture for Oracle sources you must use the correct Oracle client libraries for your operating system.

A full Oracle client is not required to run the Q Capture program with an Oracle source database. You can use what Oracle refers to as an Instant Client, which is a small compressed file that includes the essential DLLs or shared libraries that are required by an Oracle client application.

To use the Instant Client with Q Capture for Oracle:

1. Extract the instant client compressed file to a dedicated directory (do not mix with the full Oracle client directory).
2. Ensure that the instant client directory is included in one of the following environment variables: PATH (Windows), LIBPATH (UNIX), LD_LIBRARY_PATH (Linux or Solaris), or SHLIB_PATH (HP-UX).
3. If you have another client installed (possibly from an Oracle installation) that does not match the required version below, this client should not be mixed with the Instant Client in the environment variables, or you should ensure that the path to the Instant Client directory occurs before the path to the installed Oracle client directory.

The tables below list supported operating systems, along with the level of Oracle client that is required for the Q Capture program. The Oracle client does not have to match the Oracle server, but the release of the database server software should be equal to or higher than the client software release.

InfoSphere Replication Server Version 9.7 Fix Pack 3a and earlier

The following table lists supported operating systems and required clients when you are using InfoSphere Replication Server Version 9.7 Fix Pack 3a and earlier to replicate from Oracle databases.

Table 13. Supported operating systems and required Oracle client library version for InfoSphere Replication Server Version 9.7 FP3a and earlier

Operating system	Oracle client library version
AIX 5.3	10.2.0.1 instant client
Linux 390 64 bit	10.2.0.1 instant client
Solaris 64 bit	10.2.0.1 instant client
Linux AMD 64 bit	10.2.0.3 instant client patched with 10.2.0.3.0 g++ 3.4.3 RHEL AS 4.0 64-bit (x86_64) version
Windows 32 bit	10.2.0.3 instant client patched with 10.2.0.3.0 VS2005 OCCI version
HP IPF 64 bit	10.2.0.1 instant client

WebSphere Replication Server Version 9.5

The following table lists supported operating systems and required clients when you are using WebSphere Replication Server Version 9.5 to replicate from Oracle databases.

Table 14. Supported operating systems and required Oracle client library version for WebSphere Replication Server Version 9.5

Operating system	Oracle client library version
AIX 5.3	10.2.0.1 instant client
Linux 390 64 bit	10.2.0.1 instant client

Table 14. Supported operating systems and required Oracle client library version for WebSphere Replication Server Version 9.5 (continued)

Operating system	Oracle client library version
Solaris 64 bit	10.2.0.1 instant client
Linux AMD 64 bit	10.2.0.1 instant client
Windows 32 bit	10.2.0.3 instant client patched with 10.2.0.3.0 VS2005 OCCI version
HP IPF 64 bit	10.2.0.1 instant client

Configuring the Oracle source

To enable capture of transactions from the Oracle source database, you need to set required logging options, set the IBMQREPLPATH environment variable, and enable the Q Capture program to work with the Oracle LogMiner utility.

Configuring an Oracle source database to work with a Q Capture program

You must enable archive logging, minimal supplemental logging, and table-level supplemental logging at an Oracle source database before you start the Q Capture program.

About this task

The following list describes each configuration requirement:

Archive logging
The Oracle source database must run in ARCHIVELOG mode. This mode gives the LogMiner utility access to any redo logs that might be necessary to bring the source and target servers into convergence. The Q Capture program uses the LogMiner utility with the CONTINUOUS_MINE option. To perform a continuous mine from the Oracle logs using LogMiner, archive logging must be enabled. On initialization, if the Q Capture program detects that the database is running in NOARCHIVELOG mode, the program will not start.

Minimal supplemental logging
The LogMiner utility requires supplemental logging to be enabled. By default, Oracle does not provide supplemental logging. You must enable at least minimal supplemental logging before any redo logs are generated that will be used by replication.

Table-level supplemental logging
The Q Capture program requires that table-level supplemental logging be enabled for any Oracle source table that is part of a Q subscription. Data is not captured for a table until table-level supplemental logging is enabled.

Procedure

To configure an Oracle source database to work with the Q Capture program, perform these steps:

- Ensure that archive logging is enabled. To determine the database log mode (ARCHIVELOG or NOARCHIVELOG), use the following statement:

```
SELECT LOG_MODE FROM SYS.V$DATABASE;
```

- Set minimal supplemental logging by using the following statement:
  ```
  ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
  ```
  To check whether supplemental logging is enabled, query the V$DATABASE view:
  ```
  SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE;
  ```
  If the query returns a value of YES or IMPLICIT, minimal supplemental logging is enabled.
- For each source table, enable table-level supplemental logging by using the following statement:
  ```
  ALTER TABLE table_owner.table_name ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
  ```

For more information on enabling archive logging, see the "Managing Archived Redo Logs" chapter in the *Oracle Database Administrator's Guide*. For more information on enabling supplemental logging, see "Supplemental Logging" in *Oracle Database Utilities*.

How a Q Capture program works with the Oracle LogMiner utility

The Q Capture program uses the Oracle LogMiner utility to retrieve changed data from Oracle source tables before turning the data into WebSphere MQ messages for replicating or publishing.

The following figure shows the major objects and applications that are involved in capturing data from Oracle source tables.

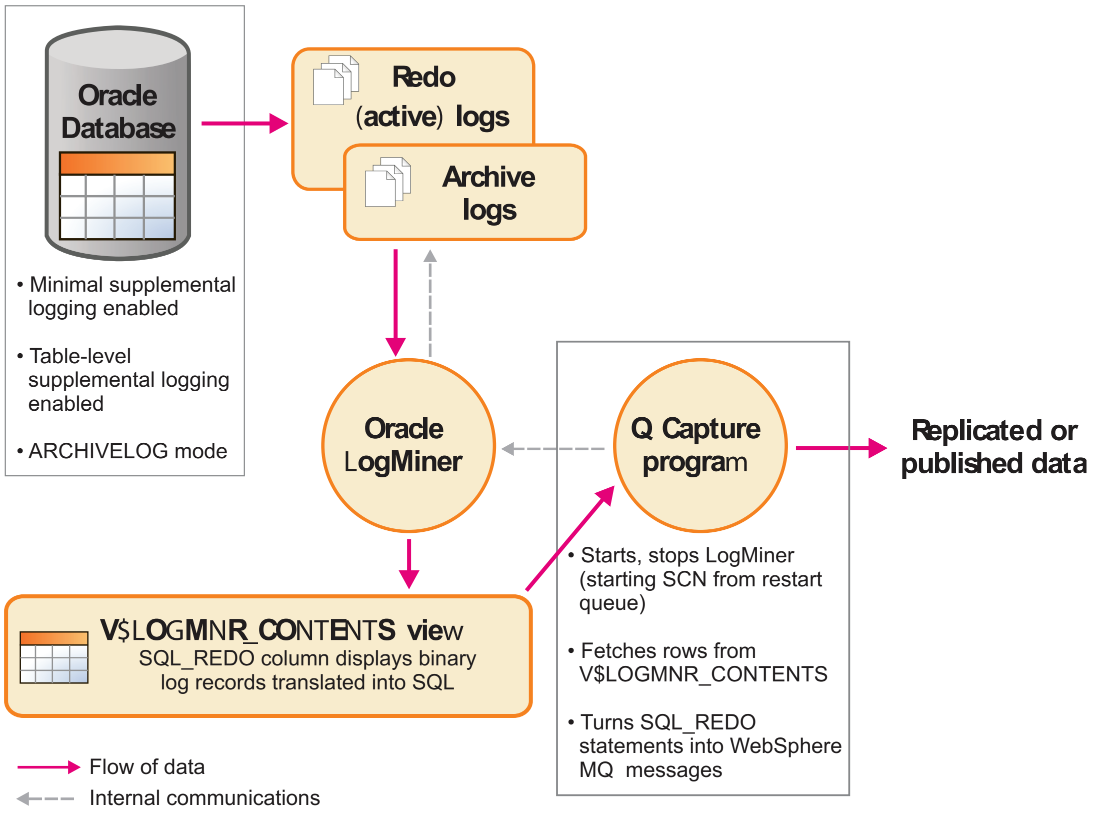

Oracle Database
• Minimal supplemental logging enabled
• Table-level supplemental logging enabled
• ARCHIVELOG mode
Redo (active) logs
Archive logs
Oracle LogMiner
Q Capture program
Replicated or published data
V$LOGMNR_CONTENTS view
SQL_REDO column displays binary log records translated into SQL
• Starts, stops LogMiner (starting SCN from restart queue)
• Fetches rows from V$LOGMNR_CONTENTS
• Turns SQL_REDO statements into WebSphere MQ messages
Flow of data
Internal communications

Figure 23. How a Q Capture program works with LogMiner to retrieve changed data

The following list describes how these objects work together:

Redo log
: The Oracle redo log consists of preallocated files that contain all changes made to the database. For Q Replication or Event Publishing, the database must operate in ARCHIVELOG mode so that the database manager never overwrites redo logs that are needed for replication or publishing.

LogMiner utility
: The utility connects to the Oracle source database, reads archive logs and active redo logs, and presents log record data in a pseudo-view called V$LOGMNR_CONTENTS. The data is reproduced in an SQL format that is used by the Q Capture program to re-create changes.

Dictionary
: LogMiner uses a dictionary to translate binary log records into plain text SQL statements. Q Replication and Event Publishing specify an online dictionary. An online dictionary always reflects the current database schema. For more details, see "How Q Capture handles alterations of Oracle source tables" on page 164.

V$LOGMNR_CONTENTS
: LogMiner creates the V$LOGMNR_CONTENTS view in response to SQL queries from Q Capture. Q Capture uses rebuilt SQL statements from V$LOGMNR_CONTENTS to create WebSphere MQ messages for replication or publishing.

Q Capture program
: The Q Capture program starts, stops, and issues queries to the LogMiner utility. Q Capture administers the utility by using two Oracle PL/SQL

packages: DBMS_LOGMNR_D to build the online dictionary, and DBMS_LOGMNR to start and stop the utility, add and remove log files, and prompt the utility to extract data from the log files. The user account that runs the Q Capture program needs special permissions to administer the LogMiner utility.

Configuring the Oracle LogMiner utility

To configure the Oracle LogMiner utility for Q Replication or Event Publishing, you create a table space for LogMiner redo tables and create a LogMiner administrator account with appropriate permissions.

About this task

The LogMiner presents changed data from the redo log in a relational pseudo-view called V$LOGMNR_CONTENTS. By default, the utility uses the SYSTEM table space for the V$LOGMNR_CONTENTS view. Oracle recommends that a different table space be created and used for LogMiner operations.

To allow the Q Capture program to access LogMiner views, you must create specific permissions. Permissions also must be granted to allow the LogMiner to access data within Oracle system tables and views that are owned by the SYS user.

Sample SQL scripts are provided in the sqllib/samples/repl/oracle directory that you can modify to specify a table space for LogMiner and to set the correct permissions. You can also use the following procedure to accomplish these tasks.

Recommendation: While Oracle supports configurations where LogMiner runs in a different database than the database instance that generates redo and archive logs, IBM does not recommend this configuration for Q Replication. The Q Capture program invokes LogMiner with the CONTINUOUS_MINE option. This option prompts LogMiner to dynamically identify and add the log files that it needs. If you run LogMiner on a remote mining database, you would need to manually make active log and archive log files available to LogMiner.

Procedure

1. Create a table space for the LogMiner utility on the Oracle source database. Follow these steps:
   a. Issue the CREATE TABLESPACE SQL statement. You can modify the following UNIX example to fit your environment:
      ```
      CREATE TABLESPACE logmnrts$
      DATAFILE
      '/dbms/oracle10/product/10.1.0/oradata/ORA10SID/logmnr01.dbf'
      SIZE 4M AUTOEXTEND ON
      NOLOGGING EXTENT MANAGEMENT LOCAL;
      ```
   b. Assign the new table space to the LogMiner utility. The following statement assigns the logmnrts$ table space that was created in Step a:
      ```
      EXECUTE SYS.DBMS_LOGMNR_D.SET_TABLESPACE('logmnrts$');
      ```
2. Create a LogMiner user account for Q Replication on the Oracle source. The following sample statements create the user asn and grant the user appropriate permissions ("asn" is not a required name for the user account; the name can be changed to meet your requirements):
   ```
   create user asn identified by asn
   default tablespace users
   temporary tablespace temp
   profile default account unlock;
   ```

```
grant connect, resource to asn;
grant create session to asn;
alter user asn quota unlimited on logmnrts$;
grant select any transaction to asn;
grant execute_catalog_role to asn;
grant select any table to asn;
grant select on sys.v_$database to asn;
grant select on sys.v_$logmnr_contents to asn;
grant select on sys.v_$logmnr_dictionary to asn;
grant select on sys.v_$logmnr_logfile to asn;
grant select on sys.v_$logmnr_logs to asn;
grant select on sys.v_$logmnr_parameters to asn;
grant select on sys.v_$logmnr_session to asn;
grant select on sys.v_$logmnr_transaction to asn;
grant select on sys.v_$log to asn;
grant select on sys.v_$logfile to asn;
grant select on sys.v_$archived_log to asn;
```

Creating an ASNCLP configuration file

The ASNCLP command-line program uses a configuration file for connecting to Oracle data sources. The configuration file is a plain text file.

About this task

You can define connection information for multiple data sources in the configuration file. You can save the configuration file to your choice of location. The **SET SERVER** command has a parameter for providing the ASNCLP program with the location of the configuration file.

Procedure

To create an ASNCLP configuration file:

1. In a text file, add a unique name enclosed in square brackets to identify the data source. The name must be eight characters or less. For example, `[Oracle1]`.
2. Enter the TYPE, DATA SOURCE, HOST, PORT parameters in your configuration file and replace the sample values with your own. The following example shows values for an Oracle source:
   ```
   [Oracle1]
   Type=ORACLE
   Data source=ORADB
   Host=9.30.155.156
   Port=1521
   ```
3. Save the file to a location of your choice. When you use the **SET SERVER** command, you must enter the full path to the file. The following is an example of the SET SERVER command that specifies an ASNCLP configuration file:
   ```
   SET SERVER capture TO CONFIG SERVER ORADB
        FILE "c:\ora_setup\asnservers.ini"
        ID myuserid PASSWORD "mypassword";
   ```

Creating Q Capture control tables for an Oracle source

Before you can publish or replicate data from an Oracle source, you must create *control tables* for a Q Capture program in the source Oracle database. Control tables store information about Q subscriptions and publications, message queues, operational parameters, and user preferences.

Before you begin

- The ASNCLP command-line program or Replication Center must be able to connect to the source Oracle database where you want to create the Q Capture control tables. To permit connections with the ASNCLP, create an ASNCLP configuration file.
- You must have the names of the following WebSphere MQ objects:
  - A queue manager that the Q Capture program works with
  - A local, persistent queue to serve as the administration queue
  - A local, persistent queue to serve as the restart queue

  Tip: Use the **`VALIDATE WEBSPHERE MQ ENVIRONMENT FOR`** command in the ASNCLP or the validate controls in the Replication Center to ensure that the queue managers and queues that you specify for the control tables exist and have the correct properties. For this **`VALIDATE`** command to work, the Q Capture program and the WebSphere MQ objects must be located on the same server as the Oracle database.

About this task

Each instance of the Q Capture program has its own set of control tables, which are identified by the Q Capture schema. For example, the control table that stores operational parameters for a Q Capture program with a schema of ASN1 would be named ASN1.IBMQREP_CAPPARMS.

The Q Capture control tables are created natively in the Oracle database, by default in a single table space. You can customize where each control table is created, and you can specify existing table spaces or create new table spaces.

Procedure

To create Q Capture control tables on an Oracle source, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **`CREATE CONTROL TABLES FOR`** command. For example, the following commands set the environment and create control tables in the SAMPLE Oracle database with a Q Capture schema of ASN1:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE CONFIG SERVER SAMPLE`
	`FILE asnservers.ini ID id1 PASSWORD pwd1;`
	`SET QMANAGER "QM1" FOR CAPTURE SCHEMA;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	
	`CREATE CONTROL TABLES FOR CAPTURE SERVER`
	`USING RESTARTQ "ASN1.QM1.RESTARTQ"`
	`ADMINQ "ASN1.QM1.ADMINQ" MEMORY LIMIT 64`
	`MONITOR INTERVAL 600000;`
	
	The **`CREATE`** command specifies a restart queue and administration queue, doubles the default amount of memory that is available to build transactions to 64 MB, and reduces the default interval for recording performance information to 600000 milliseconds (one minute).

Method	Description
Replication Center	When you create Q Capture control tables on an Oracle source or add an existing Q Capture server to the Replication Center, use the Specify an Oracle Server window. Right-click the **Q Capture Servers** folder and click **Create Q Capture control tables**. Click **Add Oracle Server** and specify the Oracle source database at the Specify an Oracle Server window: **Database name** The Oracle service, or system identifier, of the database that contains the Q Capture control tables **Database alias** An 8-character TNS_Alias for the database that contains the Q Capture control tables **System name** The host name or IP address of the system where the database runs **Port number** The port number where the database looks for incoming connection requests **User ID** A user ID for connecting to the Oracle source server **Password** A password for connecting to the Oracle source server

Creating Q subscriptions for Oracle sources

A Q subscription maps the Oracle table that contains your source data to a copy of that table at the DB2 or federated target. When you create a Q subscription, you specify a replication queue map, target table options, and other preferences. You create one Q subscription for each table that you want to replicate.

Before you begin

- Plan how you want to group replication queue maps and Q subscriptions.
- Create the control tables for the Q Capture program in the Oracle source server that contains the source table for the Q subscription.
- Create the control tables for the Q Apply program in the server that contains the target for the Q subscription.
- Specify the queues for replicating and their attributes by creating a replication queue map. (You can do this task before you create a Q subscription or while you create a Q subscription.)
- Prepare the stored procedure if you want the Q Apply program to pass source changes to a stored procedure instead of to a target table.

If you choose to load the target tables automatically with data from an Oracle source, you must use a federated database. The federation functions for Oracle and other databases are built into InfoSphere Replication Server. The federation features allow the Q Apply program to connect and pull data from the Oracle source into the target tables. You must create the following objects in the federated database that point to the Oracle source tables:

- Oracle wrapper
- Remote server definition

- User mapping

Restrictions

- Multidirectional Q subscriptions are not supported for Oracle sources.
- If you use a search condition to filter changes to replicate, the predicate must use valid Oracle syntax.
- Truncate table operations at the Oracle source database are not replicated.

Procedure

To create Q subscriptions for Oracle sources:

Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **CREATE QSUB** command. For example, the following commands set the environment and create a Q subscription with the following characteristics:
	• The Oracle source server is LONDON and the ASNCLP connection information is stored in the `asnservers.ini` file under the LONDON entry.
	• The DB2 target server is DALLAS.
	• The replication queue map is LONDON_ASN_TO_DALLAS_ASN
	• The source table is SALES and the target table is TGTSALES.
	• The Q subscription name is SALES_REPORTING.
	• The table is loaded automatically by using LOAD FROM CURSOR option (HAS LOAD PHASE I and LOAD TYPE 1).
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO CONFIG SERVER LONDON`
	`   FILE asnservers.ini ID MYUSERID PASSWORD MYPASS;`
	`SET SERVER TARGET TO DB DALLAS;`
	`SET CAPTURE SCHEMA SOURCE ASN;`
	`SET APPLY SCHEMA ASN;`
	`SET QMANAGER QM1 FOR CAPTURE SCHEMA;`
	`SET QMANAGER QM2 FOR APPLY SCHEMA;`
	`CREATE QSUB (SUBNAME SALES_REPORTING`
	`  REPLQMAP LONDON_ASN_TO_DALLAS_ASN SALES`
	`  OPTIONS HAS LOAD PHASE I TARGET NAME TGTSALES`
	`  TYPE USERTABLE NEW NICKNAME RMT SERVERNAME RMTLONDN`
	`  NICKSALES LOAD TYPE 1 );`

Method	Description
Replication Center	Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture schema or Q Apply schema in the object tree, right-click the **Q Subscriptions** folder, and click **Create**. **Servers page** Specify the Oracle server where the Q Capture program runs as the source server. **Rows and Columns page** The **Data type** column shows the Oracle data type of the column. The Q Capture program converts supported Oracle data types to DB2 native data types. You can view the data type mapping in the Column Mapping window. For details, see Oracle data types. **Loading Target Tables page** If you want to perform an automatic load of the target tables, you select the **Automatic** radio button and the **Always use LOAD FROM CURSOR** option of the **LOAD utility** radio button. Use the controls in the **Nickname** area to prompt the Replication Center to create a new nickname or to specify an existing nickname for the loading process.

Load options for Q Replication from an Oracle source

When the source table for a Q subscription is in an Oracle database, you can choose among automatic load, manual load, and no load options for the target table.

The following table describes each option.

Table 15. Load options for Q subscriptions with Oracle sources

Option	Description
Automatic	To automatically load target tables, use the federated database function in InfoSphere Replication Server to prompt the Q Apply program to pull data from the Oracle source tables into DB2 target tables. The Q Apply program uses the LOAD FROM CURSOR option of the DB2 LOAD utility. The LOAD utility fetches data from Oracle by going through nicknames in the federated database. The configuration for loading differs depending on the DB2 target: z/OS For target tables in DB2 for z/OS, you need a federated database on Linux, UNIX, or Windows so that the LOAD utility can load the target tables on DB2 for z/OS. Update the database manager configuration using `FEDERATED YES`, create a federated database, and provide the database with information about your Oracle database and tables. **Note:** The server where the federated database resides must already have an Oracle client installed and configured to connect to the Oracle database. To set up the loading process, you can download two sample scripts from the replication samples collection on IBM developerWorks: **001_federated_setup.db2** This script creates a federated database, an Oracle wrapper, federated server definition, user mapping, and nickname for the Oracle source table. **002_zos_setup.db2** This script allows the z/OS target access to the federated nickname that is created in `001_federated_setup.db2`. **Note:** Create Q subscriptions for any Oracle source tables before you use these samples. The federated database and nickname must match the Oracle database and source table so that Q Apply will use LOAD FROM CURSOR. Linux UNIX Windows For distributed targets, you must enable federated database function in the target database and provide the federated server with information about your Oracle database and tables. Before you create Q subscriptions that map Oracle source tables to DB2 target tables, you first create the following objects in the federated server to enable access to Oracle: • Oracle wrapper • Remote server definition • User mapping When you create Q subscriptions that use LOAD FROM CURSOR, you specify the federated objects to use. You can have the ASNCLP command-line program or Replication Center create the nicknames that point to the Oracle source tables, or you can use existing nicknames.
Manual	Manual loading, in which you load the target tables with a utility of your choice and then signal the replication programs when the load is complete, is supported for Oracle sources. For example, you might use the IBM Migration Toolkit to load the target tables with data from the Oracle source, and then use the ASNCLP or Replication Center to indicate when the load is finished.

Table 15. Load options for Q subscriptions with Oracle sources (continued)

Option	Description
No load	When you specify the no load option for a Q subscription, the Q Apply program begins applying transactions to a target table as soon as the Q subscription becomes active. If you choose the no load option, make sure that the values of the primary key or unique index from the source table are also present in the primary key or unique index of the target table.

Starting a Q Capture program for Oracle sources

Start a Q Capture program to begin capturing transactions or row-level changes from Oracle for active or new Q subscriptions or publications.

Before you begin

- Create a WebSphere MQ queue manager, queues, and other required objects.
- Ensure that you have authorization for Q Replication and event publishing objects and WebSphere MQ objects.
- Create control tables on the Oracle source database for the appropriate Q Capture schema.
- Configure the Oracle source database to work with the Q Capture program.

About this task

When you initially start a Q Capture program without specifying a start mode, it uses the default start mode, warmsi. In this mode, the program tries to read the log at the point where it left off. Because the program is being started for the first time, it switches to cold start mode and begins processing Q subscriptions or publications that are in N (new) or A (active) state. Any Q subscriptions or publications that are in I (inactive) state must be activated for the program to begin capturing changes.

You can start a Q Capture program even if no Q subscriptions or publications are in A (active) state. When you activate the Q subscriptions or publications, the Q Capture program begins capturing changes for those Q subscriptions or publications.

When you start a Q Capture program, you can specify startup parameter values, and the program will use the new values until you take one of the following actions:

- Change the parameter values while the program is running.
- Stop and restart the program, which prompts it to read the IBMQREP_CAPPARMS table and use the values saved there.

Procedure

To start a Q Capture program, use one of the following methods:

Method	Description

asnoqcap **command**	Use the **asnoqcap** command to start a Q Capture program for Oracle sources and specify startup parameters. For example:
	`asnoqcap capture_server=`*`server_name`*
	`capture_schema=`*`schema`*
	`parameters`
	
	Where *server_name* is the name of the database that contains the Q Capture control tables, *schema* identifies the Q Capture program that you want to start, and *parameters* is one or more parameters that you can specify at startup.

Windows Windows services	You can create a replication service on Windows operating systems to start the Q Capture program automatically when the system is started.

You can verify whether a Q Capture program started by using one of the following methods:
- Issue the **asnoqccmd status** command.
- Examine the Q Capture diagnostic log file (*`capture_server.capture_schema`*`.QCAP.log`) for a message that indicates that the program is capturing changes.
- Check the IBMQREP_CAPTRACE table for a message that indicates that the program is capturing changes.

How Q Capture handles alterations of Oracle source tables

When you alter an Oracle source table, the Q Capture program continues to replicate data as long as the existing column definitions are not changed. The table alterations are not automatically replicated to the target.

The following table describes how Q Capture handles different types of DDL and what you need to do for any affected Q subscriptions:

Table 16. How Q Capture handles DDL changes to source tables and what you need to do

DDL operation	How it is handled	What you need to do
ALTER TABLE ADD (COLUMN)	Q Capture leaves the Q subscription active, but does not replicate the added column until it receives an ADDCOL signal.	Use the ASNCLP ALTER ADD COLUMN command, or manually insert an ADDCOL signal, to indicate that you want to replicate the new column.
DROP TABLE	Q Capture leaves the Q subscription active, but there will be no log records to read for the source table.	Stop the Q subscription, and then drop the Q subscription.

Table 16. How Q Capture handles DDL changes to source tables and what you need to do (continued)

DDL operation	How it is handled	What you need to do
DDL that alters the structure of a table Examples: • DROP COLUMN • SET UNUSED • MODIFY COLUMN • RENAME COLUMN	The Q subscription for any altered table is stopped. Q Capture keeps running.	1. Drop and recreate the Q subscription to synchronize the source and target table definitions in the Q Capture and Q Apply control tables. 2. Restart the Q subscription to trigger a new load of the target table.
DDL that does not alter the table structure Examples: • CREATE INDEX • ADD/MODIFY CONSTRAINT • ADD DEFAULT VALUE	Q Capture leaves the Q subscription active.	Ensure that unique constraints, primary keys, and referential integrity constraints match between the source and target tables. If you change any of these properties at the source, make a corresponding change to the target to avoid unexpected behavior.

Oracle data types

The Q Capture program converts the supported Oracle data types to DB2 native data types. Some Oracle data types are not supported for Q Replication or Event Publishing.

Supported Oracle data types

The following Oracle data types are supported for Q Replication and Event Publishing:

- CHAR
- NCHAR
- VARCHAR2
- NVARCHAR2
- NUMBER
- DATE
- TIMESTAMP
- RAW
- CLOB
- NCLOB
- BLOB
- BINARY_FLOAT
- BINARY_DOUBLE

Support for LOB data types is available only for redo logs that are generated by a database with compatibility set to a value of 9.2.0.0 or higher.

You cannot replicate LONG or LONG RAW columns. If you convert these columns to LOB columns, you can replicate the data.

Data type mapping

The Q Capture program converts the supported Oracle data types to DB2 native data types. The Q Capture program converts the data types according to the table below.

Table 17. Oracle to DB2 data type mapping

Oracle data type	Oracle Lower Length	Oracle Upper Length	Oracle Lower Scale	Oracle Upper Scale	Oracle Bit Data	DB2 data type	DB2 Length	DB2 Scale	DB2 Bit Data
BLOB	0	0	0	0	-	BLOB	2147483647	0	Y
CHAR	1	254	0	0	-	CHAR	0	0	N
CHAR	255	2000	0	0	-	VARCHAR	0	0	N
CLOB	0	0	0	0	-	CLOB	2147483647	0	N
DATE	0	0	0	0	-	TIMESTAMP	0	0	N
FLOAT	1	126	0	0	-	DOUBLE	0	0	N
LONG	0	0	0	0	-	CLOB	2147483647	0	N
LONG RAW	0	0	0	0	-	BLOB	2147483647	0	Y
NUMBER	1	38	-84	127	-	DOUBLE	0	0	N
NUMBER	1	31	0	31	-	DECIMAL	0	0	N
NUMBER	1	4	0	0	-	SMALLINT	0	0	N
NUMBER	5	9	0	0	-	INTEGER	0	0	N
NUMBER	-	10	0	0	-	DECIMAL	0	0	N
RAW	1	2000	0	0	-	VARCHAR	0	0	Y
ROWID	0	0	0	NULL	-	CHAR	18	0	N
TIMESTAMP	-	-	-	-	-	TIMESTAMP	10	-	-
VARCHAR2	1	4000	0	0	-	VARCHAR	0	0	N

Data type limitations

Some Oracle NUMBER values might be rounded by replication.

Oracle data types are converted to DB2 data types when they are replicated. Because of differences between the mapped data types, Oracle NUMBERS might be rounded depending on their length and scale.

Oracle NUMBER columns with the following attributes might be rounded:

- NUMBER columns that are defined with length 31 to 38 might be rounded.
- NUMBER columns that are defined with length 1 to 31 and scale is -84 to 0, or scale is 31 to 127 might be rounded.

Note: When the length is not specified for a NUMBER, the NUMBER is treated as having length 38 and scale 0.
The DB2 DOUBLE data type has a maximum of 17 significant digits. This means that if a value has more than 17 significant figures, the value will be rounded. For example, if your Oracle NUMBER value is 9876543210987654321, your DB2 DOUBLE value will be 9.8765432109876543E 16, which rounds off the last two digits of the Oracle

value.
To avoid rounding, specify both length (and scale if required) in NUMBER when a column is declared.

Using Oracle TIMESTAMP columns as key for replication or publishing requires 10.2.0.5 patch set
TIMESTAMP columns for Oracle source tables can only be used as keys for replication or publishing (IS_KEY=Y) if you install the Oracle 10.2.0.5 patch set. Without the upgrade, the LogMiner utility does not return the subsecond value for timestamps, and when you replicate timestamp values, the precision is in seconds rather than subseconds.

Chapter 10. Manipulating data by using stored procedures as a target

When you create a Q subscription for unidirectional replication, you can specify that the changes be replicated from the source table to a stored procedure instead of directly to a target table. You can use the stored procedure to manipulate data that is captured from the source before the data is applied to the target table.

Stored procedures for manipulating source data for Q Replication

Typically in Q subscriptions, data from a source table is mapped to a target table; however, for Q subscriptions in unidirectional replication, you can instead have the Q Apply program call a stored procedure and pass the source data as input parameters to the stored procedure.

The source columns map to parameters in the stored procedure instead of to target columns. By mapping source columns directly to parameters in a stored procedure, you avoid the need to parse the incoming data and have a clean, simple programming model.

When you specify that you want the target of a Q subscription to be a stored procedure, you still have all of the options that you have for a Q subscription with a target table. For example, with a stored procedure, you still choose error options.The stored procedure can refer to either a DB2 table or a nickname in a DB2 federated database.

The stored procedure returns a return code to the Q Apply program that indicates whether the data was applied to the target table. The developer that creates the stored procedure must ensure that the stored procedure returns an appropriate SQL return code to the Q Apply program. The stored procedure is responsible for getting the source data to its final destination.

Version 10.1 and later: The mandatory stored procedure parameter `src_commit_lsn` must specify a data type of VARCHAR(16) under these conditions:

- The source database is DB2 10.1 for Linux, UNIX, and Windows or later
- The Q Capture program is at Version 10.1 or later and the value of the **`compatibility`** parameter is 1001 or higher

If the source database and Q Capture program are at V10.1 but **`compatibility`** is lower than 1001, the value of the `src_commit_lsn` parameter can be CHAR(10) because Q Capture continues to use 10-byte log sequence numbers.

Example

The following example shows a signature of a stored procedure that accepts values from five columns in the source table, two of which are key columns.

```
CREATE PROCEDURE storedprocedure_name(
                                        INOUT operation integer,
                                        IN suppression_ind VARCHAR(size),
                                        IN src_commit_lsn CHAR(10),
                                        IN src_trans_time TIMESTAMP,
                                        XParm1,
                                        Parm1,
                                        XParm2
```

```
                        Parm2,
                        Parm3,
                        Parm4,
                        Parm5
                       )
```

This example shows the four mandatory parameters:

operation
: This INOUT parameter identifies the type of operation.

suppression_ind
: This IN parameter identifies the parameters that have been suppressed.

src_commit_lsn
: This IN parameter identifies the log sequence number of when the source server issued the COMMIT for the transaction.

src_trans_time
: This IN parameter identifies the timestamp of when the source server issued the COMMIT for the transaction.

The signature also contains the before and after values for key columns and only the after values for non-key columns. The following parameters accept values from source columns:

- Parameters Parm1 and Parm2 map to the key columns in the source table.
- Parameters XParm1 and XParm2 accept the before values of key columns in the source table.
- Parameters Parm3, Parm4, and Parm5 map to non-key columns in the source table.

See the sample programs for examples of stored procedures for Q Replication that are written in C and in SQL.

The stored procedure must not perform COMMIT or ROLLBACK functions because the Q Apply program commits or rolls back transactions.

The Q Apply program handles LOB data for stored procedures similarly to how it handles LOB data for target tables. For Q subscriptions with stored procedures, LOB changes are replicated by sending them in multiple messages, depending on the size of the LOB and the message size that you allowed the Q Capture program to send. The stored procedure call that involves the first LOB message for a single LOB change is the original row operation. All of the remaining LOB data for the single LOB change is sent by additional calls to the stored procedure. The Q Apply program transforms the operation into an update. The suppression_ind index marks all parameters as suppressed, except for the parameter that maps to the LOB column in the source table. The stored procedure is called each time a LOB message is on the receive queue.

If a LOB value is small enough to be sent in a single message, the special update operation with append (operation value 34) is not necessary. The stored procedure writer must write logic to handle each operation code as well as column suppression; the Q Apply program has all other logic for handling LOB values.

If a LOB value is too big to be sent in one message (because the size exceeds the maximum message size) and, therefore, is split into several messages, the first message is handled differently than the subsequent messages. The Q Apply program appends any portions of a LOB value that are not sent in the first message using the concatenation operation ||. A special operation code is passed to indicate this special form of update.

Example of a LOB insert

In a stored procedure that has two LOB columns, C1 and C2, the following would occur in an insert:

SQL	Operation value
`INSERT C1 = first portion C2 = some dummy value`	16
`UPDATE C1 = C1 || next portion`	34
`UPDATE C1 = C1 || next portion`	34
`UPDATE C2 = first portion`	32
`UPDATE C2 = C2 || next portion`	34
`UPDATE C2 = C2 || next portion`	34

Writing stored procedures to manipulate source data for Q Replication

You can write a stored procedure so that you can use it as a target in a Q subscription. You must write the stored procedure before you can declare it as the target in a Q subscription.

Restrictions

- The stored procedure must not perform COMMIT or ROLLBACK functions.
- You cannot send the before value from a non-key column to a parameter in the stored procedure.
- Because the Q Apply program knows only about the stored procedure and not about the table that the stored procedure might pass the manipulated data into, Q subscriptions that have stored procedures associated with them cannot maintain referential integrity.
- If the stored procedure maintains a target table and you want the Q Apply program to resolve conflicts, then do not create secondary unique indexes on the target table.

About this task

The stored procedure consists of four mandatory parameters (one that identifies the operation and three that identify the transaction) and additional parameters that map to source columns. You can write a stored procedure that refers to either a DB2 table or a nickname in a DB2 federated database.

Procedure

To write a stored procedure to be the target in a Q subscription and to set up the stored procedure:

1. Write a stored procedure that contains four mandatory parameters and additional parameters that map to source columns. The parameters in the stored procedure are positional. The first parameter must have the parameter mode INOUT. All other parameters must have parameter mode IN. See the following topics:

- "Stored procedure parameter that identifies the type of operation for Q Replication"
- "Stored procedure parameter that identifies whether each source column was suppressed" on page 174
- "Stored procedure parameters that identify the transaction for Q Replication" on page 175, which include the following two mandatory parameters:
  - The log sequence number of when the source server issued the commit statement for the transaction
  - The timestamp of when the source server issued the commit statement for the transaction
- "Stored procedure parameters that map to source columns for Q Replication" on page 175

Tip: See the sample programs for examples of stored procedures for Q Replication that are written in C and in SQL.

2. Run the `CREATE PROCEDURE` statement runs properly so that the name and appropriate parameters of the stored procedure are registered with the DB2 database.

   Tips:
   - Use the GENERAL WITH NULLS parameter style when you declare the stored procedure. This style increases portability of the stored procedure across the DB2 family.
   - Linux UNIX Windows Make sure that the stored procedure exists in the sqllib/function directory.
3. Compile and bind the stored procedure at the target server.

   z/OS Include the DBRM of the stored procedure in the Q Apply package list.
4. Ensure that the Q Apply program has the proper authority to call the stored procedure.

   After the call to the stored procedure, the Q Apply program always checks the SQL code before checking the return code to validate whether the stored procedure is successful.
5. If the stored procedure maintains a target, ensure that the target exists.
6. Declare the stored procedure as the target in a Q subscription.

Stored procedure parameter that identifies the type of operation for Q Replication

The attributes that you specify for the Q subscription determine how the Q Apply program handles unexpected conditions in the target (in this case, in the stored procedure).

Declare the first parameter in the stored procedure as an INOUT parameter that the Q Apply program uses to pass in the type of operation (`INSERT`, `UPDATE`, `DELETE`, or `KEY UPDATE`) and that the stored procedure uses to pass a return code back to the Q Apply program about whether the operation was successful.

Tip: See the sample programs for examples of the `operation` parameter in stored procedures that are written in C and SQL. Note how the `operation` parameter is used to pass the return code back to the Q Apply program.

Table 18 shows the operation values that the Q Apply program passes to the stored procedure and what each value means.

Table 18. SQL return codes that the Q Apply program passes to the stored procedure

Operation value	Type of operation
64	Delete
16	Insert
32	Update to non-key columns
128	Update to key columns
34	Update and append

Set the `INOUT operation integer` parameter to the SQL code for the Q Apply program to evaluate to see if the change was successfully applied to the target. If the stored procedure does not return a SQL code to the Q Apply program, then the Q Apply program does not know what happened inside the stored procedure.

The stored procedure must not perform COMMIT or ROLLBACK functions because the Q Apply program commits or rolls back transactions.

If other applications are also making changes to the table that the stored procedure is applying changes to, then the stored procedure might encounter unexpected conditions in that table. For example, the stored procedure might try to update a row that another application already deleted. You have the following choices for how to resolve unexpected conditions that might occur when the stored procedure attempts to insert the manipulated source data into the target table:

- "The Q Apply program handles the unexpected condition"
- "The stored procedure handles the unexpected condition"

The Q Apply program handles the unexpected condition

If you want the Q Apply program to handle the unexpected condition in the target that the stored procedure maintains, then write the stored procedure so that it returns one of the SQL codes listed below (Table 19 on page 174) to the Q Apply program, and define the Q subscription for the Q Apply program to force changes into the target that the stored procedure maintains. The Q Apply program forces the source change. In the case of certain SQL return codes, the Q Apply program transforms the row operation, logs the exception in the IBMQREP_EXCEPTIONS table, and passes the transformed row operation back to the stored procedure.

The stored procedure handles the unexpected condition

If you do not want the Q Apply program to handle unexpected conditions in the target by forcing source changes into the target, you can build other error-handling logic into the stored procedure. The stored procedure must always return an SQL code to the Q Apply program, but the stored procedure can return 0 (zero) so that the Q Apply program does not know about unexpected conditions or actual failures in the target table. After the stored procedure passes SQL return codes back to the Q Apply program in the `operation` parameter, the Q Apply program interprets each SQL code and starts the appropriate action (depending on how you specified that the Q Apply program should handle unexpected conditions). The Q Apply program handles return codes of +100 and -803 as conflicts, and any other return code as an error. Table 19 on page 174 shows the types of +100 and -803 return codes that the stored procedure outputs, how the Q Apply program interprets that type of return code, and what action the Q Apply program takes as a result. The information in the table below assumes that you specified for the Q Apply program to force target changes.

Table 19. SQL return codes that the stored procedure passes to the Q Apply program and how the Q Apply program responds.

SQL return code	Type of operation	What the return code means	How Q Apply reacts
0	Insert	The row was inserted successfully.	Q Apply processes the next row.
0	Update	The row was updated successfully.	Q Apply processes the next row.
0	Delete	The row was deleted successfully.	Q Apply processes the next row.
+100	Delete	The row was not found in the target.	Q Apply does not retry the call.
+100	Update	The row was not found in the target.	If you specified for Q Apply to force changes, Q Apply changes the update into an insert and tries the call again.
-803	Insert	The row already exists in the target	If you specified for Q Apply to force changes, Q Apply transforms the insert into an update and tries the call again.
-803	Key update	The new key already exists in the target.	If you specified for Q Apply to force changes, Q Apply transforms the operation to an update operation that has the new keys and tries the call again.

Stored procedure parameter that identifies whether each source column was suppressed

After the parameter that identifies the operation (`operation`), declare the second parameter (`suppression_ind`) in the stored procedure as an IN parameter that identifies whether each source column was suppressed.

The IN parameter `suppression_ind` is a character array (`colsupind`) that holds a character for each parameter except for the four mandatory parameters. The character indicates whether the value of the source column was suppressed:

1 The value of the source column that correlates with the parameter was suppressed.

0 The value of the source column that correlates with the parameter was not suppressed

The array must be used inside the stored procedure for evaluating the functions parameters. The Q Apply program checks to ensure that exactly one before value exists for each parameter that maps to a key column.

Tip: See the sample programs for examples of the parameter that identifies suppressed columns in stored procedures that are written in C and SQL.

Stored procedure parameters that identify the transaction for Q Replication

After the parameter that identifies whether any source columns were (`suppression_ind`), declare the next two parameters in the stored procedure as IN parameters that take in information from the message header from the Q Apply program that identifies the transaction. Because the Q Apply program passes rows and not transactions to the stored procedure, the transactional parameters give you information to help you identify which transaction this row change belongs to, and the time stamps and log sequence numbers also help with error reporting or diagnostics.

The following two parameters identify the transaction:

`IN src_commit_lsn CHAR(10)`
: The log sequence number of when the source server issued the commit statement for the transaction. The data type that you specify for this parameter must be VARCHAR(16) if the source database is DB2 10.1 for Linux, UNIX, and Windows or later and the Q Capture program is at Version 10.1 or later with a value of 1001 or higher for the **`compatibility`** parameter.

`IN src_trans_time TIMESTAMP`
: The timestamp of when the source server issued the commit statement for the transaction.

Tip: See the sample programs for examples of the two transactional parameters in stored procedures that are written in C and SQL.

Stored procedure parameters that map to source columns for Q Replication

After the parameters in the stored procedure that identify the transaction (`src_commit_lsn` and `src_trans_time`), declare the next parameters in the stored procedure as IN parameters that take in the source data from each column. You can list the additional parameters that map to columns in any order.

After the operational and transactional parameters, include only parameters in the stored procedure that map to the TARGET_COLNAME values in the IBMQREP_TRG_COLS table. No other additional parameters are allowed in the stored procedure, even if they are declared as OUT or INOUT parameters.

The names that are listed in the TARGET_COLNAME column must be the same names as the stored procedure parameters. Table 20 shows an example of how the source column names correspond to parameter names in the IBMQREP_TRG_COLS table. The table shows only the three relevant columns from the control table.

Table 20. Source column names and names of parameters in the stored procedure that are stored in the IBMQREP_TRG_COLS table

...	SOURCE_COLNAME	TARGET_COLNAME	IS_KEY	...
	Col1	Parm1	Y	

Table 20. Source column names and names of parameters in the stored procedure that are stored in the IBMQREP_TRG_COLS table (continued)

...	SOURCE_COLNAME	TARGET_COLNAME	IS_KEY	...
	Col2	Parm2	Y	
	Col3	Parm3	N	
	Col4	Parm4	N	
	Col5	Parm5	N	

You must declare some parameters in the stored procedure to be part of the key. The parameters that you declare as target key columns must map to key columns at the source; the source and target keys must match. The Q Apply program uses these key columns to correctly order transactions that deal with the same rows.

If the parameter maps to a key column in the source table, then you must perform the following actions:

- Declare the corresponding before-value column as a parameter.
- Ensure that the parameter for the key's before value appears directly in front of the parameter for the key's after value.
- Start the name of the parameter for the key's before value with 'X' and then the parameter name. For example, for the key column named `Parm1`, the parameter for the before value is `XParm1`.

Do not declare parameters for before values of non-key columns.

Tip: If you have many source columns that you are mapping to parameters in the stored procedure, consider naming the parameters with the names of the source columns. If the parameters are named identically to the source columns to which they correspond, then the Replication Center can automatically map the source column names to the target parameter names.

Example of a valid signature of a stored procedure with key parameters: In this example, `Parm1` and `Parm2` are parameters that map to key columns.

```
CREATE PROCEDURE storedprocedure_name(
                                      INOUT operation integer,
                                      IN suppression_ind VARCHAR(size),
                                      IN src_commit_lsn CHAR(10),
                                      IN src_trans_time TIMESTAMP,
                                      XParm1,
                                      Parm1,
                                      XParm2,
                                      Parm2,
                                      Parm3,
                                      Parm4,
                                      Parm5
                                     )
```

Example of an invalid stored procedure with key parameters: In this invalid example, `Parm1` and `Parm2` are parameters that map to key columns. This example is invalid because there is no parameter that accepts the before value of the key parameter `Parm2`.

```
#-----------------------------------------------------------------
#  Incorrect:  Don't do this
#-----------------------------------------------------------------

CREATE PROCEDURE storedprocedure_name(
                                     INOUT operation integer,
                                     IN suppression_ind VARCHAR(size) ,
 IN src_commit_lsn CHAR(10),
 IN src_trans_time TIMESTAMP,
 XParm1,
 Parm1,
 Parm2,
 XParm3,
 Parm3,
 Parm4,
 Parm5
)
```

Tip: See the sample programs for examples of parameters that map to source columns in stored procedures that are written in C and SQL.

Version 10.1 and later: The mandatory stored procedure parameter `src_commit_lsn` must specify a data type of VARCHAR(16) under these conditions:

- The source database is DB2 10.1 for Linux, UNIX, and Windows or later
- The Q Capture program is at Version 10.1 or later and the value of the **`compatibility`** parameter is 1001 or higher

Chapter 11. Loading target tables for Q Replication

When you create a Q subscription, you can choose among several options for loading target tables with data from the source.

Automatic load
: The Q Apply program manages the loading of target tables. You can select which load utility the Q Apply program calls, or you can let the Q Apply program choose the best available utility for your operating system and version.

Manual load
: You handle the loading of target tables, and then signal the replication programs when loading is done. This option is also known as external load.

No load
: You either do not load target tables, or you load target tables outside the context of the replication programs.

Recommendations for loading target tables for Q Replication

Q Replication can be configured for automatic loading of target tables, and is designed so that replication can continue during the loading process without any loss of data.

Here are a few recommendation for making sure that the process goes smoothly:

Applications at the target
: Do not allow applications to update the target tables while they are being loaded. Data in the tables will be inconsistent while the tables are being loaded, and the Q Apply program drops referential integrity constraints until the target table and any other related target tables are loaded.

  Applications can safely use target tables again when the Q Apply program makes the following changes to the IBMQREP_TARGETS table:

  - Sets the STATE column to A (active).
  - Sets the STATE_INFO column to ASN7606I, which means that if the target table had referential integrity constraints, they have been restored.

  You can use the Manage Q Subscriptions window in the Replication Center to verify that these two changes have been made. To open the window, right-click the Q Capture server where the source table for the Q subscription is located and select **Manage** > **Q Subscriptions**.

  If the loading process fails, or if the Q Apply program stops during the load, any data that was in the target table before the load began is deleted. Changes that were made to the source table during the load process are not lost, and will be applied to the target table after it is successfully loaded.

Applications at the source
: Load target tables during a time of relative inactivity at the source.

Automatic load option for Q Replication

You can choose to let the Q Apply program load the target table for a Q subscription when the Q subscription is activated. This option, known as an *automatic load*, is the default for Q Replication.

By default, when you specify an automatic load the Q Apply program chooses the best available load utility for your operating system and version. If you prefer, you can specify which load utility the Q Apply program uses when you create a Q subscription.

During the automatic loading process, any source transactions that are captured and sent to the Q Apply program are placed in a temporary spill queue by the Q Apply program. This allows replication to continue during the loading process. The Q Apply program applies these transactions after the target table is loaded.

Utilities used for automatic load option for Q Replication

If you choose an automatic load, you can let the Q Apply program select the best available load utility, or you can specify a utility.

The following list shows the available load utilities.

LOAD from CURSOR
Uses an option of the DB2 LOAD utility to move data from the source table to the target table without creating an intermediate exported file.

You can modify and use the following SQL statements to define the wrapper, server, user mapping, and nickname that are needed for LOAD from CURSOR:

```
CREATE WRAPPER DRDA;
CREATE SERVER MVS TYPE DB2/MVS VERSION 8 WRAPPER DRDA AUTHID AZUMA
PASSWORD AZUMA OPTIONS (ADD DBNAME 'ONOGAWA', PASSWORD 'Y');
CREATE USER MAPPING FOR USER
SERVER MVS
OPTIONS (REMOTE AUTHID 'AZUMA',
REMOTE_PASSWORD 'AZUMA');
CREATE NICKNAME T1NK FOR MVS.AZUMA.T1;
```

The following statement drops the nickname after the load is finished:

```
DROP NICKNAME T1NK;
```

Important: If you plan to use LOAD from CURSOR with a nickname to load from a DB2 source that is at Version 9.7 or newer, you must ensure that the following option is set for the server that owns the nickname:

```
db2 alter server server_name OPTIONS(ADD CONCURRENT_ACCESS_RESOLUTION 'W');
```

EXPORT and LOAD utilities
Uses a combination of the DB2 EXPORT utility and the DB2 LOAD utility.

EXPORT and IMPORT utilities
Uses a combination of the DB2 EXPORT utility and the DB2 IMPORT utility.

Important: If you plan to use the DB2 EXPORT utility to load target tables from a DB2 source that is at Version 9.7 or newer, and the user ID that starts the Q Apply program does not have BINDADD authority, you must perform the following bind before Q Apply starts:

```
db2 bind @db2ubind.lst CONCURRENTACCESSRESOLUTION WAIT FOR OUTCOME COLLECTION ASN
```

If you use the EXPORT utility, the Q Apply program requires a password file to connect to the Q Capture server, unless the source and target server are the same. To create the password file, use the **asnpwd** command. The IXF or comma-delimited file is created in the path that is specified by the **apply_path** parameter.

z/OS The Q Apply program uses LOAD from CURSOR on DB2 Version 7.1 for z/OS or later. InfoSphere Replication Server for z/OS, 10.1 works only with DB2 for z/OS Version 8 and later.

If you specify a load utility that is not available, the Q Apply program stops the Q subscription.

Restrictions

The following restrictions apply to the use of utilities for automatic loads:

LOB data
If you are replicating from tables with large object (LOB) data and the servers are remote from each other, EXPORT/LOAD is not a valid load option. If you are using bidirectional or peer-to-peer replication that involves tables with LOB data and remote servers, the IMPORT utility is not a valid option. Use LOAD from CURSOR in both of these situations.

Partitioned database targets with z/OS sources
If you are replicating from a z/OS source to a partitioned database on Linux, UNIX, or Windows, the EXPORT/LOAD and EXPORT/IMPORT utilities are not supported because EXPORT from z/OS only supports the IXF file format while IMPORT/LOAD into partitioned databases only supports the DEL (comma-delimited) file format. For automatic load you must use LOAD from CURSOR in this situation.

Automatic load considerations for z/OS

When a Q Apply program that is running on the z/OS platform performs an automatic load of target tables, you might need to consider setting the NUMTCB parameter and the size of table spaces for multiple simultaneous loads.

Setting the NUMTCB parameter

The Q Apply program uses the LOAD from CURSOR utility to perform automatic loading of target tables on z/OS. To invoke the utility, the Q Apply program calls the DSNUTILS stored procedure that is shipped with DB2 for z/OS.

DSNUTILS must run in a Work Load Manager (WLM) environment. You must set the NUMTCB parameter, which is used to start WLM, as follows:

```
NUMTCB=1
```

For more detail on DSNUTILS, see the DB2 for z/OS *Utility Guide and Reference* for your version.

Table space considerations for parallel loads

On z/OS, if you activate multiple Q subscriptions at the same time and the Q Apply program is performing an automatic load of target tables, the Q Apply program will load the target tables in parallel. In this case, you must ensure that each target table is in a separate table space.

An alternative to putting each target table in a separate table space is to start each Q subscription sequentially so that the load for one Q subscription finishes before the load for the next Q subscription begins.

To avoid a parallel load:
1. Start the first Q subscription.
2. Wait for the Q subscription state to change to A (active).

   You can verify the Q subscription state by using the Manage Q Subscriptions window in the Replication Center, or looking at the STATE column of the IBMQREP_TARGETS control table.
3. Activate the next Q subscription.

Specifying nicknames for the automatic load option for Q Replication

Some Q subscriptions that use the LOAD from CURSOR utility to load target tables require nicknames. The nickname is defined on the Q Apply server to refer to the source table on the Q Capture server.

Before you begin

- The Q Apply server must be a federated server.
- If you want the Replication Center or ASNCLP program to create nicknames, you must create a server definition, wrapper, and user mapping.

About this task

When you create Q subscriptions, you can have the Replication Center or ASNCLP program create nicknames, or you can specify existing nicknames.

Procedure

Ensuring that nicknames used for load have correct concurrent access setting

If you use the LOAD from CURSOR utility to load target tables from a DB2 Version 9.7 or newer source and you manually create the nickname that is used for loading, you need to ensure that the nickname uses the correct federated server option for concurrent access.

About this task

Starting with Version 9.7, a new federated server option, CONCURRENT_ACCESS_RESOLUTION=W, is used to ensure that LOAD from CURSOR waits until all in-progress transactions that modify the source table are completed before beginning the load. This behavior is known as "wait for outcome." The change was required to account for the default currently committed access behavior in DB2 for Linux, UNIX, and Windows Version 9.7 and newer.

Note the following considerations:
- If you let the Replication Center or ASNCLP create the nickname that is used for LOAD from CURSOR, the server option CONCURRENT_ACCESS_RESOLUTION=W is added to the server for that nickname. In some situations you might need to manually create the nickname,

or the nickname that is used for loading might be shared by other applications. In these situations, you must set CONCURRENT_ACCESS_RESOLUTION=W manually for the nickname.

- The procedures for setting concurrent access are different if the Q Apply server is at Version 9.7 or newer, or pre-Version 9.7.
- z/OS There is currently no solution to enforce wait for outcome behavior when Q Apply on z/OS uses LOAD from CURSOR on a DB2 V9.7 source database on Linux, UNIX, or Windows to perform the load. In this case, the best solution is to suspend any applications that update the source table from the time the Q subscription is started until the load phase begins (identified by Q subscription state change to L or A in the IBMQREP_SUBS table).

Note: You can only set this option for a registered server of type DB2/UDB Version 9.7 or newer.

Procedure

To ensure that nicknames that are used for load have correct concurrent access setting, use one of the following procedures depending on whether the Q Apply program is at Version 9.7 or newer, or older than Version 9.7:

Version of Q Apply program	Procedure

Version 9.7 or newer	Issue the following command at the Q Apply server:
	`db2 alter server` *`server_name`*
	`OPTIONS(ADD CONCURRENT_ACCESS_RESOLUTION 'W');`

Pre-Version 9.7	**Note:** If you are unable to follow this procedure, suspend any applications that update the source table during the beginning of the load.
	1. From the Q Apply server, connect to the source database.
	2. Bind the SQL packages that are used for Call Level Interface (CLI) connections with a generic bind option in a specific package by using the following command:
	`db2 bind @db2cli.lst generic`
	`"CONCURRENTACCESSRESOLUTION WAIT_FOR_OUTCOME"`
	`COLLECTION ASN`
	3. Add the following name-value pair to the `db2cli.ini` file at the federated database, below the stanza that declares the options for the server definition to which the nickname belongs:
	`[`*`data_source_name`*`]`
	`CURRENTPACKAGESET=ASN`
	Where *data_source_name* is the source database that the `db2cli.bnd` packages were bound against.

Recommendation: If you use a federated server for both replication and other purposes, create a new dedicated server for use by replication that has the CONCURRENT_ACCESS_RESOLUTION=W option set, and allow other applications to use the existing server name.

Manually loading a target table

When you specify a *manual load* for a Q subscription, you load the target table using a utility of your choice, and then notify the Q Capture program when the table is loaded.

Before you begin

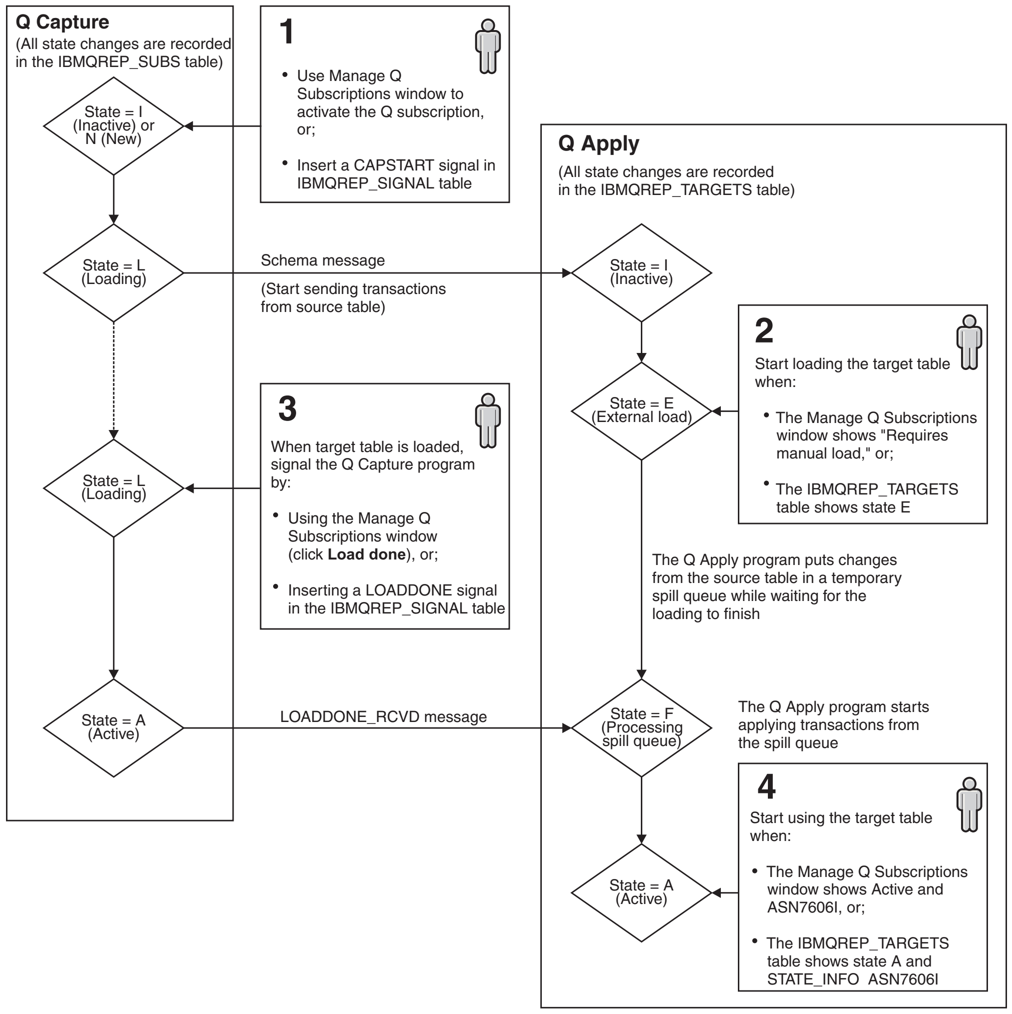
- Ensure that the HAS_LOADPHASE column in the IBMQREP_SUBS table has a value of E to indicate that a manual load will be performed.
- Ensure that no applications are updating the target table that you plan to load. Data in the tables will be inconsistent while the replication programs synchronize the source and target tables after the loading process. The Q Apply program drops referential integrity constraints until the target table and any other related target tables are loaded.

About this task

While the target table is being loaded, the Q Capture program continues to send transactions from the source table. The Q Apply program puts these transactions in a temporary spill queue, and applies them after the load is complete. The Q Apply program waits until any dependent Q subscriptions have completed their load phase before putting referential integrity constraints back on the target table.

Recommendation: Load target tables during a time of relative inactivity at the source.

Figure 24 on page 185 illustrates the stages of the manual loading process.



Q Capture
(All state changes are recorded in the IBMQREP_SUBS table)
State = I (Inactive) or N (New)
State = L (Loading)
State = L (Loading)
State = A (Active)
1
• Use Manage Q Subscriptions window to activate the Q subscription, or;
• Insert a CAPSTART signal in IBMQREP_SIGNAL table
Schema message
(Start sending transactions from source table)
3
When target table is loaded, signal the Q Capture program by:
• Using the Manage Q Subscriptions window (click Load done), or;
• Inserting a LOADDONE signal in the IBMQREP_SIGNAL table
LOADDONE_RCVD message
Q Apply
(All state changes are recorded in the IBMQREP_TARGETS table)
State = I (Inactive)
State = E (External load)
2
Start loading the target table when:
• The Manage Q Subscriptions window shows "Requires manual load," or;
• The IBMQREP_TARGETS table shows state E
The Q Apply program puts changes from the source table in a temporary spill queue while waiting for the loading to finish
State = F (Processing spill queue)
The Q Apply program starts applying transactions from the spill queue
4
Start using the target table when:
• The Manage Q Subscriptions window shows Active and ASN7606I, or;
• The IBMQREP_TARGETS table shows state A and STATE_INFO ASN7606I
State = A (Active)

Figure 24. Stages of the manual loading process

Procedure

To manually load a target table:

1. Start the Q subscription by using either the Replication Center or the **START QSUB** command. If you chose automatic start when you created the Q subscription, it will be started when the Q Capture program is reinitialized.
2. Verify that the Q Apply program is waiting for the target table to be loaded. Use one of the following methods:

Method	Description
Replication Center	Use the Manage Q Subscriptions window. To open the window, right-click the Q Capture server where the source table for the Q subscription is located and select **Manage** > **Q Subscriptions**. Locate the Q subscription on the window and verify that its state is "Requires manual load."
SQL	Issue a SELECT statement for the IBMQREP_TARGETS table at the Q Apply server and verify that the value in the STATE column is E.

3. Load the target table by using your chosen utility.

4. Notify the Q Capture program when the load is complete. Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the LOAD DONE command. For example, the following commands set the environment and generate the SQL signal to inform a Q Capture program at the SAMPLE database that loading is complete for the EMPLOYEE0001 Q subscription:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`LOAD DONE QSUB SUBNAME EMPLOYEE0001;`

Replication Center	In the Manage Q Subscriptions window, select the Q subscription for the target table that you loaded and click **Load done**.

SQL	Insert a LOADDONE signal into the IBMQREP_SIGNAL table at the Q Capture server, as follows:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'LOADDONE',`
	`    'subname',`
	`    'P');`
	
	Where *schema* identifies the Q Capture program that you want to signal, and *subname* is the name of the Q subscription for which you are performing a manual load.

5. Verify that applications can safely use target tables again by checking to see if the Q Apply program makes the following changes to the IBMQREP_TARGETS table:
   - Sets the STATE column to A (active).
   - Sets the STATE_INFO column to ASN7606I, which means that any referential integrity constraints on the target table are restored.

   You can use the Manage Q Subscriptions window in the Replication Center to verify that these two changes have been made. Look for Active and ASN7606I in the row for the Q subscription.

No load option for Q Replication

The no load option is appropriate when the source and target tables are synchronized before any Q subscriptions become active and replication begins.

When you specify the no load option for a Q subscription, the Q Apply program begins applying transactions to a target table as soon as the Q subscription becomes active.

If you choose the no load option, make sure that the values of the primary key or unique index from the source table are also present in the primary key or unique index of the target table.

You might specify no load when adding a large number of new tables during a period of relative inactivity in the source and target databases or subsystems. After you quiesce the source tables, you load the target tables, and then activate the Q subscriptions.

You might also specify a no load option if you back up a source database and then restore the database on the target server.

Load options for different types of Q Replication

Options for loading target tables depend on the type of Q Replication that you are setting up.

The following sections explain what load options are available for each type of Q Replication, which server is used for the initial load, and the steps that you must take to begin the load:
- "Load options for unidirectional replication"
- "Load options for bidirectional and peer-to-peer replication with two servers" on page 188
- "Load options for peer-to-peer replication with three or more servers" on page 189

Load options for unidirectional replication

This section explains the load options for unidirectional replication.

Load options
The available options depend on the target type:

DB2 targets
All load options are available.

CCD targets
If the CCD table is noncomplete, the only valid choice is no load. If the CCD table is complete, all load options are available.

Non-DB2 targets
For automatic loads, the EXPORT and IMPORT utilities are supported.You can also specify that Q Apply use an ODBC select to fetch data from the source, and for Oracle targets, you can specify that Q Apply call the SQL*Loader utility.

Which server is used for the initial load
Q Capture server

What you must do
By default, the Q Apply program begins the loading process when you start the Q Capture program for the Q subscription's source table. If you create the Q subscription so that it is not started automatically when the Q Capture program starts, then you must start the Q subscription for the load to begin.

Example

Assume that you want to replicate data in one direction from the DEPARTMENT table at Server A to the DEPARTMENT table at Server B and use the most automatic method. You want the Q Apply program to handle the load and use the best available utility.

1. You create a Q subscription for the DEPARTMENT table that specifies an automatic load that uses the best available utility.
2. You start the Q Capture program at Server A and the Q Apply program at Server B.

   The Q Apply program calls a load utility that copies the data from the DEPARTMENT table at Server A to the DEPARTMENT table at Server B. Once the loading process is finished, replication begins in one direction from Server A to Server B.

Load options for bidirectional and peer-to-peer replication with two servers

This section explains the load options for bidirectional and peer-to-peer replication with two servers.

Load options
All load options are available. However, if you specify an automatic load, by default the Q Apply program will choose between a combination of the EXPORT and LOAD utilities and a combination of the EXPORT and IMPORT utilities, depending on your operating system and version. You can override this behavior and instruct the Q Apply program to use the LOAD from CURSOR utility by opening the Q Subscription Properties notebook for individual Q subscriptions.

Which server is used for the initial load
When you create the two Q subscriptions for bidirectional or peer-to-peer replication with two servers, you choose which server will be the initial load source. This server contains the table whose data you want to copy to a table on the other server.

For subsequent loads (for example, if you stop the Q subscriptions for the logical table and then start them), you specify which server will be the load source when you decide which of the two Q subscriptions to start. The source table for the Q subscription that you start will be the load source.

What you must do
The process of initiating a load differs depending on whether you specify an automatic or manual load:

Automatic load
If you created the Q subscriptions to start automatically when the Q Capture program starts, you only need to start the Q Capture and Q Apply programs at both servers for the loading process to begin.

If you chose not to have the Q subscriptions start automatically, you must take the following actions:

- Start the Q Capture and Q Apply programs at both servers.
- Start the Q subscription whose source table you specified as the load source.

Manual load

1. Start the Q Capture and Q Apply programs at both servers.
2. Start the Q subscription whose source table you want to be the load source.

   The Q subscription will go into load pending state.

3. Load the target table for the Q subscription, using any method.
4. When you are done with the load, tell the Replication Center that the load is finished or insert a LOADDONE signal into the IBMQREP_SIGNAL table at the source server for the Q subscription.

Example

Assume that you wanted to replicate the EMPLOYEE table in a bidirectional setup on Server A and Server B, and use the most automatic method. You want Server A to be the initial load source:

1. You create two Q subscriptions, EMP_A2B and EMP_B2A. When you create EMP_A2B, you specify Server A as the initial load source and specify an automatic load in which the Q Apply program chooses the best available load utility.
2. You initiate the load by starting the Q Capture and Q Apply programs at Server A and Server B.

   The Q Apply program at Server B initiates the load for EMP_A2B by calling a load utility to copy the data from the EMPLOYEE table at Server A to the EMPLOYEE table at Server B. When the loading completes, replication begins in both directions between Server A and Server B.

Load options for peer-to-peer replication with three or more servers

This section explains the load options for peer-to-peer replication with three or more servers.

Load options

All load options are available. However, if you specify an automatic load, by default the Q Apply program chooses between a combination of the EXPORT and LOAD utilities, and a combination of the EXPORT and IMPORT utilities, depending on your operating system and version. You can override this behavior and instruct the Q Apply program to use the LOAD from CURSOR utility by opening the Q Subscription Properties notebook for individual Q subscriptions.

Which server is used for the initial load

In a peer-to-peer group with three or more servers, you start replication in stages. First you start replication between two servers, and then you bring additional servers into the group by starting replication between an active server and a new server. Follow these guidelines:

- When you start replication between the first two servers, choose one server as the load source. Start the Q subscription that specifies this server as its source. The Q Apply program at the second server begins the loading process for the table at the second server.
- To add a new server, choose one of the active servers as the load source. Start the Q subscription that specifies this server as its source and the new server as its target. The Q Apply program at the new server begins the loading process for the table at the new server.

What you must do

In a peer-to-peer configuration with three or more servers, you cannot create Q subscriptions that start automatically. You must manually start the Q subscriptions in stages. Follow these steps:

1. Start the Q Capture and Q Apply programs at the first two servers in the group.
2. Start one of the two Q subscriptions between the servers. The source table for the Q subscription that you start will be the load source, and the target table will be loaded.
3. Start the Q Capture and Q Apply programs at a new server.
4. Start a Q subscription that specifies one of the active servers as its source, and the new server as its target. The source table for the Q subscription that you start will be the load source, and the table at the new server will be loaded.
5. Follow Steps 3 and 4 until all the servers in the group are loaded.

Manual load: If you choose a manual load, you must load the target table after you start each Q subscription, and then notify the replication programs when the target table is loaded.

Example

Assume that you want to initiate the loading process for a peer-to-peer Q subscription group that includes Server A, Server B, and Server C, with a single logical table, the DEPARTMENT table. You want the Q Apply program to handle the loading and use the best available load utility. You will use Server A as the load source for the tables at both Server B and Server C.

1. You create six Q subscriptions, DEP_A2B, DEP_B2A, DEP_A2C, DEP_C2A, DEP_B2C, and DEP_C2B, all specifying an automatic load using the best available utility.
2. You start the Q Capture and Q Apply programs at Server A and Server B.
3. You start the Q subscription DEP_A2B.

   The Q Apply program at Server B calls a utility to load the DEPARTMENT table at Server B with data from the DEPARTMENT table at Server A. When the loading completes, replication begins in both directions between Server A and Server B.
4. To begin the load at Server C, you first start the Q Capture and Q Apply programs at Server C.
5. Next, you start the Q subscription DEP_A2C.

   The Q Apply program at Server C calls a utility to load the DEPARTMENT table at Server C with data from the DEPARTMENTS table at Server A. When the loading completes, replication begins in all directions between all three servers.

Replicating load operations at the source table

You can specify that load operations at the source table that use the DB2 LOAD utility are replicated to the target table.

About this task

By default, when the Q Capture program reads a log record that indicates the source table was successfully loaded, it issues a warning message. You can change this default behavior when you create a Q subscription for a source table by specifying that Q Capture replicate some types of load operations at the source table.

When this function is enabled, Q Capture stops and starts the Q subscription for the source table, prompting a load of the target table if one is specified for the Q subscription and based on the load options that were set for the Q subscription.

When Q Capture detects the following DB2 operations, its subsequent actions are based on whether you enabled replication of source table loads:

z/OS

- LOAD SHRLEVEL NONE RESUME YES
- LOAD SHRLEVEL NONE REPLACE
- REORG TABLESPACE DISCARD

Linux UNIX Windows

- LOAD REPLACE
- LOAD INSERT

The following options for replication of source table loads are stored in the CAPTURE_LOAD column of the IBMQREP_SUBS table:

R (restart)
The Q subscription is restarted and the target table is loaded with the data from the source table. The type of load is determined by the LOAD_TYPE value in the IBMQREP_TARGETS control table.

W (warning)
Only a warning message is issued. The Q subscription is not restarted.

Restrictions

- z/OS The source database in DB2 for z/OS Version 9 must be at DB2 APAR PK78558 or newer and the Q Capture program must be at the equivalent of Version 9.7 or newer. Also, this function is not supported if the source or target table share a table space with other tables. CAPTURE_LOAD is a table-space-level option.
- Linux UNIX Windows The source database in DB2 for Linux, UNIX, and Windows and the Q Capture program must be at Version 9.7 or newer.
- Replication of source table loads is not supported for peer-to-peer replication with three or more servers.
- When you load a table that is involved in bidirectional replication, you must let the replication of that load operation complete and the Q subscription state change to active (A) before you start another load operation for the matching table at the other server. Otherwise, the two load operations conflict and might lead to unexpected behavior.
- Loading a source table while any Q subscriptions for the table are still in loading state (for example, performing two consecutive load operations) causes the Q subscriptions for the table to stop. Always wait until all Q subscriptions for the source table are in active (A) state before performing another load operation.

Procedure

To replicate load operations at the source table, use one of the following methods:

Method	Description
ASNCLP command-line program	In the CREATE QSUB command, specify the CAPTURE_LOAD keyword with the R option, as in the following example: `CREATE QSUB USING REPLQMAP SAMPLE_ASN_TO_TARGET_ASN (SUBNAME EMPLOYEE0001 EMPLOYEE OPTIONS HAS LOAD PHASE I CAPTURE_LOAD R TARGET NAME TGTEMPLOYEE LOAD TYPE 2);`
Replication Center	Select the **Reload the target table if the source table is reloaded** checkbox on the Loading the Target Tables page of the Create Q Subscriptions wizard. The same control can also be found on the **Load** tab of the Q Subscription Properties notebook.

How constraints on the source table affect replication of load operations

Replication of load operations at the source table requires special consideration when constraints are defined on the source table.

After the source table is loaded, READ access that is required by the DB2 LOAD utility to load the data into the target table might be restricted:

z/OS

On z/OS, access might be restricted if constraints are defined on the source table and the LOAD utility is invoked with the REPLACE or RESUME ENFORCE NO option. This situation prompts the LOAD utility to put the table into CHECK PENDING state, and Q Apply stops the Q subscription. To correct the problem you must run CHECK DATA on the source table to make the table space accessible, and then start the Q subscription.

Linux UNIX Windows

On Linux, UNIX, and Windows, access might be restricted if constraints are defined on the source table and the LOAD utility is invoked with the REPLACE or INSERT NO ACCESS option. This situation prompts the LOAD utility to put the table into SET INTEGRITY CHECK PENDING state, and Q Apply stops the Q subscription. To correct the problem, you must run SET INTEGRITY on the source table to make the table space accessible, and then start the Q subscription.

Attention: The INSERT ALLOW READ ACCESS option of the LOAD utility is not recommended if you set the option to replication source table loads. If the utility is invoked with this option and constraints are defined on the source table, the table is put into SET INTEGRITY PENDING state and READ ACCESS state. In these two states, the newly loaded data is not accessible. If you expect to use this option, use one of the following methods to prevent the source and target tables from losing synchronization:

- Before the load commences, stop the Q subscription, load the source table, and then start the Q subscription. This procedure triggers a complete new load of the target table (full refresh).
- Set the CAPTURE_LOAD option of the Q subscription to W and load the source table while the Q subscription is active. None of the newly loaded data is replicated, and you can then do one of the following things:
  - Use the asntdiff and asntrep commands to find and resolve differences between the source and target tables.
  - Stop and start the Q subscription, triggering an automatic load of the target.

Loading target tables in a data consolidation scenario

When multiple source tables are replicated to a single target table in Q Replication, you use a special procedure to ensure that all of the data from multiple loads is preserved in the target table.

About this task

This procedure allows you to incrementally load the target from multiple sources and ensure that none of the loads overwrite data from previous loads.

When you create Q subscriptions for the source tables, you specify a load type that uses the REPLACE option for one Q subscription, and a load type that uses the RESUME option (z/OS) or INSERT option (Linux, UNIX, and Windows) for all of the other Q subscriptions. Then you start the Q subscription with the REPLACE load type first, and start the other Q subscriptions after the first load is finished.

The first load replaces any data in the target table, and the subsequent loads add data from the other source tables without affecting the data from the first load.

Table 21 shows the load types that you should use in combination for this procedure. The load type information is stored in the IBMQREP_TARGETS table.

Table 21. Load type combinations for data consolidation

Load type for first load (with REPLACE option)	Load type for subsequent loads (with RESUME or INSERT option)
1 (LOAD from CURSOR)	101
2 (EXPORT/IMPORT)	102
3 (EXPORT/LOAD)	103

Figure 25 on page 194 shows how the procedure works for one of these combinations, LOAD from CURSOR with the REPLACE option for the first load and LOAD from CURSOR with the RESUME option for the subsequent loads.

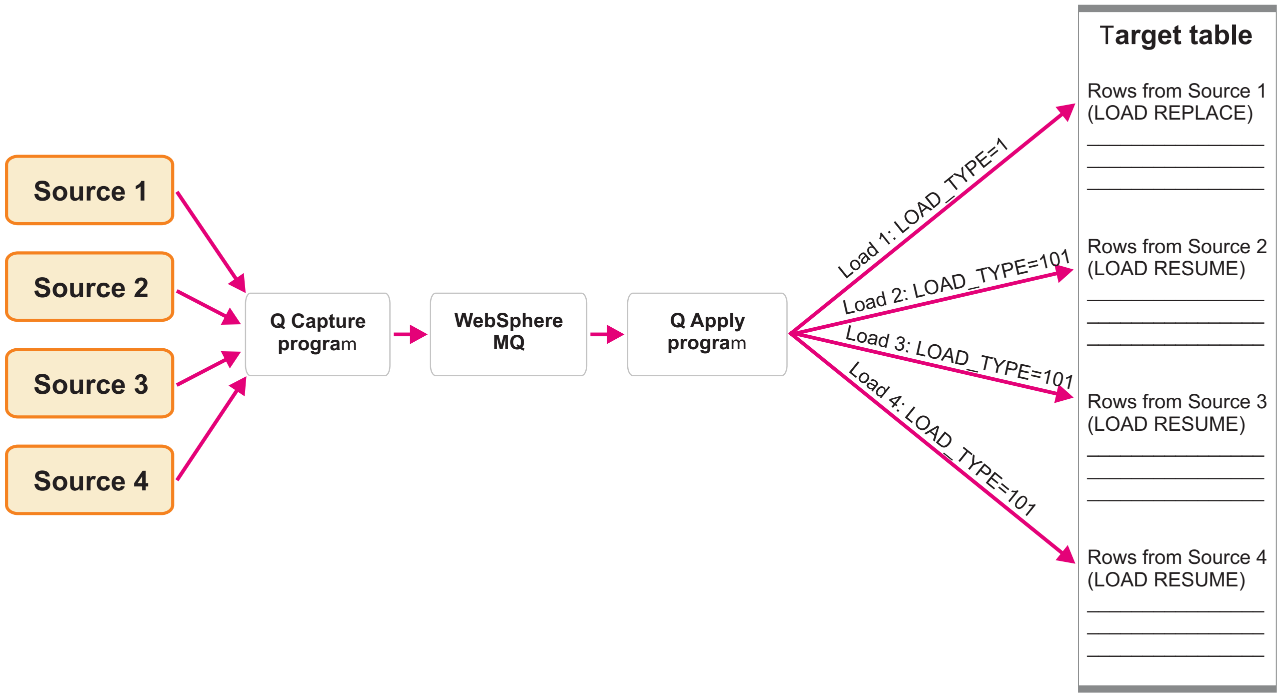


Figure 25. Example of load procedure for a data consolidation scenario that uses LOAD from CURSOR

Procedure

1. When you create Q subscriptions for the consolidated target table, set LOAD_TYPE to 1, 2, or 3 for one Q subscription in the group and LOAD_TYPE to 101, 102, or 103 for all remaining Q subscriptions in the group, following the combinations shown in Table 21 on page 193.

   Note: Specify that the Q subscriptions with LOAD_TYPE of 101, 102, or 103 are created in inactive (I) state. You can do this in the ASNCLP command-line program by using the START AUTOMATICALLY NO keywords. In the Replication Center, on the Loading the Target Tables page of the Create Q Subscriptions wizard, clear the **Start all Q subscriptions automatically** checkbox. The Q subscription with LOAD_TYPE of 1, 2, or 3 starts automatically.

   Also, do not set the conflict action and error action for the Q subscriptions to D (stop the Q subscription).
2. After loading completes for the first Q subscription, start the remaining Q subscriptions.

   The Q Apply program sets the state of the Q subscription to A (active) when a load completes. You can use the Q Replication Dashboard or the Manage Q Subscriptions window in the Replication Center to verify the Q subscription state.

If the target table is empty, or if you manually delete all of the rows, you can set LOAD_TYPE to 101, 102, or 103 for all Q subscriptions in the group and start them one at a time.

If you need to reload the data from only one source table (for example because one Q subscription was in error), manually delete the rows in the target table using a suitable range, set LOAD_TYPE to 101, 102, or 103 for the Q subscription that was in error, and start the Q subscription.

If you need to do a full refresh of the entire target table, follow the procedure above.

Q Replication and Event Publishing for multiple database partitions

Q Replication and Event Publishing support capture of data from DB2 source tables that are spread across multiple database partitions.

When you create Q Capture control tables in a multiple partitioned database, all of the table spaces used by those control tables must be on the catalog partition. If you use an existing table space, the table space must be non-partitioned and on the catalog partition.

The Q Capture program keeps a list of database partitions within the restart message. Whenever the Q Capture program is started in warm mode, it reads the list of database partitions from the restart message. Q Capture compares the number of database partitions that are known to DB2 with the number of database partitions that are listed in the restart message. If the numbers do not match, the Q Capture program stops.

If you added one or more database partitions since the last time you ran the Q Capture program, you must tell the Q Capture program about the new partitions. You can do this by starting Q Capture with the **add_partition** parameter.

For example, the following command specifies that the Q Capture program on server SAMPLE should start reading the log file for newly added partitions:

```
asnqcap capture_server=sample capture_schema=asn1 add_partition=y
```

Replication of DB2 partitioned tables: Version 9.7 Fix Pack 1 or earlier (Linux, UNIX, Windows)

Q Replication supports DB2 tables that are partitioned by range (using the PARTITION BY clause of the CREATE TABLE statement). These tables are sometimes known as range-partitioned tables.

Version and fix pack requirements exist for the Q Capture program if a source table is partitioned by range. This topic covers these requirements and support for replication of partitioned tables when your replication programs are at Version 9.7 Fix Pack 1 or earlier. If only target tables (no source tables) are partitioned by range, then Q Replication has no version or fix pack requirements specific to these tables.

To capture changed data for range-partitioned tables, your Q Capture program must be at Version 9.7 or later. It can capture changes from range-partitioned tables on DB2 V9.1, V9.5, or V9.7. However, restrictions exist for range-partitioned tables earlier than V9.7. The restrictions are also discussed in this topic.

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table.

Replication and publishing treat all data partitions of a source table as a single table. For example, when you create a Q subscription or publication that specifies a partitioned table, you specify the entire table rather than one or more data partitions of the table. All row operations for the table, regardless of the data partition at which they occur, are replicated or published.

You can perform several alterations on a partitioned table, including adding a data partition, attaching a data partition, or detaching a data partition. These ALTER operations on the source table are not replicated to the target. You must alter the target table independently of the source table if you want to maintain an identical partitioning scheme.

Replication and publishing treat these ALTER operations differently:

Note for Version 9.5 and earlier: The Q Capture program does not recognize the addition, attachment, or detachment of a data partition until the program is reinitialized or stopped and restarted.

ADD Adds a new, empty data partition to the source table. If you require the new data partition at the target, you must manually add it. Q Capture program behavior and the procedure that you need to follow depend on the release of your DB2:

Version 9.7 or higher
Add the data partition at the target before adding it at the source. Q Capture automatically begins replicating changes to the data partition.

Version 9.5 or 9.1
Add the data partition at both the source and target before restarting Q Capture. Do not change data in the source data partition until Q Capture is restarted.

ATTACH
Creates a new data partition at the source by using an existing table. The ATTACH operation is not replicated and the data in the new data partition is not replicated to the target. If you require the new data partition at the target you must manually add it. If you require the attached data at the target, you must manually load the data into the target before you attach the data partition at the target.

Note: If the Q Capture program is stopped when a data partition is attached, rows that are inserted into, updated, or deleted from the table before it is attached as a partition are replicated. If Q Capture is running when the data partition is attached, these rows are not replicated.
To ensure consistent behavior, before you attach a table as a new data partition, set the DATA CAPTURE CHANGES clause for the table to OFF if you need to make any changes to the table. For example, the following statements create a table, insert values into the table, and then attach the table as a data partition to an existing partitioned table:

```
db2 create table temp1 like t1;
--  NOTE: data capture changes is off by default
db2 insert into temp1 values (44,44);
-- NOTE: Turn on data capture changes after insert/update/deletes
-- and before attach partition
db2 alter table temp1 data capture changes;
db2 alter table t1 attach partition part4 starting from 41
ending at 50 from temp1;
db2 set integrity for t1 allow write access immediate checked;
```

DETACH
Turns an existing data partition into a separate table. The DETACH operation is not replicated. The data that is deleted from the source table by the DETACH operation is not deleted from the target table. If you need to change the target data partition into a separate table, you need to do so manually.

Note: DB2 logs updates that cause rows to move across data partitions as delete/insert pairs. The Q subscription or publication option to suppress delete operations from the source table (SUPPRESS_DELETES=Y) can cause a single UPDATE operation at the source to appear as two rows at the target. It is recommended that you avoid using the suppress delete option with partitioned source tables.

Replication of DB2 partitioned tables: Version 9.7 Fix Pack 2 and later (Linux, UNIX, Windows)

Q Replication supports DB2 tables that are partitioned by range (using the PARTITION BY clause of the CREATE TABLE statement). These tables are sometimes known as range-partitioned tables.

Version and fix pack requirements exist for the Q Capture program if a source table is partitioned by range. This topic covers these requirements and support for replication of partitioned tables when your replication programs are at Version 9.7 Fix Pack 2 or later. If only target tables (no source tables) are partitioned by range, then Q Replication has no version or fix pack requirements specific to these tables.

To capture changed data for range-partitioned tables, your Q Capture program must be at Version 9.7 or later. It can capture changes from range-partitioned tables on DB2 V9.1, V9.5, or V9.7. However, restrictions exist for range-partitioned tables earlier than V9.7. The restrictions are also discussed in this topic.

Important: If your replication programs are at Version 9.7 Fix Pack 2 or later and you plan to replicate range-partitioned tables, you must run the Version 9.7 Fix Pack 2 migration script, `asnqcapluwv97fp2.sql`. The script adds a new control table, IBMQREP_PART_HIST, to help the replication programs handle data partition changes such as add, attach, or detach. The script is located in the `sqllib/samples/repl/mig97/q/` directory. The Q Capture program does not use the IBMQREP_PART_HIST table for partitioned source tables on DB2 Version 9.5 or Version 9.1.

Partitioned tables use a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table.

Replication and publishing treat all data partitions of a source table as a single table. For example, when you create a Q subscription or publication that specifies a partitioned table, you specify the entire table rather than one or more data partitions of the table. All row operations for the table, regardless of the data partition at which they occur, are replicated or published.

You can perform several alterations on a partitioned table, including adding a data partition, attaching a data partition, or detaching a data partition. These ALTER operations on the source table are not replicated to the target. You must alter the target table independently of the source table if you want to maintain an identical partitioning scheme.

Replication and publishing treat these ALTER operations differently:

Note for Version 9.5 and earlier: The Q Capture program does not recognize the addition, attachment, or detachment of a data partition until the program is reinitialized or stopped and restarted.

ADD Adds a new, empty data partition to the source table. If you require the new data partition at the target, you must manually add it. Q Capture program behavior and the procedure that you need to follow depend on the release of your DB2:

Version 9.7 or higher
Add the data partition at the target before adding it at the source. Q Capture automatically begins replicating changes to the data partition.

Version 9.5 or 9.1
Add the data partition at both the source and target before restarting Q Capture. Do not change data in the source data partition until Q Capture is restarted.

ATTACH
Creates a new data partition at the source by using an existing table. The ATTACH operation is not replicated and the data in the new data partition is not replicated to the target. If you require the new data partition at the target you must manually add it. If you require the attached data at the target, you must manually load the data into the target before you attach the data partition at the target.

DETACH
Turns an existing data partition into a separate table. The DETACH operation is not replicated. The data that is deleted from the source table by the DETACH operation is not deleted from the target table. If you need to change the target data partition into a separate table, you need to do so manually.

Note: DB2 logs updates that cause rows to move across data partitions as delete-insert pairs. The Q subscription or publication option to suppress delete operations from the source table (SUPPRESS_DELETES=Y) can cause a single UPDATE operation at the source to appear as two rows at the target. It is recommended that you avoid using the suppress delete option with partitioned source tables.

CCD tables in Q Replication

Consistent-change-data (CCD) tables provide committed transactional data that can be read and used by other applications, for example InfoSphere DataStage® or the Apply program for SQL replication.

By using a CCD table as your target type, you can also keep a history of source changes. For example, you can track before and after comparisons of the data, when changes occurred, and which user ID updated the source table.

You can specify CCD tables as both sources and targets in unidirectional Q Replication. You can also specify a CCD table that is populated by the Q Replication programs as a source for SQL replication.

The following sections provide more detail about CCD tables in Q Replication:
- "Condensed and complete attributes"
- "Load options for CCD tables" on page 200
- "Options for errors or conflicts" on page 200
- "Default columns in CCD tables" on page 200
- "Optional auditing columns" on page 201
- "ASNCLP sample for creating CCD target table" on page 202
- "Using CCD tables as Q Replication sources" on page 202

Important for Version 10.1 on Linux, UNIX, and Windows: If the source database is DB2 10.1 for Linux, UNIX, and Windows with multiple DB2 pureScale® members, Q Replication does not support CCD target tables. With Version 10.1 as a source, CCD targets are only supported in single-member databases with the Q Capture **`compatibility`** parameter set to 0907 or lower.

Condensed and complete attributes

Two attributes define a CCD table: condensed and complete. The following list summarizes these attributes:

Complete (COMPLETE=Y)
A complete CCD table contains every row of interest from the source table and is initialized with a full set of source data.

All target table loading options are valid for complete CCDs (automatic, manual, or no load).

Noncomplete (COMPLETE=N)
A noncomplete CCD table contains only changes to the source table and starts with no data.

The only valid load option for noncomplete CCD tables is no load.

Condensed (CONDENSED=Y)
A condensed CCD table contains one row for every key value in the source table and contains only the latest value for the row.

For condensed CCD tables, a primary key is required to ensure there are no duplicate rows. In case of an update conflict, all the source columns will be forced into the row. The required settings for conflict rule and conflict action are CONFLICT_RULE=K and CONFLICT_ACTION=F.

Noncondensed (CONDENSED=N)
A noncondensed CCD table contains multiple rows with the same key value, one row for every UPDATE, INSERT, or DELETE operation at the source table.

When added to the CCD table, all of the rows become INSERT operations. Noncondensed CCD tables cannot have a unique index or primary key.

A CCD table that is used for keeping a history of changes to the source table is complete (COMPLETE=Y) and noncondensed (CONDENSED=N).

Load options for CCD tables

The following load options apply to CCD target tables:

Complete
You can specify an automatic load by the Q Apply program, a manual load, or no load.

Noncomplete
You must specify no load.

Options for errors or conflicts

The following options are available for handling unexpected conditions in CCD target tables:

Condensed and complete
Two choices are valid for conflict detection:

- Force the source change into the target table (CONFLICT_ACTION=F).
- Ignore the condition and continue (CONFLICT_ACTION=I).

For all CCD table types, the only valid conflict option is to check only key columns (CONFLICT_RULE=K).

Notes:

1. For noncomplete CCD targets, the IBMQREP_EXCEPTIONS table is not updated after "row not found" or duplicate row conflicts because rows from the source table might not exist in a noncomplete CCD target and rework might occur frequently.
2. For complete CCD targets, the IBMQREP_EXCEPTIONS table is updated after conflicts with one exception: duplicate row violations are not logged because rows from delete operations remain in CCD targets and rework might occur frequently.
3. For condensed CCD tables, a key update at the source table might result in the removal of a row at the CCD target. The removed row is typically a "delete" row (one with a value of D in the IBMSNAP_OPERATION column). This type of row is removed only if it has the same key value as the new value of the key update.
4. For condensed CCD tables, CONFLICT_ACTION=I implies that conflicts at the target are ignored, with one exception: If a key update at the source table results in a rework at the CCD target, a rework is necessary to keep the CCD data consistent.

Default columns in CCD tables

By definition, a CCD table always includes the following columns in addition to the replicated columns from the source table. These columns contain information from the recovery log at the source database that provide more details about each row change:

Column	Description
IBMSNAP_INTENTSEQ	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No A sequence number that uniquely identifies a change. This value is ascending in a transaction. z/OS The log sequence number (LRSN or RBA) of each update, delete, and insert.
IBMSNAP_OPERATION	**Data type:** CHAR(1); **Nullable:** No A flag that indicates the type of operation: I (INSERT), U (UPDATE), or D (DELETE).
IBMSNAP_COMMITSEQ	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No A sequence number for each row within a transaction. z/OS The log sequence number (LRSN or RBA) of the source commit record.
IBMSNAP_LOGMARKER	**Data type:** TIMESTAMP; **Nullable:** No The approximate time at the source server that the data was committed, measured in Greenwich Mean Time (GMT). You can use the Q Apply **`logmarkertz`** parameter to specify that the timestamp uses the local time of the Q Apply server. Specifying **`logmarkertz`**=local is only useful when the Q Capture and Q Apply servers are in the same time zone with the same Daylight Savings Time or other time change in effect.

Optional auditing columns

When you create a noncomplete (COMPLETE=N) CCD table with the ASNCLP command-line program or Replication Center, you can specify additional auditing columns. The following table describes these columns:

Column	Description
IBMSNAP_AUTHID	**Data type:** VARCHAR(30); **Nullable:** Yes The user ID that updated the source table. z/OS This column is the primary authorization ID.

Column	Description
IBMSNAP_AUTHTKN	**Data type:** VARCHAR(30); **Nullable:** Yes The authorization token that is associated with the transaction. z/OS The correlation ID (normally a job name) that ran the source update.
IBMSNAP_PLANID	**Data type:** VARCHAR(8); **Nullable:** Yes z/OS The plan name that is associated with the transaction. This column will be null for DB2 for Linux, UNIX, and Windows.
IBMSNAP_UOWID	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** Yes The unit-of-work (UOW) identifier from the log record for a row. z/OS The unit-of-work identifier, sometimes called the unit-of-recovery ID (URID) of the transaction.

ASNCLP sample for creating CCD target table

You can use the ASNCLP command-line program or Replication Center to create CCD tables as part of the process of creating a Q subscription for the source table. The following ASNCLP sample sets the environment and then creates a Q subscription for the EMPLOYEE table in the SAMPLE database, also creating a CCD target table:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET SERVER CAPTURE TO DB SAMPLE;
SET SERVER TARGET TO DB TARGET;
CREATE QSUB USING REPLQMAP REPQMAP1
(SUBNAME SUB1 EMPLOYEE TYPE CCD);
```

Using CCD tables as Q Replication sources

You can use a CCD target table that is populated by the Q Apply program as a source table for another Q Replication target database. Typically, this configuration uses three tiers, where the log record values from the source database at Tier 1 are sent to the Q Apply program at Tier 2 and applied to a target CCD table. The CCD table at Tier 2 is then mapped to a CCD table at Tier 3, and the original values from Tier 1 are then propagated to Tier 3.

To set up a three-tier configuration like this, use the ASNCLP program to create a Q subscription from Tier 1 to Tier 2. In the CREATE QSUB command, use the TYPE CCD keywords to specify that the target CCD table at Tier 2 will be populated with values taken directly from the recovery log at Tier 1. Then, when you create the Q subscription from Tier 2 to Tier 3, replace the TYPE CCD keywords in the CREATE QSUB command with TYPE USERTABLE so that the CCD columns at Tier 2 are mapped to the matching columns at Tier 3.

The following examples of CREATE QSUB commands illustrate this method. The first example uses the TYPE CCD keywords to create a Q subscription from Tier 1 to Tier 2:

```
CREATE QSUB USING TIER1_ASN_TO_TIER2_ASN
(SUBNAME EMPLOYEE0001 HR.EMPLOYEE
 TARGET NAME HR.TIER2EMPLOYEE
 TYPE CCD
 KEYS (C1));
```

The second example omits the TYPE CCD keywords and instead uses TYPE USERTABLE. This syntax prompts the Q Apply program to create a table with identical columns at Tier 3 and then propagate the values in the Tier 2 CCD table to matching columns at Tier 3:

```
CREATE QSUB USING TIER2_ASN_TO_TIER3_ASN
(SUBNAME EMPLOYEE0001 HR.TIER2EMPLOYEE
 TARGET NAME HR.TIER3EMPLOYEE
 TYPE USERTABLE
 KEYS (C1));
```

Using a Q Replication CCD table as a source requires Version 9.7 Fix Pack 5 or later on Linux, UNIX, and Windows and Version 10.1 on z/OS with the PTF that corresponds to Fix Pack 5.

Chapter 12. Creating publications

With event publishing, you can publish changed rows or transactions from a source table to a user application. The Q Capture program publishes changes from a source table and puts those changes on a send queue. You are then responsible for having an application of your choice retrieve those messages.

Tip: The asnqwxml sample program provides an example of a Web-based application that consumes XML messages that the Q Capture program publishes. The sample demonstrates how to use publications in a business scenario.

Grouping publishing queue maps and publications

Before you define publications and publishing queue maps, you should first plan how you want to group them.

Each publication identifies a single source table from which changes will be published in an XML format or delimited format such as comma-separated values (CSV). When you define a publication, you must also define which publishing queue map is used to transport the data for that source table. Among other things, each publishing queue map identifies the WebSphere MQ queue that the Q Capture program sends changes to. A single publishing queue map can be used to transport data for several publications, so you must decide which publications use the same publishing queue map to transport data.

When you plan how to group publications and publishing queue maps, keep in mind the following rules:
- A WebSphere MQ queue cannot be shared by multiple Q Capture programs.
- A single Q Capture program can write to multiple send queues.
- You can create one or multiple publishing queue maps from a single Q Capture program.

How the Q Capture program works with the send queue

For a publishing queue map, the Q Capture program captures changes from the database log for all tables for which there are active publications. The Q Capture program stores these changes in memory until it reads the corresponding commit or abort record from the database log. The Q Capture program then sends information on committed transactions to all WebSphere MQ send queues that were defined for the publications.

Suggestions for grouping similar publications with publishing queue maps

For tables that are involved in transactions with one or more applications, you should create publications for these tables so that they all share a common publishing queue map. Grouping similar publications with the same publishing queue map assures the transactional consistency of the data that is sent to the send queue.

It is important to have publications that have dependencies share the same publishing queue map. If you define publications that are involved in related transactions to send data through independent publishing queue maps, then the Q Capture program splits the data between the multiple send queues.

If multiple applications update the source server but do not update the same tables, and you configure a single Q Capture program to publish data from the source server to a target server, then you might consider defining multiple publishing queue maps for this Q Capture program to use. All of the publications that are associated in transactions for each application are then published over one of these publishing queue maps. Such a configuration could provide advantages, such as failure isolation or increased throughput. You can gain higher throughput and failure isolation by configuring each Q Capture program with its own publishing queue map. However, you must balance these gains against increased CPU consumption and a more complex publishing environment.

Creating publishing queue maps

When you create publications, you specify which WebSphere MQ queue to send the data to by associating each publication with a publishing queue map. You can create a publishing queue map before you begin creating publications or as one of the steps while you are creating publications.

Before you begin

- Plan how you want to group publishing queue maps and publications.
- On the server that contains the source tables for the publications, create the control tables for the Q Capture program.
- Ensure that you have defined the appropriate objects in WebSphere MQ.

Restrictions

The same send queue cannot be used for both Q replication and event publishing because a send queue can transport compact messages (for Q replication) or XML or delimited messages (for event publishing), but not both.

Procedure

To create a publishing queue map, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the CREATE PUBQMAP command. For example, the following commands set the environment and create a publishing queue map SAMPLE_EP1_TO_SUBSCRIBER:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	`SET RUN SCRIPT LATER;`
	
	`CREATE PUBQMAP SAMPLE_EP1_TO_SUBSCRIBER`
	`USING SENDQ "EP1.QM1.PUBDATAQ"`
	`MESSAGE CONTENT TYPE R`
	`MAX MESSAGE SIZE 128`
	`HEARTBEAT INTERVAL 5;`

Replication Center	Use the Create Publishing Queue Map window. To open the window, expand the Q Capture schema that identifies the Q Capture program that uses the queue map. Right-click the **Publishing Queue Maps** folder and select **Create**.

Tip: You can use either replication administration tool to validate the send queue that you specify for a publishing queue map. Click **Validate queue** on the Create Publishing Queue Map window or use the **`VALIDATE WSMQ ENVIRONMENT FOR`** command in the ASNCLP.

When you create a publishing queue map, you specify the following options:

Send queue
The WebSphere MQ queue where the Q Capture program sends source data and informational messages.

Message content
You can specify for the Q Capture program to send messages that contain either of the following types of content:

- Individual row operations. (This type of message from the Q Capture program is called a *row operation message*.)
- Full transactions. (This type of message from the Q Capture program is called a *transaction message*.)

For either type of message content, the operation is not sent until the transaction that it is part of has committed. The type of message content that you choose determines how the Q Capture program sends data for all publications that use this publishing queue map.

For LOB data types: Regardless of which option that you choose, LOB data types are sent separately as individual physical messages that are associated with either the transaction message or row operation message.

Maximum message length
The maximum size of a message (in kilobytes) that the Q Capture program can put on this send queue. This maximum message length must be equal to or less than the WebSphere MQ maximum message size attribute (MAXMSGL) that is defined for the queue or queue manager.

Queue error action
The actions that the Q Capture program takes when a send queue is no longer accepting messages because of an error, for example when the queue is full:

- Stops running
- Stops putting messages on the queue in error but continues to put messages on other queues

Heartbeat interval
How often, in seconds, that the Q Capture program sends messages on this queue to tell the user application that the Q Capture program is still running when there are no changes to publish. The heartbeat message is sent on the first commit interval after the heartbeat interval expires. A value of 0 tells the Q Capture program not to send heartbeat messages.

Note: This heartbeat interval is different from the WebSphere MQ parameter HBINT (heartbeat interval) that you can define for a WebSphere MQ channel.

Code page conversion errors
Whether character data is published when the data causes a code page conversion error. By default, no data is sent for the character field that failed code page conversion for XML messages. For delimited message format, by default none of the data from character columns in the row is sent when any single character field in the row fails code page conversion.

Instead, all the character columns are sent as null values. Check the box to specify that hexadecimal values be send rather than null values.

Message header
Whether to include a JMS-compliant (MQRFH2) header in all messages that are put on the send queue. When you create a publication that uses the send queue, you can specify a topic for the topic field in the header. A topic is a character string that describes the nature of the data that is published.

Delimited format
Whether you want the Q Capture program to publish messages in delimited format. You can accept the default delimiters for column, character data, new lines, and decimals, or use the four fields to specify your own delimiters.

If you specify your own delimiters, each of the delimiter fields must have a different value. You can select from the drop-down lists of valid delimiters, type a character, or type a character code point. In a character code point, you use the syntax 0xJJ or XJJ where JJ is the hexadecimal representation of the code point. For example, for the delimiter # you can use #, 0x23, or X23.

Code page for publishing delimited messages
Displays the code page that the Q Capture program uses to publish delimited messages. If this code page differs from the code page of the source table, the Q Capture program will convert the data. To avoid code page conversion, specify the code page that is used for the source table.

Column delimiter
Specifies a delimiter to separate the values in each column of the source table. The default delimiter is a comma (,). A null column value is represented by two consecutive column delimiters.

Character string delimiter
Specifies a delimiter to enclose all character data. The default delimiter is a double quotation mark ("). If the character data contains its delimiter, the delimiter is escaped by prefixing it with the same delimiter.

New line value
Specifies a new-line delimiter to separate the change-data records in one message. The default is a line feed (LF).

Decimal value
Specifies a character to use for the decimal point. The default is a period (.).

Creating publications

With event publishing, you can publish changed rows or transactions from a source table to a user application by creating publications.

Each publication is a single object that identifies:

- The source table that you want to publish changes from
- The columns and rows from the source table that you want to be published
- The publishing queue map, which names the WebSphere MQ queue that changes are published to

In event publishing, you can create one or multiple publications at one time.

Attention: Publications are separate objects from Q subscriptions. Publications do not publish data to the Q Apply program, but to an application of your choice. Publications are for publishing data, and Q subscriptions are for replicating data. If you want to replicate changes from a source table and want the Q Apply program to apply those source changes to a target table or pass them to a stored procedure for data manipulation, create a Q subscription, not a publication.

The following topics explain how to create one or many publications and further customize the publications based on your business needs.

Creating publications

By creating a publication, you define how data is published as XML or delimited messages from a source table to WebSphere MQ so that a subscribing application can retrieve and use those messages.

Before you begin

- Plan how you want to group publishing queue maps and publications.
- On the server that contains the source table for the publication, create the control tables for the Q Capture program.
- Create a publishing queue map. (You can do this task before you create a publication or while you create a publication.)

Restrictions

- A view cannot be a source for a publication.

About this task

Figure 26 shows how a single publication connects a source table to a WebSphere MQ send queue.

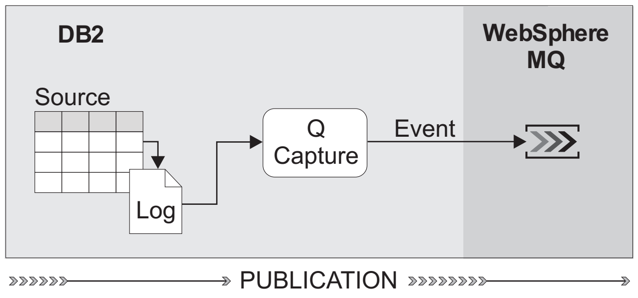


Figure 26. Single publication. Changes from a source table are published over WebSphere MQ queues.

Figure 27 on page 210 shows how multiple publications can use the same publishing queue map and Q Capture program.

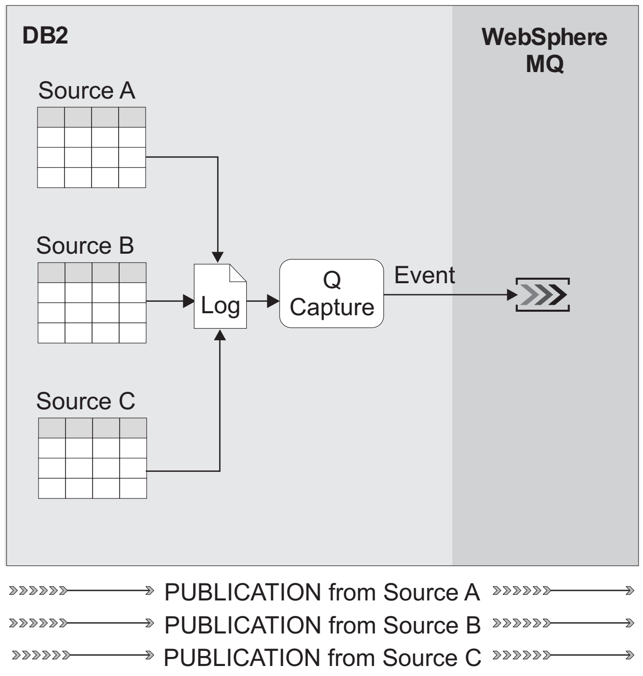

DB2
WebSphere MQ
Source A
Source B
Source C
Log
Q Capture
Event
PUBLICATION from Source A
PUBLICATION from Source B
PUBLICATION from Source C

Figure 27. Multiple publications. Changes from source tables are published over WebSphere MQ queues.

Procedure

To create publications, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **CREATE PUB** command. For example, the following commands set the environment and create the publication DEPARTMENT0001:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	`SET RUN SCRIPT LATER;`
	
	`CREATE PUB USING PUBQMAP`
	`SAMPLE_EP1_TO_SUBSCRIBER`
	`(PUBNAME "DEPARTMENT0001" JK.DEPARTMENT`
	`ALL CHANGED ROWS Y SUPPRESS DELETES Y);`

Replication Center	Use the Create Publications wizard. To open the wizard, expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the publication. Right-click the **Publications** folder and select **Create**. You can create one publication or many by using the wizard. When you create multiple publications at one time, the Replication Center assumes that you want to publish all columns and rows from each source table. At the end of the wizard, before the Replication Center builds the publications, you can modify individual publications so that only a subset of the source columns and rows are published.

Source columns for publications

By default when you create publications, changes to all columns that are in the source table are published. However, you can publish a subset of the columns if you do not want to make all of the columns that are in the source table available to the user application.

You might also want to publish a subset of the columns if the user application for a publication does not support all of the data types that are defined for the source table.

To publish a subset of the columns, select only the source columns that you want to be published to the user application. If you are creating a single publication, then the Create Publications wizard in the Replication Center gives you options for how to publish a subset of the columns from the source table. If you are creating multiple publications at one time, then, on the Review page of the Create Publications wizard, select the individual publication for which you want to publish only a subset of the columns, and then edit the properties for that publication.

Important for LOB columns: If you select columns that contain LOB data types for a publication, make sure that the source table enforces at least one unique database constraint (for example, a unique index or primary key). You do not need to select the columns that make up this uniqueness property for the publication.

When the Q Capture program publishes a message for publications

When you create publications, you can specify that the Q Capture program publishes a message either every time that a column in the source table changes, or only when columns that are part of a publication change.

The following sections describe the two different types of events that can cause the Q Capture program to publish a message:
- “Message is sent only when columns in publications change”
- “Message is sent every time a change occurs in the source table” on page 212

Recommendation: In general, the appropriate choice is that only changes that affect a selected column should be published. However, some applications need only a portion of a row, such as the key columns, whenever a change occurs. This published information can serve as an event notification, which can trigger other actions to occur. If you publish all of the columns that are in the source table, then these two options result in the same action.

Message is sent only when columns in publications change

By default, the Q Capture program publishes a message only when the change occurs in columns that you selected for the publications.

Example: Assume that you have 100 columns in your source table and you select 25 of those columns to be published in a publication. If you specify for a message to be sent only when columns in publications change, then any time a change is made to any of the 25 columns that are part of the publication, the Q Capture program publishes an message. Any time a change is made in any of the 75 columns that are not part of the publication, the Q Capture program does not publish an message.

Message is sent every time a change occurs in the source table

You can define publications so that the Q Capture program publishes a message every time a change occurs in the source table. If you are publishing only a subset of the columns in the source table, then the Q Capture program publishes a message even if the change occurs in a column that is not part of a publication.

Example: Assume that you have 100 columns in your source table and you select 25 of those columns to be published in a publication. If you specify for a message to be sent every time a change occurs in the source table, any time that a change is made to any of the of the 100 columns in your source table, the Q Capture program publishes an message.

Search conditions to filter rows in publications

By default when you create publications, all rows from the source table are published. However, when you create a publication, you can specify a WHERE clause with a search condition to identify the rows that you want to be published.

When the Q Capture program detects a change in the DB2 recovery log that is associated with a source table, the Q Capture program evaluates the change against the search condition to determine whether to publish the change.

If you are creating a single publication, then the Create publications wizard in the Replication Center helps you add a WHERE clause to publish a subset of the rows from the source table. If you are creating multiple publications at one time, then, on the Review page of the Create Publications wizard, select the individual publication for which you want to subset rows and edit the properties for that publication to add the WHERE clause.

When you specify a WHERE clause, you can specify whether the column is evaluated with values from the current log record. If you want a column in the WHERE clause to be evaluated with values from the current log record, place a single colon directly in front of the column name.

Example of WHERE clause that evaluates a column with values from the current log record

```
WHERE :LOCATION = 'EAST' AND :SALES > 100000
```

In the above example, `LOCATION` and `SALES` are column names in the source table that are evaluated with values from the current log record. Here, the Q Capture program sends only the changes from the source table that involve sales in the East that exceed $100,000. When you type a column name, the characters fold to uppercase unless you enclose the name in double quotation marks. For example, type `"Location"` if the column name is mixed case.

If the Q Capture program publishes a column that is part of the WHERE clause, it might need to change the type of operation that needs to be sent to the target table or stored procedure.

Example where the Q Capture program must change the type of operation because of a WHERE clause

```
WHERE :LOCATION = 'EAST'
AND :SALES > 100000
```

Suppose that the following change occurs at the source table:

```
INSERT VALUES ( 'EAST', 50000 )
UPDATE SET SALES = 200000 WHERE LOCATION = 'EAST'
```

Because the before value does not meet the search condition of the WHERE clause, the Q Capture program sends the operation as an INSERT instead of an UPDATE.

Likewise, if the before value meets the search condition but the after value does not, then the Q Capture program changes the UPDATE to a DELETE. For example, if you have the same WHERE clause as before:

```
WHERE :LOCATION = 'EAST'
AND :SALES > 100000
```

Now suppose that the following change occurs at the source table:

```
INSERT VALUES ( 'EAST', 200000 )
UPDATE SET SALES = 50000 WHERE LOCATION = 'EAST'
```

The first change, the insert, is sent to the target table or stored procedure because it meets the search condition of the WHERE clause (200000 > 100000 is true). However, the second change, the update, does not meet the search condition (50000 >100000 is false). The Q Capture program sends the change as a DELETE so that the value will be deleted from the target table or stored procedure.

Complex search conditions

Event publishing allows you to specify more complex WHERE clauses. However, complex search conditions might impact performance. For example, you can specify a more complex WHERE clause with a subselect that references other tables or records from either the source table or another table.

Example of WHERE clause with a subselect

```
WHERE :LOCATION = 'EAST'
AND :SALES > (SELECT SUM(EXPENSE) FROM STORES WHERE STORES.DEPTNO = :DEPTNO)
```

In the above example, the Q Capture program sends only the changes from the East that resulted in a profit, where the value of the sale is greater than the total expense. The subselect references the STORES table and the following columns in the source table: LOCATION, SALES, and DEPTNO.

When you define a publication with a subselect in a WHERE clause, the following problems might occur:

- Performance might be slower because, for each change in the source table, the Q Capture program computes a large select on the STORES table to compute the SUM(EXPENSE) value. Also, this type of select might compete for locks on the tables.
- The subselect might produce unexpected results. For example, because the subselect is evaluated against the current database values, the example above produces a wrong answer if the EXPENSE value changes in the database, whereas columns in the WHERE clause are substituted with the older log record values. If the table name that the subselect references does not change, then the search condition produces the proper results.

Restrictions for search conditions

- Search conditions cannot contain column functions, unless the column function appears within a subselect statement.

  Invalid WHERE clause with column functions:

```
#-----------------------------------------------------------------
#  Incorrect:  Don't do this
#-----------------------------------------------------------------

WHERE :LOCATION = 'EAST' AND SUM(:SALES) > 1000000
```

The Replication Center validates search conditions when the Q Capture program evaluates them, not when the Replication Center creates the publication. If a publication contains an invalid search condition, then that publication will fail when the invalid condition is evaluated, and the publications will be deactivated.

- Search conditions cannot contain an ORDER BY or GROUP BY clause unless the clause is within a subselect statement.

  Invalid WHERE clause with GROUP BY:

  ```
  #-----------------------------------------------------------------
  #  Incorrect:  Don't do this
  #-----------------------------------------------------------------

  WHERE :COL1 > 3 GROUP BY COL1, COL2
  ```

  Valid WHERE clause with GROUP BY:

  ```
  WHERE :COL2 = (SELECT COL2 FROM T2 WHERE COL1=1 GROUP BY COL1, COL2)
  ```
- Search conditions cannot reference the actual name of the source table that you are publishing changes from. Do not use the schema.tablename notation in a WHERE clause for the actual name of the source table. However, you can reference another table name in a subselect by using schema.tablename notation.

  Invalid WHERE clause with actual name of source table and column name:

  ```
  #-----------------------------------------------------------------
  #  Incorrect:  Don't do this
  #-----------------------------------------------------------------

  WHERE :ADMINISTRATOR.SALES > 100000
  ```

  In this example of a WHERE clause that has the actual names of the source table and columns, the table that is published is ADMINISTRATOR and SALES is the column name. This invalid WHERE clause is intended to select only the values of the SALES column of the ADMINISTRATOR table, for which SALES is greater than 100000.

  Valid WHERE clause with column name:

  ```
  WHERE :SALES > 100000
  ```

  In this example of a WHERE clause that has a column name, SALES is the column name.
- Search conditions cannot reference values that were in columns before a change occurred; they can reference only after values.
- Search conditions cannot contain EXISTS predicates.
- Search conditions cannot contain a quantified predicate, which is a predicate using SOME, ANY, or ALL.
- Search conditions cannot reference LOB values.

Key columns for publications

For each publication, you must specify which columns in the source table are key columns. Event publishing requires key columns to enforce that each row is unique. You can have the Replication Center recommend which columns in the source table should be used to identify uniqueness, or you can select the key columns yourself.

If you are creating a single publication, then the Create Publications wizard in the Replication Center launches the Select Key Column window to help you select the key columns from the source table. If you are creating publications at one time, then you can use the Review page of the Create Publications wizard to customize which key columns to use.

Options for including unchanged columns in messages for publications

When you create publications that publish a subset of the source columns, you can specify what column values from each row the Q Capture program includes in the message that it publishes.

The following sections describe the values that the Q Capture program can include in the message:

- "Only changed columns are sent"
- "Both changed and unchanged columns are sent"

This option applies only to values in non-key columns. The Q Capture program always publishes values in key columns.

Only changed columns are sent

By default when you create publications, the Q Capture program sends the values in the columns that you selected for the publications only if the column values change.

Example: Assume that you have 100 columns in your source table and you select 25 of those columns to be published in a publication. If you specify that only changed columns are sent, then any time that a change occurs in any of the 25 selected columns, the Q Capture program publishes only the columns that changed. For instance, if changes occur in 17 of the 25 selected columns, then the Q Capture program sends those 17 changed values.

Recommendation: Use this option to minimize the amount of unnecessary data that goes across the queues.

Both changed and unchanged columns are sent

You can also define publications so that the Q Capture program always sends the values in the columns that you selected for the publications, whether those values changed or not.

Example: Assume that you have 100 columns in your source table and you select 25 of those columns to be published in a publication. If you specify that both changed and unchanged columns are sent, then any time that a change occurs in any of the 25 selected columns, the Q Capture program publishes all of the selected columns. For instance, if changes occur in 17 of the 25 selected columns, then the Q Capture program still sends the values from all 25 columns.

Options for including before values in messages for publications

When an update occurs in columns that are not part of the target key, the Q Capture program either sends the value in the column after the change occurred, or it sends both the value in the column before the change occurred and the value in the column after the change occurred. When an update occurs in a key column, the before value and after value are always sent.

Because a delete operation always applies to a row and not to a specific column value, deletes are handled differently. For deletes, only before values are ever sent. Before values of key columns are always sent. If you specify for the message to include column values from before and after the change, then, if values in non-key columns are deleted, the before values of the non-key columns are sent.

The sections below describe your two options for before values and after values:
- "Send new data values only"
- "Send both old and new data values"

Send new data values only

By default when an update occurs at the source table, the Q Capture program publishes the values that are in the non-key columns after the change occurs. If you specify for the message to include only new data values (values from after the change), then the message does not include the values that were in the non-key columns before the change occurred.

Recommendation: If the application that receives the messages for the publications never uses the value that was in each non-key column before the change, then specify that the Q Capture program send only column values from after the change.

Send both old and new data values

If you specify that the message is to include both old and new data values (values from both before and after the change), then when a non-key column is updated, the Q Capture program publishes the value that is in the column before the change occurs and the value that is in the column after the change occurs.

Recommendation: If the application that receives the messages for the publications uses the value that was in each column before the change, then specify for the Q Capture program to send column values from before and after the change.

Restrictions for LOB data types: Before values for columns with LOB data types are not sent in the messages. If you specify for the message to include both before and after values, then this option does not apply for columns with LOB data types, and their before values are not sent.

Chapter 13. Data type considerations

When you replicate or publish certain data types, such as LONG VARCHAR or LOB data types, you should be aware of certain conditions and restrictions.

General data restrictions for Q Replication and Event Publishing

Some data types are not supported in Q Replication and Event Publishing and some data types can be used only under certain circumstances.

Data encryption restrictions

You can replicate or publish some types of encrypted data:

z/OS **EDITPROC**

DB2 for z/OS source tables that are defined with an edit routine (EDITPROC) to provide additional data security are supported. To use these tables as sources, the DB2 subsystem that contains the tables must be at Version 8 or higher with APAR PK13542 or higher.

Linux UNIX Windows **Encrypt scalar function**

Column data can be encrypted and decrypted using the encrypt scalar function in DB2 for Linux, UNIX, and Windows. To use this with replication or publishing, the data type must be VARCHAR FOR BIT DATA at the source. This data replicates successfully as long as the source and target use the same code page and the decrypt functions are available. Replication of columns with encrypted data should only be used with servers that support the DECRYPT_BIN or DECRYPT_CHAR function.

z/OS **FIELDPROC**

Q Replication supports columns that are defined on DB2 for z/OS tables with field procedures (FIELDPROC) to transform values. The DB2 subsystem that contains the tables with FIELDPROC columns must be at APAR PK75340 or higher.

If possible, you should create the following index on your SYSIBM.SYSFIELDS table to improve performance:

```
CREATE INDEX "SYSIBM"."FIELDSX"
ON "SYSIBM"."SYSFIELDS"
(TBCREATOR ASC,
TBNAME ASC,
NAME ASC)
USING STOGROUP SYSDEFLT PRIQTY 100 SECQTY 100
CLOSE NO;
COMMIT;
```

Data type restrictions

Currently, the following data cannot be replicated or published:

- Spatial data types
- z/OS Any column on which a VALIDPROC is defined
- You can replicate BINARY or VARBINARY data types when the source and target are on z/OS. Replication of these data types from a z/OS source to a DB2 for Linux, UNIX, and Windows target or federated target is not supported. BINARY and VARBINARY data types are

supported as targets of source expressions only if the source datatype is CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, or ROWID.

You can replicate or publish the following types of data only under certain circumstances:

LONG VARCHAR and LONG VARGRAPHIC
Columns with long variable character (LONG VARCHAR) and long variable graphic (LONG VARGRAPHIC) data types have the following restrictions:

- LONG VARCHAR and LONG VARGRAPHIC cannot be replicated from DB2 for Linux, UNIX, and Windows to DB2 for z/OS. Fields in DB2 for z/OS that contain long variable characters have a smaller maximum length than the fields in DB2 for Linux, UNIX, and Windows. Therefore, replication of these types of fields to DB2 for z/OS from DB2 for Linux, UNIX, and Windows might result in truncation.
- When you specify DATA CAPTURE CHANGES for a source table when the table is created, any LONG VARCHAR and LONG VARGRAPHIC columns are automatically enabled for replication. If you add LONG VARCHAR columns to the table after you create a Q subscription and the table previously had no LONG columns, you must use the ALTER TABLE statement to enable DATA CAPTURE CHANGES INCLUDE LONGVAR COLUMNS for the new LONG VARCHAR or LONG VARGRAPHIC columns.

User-defined data types
You can replicate or publish user-defined distinct data types, but not user-defined structured and reference data types. User-defined distinct data types (distinct data types in DB2) are converted to the base data type before they are replicated. If the target table is created when the Q subscription is created, user-defined distinct data types are converted to the base data type in the new target table.

GRAPHIC data type
Columns with the GRAPHIC data type at the source and target might not match when you use the asntdiff utility to check that the source and target tables are the same. DB2 columns with the GRAPHIC data type have blank padding after the graphic data. This padding might be single-byte or double-byte spaces, depending on the code page that the database was created in. This padding potentially can cause data to not match between the source and the target tables, especially if the source and target tables are in different code pages. This padding applies only to GRAPHIC data types and not other graphic data types such as VARGRAPHIC or LONG VARGRAPHIC.

To compare columns with GRAPHIC data types, you must remove the blank padding in the data before you compare the source and target tables by using the DB2 scalar function:

```
rtrim(column)
```

This function eliminates the code page differences for single-byte or double-byte spaces and ensures that the asntdiff utility compares the GRAPHIC data in a consistent manner.

TIMESTAMP WITH TIME ZONE
You can replicate the TIMESTAMP WITH TIMEZONE data type that was introduced in DB2 for z/OS Version 10, with some restrictions. Table 22 on page 219 shows the supported mappings

Table 22. Supported column mappings for TIMESTAMP WITH TIMEZONE data type

		Target column	
		TIMESTAMP WITH TIMEZONE	TIMESTAMP WITHOUT TIMEZONE
Source column	TIMESTAMP WITH TIMEZONE	Allowed if source column length is less than or equal to target column length	Not allowed
	TIMESTAMP WITHOUT TIMEZONE	Not allowed.	Allowed

Note: DB2 for Linux, UNIX, and Windows does not support TIMESTAMP WITH TIMEZONE.

Considerations for large object (LOB) data types for Q Replication and Event Publishing

Replication handles binary large object (BLOB), character large object (CLOB), and double-byte character large object (DBCLOB) data types differently depending on the setting of the Q Capture **lob_send_option** parameter and the size of the LOB data.

The parameter has these values:

I (inline)
LOB data is sent within the transaction message. This method can improve performance. You might need to adjust the value of the **max_message_size** option for the replication queue map to ensure that the messages are large enough to hold LOB values.

S (separate)
LOB data is sent after the transaction message in one or more LOB messages. The number of messages depends on the value of **max_message_size**.

Regardless of the setting for **lob_send_option**, starting with DB2 10.1 for Linux, UNIX, and Windows the Q Capture program can read LOB data directly from the DB2 recovery log rather than having to connect to the database and fetch from the source table. This method can also improve performance.

For source databases at DB2 Version 10.1 or later, LOB data is read from the log under these circumstances:

- The data fits within a source table row, which is affected by the INLINE LENGTH setting in the definition of a LOB column. This setting indicates the maximum byte size of a LOB value that can be stored in a base table row.
- The LOB column was created with the LOGGED attribute, which specifies that changes to the column are written to the log even if the size of the data exceeds the value of INLINE LENGTH.

When a LOB value does not fit into a message, the LOB_TOO_BIG_ACTION value in the IBMQREP_SENDQUEUES table determines what action Q Capture takes:

Q (default)
The Q Capture program follows the error action that is defined for the send queue.

E The Q Capture program sends empty LOB values if the data does not fit in a single transaction message. If the substitute value does not fit into a message, Q Capture follows the error action for the queue.

With **`lob_send_option`**=S, the Q Capture program sends LOB values for all LOB columns that are part of a Q subscription or publication for every row in a transaction. This behavior results in more WebSphere MQ messages if a LOB value is updated multiple times in a transaction or if the CHANGED_COLS_ONLY option in the Q subscription or publication is set to **`N`**. For key updates, LOB values for all LOB columns that are part of a Q subscription or publication are sent regardless of the CHANGED_COLS_ONLY setting.

Recommendation: If you are replicating the DBCLOB data type, set **`lob_send_option`**=`I`, especially if you expect DBCLOB values that are larger than 16KB.

Federated targets: You can replicate LOBs to Oracle targets, but not to any other non-DB2 target.

Other important information regarding LOB replication:
- You must have a unique index in your source table to replicate LOB data.
- An entire LOB is replicated or published, even if only a small portion of the LOB is changed.
- Q Replication and Event Publishing do not support DB2 Extenders™ for Text, Audio, Video, Image, or other extenders where additional control files that are associated with the extender's LOB column data are maintained outside of the database.
- The before values for LOB or ROWID columns are not replicated or published. When the Q Capture program sees an indication of a LOB change (a LOB descriptor) in the DB2 recovery log, the Q Capture program sends the LOB value from the source table.

XML data type

DB2 Version 9.1 and later supports native storage of XML documents. You can replicate or publish the XML documents.

The XML documents are replicated within the transaction message, similar to how other columns are replicated. If the XML documents are large, you might need to increase the maximum message size for the WebSphere MQ messages. If an XML documents exceeds the maximum message size for the send queue, the XML document is not sent. You can set the XML_TOO_BIG_ACTION for the send queue in the following ways in the IBMQREP_SENDQUEUES table to define the action that Q Capture takes:

Q (default)
The Q Capture program follows the error action that is defined for the send queue.

E The Q Capture program sends an XML placeholder. If the XML placeholder does not fit into a message, Q Capture follows the error action for the queue.

The Q Apply program adds a row to the IBMQREP_EXCEPTIONS table any time that XML documents cannot be sent.

Requirements

You must have a unique index in your source table to replicate XML data.

In addition to replicating to the DB2 XML data type, you can replicate XML documents to federated targets that support XML data types, or you can map the source XML column to a CLOB or BLOB column at the target.

Restrictions

The XML data type has the following restrictions:
- You cannot filter rows based on the contents of an XML document.
- You cannot replicate XML columns that you defined as a unique index.
- The documents are not validated by the Q Apply program.
- You cannot replicate the XML schema registrations.
- z/OS On z/OS, you cannot perform an automatic load for tables that contain XML columns. The DSNUTILS program that performs the automatic load does not support the XML data type.
- You cannot replicate XML columns from Oracle sources.
- User-defined unique indexes on XPATH expressions on XML columns are not supported. To replicate these columns, drop the unique index and use a non-unique index.

Replication between XML and LOB columns

Q Replication supports mappings between XML and large object (LOB) columns, with restrictions in some cases.

No restrictions
The following mapping is supported without restriction:
- BLOB to XML

Supported with restrictions
The following mappings are supported, but no automatic load (type I) is allowed because moving data from XML to BLOB or CLOB columns is not supported by the DB2 LOAD or IMPORT utilities that are used in automatic loads:
- XML to BLOB
- XML to CLOB

You can use automatic loads with CLOB to XML, but if you specify the EXPORT and LOAD utilities the source and target databases must be in the same code page:
- CLOB to XML

Attention: If you are replicating CLOB to XML, data loss can occur if both of the following conditions are true:
- The source database uses a non-UTF-8 code page, and the target database uses code page 1208 (codeset UTF-8).
- The Q subscription specifies an automatic load that uses the EXPORT and IMPORT utilities.

In this situation, you must use the LOAD from CURSOR utility for the Q subscription.

Not supported

The following mappings are not supported:

- XML to DBCLOB
- DBCLOB to XML

Replication of new DB2 Version 9.7 data types (Linux, UNIX, Windows)

Q Replication supports new data types that were introduced with DB2 for Linux, UNIX, and Windows Version 9.7 to make it easier to migrate applications to DB2.

Some of the new data types require special considerations for a replication environment. The following sections provide details:

- "TIMESTAMP with extended precision"
- "DATE with compatibility option" on page 223
- "NUMBER" on page 223

TIMESTAMP with extended precision

Q replication supports replication of TIMESTAMP data with extended precision that ranges from TIMESTAMP(0) to TIMESTAMP(12). If both the source and target databases and Q Capture and Q Apply are Version 9.7 or newer, the source data is padded or truncated at the target database to satisfy a mapping of non-matching TIMESTAMP columns. Mapping of non-matching TIMESTAMP columns is supported only for unidirectional replication.

If a V9.7 Q Capture program is sending data to an older Q Apply program (COMPATIBILITY in the IBMQREP_CAPPARMS table is less than 0907), Q Capture sends only TIMESTAMP(6) data to match the capability of the target. This situation might require Q Capture to pad or truncate the source data, depending on the size of the V9.7 source TIMESTAMP column.

For example, if you replicated a source at V9.7 to a target at V9.5 and a source table included a TIMESTAMP(12) column, the V9.7 Q Capture program would truncate six digits from the fractional seconds portion of the TIMESTAMP value. The truncation is necessary because DB2 V9.5 does not support extended precision, and so for V9.5 databases TIMESTAMP values have a fractional seconds portion that equates to TIMESTAMP(6). Table 23 shows a value at the source and resulting truncated value at the target.

Table 23. Truncation of TIMESTAMP(12) during replication

Source value in TIMESTAMP(12)	Target value in TIMESTAMP(6)
2009-07-10-10.33.42.458499823012	2009-07-10-10.33.42.458499

In this same scenario, if the source data is in the TIMESTAMP(0) to TIMESTAMP(5) range, DB2 automatically pads the data to the pre-V9.7 level of six digits for fractional seconds.

z/OS **Note:** When handling these new data types, Q Replication treats a DB2 for z/OS source or target the same as DB2 for Linux, UNIX, and Windows Version 9.5 or older.

If Q Apply is at V9.7 or newer and Q Capture is older, DB2 automatically pads or truncates source TIMESTAMP values to match the precision of the target TIMESTAMP column. Table 24 shows an example of padding.

Table 24. Padding of older TIMESTAMP value during replication

Source value in TIMESTAMP	Target value in TIMESTAMP(12)
2009-07-10-10.33.42.458499	2009-07-10-10.33.42.458499000000

DATE with compatibility option

The date compatibility option stores the DATE type with an additional time portion (HH:MM:SS). This format conforms to the date representation by other relational database management systems such as Oracle, where the DATE data type includes YYYY-MM-DD HH:MM:SS.

Q Replication treats databases without date compatibility the same as DB2 databases prior to V9.7, and the same as DB2 for z/OS subsystems. When date compatibility is enabled, DB2 handles columns that are defined as DATE in the same way that it handles columns defined as TIMESTAMP(0).

Enable the DATE as TIMESTAMP(0) support by setting bit position number 7 (0x40) of the DB2_COMPATIBILITY_VECTOR registry variable before you create a database. With unidirectional Q Replication you can create the following column mappings between DATE and TIMESTAMP(0):

DATE to TIMESTAMP(0)
If the source database does not have date compatibility enabled, the target value is padded to YYYY-MM-DD-00:00:00.

TIMESTAMP(0) to DATE
If the target database does not have date compatibility enabled, the TIMESTAMP(0) value is truncated to YYYY-MM-DD.

NUMBER

The NUMBER data type supports applications that use the Oracle NUMBER data type. DB2 treats NUMBER data internally as DECFLOAT if no precision or scale are specified, and as DECIMAL with precision or scale if these attributes are specified.

Because Q Replication already supports DECFLOAT and DECIMAL, for unidirectional replication you can map columns defined with any of these three numeric types to each other: NUMBER to DECFLOAT or DECIMAL, DECFLOAT to NUMBER or DECIMAL, and DECIMAL to NUMBER or DECFLOAT.

Replication of tables with identity columns

Q Replication allows identity columns in both source and target tables, but because of DB2 restrictions you might need to take extra steps if your source table has columns that are defined with the AS IDENTITY GENERATED ALWAYS clause.

Identity columns are handled differently by replication depending on whether they are in the source or target table:

Source table
If you have an identity column in a source table and you want to replicate

it to a target table, create a Q subscription for the source table as usual. The target table is created with numeric columns to hold the values. For example, a source column that is defined as GENERATE ALWAYS might be replicated to a BIGINT column at the target. The columns in the target table cannot be identity columns themselves, so you cannot replicate an identity column in a source table to an identity column in a target table.

Target table
If you have an identity column in a target table, do not include that column in your Q subscription. The column is populated automatically when replication inserts into or updates the target table. The behavior of the identity column is the same as for inserts and updates by any other application. If you replicate the same source table to multiple target tables that have identity columns, the identity values in those target tables are independent of each another.

DB2 does not allow inserts into columns that are defined with the AS IDENTITY GENERATED ALWAYS clause, and so this clause is not supported for Q Replication target tables. However, options exist for replicating these columns:
- Create the target table without the IDENTITY clause.
- Create the target table with a column that is defined with AS IDENTITY GENERATED BY DEFAULT.

For columns that are defined with AS IDENTITY GENERATED BY DEFAULT, the range of values must be distinct between the source and the target because DB2 does not guarantee uniqueness of identity columns between two different DB2 databases.

For example, the identity column at one site could be set to even numbers (START WITH 2, INCREMENT BY 2) and at the other site the identity column could be set to odd numbers (START WITH 1, INCREMENT BY 2). You could also assign ranges to sites (for example, 1 to 10,000 at one site and 20,000 to 40,000 at the other). The odd-even approach ensures that in a conflict situation, two different rows that accidentally have the same generated identity key do not overwrite one another when the conflict action is to force the change.

The data type of the identity column (SMALLINT, INTEGER, or BIGINT) should be determined by application needs, for example the largest number that you expect in the column.

The identity columns should be NO CYCLE if numbers cannot be reused. Put a plan in place for what to do when the maximum value is reached (SQLSTATE 23522). If you use CYCLE, make sure that a new use of a number does not cause problems for any existing use of the number, including what happens during replication.

Chapter 14. Working with scripts and commands generated by the replication administration tools

The ASNCLP command-line program and Replication Center generate SQL scripts for defining and changing replication objects. The Replication Center also generates operational commands for such tasks as starting and stopping the replication programs, pruning control tables, changing parameters, or checking program status.

You can use the replication administration tools to run the scripts and commands that they generate, or you can save the scripts and commands, modify them, and run them later.

Running and saving scripts generated by the replication administration tools

To create replication and publishing objects, you run SQL scripts that are generated by the ASNCLP command-line program or Replication Center. You can modify the scripts, use the tools to run the scripts, or run the scripts from a DB2 command line.

About this task

When editing the generated SQL scripts, be careful not to change the termination characters. Also, do not change the script separators if there are multiple scripts saved to a file.

You might want to customize the SQL scripts for your environment to perform the following tasks:

- Create multiple copies of the same replication action, customized for multiple servers.
- Combine definitions together and run as a batch job.
- Defer the replication action until a specified time.
- Create libraries of SQL scripts for backup, site-specific customization, or to run standalone at distributed sites, such as for an occasionally connected environment.

Procedure

1. Use one of the following methods to run or save scripts that are generated by the replication administration tools:

Method	Description
ASNCLP command-line program	Use the SET RUN SCRIPT command to control whether to automatically run SQL statements that are generated by each ASNCLP task command before processing the next command or to manually run them later in a DB2 command prompt. **NOW** This option automatically runs the generated SQL. **LATER** This option saves the generated SQL to a file. You can specify the file path and name by using the SET OUTPUT command. For example the following command specifies to automatically run the SQL script but stop processing the ASNCLP commands if an error occurs: `SET RUN SCRIPT NOW STOP ON SQL ERROR ON`
Replication Center	Use the Run Now or Save Script window. The **Run now** radio button is initially selected so that when you click **OK** the Replication Center issues the SQL statements to create or change an object. Subsequently, the window selects the last processing option that you picked. For example, if you previously choose to save the scripts to a file, the next time this window is displayed the **Save to file** radio button is selected and the same system and path appear in the **Run specifications** box. Linux UNIX Windows You can also schedule SQL scripts to run as tasks in the Task Center on Linux, UNIX, and Windows. The **Apply** button allows you to run the script and leave this window open so that you can still save the script as a file or as a task.

2. Optional: Use one of the following methods to run the files containing SQL scripts from a DB2 command line:
   - Use this command if the SQL script has a semicolon (;) as a termination character:
     ```
     db2 -tvf filename
     ```
   - Use this command if the SQL script has some other character as the delimiter:
     ```
     db2 -tdchar -vf filename
     ```
     Where *char* is a termination character such as a pound sign (#).

   If you run the SQL scripts from a DB2 command line, you must connect to servers manually when you run the SQL script. The script is generated with CONNECT statements. Before you run the SQL script, you must edit the SQL statements to specify the user ID and password for the server. For example, look for a line that resembles the following example and add your information by typing over the placeholders (XXXX):
   ```
   CONNECT TO database_name USER XXXX USING XXXX ;
   ```

Running and saving commands (Replication Center)

You can run commands directly from the Replication Center, save and run commands as batch files from a command line, or save commands as task objects for the Task Center.

Procedure

To work with commands that are generated by the Replication Center, use the Run Now or Save Command window.

1. Optional: In the text area, modify the command.
2. Choose one of the following options:
   - To run the command immediately, click **Run now**, fill in the **Run specifications** fields, and click **OK**.
   - To save the command, click **Save to file**, fill in the **Run specifications** fields, and click **OK**. The Replication Center creates a file with the name and extension that you specify, or you can save the command to an existing file.
   - To save the command as a task object for the Task Center, click **Save as task**, fill in the **Run specifications** fields, and click **OK**.

Chapter 15. Operating a Q Capture program

A Q Capture program captures transactions or row-level changes from source tables that are part of a Q subscription or publication, and then sends this transactional data as messages over WebSphere MQ queues.

You can operate a Q Capture program using the Replication Center, system commands, and system services, and you can change the Q Capture operating parameters in several ways.

Starting a Q Capture program

You start a Q Capture program to begin capturing transactions or row-level changes from the DB2 recovery log for active or new Q subscriptions or publications, and sending the transactional data as messages over WebSphere MQ queues.

Before you begin

- If you are starting a Q Capture program from a remote workstation, configure connections to the Q Capture server.
- Create a WebSphere MQ queue manager, queues, and other required objects.
- Ensure that you have authorization for Q Replication and event publishing objects and WebSphere MQ objects.
- Create control tables for the appropriate Q Capture schema.
- Configure the source database or subsystem to work with the Q Capture program.

  Important: Ensure that archive logging is turned on at the database that you are using as the Q Capture server. Use the Turn On Archive Logging window in the Replication Center to configure the Q Capture server for archive logging, and perform an offline backup of the database.
- If any Q subscriptions that specify an automatic load that uses the EXPORT utility are in N (new) or A (active) state, create a password file on the Q Apply server to allow the utility to connect to the Q Capture server.

About this task

When you initially start a Q Capture program without specifying a start mode, it uses the default start mode, warmsi. In this mode, the program tries to read the log at the point where it left off. Because this is the first time that the program is started, Q Capture switches to cold start mode and begins processing Q subscriptions or publications that are in N (new) or A (active) state. Any Q subscriptions or publications that are in I (inactive) state must be activated for the program to begin capturing changes.

You can start a Q Capture program even if no Q subscriptions or publications are in A (active) state. When you activate the Q subscriptions or publications, the Q Capture program begins capturing changes.

When you start a Q Capture program, you can specify startup parameter values and the program will use the new values until you take one of the following actions:

- Change the parameter values while the program is running.
- Stop and restart the program, which prompts it to read the IBMQREP_CAPPARMS table and use the values saved there.

Procedure

To start a Q Capture program, use one of the following methods:

Method	Description

z/OS
 	
z/OS console or TSO
	On z/OS, you can start a Q Capture program by using JCL or as a system-started task. You can specify new invocation parameter values when you start a Q Capture program with JCL.
	
	z/OS has a 100-byte limit for the total length of parameters that you can specify in the PARMS= field. To overcome this limitation, replication programs now allow you to specify as many additional parameters as needed in the SYSIN data set.
	
	When the SYSIN DD statement is included in the invocation JCL, the Q Capture program automatically concatenates what is specified in the SYSIN dataset to the PARMS= parameters. You can only specify Q Capture parameters in the SYSIN data set. Any LE parameters must be specified in the PARMS= field or in LE _CEE_ENVFILE=DD, followed by a slash(/).
	
	Example:
	`//* asterisk indicates a comment line`
	`// QCAP EXEC PGM=ASNQCAP,PARMS='LE/Q Capture parameters'`
	`//* Parameters can be any or no LE parameters and any or`
	`//* no Q Capture parameters`
	`//SYSIN DD *`
	`//* additional Q Capture parameters, one or more`
	`//* parameters on each line`
	`  CAPTURE_SERVER=DSN!! CAPTURE_SCHEMA=CAPCAT`
	`  DEBUG=Y LOGSTDOUT=N`

`asnqcap` command	Use the **`asnqcap`** command to start a Q Capture program and specify startup parameters. For example:
	`asnqcap capture_server=`*`server_name`*
	`capture_schema=`*`schema`*
	`parameters`
	
	Where *server_name* is the name of the database or subsystem that contains the Q Capture control tables, *schema* identifies the Q Capture program that you want to start, and *parameters* is one or more parameters that you can specify at startup.

Windows Windows services	You can create a replication service on Windows operating systems to start the Q Capture program automatically when the system is started.

You can verify whether a Q Capture program started by using one of the following methods:

- Examine the Q Capture diagnostic log file (*`capture_server.capture_schema`*`.QCAP.log` on z/OS and *db2instance.capture_server.capture_schema*.QCAP.log on Linux, UNIX, and Windows) for a message that indicates that the program is capturing changes.

- Check the IBMQREP_CAPTRACE table for a message that indicates that the program is capturing changes.
- z/OS If you are running in batch mode, examine the z/OS console or z/OS job log for messages that indicate that the program started.
- Use the Q Capture Messages window in the Replication Center to see a message that indicates that the program started. To open the window, right-click the Q Capture server that contains the Q Capture program whose messages you want to view and select **Reports** > **Q Capture Messages**.

Starting Q Capture from a known point in the DB2 log

You can use command-line parameters to start the Q Capture program at a known point in the DB2 log without triggering a load of the target table.

About this task

Typically, when the Q Capture program is stopped you use a warm restart to begin reading the DB2 recovery log where Q Capture left off. You typically use a cold restart to begin reading at the end of the log. A cold restart automatically prompts the Q Apply program to reload the target table with the latest data from the source.

In some situations, you might want to restart the Q Capture program from a known point in the log. For example, in a high availability failover scenario you can record information from the DB2 log on one server and then use a command to start capturing data on the backup server after the second server takes over.

Procedure

To start the Q Capture program from a known point in the DB2 log:

1. Determine the point in the log where you want Q Capture to start reading. You will need the values for two command parameters:

   lsn The log sequence number (LSN) of the oldest uncommitted transaction that you want to capture.

   maxcmtseq
   The LSN of the most recently committed transaction that was put on the send queue.
2. Use JCL or the **asnqcap** command and specify both the **lsn** and **maxcmtseq** parameters.

   You must use both parameters in the same JCL statement or command invocation, and you cannot use these parameters if the value of **startmode** is cold.

   For example, to start the Q Capture program on a server named SAMPLE with an **lsn** value of 0000:C60F:2722:E137:0001 and a **maxcmtseq** value of 0000:C60F:2722:E080:0001, use one of the following methods:

   JCL
   ```
   //QCAPDEC EXEC PGM=ASNQCAP,REGION=0M,TIME=NOLIMIT,
   // PARM='STORAGE(FF,FF,FF)/SSTR CAPTURE_SCHEMA=QDECODER
   //             STALE=20 STARTALLQ=N'
   .
   .
   ```

```
.
//SYSIN DD *
    MAXCMTSEQ=0000:C60F:2722:E080:0001
    LSN=0000:C60F:2722:E137:0001
//SYSTERM  DD DUMMY
```

asnqcap command

```
asnqcap SAMPLE LSN=0000:0000:0000:115b:7704
MAXCMTSEQ=41c2:2264:0000:0004:0000
```

The following examples show how you can restart Q Capture from various points in the log, and find the values of **lsn** and **maxcmtseq** in different scenarios.

Example 1:

To start from the end of the log without triggering a load (full refresh) of the target table, specify one of the following values in the **asnqcap** command, depending on your DB2 version:

Version 9.7 and below
: **lsn**=FFFF:FFFF:FFFF:FFFF:FFFF and **maxcmtseq**=FFFF:FFFF:FFFF:FFFF:FFFF.

Version 10.1 or higher with compatibility of 1001 or higher, or Version 9.8
: **lsn**=FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF and **maxcmtseq**=FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF.

Example 2:

In some cases you might not be able to use a warm restart, for example if the restart message or restart queue is lost or corrupted. To find the values of **lsn** and **maxcmtseq** at the point where the Q Capture program stopped reading the log, use one of the following methods:

Method	Description

asnqmfmt command	Run the **asnqmfmt** command to format the restart message, which contains the point where Q Capture stopped reading.
	
	Specify the queue manager that the Q Capture programs works with, and the name of the restart queue. For example:
	`asnqmfmt asn.qm1.restartq qm1`
	
	In the command output, look for the values in the restartLSN and qRestartMsg.lastCommitSEQ fields. For example:
	`restartLSN=0000:0000:0000:03ab:65ab`
	`qRestartMsg.lastCommitSEQ: 4a02:3396:0000:0002:0000`
	
	You use the value of qRestartMsg.restartLSN for the **lsn** parameter and the value of qRestartMsg.lastCommitSEQ for the **maxcmtseq** parameter.
	Note: Starting with Version 9.5 Fix Pack 3, the format of the restart message changed to indicate restart points for each send queue that a Q Capture program works with. If you are using a version at this level or later, the restart LSN and last commit indicator appear as in the following example:
	`sendq lastCommitSEQ: 4a02:3396:0000:0002:0000`
	`sendq restartLSN=0000:0000:0000:03ab:65ab`
	
	You use the value of sendq restartLSN for the **lsn** parameter and the value of sendq lastCommitSEQ for the **maxcmtseq** parameter.

Method	Description
Q Capture diagnostic log	Look for message ASN7109I in the log. The log file is found in the directory specified by the **capture_path** parameter or by default in SQLLIB\bin. The value of **lsn** is described as "the lowest log sequence number of a transaction still to be committed." The value of **maxcmtseq** is described as "the highest log sequence number of a successfully processed transaction." For example: `2006-01-11-16.12.49.583183 <asnqwk>ASN7109I "Q Capture" : "ASN" : "WorkerThread" : At program termination, the highest log sequence number of a successfully processed transaction is "43C5:9F00:0000:0001:0000" and the lowest log sequence number of a transaction still to be committed is "0000:0000:0000:2B39:606D".`

Example 3:

To start Q Capture from the beginning of its last run, look for message ASN7108I in the Q Capture diagnostic log for the values of **lsn** and **maxcmtseq** at the time that Q Capture started.

For example:

```
2006-01-11-16.12.31.360146 <asnqwk> ASN7108I  "Q Capture" : "ASN" : "WorkerThread" :
At program initialization, the highest log sequence number of a successfully
processed transaction is "43C5:9EE5:0000:0001:0000" and the lowest log sequence
number of a transaction still to be committed is "0000:0000:0000:2B38:82F6".
```

You can also look for message ASN7699I in the Q Apply diagnostic log to obtain the value for **maxcmtseq** if you want to avoid having the Q Capture program resend transactions that Q Apply already received.

Specifying Q Capture restart points for individual send queues or data partitions (z/OS)

When you start a Q Capture program in warm mode, you can override the global restart information in the restart message and specify different restart points for individual send queues and for individual partitions in a partitioned source database.

Before you begin

The user ID that starts the Q Capture program must have authorization to open and read the restart file.

About this task

To specify restart points for individual send queues or individual data partitions, you use a Q Capture initialization parameter that enables you to override the restart information in the restart message with information from a file.

The parameter, **override_restartq**, prompts Q Capture to look in the restart file as well as the restart message for the point where it should start reading in the log for a given send queue or data partition when it starts. Any restart information that is contained in the file overrides the restart information in the restart message.

The ability to start Q Capture from different points for individual send queues or data partitions can help in the following situations:

- You want to restart Q Capture so that it republishes data for one or more targets that are served by different send queues without affecting other targets. In case of disaster recovery where the primary site connects to multiple target sites, the fallback procedure would consist of restarting Q Capture from different points for each send queue that connects to a different target site to resend any changes that were lost.
- In a partitioned database environment, you want to start Q Capture from a known point in the log that differs from the point that Q Capture saved in its restart message, or start Q Capture in warm mode when its restart message is lost or corrupted.
- If the file system that stores WebSphere MQ objects is lost or the Q Capture restart message is accidentally deleted and multiple send queues are used, you can restart Q Capture from individual restart points.

Procedure

To start Q Capture from a specified point in the log for one or more send queues:

1. Do one of the following:
   - If are specifying a restart point at the send queue or partition level for the first time, create a restart file in the directory that is specified by the **`capture_path`** parameter, or in the directory from which Q Capture was started if nothing is specified for **`capture_path`**. Use the following naming convention:

     `capture_server``.`*`capture_schema`*`.QCAP.QRESTART`. Q Capture creates the data set with these attributes: DCB=(RECFM=FB,LRECL=128).

     Linux UNIX Windows

     `db2_instance``.`*`capture_server`*`.`*`capture_schema`*`.QCAP.QRESTART`
   - If the QCAP.QRESTART file already exists, follow the steps below to update its contents.
2. Look on the standard output or Q Capture log file for informational message ASN7207I, which Q Capture issues just before it stops. This message contains the following information for each send queue:
   - Queue name
   - Restart LSN (the lowest LSN of a transaction still to be committed)
   - commitSEQ (the last successfully committed transaction)
   - Partition number for partitioned databases on Linux, UNIX, and Windows

   The following example shows an ASN7207I message:

   ```
   ASN7207I "Q Capture" : "QALLTYPE" : "WorkerThread" : The restartq override
   file contents follow: queue name restart lsn maxcmtseq partition
   (only for partitioned databases).
   QALLTYPEL12 0000:0000:4F07:634E:0000 0000:0000:4F00:547A:0000
   ```
3. For each send queue or data partition for which you want Q Capture to start capturing transactions from the recovery log at a specific point, enter a line in the `QCAP.QRESTART` file in the following format:

Send queue name	Space	restartLSN	Space	commitSEQ	Space (optional)	partition (optional)

   The following example shows the contents of a restart file for two send queues, SQ1 and SQ2, on data partitions 0 and 1:

   ```
   SQ1 0000:0000:4F07:634E:0000 0000:0000:4F00:547A:0000 0
   SQ2 0000:0000:4FB9:0864:0000 0000:0000:4FB9:0864:0000 1
   ```

The data partition information is not needed for z/OS or for non-partitioned databases on Linux, UNIX, and Windows. Colons (:) are optional in the restartLSN and commitSEQ values.

4. Save the `QCAP.QRESTART` file.
5. Start the Q Capture program with **`startmode`**=warmns and **`override_restartq`**=y.

   Q Capture reads the restart file when it initializes, and starts reading the recovery log at the specified restartLSN for each send queue for which information exists in the file. For any other send queues, Q Capture uses the global restart information in the restart message.

   When Q Capture starts, it clears the restart file. When Q Capture stops, it writes the most recent restart information to the file and to the standard output, its log file, and the IBMQREP_CAPTRACE table.

   Note: You cannot specify **`override_restartq`**=y if you also specify the **`lsn`** and **`maxcmtseq`** parameters. These parameters are used to start Q Capture from a specified point in the log for all send queues, rather than for individual send queues.

The next time that you need to start Q Capture from a specified point for a given send queue or data partition, you can open `QCAP.QRESTART` and modify any lines that pertain to the send queues that you want to be started from a given point by changing the restartLSN or commitSEQ values.

During warm starts when **`override_restartq`** is specified, Q Capture follows this protocol depending on whether a restart message exists in the restart queue:

Restart message exists
: The restart file must contain one line for each send queue in the restart message whose restartLSN and commitSEQ you want to modify. Q Capture replaces the send queue restartLSN and commitSEQ information in the restart message with corresponding restart information from the file.

Restart message does not exist
: The restart file must contain one line for each send queue that was in the restart message. Q Capture builds the restart message from restart information in the restart file.

Considerations for using the cold start mode

You can prevent unwanted loading of the target table and other possible issues by considering these issues related to the cold start mode.

When you cold start a Q Capture program any time after the Q Capture program starts initially, the Q Capture program starts reading the DB2 log from the end instead of from the last restart point. The following results can occur:

The source and target can become out of sync, which requires you to load the target
: The Q Capture program might skip log records for data that it otherwise would have passed to the Q Apply program. If these records contain updates or inserts to the source table, the only way to synchronize the source and target tables is to load the target table (sometimes called a full refresh). Loading the target can cause you to lose historical data.

The load can take a long time
: The load requires significant time and effort for environments that contain many tables or large amounts of data. For these environments, the full

refresh can cause costly outages, especially on production systems. Use the cold start option as a last resort except when you start a Q Capture program initially.

You can follow these guidelines to prevent unexpected cold starts:

Set the startmode parameter
The **startmode** parameter should not have the value cold. Specify the warmns or warmsi start mode whenever possible. When you specify these start modes, the Q Capture program will not cold start if the warm start information is not available.

Monitor the status of the Q Capture programs with the Replication Alert Monitor
For example, you can use the QCAPTURE_STATUS alert condition to send you an e-mail alert whenever the monitor detects that a Q Capture program stopped. If the program is down long enough, you might require a cold start to synchronize the source and target tables because the Q Capture program behind in reading log records.

Retain sufficient DB2 log data and ensure the data is available to Q Capture
If log files are not available to the Q Capture program, the program cannot continue to capture the changes that are made to the source tables and might require a cold start.

Changing the Q Capture parameters

You can change the Q Capture operating parameters when you start the program, while the program is running, or by updating the IBMQREP_CAPPARMS control table.

Methods of changing the Q Capture operating parameters

This topic provides a brief description of the three different ways that you can change the parameters, followed by an example to help clarify the differences.

Changing saved parameters in the IBMQREP_CAPPARMS table
The Q Capture parameter values are saved in the IBMQREP_CAPPARMS control table. After installation, this table is filled with the shipped default values for the program. A Q Capture program reads the table when it starts. You can use other methods to change the parameter values when you start a Q Capture program or while it is running, but these changes stay only in memory. When you stop and restart the Q Capture program, it uses the parameter values that are saved in the IBMQREP_CAPPARMS table. You can update this table using the Q Replication Dashboard or SQL.

Setting parameter values at startup
When you start a Q Capture program, you can override the parameter values that are saved in the IBMQREP_CAPPARMS table. You can use JCL or the **asnqcap** system command to set values for the operating parameters. Your changes take effect when the program starts, but last only while the program is running.

Dynamically changing parameters while a Q Capture program is running
You can dynamically change parameter values without needing to stop capturing changes from the source. Use the **chgparms** parameter of the MODIFY command on z/OS or asnqccmd command on Linux, UNIX, Windows or UNIX System Services on z/OS to change values while a Q

Capture program is running. Your changes last only until the program stops running, or until the next change-parameters request.

Three ways to change Q Capture operating parameters

Assume that you want to shorten the default setting for the **commit_interval** parameter of 500 milliseconds (one half second) for a Q Capture program that is identified by schema ASN1:

1. Update the IBMQREP_CAPPARMS table for Q Capture schema ASN1. Set the commit interval to 250 milliseconds. You can use the Programs tab in the Q Replication Dashboard, or the following SQL:

   ```
   update asn1.ibmqrep_capparms set commit_interval=250
   ```

   When you start this Q Capture program in the future, the commit interval defaults to 250 milliseconds.
2. You want to see the effect of a longer commit interval on replication throughput (the number of transactions published for a given period of time). Rather than change the saved value in the control table, you start the Q Capture program with **commit_interval** set to 1000 milliseconds (one second). You can use the **asnqcap** command:

   ```
   asnqcap capture_server=srcdb1 capture_schema="ASN1" commit_interval=1000
   ```

   While the program runs using a 1-second commit interval, you monitor its performance.
3. Based on performance, you decide to lower the commit interval. Instead of stopping the Q Capture program, you dynamically change the parameter while the program is running to 750 milliseconds (0.75 seconds), and monitor the change. You can use the **chgparms** parameter with the **MODIFY** command or **asnqccmd** command:

   ```
   f myqcap,chgparms commit_interval=750
   ```

   ```
   asnqccmd capture_server=srcdb1 capture_schema="ASN1" chgparms
    commit_interval=750
   ```

You can continue to monitor the throughput and latency statistics and tune the **commit_interval** parameter. When you find the value that meets your requirements, you can update the IBMQREP_CAPPARMS table (as described in step 1). The next time you start a Q Capture program, it uses the new value as the default commit interval.

Changing parameters while a Q Capture program is running

You can modify the behavior of a Q Capture program while it continues to capture changes from the source. The Q Capture program begins using the new settings almost immediately, but the changes are not saved in the IBMQREP_CAPPARMS control table. If you stop and then restart the program, it uses the saved values in the control table.

About this task

You can change the following Q Capture parameters while the program is running:

- **autostop**
- **commit_interval**
- **logreuse**
- **logstdout**

- **`memory_limit`**
- **`monitor_interval`**
- **`monitor_limit`**
- **`prune_interval`**
- **`qfull_num_retries`**
- **`qfull_retry_delay`**
- **`signal_limit`**
- **`sleep_interval`**
- **`term`**
- **`trace_limit`**

z/OS **Restriction:** The amount of memory that the Q Capture program can use to build messages is determined when the Q Capture program starts, based on the value of the **`memory_limit`** parameter and the REGION size that is specified in the JCL. The value of **`memory_limit`** cannot be altered with the Q Capture program is running. To change the value you must first stop the Q Capture program.

Procedure

To dynamically change parameters while a Q Capture program is running, use the **`chgparms`** parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services for z/OS.

MODIFY

```
f myqcap,chgparms parameter=value
```

Where `myqcap` is the Q Capture job name.

asnqccmd

```
asnqccmd capture_server=server
capture_schema=schema
chgparms parameters
```

Where *server* is the name of the Q Capture server, *schema* identifies a running Q Capture program, and *parameters* is one or more parameters that you want to change.

When you change the values, a delay of 1 to 2 seconds can occur between the time that you issue the command and the time that a Q Capture program changes its operation.

Changing saved Q Capture parameters in the IBMQREP_CAPPARMS table

A Q Capture program stores its operating parameters in the IBMQREP_CAPPARMS control table. To change the saved parameter values, you must update the control table.

About this task

If you override these saved parameters when you start the program or while it is running, the changes stay only in memory. The next time that you start the Q Capture program, it uses the values saved in the control table.

The IBMQREP_CAPPARMS table contains a single row. If this table has no row, or more than one row, the Q Capture program will not run.

If you want to change one or more saved parameter values, you can update the IBMQREP_CAPPARMS table. Because a Q Capture program reads this table when it starts, you must stop and restart the program for the updates to take effect. Reinitializing a Q Capture program will not prompt the program to read new values in the IBMQREP_CAPPARMS table.

Procedure

To change saved Q Capture parameters in the IBMQREP_CAPPARMS table, use one of the following methods:

Method	Description
Q Replication Dashboard	On the Programs tab, click **Properties** and then select a Q Capture server from the **Program** field to view or change saved values.

SQL	From a command prompt or one of the DB2 command line tools issue an SQL UPDATE statement for the IBMQREP_CAPPARMS table. For example, to change the defaults for **`monitor_interval`** and **`logstdout`**:
	`update` *`schema`*`.ibmqrep_capparms`
	`set monitor_interval=600, logstdout=Y`
	
	Where *schema* identifies the Q Capture program whose saved parameter values you want to change.

Replication Center	Use the Change Parameters – Saved window. To open the window, right-click the Q Capture server that contains the Q Capture program whose saved parameters you want to view or change and select **Change Parameters** > **Saved**. **Note:** Starting with Version 10.1, not all new Q Capture parameters were added to the Replication Center. To see all parameters, use the Q Replication Dashboard.

Prompting a Q Capture program to ignore unwanted transactions

You can specify that a Q Capture program ignore unwanted transactions, and these transactions are not captured for replication or publishing.

About this task

You can specify which transactions to ignore by using one or more of the following identifiers:
- Transaction ID
- Authorization ID
- z/OS Authorization token
- z/OS Plan name

For example, your operation might run very large purge jobs once a month and you prefer to run these jobs against each database rather than replicate the data. In this case you could use a specific and unique authorization ID for the purge jobs, and specify that Q Capture ignore transactions from that ID.

To prompt the Q Capture program to ignore a single transaction based on its transaction identifier, you use an **asnqcap** command parameter when you start the Q Capture program.

To ignore transactions based on authorization ID, authorization token, or plan name, you use SQL to insert the identifiers into the IBMQREP_IGNTRAN control table at the Q Capture server. You can use a wild card character (%) to ignore groups of transactions (see below for more detail).

Caution: Ignoring a transaction that will ultimately be aborted causes no data integrity issues. But ignoring a transaction that was committed at the source server typically causes divergence between source and target servers, and you might need to take other actions to synchronize the servers, for example triggering a new load of the target table or using the asntdiff table compare utility.

Procedure

To prompt the Q Capture program to ignore transactions:

1. Use one of the following methods depending on the identifier you plan to use:

Identifier	Procedure

Transaction ID	Use the **`asnqcap`** command with the **`ignore_transid`** parameter to specify one transaction to be ignored. The format of the command is as follows:
	`asnqcap capture_server=`*`q_capture_server`*
	`capture_schema=`*`q_capture_schema`*
	`ignore_transid=`*`transaction_ID`*
	
	DB2 sources
	The *transaction_ID* is a 10-byte hexadecimal identifier in the following format:
	
	z/OS
	0000:*xxxx*:*xxxx*:*xxxx*:*mmmm*
	
	Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *mmmm* is the data-sharing member ID. You can find the member ID in the last 2 bytes of the log record header in the LOGP output. The member ID is 0000 if data-sharing is not enabled.
	
	For example, the following command specifies that a transaction be ignored in a data-sharing environment, with a member ID of 0001:
	`asnqcap capture_server=sample`
	`capture_schema=ASN`
	`ignore_transid`
	`=0000:BD71:1E23:B089:0001`
	
	Linux UNIX Windows
	
	Version 9.7 and below
	nnnn:0000:*xxxx*:*xxxx*:*xxxx*
	
	Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *nnnn* is the partition identifier for partitioned databases (this value is 0000 for non-partitioned databases).
	
	For example, the following command specifies that a transaction be ignored in a non-partitioned database:
	`asnqcap capture_server=sample`
	`capture_schema=ASN`
	`ignore_transid`
	`=0000:0000:0000:0000:BE97`

Identifier	Procedure

Transaction ID
 	
(Continued)
	Version 10.1 or higher with compatibility of 1001 or higher, or Version 9.8
	0000:*llll*:*xxxx*:*xxxx*:*xxxx*
	
	Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *llll* is the log stream identifier for databases with the DB2 pureScale Feature. The first four characters are always 0000.
	
	For example, the following command specifies that a transaction be ignored in a database with the DB2 pureScale Feature:
	
	`asnqcap capture_server=sample`
	`capture_schema=ASN`
	`ignore_transid`
	`=0000:0001:0000:0000:BE97`
	
	Oracle sources
	The transaction ID is the value of XID from the V$CONTENTS view of the Oracle Log Miner utility. The ID is a RAW(8) value. When displayed in text, it is formatted as a hexadecimal string (16 digits total).
	
	To ignore more than one transaction, stop the Q Capture program and start it in warm mode with another transaction identifier specified.

Identifier	Procedure

Authorization ID, authorization token (z/OS), or plan name (z/OS)	Use SQL to insert one or more of the identifiers into the IBMQREP_IGNTRAN control table. Insert the appropriate identifiers into the following columns:
	
	AUTHID
	The authorization ID.
	Oracle sources: The authorization ID is the value of USERNAME from the V$CONTENTS view of the Oracle Log Miner utility.
	
	z/OS **AUTHTOKEN**
	The authorization token (job name).
	
	z/OS **PLANNAME**
	The plan name.
	
	For example, the following statement specifies that the Q Capture program ignore a transaction with an authorization ID of repldba:
	`insert into schema.IBMQREP_IGNTRAN (`
	`  AUTHID,`
	`  AUTHTOKEN,`
	`  PLANNAME)`
	`values (`
	`  'repldba',`
	`  NULL,`
	`  NULL);`
	
	You can use a wild card character, the percentage sign (%), to represent any number of characters or none, which allows you to ignore groups of authorization IDs, authorization tokens, or plan names.
	
	For example, the following insert statement tells Q Capture to ignore any transaction with plan names that ends with the string "PLAN11":
	`INSERT INTO`
	`IBMQREP_IGNTRAN (AUTHID, AUTHTOKEN, PLANNAME, IGNTRANTRC)`
	`VALUES (null, null, '%PLAN11', 'N')`
	
	This example tells Q Capture to ignore any transaction with an authorization ID that starts with "AUTH" and contains the substring "ID". Also, the authorization token has to match a string starting with "TOK11" (z/OS only):
	`INSERT INTO`
	`IBMQREP_IGNTRAN (AUTHID, AUTHTOKEN, PLANNAME, IGNTRANTRC)`
	`VALUES ('AUTH%ID%', 'TOK11%', null, 'N')`
	
	If the percentage sign is part of the data, use a backslash character (\) to escape the percentage sign. For example, this statement specifies that Q Capture ignore the authorization ID AUTH%ID:
	`INSERT INTO`
	`IBMQREP_IGNTRAN (AUTHID, AUTHTOKEN, PLANNAME, IGNTRANTRC)`
	`VALUES ('AUTH\%ID', null, null, 'N')`
	
	Pattern matching is case sensitive.

2. Optional: Specify whether you want the Q Capture program to insert a row into the IBMQREP_IGNTRANTRC table when it ignores a transaction. By default, this tracing is disabled. To turn on the tracing, use the following SQL statement:

```
update schema.IBMQREP_IGNTRAN set IGNTRANTRC='Y'
WHERE identifier = transaction_identifier
```

The *identifier* would be any one of the AUTHID, AUTHTOKEN, or PLANNAME columns that was used to identify the transaction or transactions to skip.

When you use the Q Apply parameter **`insert_bidi_signal`**=n, setting IGNTRANTRC to N (no tracing) prevents the Q Capture program from inserting a row into the IBMQREP_IGNTRANTRC table for each transaction that it does not recapture and reduces maintenance overhead on the table.

Stopping a Q Capture program

You can stop a Q Capture program, and it will stop reading from the recovery log and building transactions in memory.

About this task

When you stop Q Capture, messages that were put on queues will be committed to WebSphere MQ before the program stops. Uncommitted WebSphere MQ transactions or row changes that were in memory when you stopped the program will be recaptured from the log when the Q Capture program restarts, based on a restart point stored in the restart message.

Tip: You do not need to stop a Q Capture program to add or delete a Q subscription or publication:

- If you want to add one or two Q subscriptions or publications while the program is running, create the Q subscriptions or publications so that they do not start automatically, and then activate them.
- If you want to add a large number of Q subscriptions or publications, create them so that they start automatically, and then reinitialize the Q Capture program.
- You can delete a Q subscription or publication without stopping the Q Capture program by deactivating the Q subscription or publication and then deleting it.

Procedure

To stop a Q Capture program, use one of the following methods:

Method	Description

`stop` parameter	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to stop a Q Capture program:
	
	MODIFY
	`f myqcap,stop`
	
	Where `myqcap` is the Q Capture job name.
	
	asnqccmd
	`asnqccmd capture_server=server_name capture_schema=schema stop`
	
	Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program that you want to stop.

Method	Description

SQL	Use a command prompt or one of the DB2 command-line tools to insert a STOP signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE )`
	` values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'STOP',`
	`    'NULL',`
	`    'P' );`
	
	Where `schema` identifies the Q Capture program that you want to stop.

Windows Windows services	You can create a DB2 replication service on the Windows operating system and use the Windows Service Control Manager or the **`net stop`** command to stop a Q Capture program.

Stopping a Q Capture program at a specified point

You can perform a scheduled and controlled stop of the Q Capture program, for example when you are bringing down a site during a planned outage.

About this task

The ability to stop Q Capture at a specified point ensures that all changes up to that point are delivered or applied to a target before you stop replication. This ability is useful in situations where you are rerouting the replication workload from an active site to a standby site at a specified point in time.

The Q Capture **`stop`** command and STOP signal offer two parameters that you can use for a controlled shutdown:

`captureupto`
: The **`captureupto`** parameter specifies a point when Q Capture will stop reading the recovery log. You can provide a timestamp in the database time zone. Q Capture stops after publishing all transactions that were committed prior to that time. You can also specify the value CURRENT_TIMESTAMP to prompt Q Capture to stop reading the log at the current local time. You can also specify EOL to indicate the Q Capture should stop after it reaches the end of the log.

`stopafter`
: If you specify the **`stopafter`** parameter, Q Capture suspends reading the log until one of two conditions that you specify is true:

  data_applied
  : Q Capture stops after all changes up to the specified stopping point are applied at the target. At this point, applications can be rerouted from the take-down site to the failover site.

  data_sent
  : Q Capture stops after all messages are removed from the

transmission queue that points to the target system, or until the Q Apply program has processed all messages in the case of a shared local send queue-receive queue. For remote configurations, this method ensures that all published messages exist only on the local queue at the target system. At this point, you can perform maintenance such as a WebSphere MQ stoppage on the take-down site without risking loss of messages. The benefit of this mode is that it can operate even if Q Apply is not actively reading from queues (although there must be enough disk space on the failover system for all messages to be delivered).

You can specify **captureupto** and **stopafter** alone or in combination.

The timestamp must be specified in the time zone of the Q Capture server, in a full or partial timestamp format. The full timestamp uses the following format: YYYY-MM-DD-HH.MM.SS.mmmmmm. For example, 2010-04-10-10.35.30.555555 is the local timestamp for April 10, 2010, 10:35 a.m., 30 seconds, and 555555 microseconds. You can specify the partial timestamp in one of the following formats:

YYYY-MM-DD-HH.MM.SS
For example, 2010-04-10-23.35.30 is the partial local timestamp for April 10, 2010, 11:35 p.m., 30 seconds.
YYYY-MM-DD-HH.MM
For example, 2010-04-10-13.30 is the partial local timestamp for April 10, 2010, 1:30 p.m.
YYYY-MM-DD-HH
For example, 2010-04-10-01 is the partial local timestamp for April 10th, 2010, 1:00 a.m.
HH.MM
For example, 14:55 is the partial local timestamp for today at 2:55 p.m.
HH
For example, 14 is the partial local timestamp for today at 2 p.m.

Note: If you specify the **data_sent** parameter and Q Capture detects that another application is sharing the transmission queue, it issues a warning message and stops because it cannot reliably determine when all of its messages are removed from the queue.

Procedure

To stop a Q Capture program at a specified point, use one of the following methods:

Method	Description

stop command with captureupto option, stopafter option, or both	Use these parameters with the MODIFY command on z/OS or with the **asnqccmd** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to stop a Q Capture program at a specified point:
	
	MODIFY
	The command format is as follows:
	`F Q_CAPTURE_JOBNAME,STOP CAPTUREUPTO=timestamp|CURRENT_TIMESTAMP|EOL STOPAFTER=DATA_SENT|DATA_APPLIED`
	
	Where *Q_CAPTURE_JOBNAME* is the name of the Q Capture job.
	
	asnqccmd
	The command format is as follows:
	`asnqccmd capture_server=server_name capture_schema=schema stop captureupto=timestamp|CURRENT_TIMESTAMP|EOL stopafter=data_sent|data_applied`
	
	Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program that you want to stop.
	
	The following examples show different ways of using the controlled-stop parameters:
	`f QCAP1,STOP CAPTUREUPTO=2010-04-05-22:30:00.000000`
	`asnqccmd capture_server=sample capture_schema="bsn" stop captureupto=2010-04-05-22:30:00.000000`
	
	Q Capture publishes all changes committed up to and including 22:30:00 on 2010-04-05.
	`f QCAP1,STOP STOPAFTER=DATA_SENT`
	`asnqccmd capture_server=sample capture_schema="bsn" stop stopafter=data_sent`
	
	Q Capture stops reading the log immediately and waits until all messages have been sent from the transmission queue or local send-receive queue before the program stops.
	`f QCAP1,STOP CAPTUREUPTO=2010-04-05-22:30:00.000000 STOPAFTER=DATA_APPLIED`
	`asnqccmd capture_server=sample capture_schema="bsn" stop captureupto=2010-04-05-22:30:00.000000 stopafter=data_applied`
	
	Q Capture publishes all changes committed up to and including 22:30:00 on 2010-04-05, and waits until Q Apply finishes applying all data that was published up to that time before stopping.

Method	Description
STOP signal with captureupto option, stop after option, or both	Use a command prompt or one of the DB2 command-line tools to insert a STOP signal with one or more of the controlled-stop parameters into the IBMQREP_SIGNAL table at the Q Capture server. The signal format is as follows: `insert into *schema*.IBMQREP_SIGNAL( SIGNAL_TIME, SIGNAL_TYPE, SIGNAL_SUBTYPE, SIGNAL_INPUT_IN, SIGNAL_STATE ) values ( CURRENT TIMESTAMP, 'CMD', 'STOP', '*timestamp*|CURRENT_TIMESTAMP|EOL;DATA_SENT|DATA_APPLIED', 'P' );` Where `schema` identifies the Q Capture program that you want to stop and *timestamp* specifies the time that you want Q Capture to stop reading the log and then stop (or stop after a condition is met).

When Q Capture is waiting for a response from Q Apply or for the transmission queue to drain, it goes into a suspended state with the following characteristics:
- An informational message is issued.
- The log reader is suspended.
- Heartbeat messages are still sent.
- Administration messages are still processed.
- Statistics are still written to the monitor tables.
- Q Capture continues to respond to MODIFY or asnqccmd commands, including reporting the suspended state on a **`status`** command.

Interrupting a suspended Q Capture requires the default **`stop`** command with no parameters, or an operating system signal (for example `f` *`jobname`* `stop` on z/OS or Ctrl-C on Linux, UNIX, and Windows). Signals to the IBMQREP_SIGNALS table to cancel the controlled shutdown do not work because the log reader is suspended.

Starting Q subscriptions

You start Q subscriptions to instruct the Q Capture program to begin capturing changes to source tables and putting the change messages on WebSphere MQ queues.

About this task

Before the Q Capture program can start replicating data from source tables, the Q subscriptions that specify those source tables must be in A (active) or N (new) state. By default, newly created Q subscriptions are in N state and are started automatically when the Q Capture program is started or reinitialized. Follow this procedure to start existing Q subscriptions and put them in A state.

The Q Capture program must be running to read the CAPSTART signal. If the Q Capture program is stopped when you start Q subscriptions, the program processes the signal only if it is warm started. The signal or message will be lost if you use cold start.

If any of the Q subscriptions that you want to start specify an automatic load that uses the EXPORT utility, you must create a password file on the Q Apply server to allow the utility to connect to the Q Capture server.

Procedure

To start Q subscriptions, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the START QSUB command. For example, the following commands set the environment and generate SQL to start the DEPARTMENT0001 Q subscription:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`START QSUB SUBNAME DEPARTMENT0001;`

Q Replication Dashboard	On the Subscriptions tab, select one or more Q subscriptions and click **Start**.
Replication Center	Use the Manage Q Subscriptions window. To open the window, right-click the Q Capture server where the source table for the Q subscription is located and select **Manage** > **Q Subscriptions**.

SQL	Use a command prompt or one of the DB2 command line tools to insert a CAPSTART signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'CAPSTART',`
	`    'subname',`
	`    'P' );`
	
	Where *schema* identifies a Q Capture program, and *subname* is the name of the Q subscription that you want to start.

Stopping Q subscriptions

You stop a Q subscription to instruct the Q Capture program to stop capturing changes for the Q subscription. Stopping lets you delete or suspend activity for Q subscriptions without stopping the Q Capture program.

Before you begin

The Q Capture program must be running to read the CAPSTOP signal. If the Q Capture program is stopped when you stop a Q subscription, the program will process the signal only if it is warm started. The signal will be lost if you use cold start.

About this task

When you stop a Q subscription, the Q Capture program stops capturing changes for the Q subscription and changes its state to I (inactive) in the IBMQREP_SUBS table.

Procedure

To stop Q subscriptions, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the STOP QSUB command. For example, the following commands set the environment and generate SQL to stop the DEPARTMENT0001 Q subscription:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	
	`STOP QSUB SUBNAME DEPARTMENT0001;`

Q Replication Dashboard	On the Subscriptions tab, select one or more Q subscriptions and click **Stop**.
Replication Center	Use the Manage Q Subscriptions window. To open the window, right-click the Q Capture server where the source table for the Q subscription is located and select **Manage** > **Q Subscriptions**.

SQL	Use a command prompt or one of the DB2 command-line tools to insert a CAPSTOP signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into schema.IBMQREP_SIGNAL (`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` )  values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'CAPSTOP',`
	`     'subname',`
	`     'P' );`
	
	Where *schema* identifies a Q Capture program, and *subname* is the name of the Q subscription that you want to stop.

Starting publications

When you start publications, the Q Capture program starts capturing source changes and putting the change messages on WebSphere MQ queues.

Before you begin

- The Q Capture program must be running to read the CAPSTART signal or `activate subscription` message. If the Q Capture program is stopped when you

start publications, it will process the signal or message only if it is warm started. The signal or message will be lost if you use cold start.
- Before the Q Capture program can start publishing changes from source tables, the publications that specify those source tables must be in A (active) or N (new) state.

About this task

By default, newly created publications are in N (new) state, and they are automatically started when the Q Capture program is started or reinitialized. If you change this default, you must start publications after you create them.

Procedure

To start publications, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **START PUB** command. For example, the following commands set the environment and generate SQL to start the publication EMPLOYEE0001:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	
	`START PUB PUBNAME EMPLOYEE0001;`

Replication Center	Use the Manage Publications window. To open the window, right-click the Q Capture server where the source table for the publication is located and select **Manage** > **Publications**.

SQL	Use a command prompt or one of the DB2 command line tools to insert a CAPSTART signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'CAPSTART',`
	`    'pubname',`
	`    'P' );`
	
	Where *schema* identifies a Q Capture program and *pubname* is the name of the publication that you want to start.

Stopping publications

You stop a publication to instruct the Q Capture program to stop capturing changes for the publication. Stopping lets you delete or suspend activity for publications without stopping the Q Capture program.

Before you begin

The Q Capture program must be running to read the CAPSTOP signal. If the Q Capture program is stopped when you stop a publication, it will process the signal only if it is warm started. The signal will be lost if you use cold start.

About this task

When you stop a publication, the Q Capture program stops capturing changes for the publication and changes its state to I (inactive) in the IBMQREP_SUBS table.

Procedure

To stop publications, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **`STOP PUB`** command. For example, the following commands set the environment and generate SQL to stop the publication EMPLOYEE0001:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	
	`STOP PUB PUBNAME EMPLOYEE0001;`

Replication Center	Use the Manage Publications window. To open the window, right-click the Q Capture server where the source table for the publication is located and select **Manage** > **Publications**.

SQL	Use a command prompt or one of the DB2 command-line tools to insert a CAPSTOP signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into` *`schema`*`.IBMQREP_SIGNAL (`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` )  values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'CAPSTOP',`
	`     '`*`pubname`*`',`
	`     'P' );`
	
	Where *schema* identifies a Q Capture program and *pubname* is the name of the publication that you want to deactivate.

Managing Q Capture message activity at the send queue level

You can manage Q Capture message activity at the send queue level to prevent outages when an error occurs on a queue, to prevent the need for a full refresh (new load) of replicated tables because of a queue error, and to guarantee delivery of published events.

The Q Capture program maintains individual restart information for each send queue. If a queue stops because of an error or for scheduled maintenance, Q Capture can replay changes that occurred while the queue was stopped. This feature eliminates the need to reload target tables that use a disabled send queue.

You can also set up the Q Capture program so that it continues to put messages on send queues when one or more queues are disabled. Q Capture provides commands to stop putting messages on individual send queues so that you can fix errors or perform maintenance, and to restart selected send queues when you are ready to resume replication or publishing.

The following sections provide more detail on Q Capture features that allow you to manage individual send queues:
- "Using multiple send queues"
- "Setting the error action for send queues to Q" on page 254
- "Keeping disabled queues inactive when Q Capture starts" on page 254
- "Send queue information in the restart message" on page 254

Using multiple send queues

Typically, you configure Q Capture to spread messages over multiple send queues for two reasons:

Dividing the replication workload by application
Changes from the tables that service each application are replicated independently of the changes for other application tables, minimizing CPU and administration overhead.

Data distribution
A single Q Capture program captures the changes once, and then writes them onto each send queue for delivery to its associated remote server.

Figure 28 shows a unidirectional replication setup where the changes are divided among four send queues, each queue handling the activity from the source tables that serve a different application.

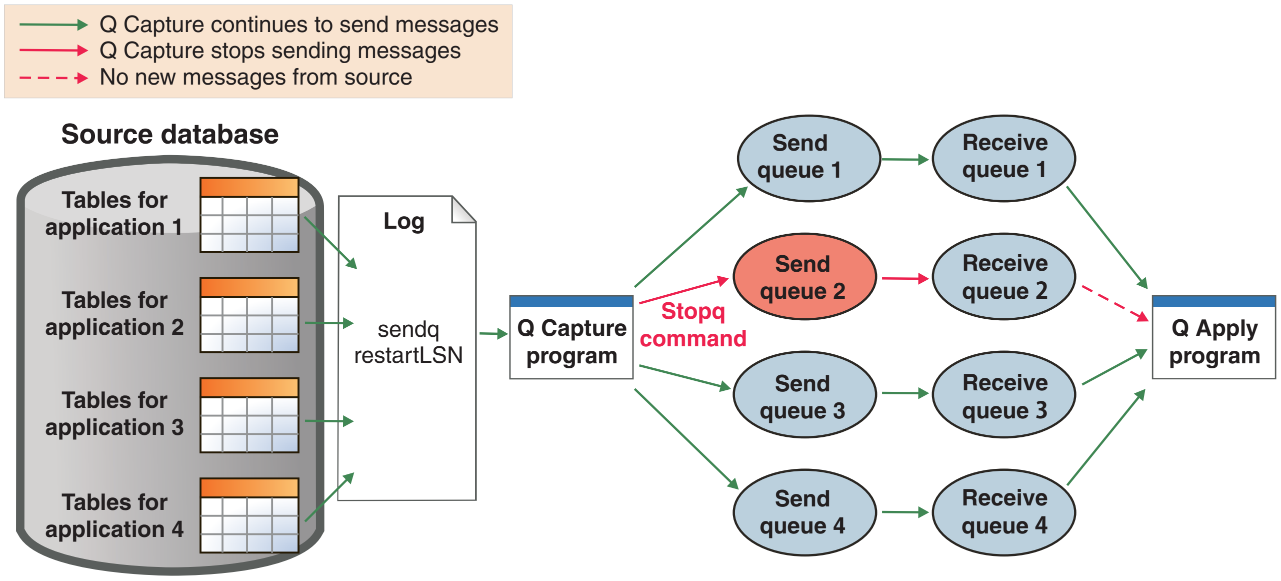


Figure 28. Managing message activity for a single send queue. The stopq command was issued for Send queue 2, which prompts the Q Capture program to stop putting messages on the queue. While activity at the source is stopped, you can perform maintenance on the tables for application 2 without stopping replication for applications 1, 3, and 4. After maintenance is complete, you can issue the **startq** command and Q Capture finds the restart point for Send queue 2 (sendqrestartLSN) in the recovery log, and begins capturing changes for the now-active send queue at that point.

Setting the error action for send queues to Q

To prevent a single send queue failure from affecting replication or publishing on other queues, set the error action for the replication or publishing queue maps that specify the queues to `Q` (**Stop send queue** in the Replication Center). With this error action, when one send queue is disabled (for example by running out of space for messages), the Q Capture program stops putting messages on the disabled queue, but continues to put messages on the other send queues.

After the problem on the disabled send queue is resolved, you can use the **startq** command to prompt Q Capture to resume reading the recovery log for Q subscriptions or publications that use the send queue, and to resume putting messages on the queue.

Keeping disabled queues inactive when Q Capture starts

By default, the Q Capture program activates all inactive send queues when it starts, or when you use the Q Capture **reinit** command to reload all Q subscriptions from the Q Capture control tables. If you need to keep any disabled send queues in inactive (`I`) state, you can start Q Capture with the **startallq** parameter set to N.

Send queue information in the restart message

Q Capture maintains a restart message on its restart queue to keep track of restart points in the recovery log. Starting with Version 9.5 Fix Pack 3, the message contains restart information for each send queue.

Important: If your Q Capture program is at Version 9.5 Fix Pack 3 or newer (on z/OS this is Version 9.1 with the PTF that corresponds to V9.5 Fix Pack 3), the program can start in warm mode by using restart information from an older level of Q Capture. But Q Capture programs that are older than Version 9.5 Fix Pack 3 cannot use the new restart message format to restart in warm mode. If you need to restart an older level of Q Capture in warm mode, you must start Q Capture from a known point in the log by specifying the **lsn** and **maxcmtseq** parameters. Look for message ASN7109I in the Q Capture diagnostic log. For more details, see “Starting Q Capture from a known point in the DB2 log” on page 231.

You can format the newer restart message by using the **asnqmfmt** or **asnqxmfmt** commands. The following example shows a restart message that contains restart information for three send queues, Q1, Q2, and Q3.

```
**** Descriptor for message number: 1
    StrucId         MD
    MsgId           414d5120535a5051544553542020202048c1cd6620004e02
    CorrelId        51505542214d41494e4c494e450000000000000000000000
    Version         1
    Report          0
    MsgType         8
    Expiry          -1
    Feedback        0
    Encoding        273
    CodedCharSetId  819
    Format          COMPACT
    Priority        0
    Persistence     1
**** Message number: 1
**** Message size:   640
  qRestartMsg for MAINLINE log reader.
  qRestartMsg.capServer : QTEST
qRestartMsg.capSchema : ASN
qRestartMsg.qRestartsgSize : 640
Number of partition at restart: 1
restartLSN=0000:0000:0000:1e46:2be0 nodeId=0
qRestartMsg.lastCommitSEQ:  0000:0000:0000:0000:0000
qRestartMsg.lastCommitTime: 2008-09-11-17.22.42.287478
qRestartMsg.reuseSEQ:       0000:0000:0000:0000:0000
qRestartMsg.reuseTime:      2008-09-11-17.22.42
Number of send queues at restart: 3
[0] sendq name:            Q2
    sendq activation time:  1221178963
    sendq next msg seq no:  00000000000000000000000000000003
    sendq lastCommitSEQ:    0000:0000:0000:0000:0000
    sendq lastCommitTime:   2008-09-11-17.22.42.287478
    sendq restartLSN=0000:0000:0000:1e46:2be0 nodeId=0
[1] sendq name:            Q1
    sendq activation time:  1221178963
    sendq next msg seq no:  00000000000000000000000000000003
    sendq lastCommitSEQ:    0000:0000:0000:0000:0000
    sendq lastCommitTime:   2008-09-11-17.22.42.287478
    sendq restartLSN=0000:0000:0000:1e46:2be0 nodeId=0
[2] sendq name:            Q3
    sendq activation time:  1221178963
    sendq next msg seq no:  00000000000000000000000000000003
    sendq lastCommitSEQ:    0000:0000:0000:0000:0000
    sendq lastCommitTime:   2008-09-11-17.22.42.287478
    sendq restartLSN=0000:0000:0000:1e46:2be0 nodeId=0
```

Starting message activity on one or more send queues

You can prompt the Q Capture program to resume putting messages on one or more inactive send queues without needing to do a full refresh (new load) for target tables that use the queue.

About this task

The ability to start one or more send queues is useful in situations where you set the error action for the replication or publishing queue map to `Q`, which tells the Q Capture program to continue putting messages on active queues if an error occurs on one or more other send queues.

When you start message activity on one or more send queues, Q Capture finds the restart point in the recovery log for any queues that were stopped and can rapidly catch up those queues with active queues. Because Q Capture keeps Q subscriptions active while the queues are stopped, when the queues are started there is no need to perform a full refresh (load) of target tables.

Note: While Q Capture is bringing restarted queues up to date, overall latency is affected because Q Capture does not put messages on other queues until it reads the log past the maximum commit point for these already-active queues.

Procedure

To start one or more send queues, use one of the following methods:

Method	Description

startq command	You can use the z/OS **MODIFY** command or **asnqccmd** command to issue **startq**. The command enables you to start a single send queue or all of the queues that a Q Capture program works with.
	
	In the examples below, the first MODIFY command prompts Q Capture to resume putting messages on the send queue asn.qm1.dataq. The second command restarts all queues:
	`f myqcap,startq=asn.qm1.dataq`
	
	`f myqcap,startq=all`
	
	The following commands use **asnqccmd** for the same actions:
	`asnqccmd capture_server=sourcedb capture_schema="asn"`
	`startq="asn.qm1.dataq"`
	
	`asnqccmd capture_server=sourcedb capture_schema="asn"`
	`startq=all`

Reinitializing the Q Capture program	The Q Capture **reinit** command starts inactive send queues if the Q Capture program was started with the **startallq**=y option (the default).
Stopping and starting the Q Capture program	When Q Capture starts, by default it resets the state of all send queues to active (A). Stopping and starting Q Capture also prompts the program to refresh all Q subscriptions or publications from its control tables and find any changes that you made. You can start Q Capture with the **startallq**=n parameter to keep any disabled send queues in inactive (I) state.
Starting a Q subscription	Starting a Q subscription starts an inactive send queue if the Q subscription that you are starting is the only active Q subscription that is using the inactive send queue.

SQL	Use a command prompt or one of the DB2 command line tools to insert a STARTQ signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into` *schema*`.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'STARTQ',`
	`    '`*send_queue_name*`',`
	`    'P' );`
	
	Where *schema* identifies a Q Capture program, and *send_queue_name* is the name of the send queue that you want to start.

When Q Capture receives the command to start message activity on one or more send queues, it takes the following actions:

- Changes the state of the queues to active (A) in the IBMQREP_SENDQUEUES table.
- Gets the restart information for the queues from its restart message.

- Resumes reading the log at the oldest restart point among all send queues, putting messages only on the restarted send queues until they catch up with all other active send queues and all queues have the same restart point

Stopping message activity on one or more send queues

You can prompt the Q Capture program to stop putting messages on selected send queues when you need to temporarily stop replication or publishing on one or more queues.

About this task

For example, you might need to fix a disabled queue or stop message activity while you perform WebSphere MQ administrative tasks.

When you stop message activity on one or more send queues, the Q Capture program continues to replicate or publish changes for active Q subscriptions or publications that use other send queues. Stopping selected queues can be preferable to stopping the Q Capture program when a queue error occurs or when maintenance is required.

You can use the **`startq`** command to prompt Q Capture to resume putting messages on the stopped queues. Because Q Capture keeps restart information for each queue, a new load (full refresh) of target tables is not required when you use this method.

Procedure

To stop message activity on a send queue, use one of the following methods:

Method	Description

stopq command	You can use the z/OS **MODIFY** command or **asnqccmd** command to issue **stopq**. The command enables you to stop a single send queue or all of the queues that a Q Capture program works with.
	
	In the examples below, the first MODIFY command prompts Q Capture to stop putting messages on the send queue asn.qm1.dataq. The second command stops message activity on all send queues:
	`F MYQCAP,STOPQ=ASN.QM1.DATAQ`
	
	`F MYQCAP,STOPQ=ALL`
	
	The following commands use **asnqccmd** for the same actions:
	`asnqccmd capture_server=sourcedb capture_schema="asn"`
	`stopq="asn.qm1.dataq"`
	
	`asnqccmd capture_server=sourcedb capture_schema="asn"`
	`stopq=all`
	
	When you specify to stop an individual queue, you can also specify the **captureupto** parameter, **stopafter** parameter, or both to instruct Q Capture to stop putting messages on the queue in a controlled manner from a specified point:
	
	captureupto
	Specify with a full or partial timestamp to instruct Q Capture to publish up to the specified point and then stop putting messages on the queue. Alternatively, you can specify CURRENT_TIMESTAMP, or specify EOL to prompt Q Capture to stop putting messages on the queue after it reaches the end of the active log.
	
	stopafter
	Specifies that Q Capture stop putting messages on the queue after one of the following conditions is true:
	
	data_applied
	All changes up to the specified stopping point are applied at the target.
	
	data_sent
	All messages are removed from the transmission queue that points to the target system, or the Q Apply program has processed all messages in the case of a shared local send queue-receive queue.
	For example, the following MODIFY and asnqccmd commands instruct Q Capture to publish up to 12:00 and then stop putting messages on the ASN.QM1.DATAQ send queue.
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=12:00`
	
	`asnqccmd capture_server=sourcedb capture_schema="asn"`
	`stopq="asn.qm1.dataq" captureupto=12:00`

Method	Description

stopq command
 	
(Continued)
	The following syntax can also be used with **`stopq`** and MODIFY:
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=EOL`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=CURRENT_TIMESTAMP`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ STOPAFTER=DATA_APPLIED`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ STOPAFTER=DATA_SENT`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=EOL STOPAFTER`
	`=DATA_APPLIED`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=EOL STOPAFTER`
	`=DATA_SENT`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=CURRENT_TIMESTAMP`
	` STOPAFTER=DATA_APPLIED`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=CURRENT_TIMESTAMP`
	` STOPAFTER=DATA_SENT`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=12:00 STOPAFTER`
	`=DATA_APPLIED`
	`F QCAP1,STOPQ=ASN.QM1.DATAQ CAPTUREUPTO=12:00`
	`STOPAFTER=DATA_SENT`

SQL	Use a command prompt or one of the DB2 command line tools to insert a STOPQ signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into schema.IBMQREP_SIGNAL(`
	`    SIGNAL_TIME,`
	`    SIGNAL_TYPE,`
	`    SIGNAL_SUBTYPE,`
	`    SIGNAL_INPUT_IN,`
	`    SIGNAL_STATE`
	`) values (`
	`     CURRENT TIMESTAMP,`
	`    'CMD',`
	`    'STOPQ',`
	`    'send_queue_name',`
	`    'P' );`
	
	Where *schema* identifies a Q Capture program, and *send_queue_name* is the name of the send queue that you want to stop.

After the Q Capture program receives the command, it takes the following actions:

- Stops publishing changes for all active Q subscriptions or publications that are associated with stopped send queues.
- Continues to put messages on active send queues.
- Changes the state of the stopped queues to inactive (I) in the IBMQREP_SENDQUEUES table.
- Stops updating restart information in its restart message for any send queues that are stopped.

If all send queues are stopped, the Q Capture program continues reading the log for signals such as CAPSTART, continues to insert into its monitor tables, and waits for commands.

Chapter 16. Operating a Q Apply program

A Q Apply program reads messages that contain transactions from source tables and applies them to targets that are defined by Q subscriptions. You can operate the Q Apply program by using the Replication Center, system commands, and system services, and you can change the Q Apply operating parameters in several ways.

Starting a Q Apply program

When you start a Q Apply program, it begins reading transaction messages from queues and applying the transactions to target tables or stored procedures.

Before you begin

- If you are starting the Q Apply program from a remote workstation, configure connections to the Q Apply server.
- Create and configure a WebSphere MQ queue manager, queues and other necessary objects.
- Ensure that you have authorization for Q Replication objects and WebSphere MQ objects.
- Create Q Apply control tables.
- Configure the target database or subsystem to work with the Q Apply program.
- If you have Q subscriptions that specify an automatic load that uses the EXPORT utility, create a password file so that Q Apply can connect to the Q Capture server.

About this task

When you start a Q Apply program, you can specify startup parameters and the program will use the new values until you take one of the following actions:
- You change the parameter values while the program is running
- You stop and restart the program, which prompts it to read the IBMQREP_APPLYPARMS table and use the values saved there.

Procedure

To start a Q Apply program, use one of the following methods:

Method	Description

z/OS
 	
z/OS console or TSO
	On z/OS, you can start a Q Apply program by using JCL or as a system-started task. You can specify new invocation parameter values when you start a Q Apply program with JCL.
	
	z/OS has a 100-byte limit for the total length of parameters that you can specify in the PARMS= field. To overcome this limitation, replication programs now allow you to specify as many additional parameters as needed in the SYSIN data set.
	
	When the SYSIN DD statement is included in the invocation JCL, the Q Apply program automatically concatenates what is specified in the SYSIN dataset to the PARMS= parameters. You can only specify Q Apply parameters in the SYSIN data set. Any LE parameters must be specified in the PARMS= field or in LE _CEE_ENVFILE=DD, followed by a slash(/).
	
	Example:
	`//* asterisk indicates a comment line`
	`// QAPP EXEC PGM=ASNQAPP,PARMS='LE/Q Apply parameters'`
	`//* Parameters can be any or no LE parameters and any or`
	`//* no Q Apply parameters`
	`//SYSIN DD *`
	`//* additional Q Apply parameters, one or more`
	`//* parameters on each line`
	`  APPLY_SERVER=DSN!! APPLY_SCHEMA=APPCAT`
	`  DEBUG=Y LOGSTDOUT=N`

`asnqapp` command	From a command line, use the **`asnqapp`** command to start a Q Apply program and optionally specify startup parameters:
	`asnqapp apply_server=`*`server_name`*
	`apply_schema=`*`schema`*
	`parameters`
	
	Where *server_name* is the name of the database or subsystem where the Q Apply control tables are defined and where the Q Apply program will apply changes to targets, *schema* identifies the Q Apply program that you want to start, and *parameters* is one or more parameters that you can specify at startup.

Windows Windows Service	You can create a replication service on the Windows operating system to start the Q Apply program automatically when the system starts.

You can verify whether a Q Apply program started by using one of the following methods:

- Examine the Q Apply diagnostic log file (*`apply_server.apply_schema`*`.QAPP.log` on z/OS and *`db2instance.apply_server.apply_schema`*`.QAPP.log` on Linux, UNIX, and Windows) for a message that indicates that the program is capturing changes.
- Check the IBMQREP_APPLYTRACE table for a message that indicates that the program is capturing changes.
- z/OS If you are running in batch mode, examine the z/OS console or z/OS job log for messages that indicate that the program started.
- Use the Q Apply Messages window in the Replication Center to see a message that indicates that the program started. To open the window, right-click the Q Apply server that contains the Q Apply program whose messages you want to view and select **Reports** > **Q Apply Messages**.

Changing the Q Apply parameters

You can change the Q Apply operating parameters when you start the program, while the program is running, or by updating the IBMQREP_APPLYPARMS control table.

For a brief description of the three different ways that you can change the parameters and an example to help clarify the differences, see "Methods of changing the Q Capture operating parameters" on page 236. The methods are the same for a Q Apply program.

Changing parameters while a Q Apply program is running

You can modify the behavior of a Q Apply program while it continues to apply transactions to targets. The Q Apply program begins using the new settings almost immediately, but the changes are not saved in the IBMQREP_APPLYPARMS control table. If you stop and then restart the Q Apply program, it uses the values saved in the control table.

About this task

You can change the following Q Apply parameters while the program is running:

- **`autostop`**
- **`logreuse`**
- **`logstdout`**
- **`monitor_interval`**
- **`monitor_limit`**
- **`prune_interval`**
- **`term`**
- **`trace_limit`**
- **`deadlock_retries`**

Procedure

To dynamically change parameters while a Q Apply program is running, use the **`chgparms`** parameter with the **`MODIFY`** command on z/OS or **`asnqacmd`** command on Linux, UNIX, Windows, and UNIX System Services for z/OS

MODIFY

```
f myqapp,chgparms parameter=value
```

Where `myqapp` is the Q Apply job name.

asnqacmd

```
asnqccmd apply_server=server
apply_schema=schema
chgparms parameters
```

Where *server* is the name of the Q Apply server, *schema* identifies a running Q Apply program, and *parameters* is one or more parameters that you want to change.

A delay of 1 to 2 seconds can occur between the time that you issue the command and the time that a Q Apply program changes its operation.

Changing saved Q Apply parameters in the IBMQREP_APPLYPARMS table

A Q Apply program stores its operating parameters in the IBMQREP_APPLYPARMS control table. If you override these saved parameters when you start the program or while it is running, the changes stay only in memory. The next time that you start the Q Apply program, it will use the values saved in the control table. To change the saved parameter values, you must update the control table.

About this task

The IBMQREP_APPLYPARMS table contains a single row. If this table has no row, or more than one row, the Q Apply program will not run.

If you want to change one or more saved parameter values, you can update individual columns in the IBMQREP_APPLYPARMS table. Because a Q Apply program reads this table when it starts, you must stop and restart the program for changes to take effect.

Procedure

To change saved parameters for a Q Apply program in the IBMQREP_APPLYPARMS table:

Use one of the following methods:

Option	Description
Q Replication Dashboard	On the Programs tab, click **Properties** and then select a Q Apply server from the **Program** field to view or change saved values.

SQL	From a command prompt or one of the DB2 command line tools, issue an SQL UPDATE statement for the IBMQREP_APPLYPARMS table. For example, to change the defaults for prune_interval and deadlock_retries:
	`update schema_name.ibmqrep_applyparms`
	`set prune_interval=600,`
	`deadlock_retries=10`
	
	Where *schema* identifies the Q Apply program whose saved parameter values you want to change.

Replication Center	Use the Change Parameters – Saved window to view or change any of the values in the IBMQREP_APPLYPARMS table. To open the window, right-click the Q Apply server that contains the Q Apply program whose saved parameters you want to view or change and select **Change Parameters** > **Saved**. **Note:** Starting with Version 10.1, not all new Q Apply parameters were added to the Replication Center. To see all parameters, use the Q Replication Dashboard.

Stopping a Q Apply program

You can stop a Q Apply program by using the Replication Center or system commands.

About this task

When you stop a Q Apply program, it takes the following actions:
- Stops reading messages from all receive queues
- Rolls back transactions that have been partially applied to targets but not committed
- Shuts down in an orderly manner

While a Q Apply program is stopped, messages from running Q Capture programs continue to collect on receive queues. When you start the Q Apply program again, it begins reading these messages, and re-reads any messages that contain rolled-back transactions. The program then goes back to applying transactions to targets. Transactions are applied only once, and no replicated data is lost.

Attention:
- If you stop the Q Apply program while a target table is being loaded, make sure that no applications are allowed to update the target table until the Q Apply program is started again and the table is loaded. When you restart the program, it deletes the contents of the target table and then starts loading the table again. Any updates to target tables that occur while the Q Apply program is stopped are lost.
- During the loading process, the Q Apply program drops referential integrity constraints on target tables. These constraints are not reapplied until the program starts again and the table is loaded. Any updates to target tables that occur while the Q Apply program is stopped will not be checked for referential integrity.

If you want to stop a Q Apply program from reading messages from a single receive queue, use the **`asnqacmd stopq`** command.

Procedure

To stop a Q Apply program, use one of the following methods:

Method	Description

`stop` **parameter**	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqacmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to stop a Q Apply program:
	
	MODIFY
	`f myqapp,stop`
	
	Where `myqapp` is the Q Apply job name.
	
	asnqacmd
	`asnqacmd apply_server=`*`server_name`*
	`apply_schema=`*`schema`* `stop`
	
	Where *server_name* is the name of the database or subsystem where the Q Apply program is running, and *schema* identifies the Q Apply program that you want to stop.

Windows Windows Service	You can create a DB2 replication service on the Windows operating system and use the Windows Service Control Manager or the **`net stop`** command to stop a Q Apply program.

Stopping message processing on a receive queue

You can use the **`stopq`** parameter or Replication Center to prompt a Q Apply program to stop message processing on a receive queue.

Before you begin

- The Q Apply program must be running.
- The receive queue must be in A (active) state.

About this task

When you stop processing for one queue:

- The Q Apply program continues to apply transactions from other receive queues.
- If any Q subscriptions that use the queue are active, messages continue to arrive on the receive queue for which processing was stopped.

You can start processing messages on the receive queue again, and no messages will be lost.

If you stop message processing on a receive queue, the Q Apply program changes the state of the queue to I (inactive) in the IBMQREP_RECVQUEUES table. Even if the Q Apply program is stopped and then restarted, it will resume reading from the queue only if you issue an **`asnqacmd startq`** command.

When you stop processing messages on a receive queue, the Q Apply program finishes applying in-memory transactions from the queue. The Q Apply program rolls back transactions that were partially applied to targets. Transactions that are rolled back will be processed the next time that the Q Apply program is started. No data is lost.

Procedure

To stop message processing on a receive queue:

Use one of the following methods:

Method	Description

`stopq` parameter	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqacmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to stop message processing on a receive queue:
	MODIFY
	`f myqapp,stopq`
	Where `myqapp` is the Q Apply job name.
	asnqacmd
	`asnqacmd apply_server=`*`server_name`* `apply_schema=`*`schema`* `stopq`
	Where *server_name* is the name of the database or subsystem where the Q Apply program is running, and *schema* identifies the Q Apply program for which you want to stop message processing.

Replication Center	Use the Manage Receive Queues window. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage** > **Receive Queues**.

To verify that the Q Apply program stopped reading messages from the queue:
- Use the Manage Receive Queues window to see whether the state of the queue changed to I (inactive).
- Check the Q Apply diagnostic log file for a message that indicates that processing stopped for the queue.
- Look at the IBMQREP_RECVQUEUES control table to see whether the state of the queue changed to I (inactive) in the STATE column:

```
SELECT RECVQ, STATE FROM schema.IBMQREP_RECVQUEUES
WHERE RECVQ = 'receive_queue_name';
```

Starting message processing on a receive queue

You can use the Replication Center or a command to instruct the Q Apply program to start processing messages on a receive queue.

Before you begin
- The Q Apply program must be running.
- The receive queue must be in I (inactive) state.

About this task

You might need to instruct a Q Apply program to start processing messages on a receive queue for several reasons:
- A conflict, SQL error, or persistent deadlocks occurred at the target, which prompted the Q Apply program to stop reading from the receive queue.
- You instructed the Q Apply program to stop processing messages from the receive queue.

Note: You can use the Replication Center or the **startallq** invocation parameter to prompt the Q Apply program to start reading from all receive queues when the program starts, even if the queues are in inactive (I) state.

Procedure

To start message processing on a receive queue:

Use one of the following methods:

Method	Description

startq parameter	Use this parameter with the **MODIFY** command on z/OS or **asnqacmd** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to start message processing on a receive queue:
	
	MODIFY
	`f myqapp,startq`
	
	Where `myqapp` is the Q Apply job name.
	
	asnqacmd
	`asnqacmd apply_server=`*`server_name`*
	`apply_schema=`*`schema`*` startq`
	
	Where *server_name* is the name of the database or subsystem where the Q Apply program is running, and *schema* identifies the Q Apply program for which you want to start message processing.

Replication Center	Use the Manage Receive Queues window. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage** > **Receive Queues**.

To verify that the Q Apply program started reading messages from the queue:
- Use the Manage Receive Queues window to see whether the state of the queue changed to A (active).
- Check the Q Apply diagnostic log file for a message that indicates that processing started for the queue.
- Look at the IBMQREP_RECVQUEUES control table to see whether the state of the queue changed to A (active) in the STATE column:

```
SELECT RECVQ, STATE FROM schema.IBMQREP_RECVQUEUES
WHERE RECVQ = 'receive_queue_name';
```

Prompting a Q Apply program to ignore transactions

You can prompt a Q Apply program to ignore transactions, and these transactions are taken off the receive queue but not applied to target tables.

About this task

To ignore one or more transactions, you use an **asnqapp** command parameter when you start the Q Apply program. You can also specify to ignore one or more transactions on a receive queue when you start the receive queue by using the **startq** command.

Stopping the program from applying transactions is useful in unplanned situations, for example:

- Q Apply receives an error while applying a row of a transaction and either shuts down or stops reading from the receive queue. On startup, you might want Q Apply to skip the entire transaction in error.
- After the failover from a disaster recovery situation, you might want to skip a range of transactions on the receive queue from the failover node to the fallback node.
- A large number of DELETE operations are replicated, slowing Q Apply processing. If you do not need to delete the rows from the target table, you could skip the transactions that contain the deleted rows and improve overall performance.

You can also prompt the Q Capture program to ignore transactions. This action is more typical when you can plan which transactions do not need to be replicated.

Note: Ignoring a transaction that was committed at the source server typically causes divergence between tables at the source and target. You might need to take other actions to synchronize the tables.

Restrictions

The ability to ignore transactions at the target is not supported for Classic replication sources.

Procedure

To prompt a Q Apply program to ignore transactions, use one of the following methods:

Situation	Method

On program startup	Use the **asnqapp** command with the **skiptrans** parameter to prompt Q Apply to not apply one or more transactions from one or more receive queues when it initializes. The format of the command is as follows:
	`asnqapp apply_server=`*`server`*` apply_schema=`*`schema`*
	`skiptrans=`*`receive queue name`*`;`*`transaction_ID`*
	
	You can find *transaction_ID* either in the qTransMsgHeader.uow_id entry in the **asnqmfmt** command output for a receive queue or in the SRC_TRANS_ID column of the IBMQREP_EXCEPTIONS table if a transaction in error needs to be ignored. The ID is a 10-byte hexadecimal identifier in the following format:
	
	z/OS
	0000:*xxxx*:*xxxx*:*xxxx*:*mmmm*
	
	Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *mmmm* is the data-sharing member ID. You can find the member ID in the last 2 bytes of the log record header in the LOGP output. The member ID is 0000 if data sharing is not enabled.
	
	For example, the following JCL specifies that a transaction be ignored in a data-sharing environment, with a member ID of 0001:
	`//QAPP EXEC PGM=ASNQAPP,`
	`//PARM='/APPLY_SERVER=DSN1 APPLY_SCHEMA=APPCAT`
	`//SKIPTRANS=Q1;0000:1171:12c3:0890:0001`
	
	If the allowed command-line limit is exceeded by the above syntax, you can also use the SYSIN data set to specify the transactions to skip. The following sample DD statement for the Q Apply job specifies to ignore one transaction on the queue Q1 and a range of transactions on the queue Q3:
	`//SYSIN DD *`
	`SKIPTRANS=Q1;0000:1171:12c3:0890:000,`
	`Q3;0000:0000:0000:79bc:0000-0000:0000:0000:79fc:0000`
	
	Linux UNIX Windows
	nnnn:0000:*xxxx*:*xxxx*:*xxxx*
	
	Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *nnnn* is the partition identifier for partitioned databases (this value is 0000 for non-partitioned databases).
	
	For example, the following command specifies that a transaction be ignored in a non-partitioned database:
	`asnqapp capture_server=target`
	`apply_schema=ASN`
	`skiptrans="recvq1;0000:0000:0000:0000:BE97"`
	You can also specify to skip a range of transactions by separating the transaction IDs with a hyphen, for example:
	`skiptrans=`
	`"Q1;0000:0000:0000:0000:51b0-0000:0000:0000:0000:51g0"`

Situation	Method

While starting a receive queue	When you start a receive queue with the **startq** parameter of the **asnqacmd** or **MODIFY** command, specify the **skiptrans** parameter to skip one or a range of transactions from the receive queue that you are starting. The format of the command for asnqacmd is as follows:
	`asnqacmd apply_server=`*`server`*` apply_schema=`*`schema`*
	`startq=`*`queue_name`*`;skiptrans="`*`transaction_ID`*`"`
	
	For example, the following command prompts Q Apply to ignore one transaction on receive queue Q1:
	`asnqacmd apply_server=target apply_schema=ASN`
	`startq=Q1;skiptrans="0000:0000:0000:0000:51a1"`

When one or more transactions are successfully ignored, the Q Apply program issues messages ASN7670I or ASN7671I in its log and in the IBMQREP_APPLYTRACE table.

Note about target table loading: If you start Q Apply with the **skiptrans** parameter, transactions that are part of an automatic target table load (one that is performed by Q Apply) are also not applied. If you specified a manual load, these transactions could be applied to target tables as part of a load operation that is external to the receive queue. You might need to manually delete these rows from the target if required.

Chapter 17. Changing a Q Replication environment

You can change the properties of replication objects such as Q subscriptions and replication queue maps after you define them, in most cases without stopping replication.

Changing the properties of unidirectional Q subscriptions

You can change the properties of Q subscriptions without stopping replication. First you make your changes. Then you reinitialize either a single Q subscription or reinitialize the Q Capture program if you are changing multiple Q subscriptions.

Restrictions

Not all properties of Q subscriptions can be changed.

- If you are using the ASNCLP command-line program to make changes, check the syntax of the ALTER QSUB command for unidirectional replication to see which properties you can change.
- In the Replication Center, controls are disabled on the Q Subscription Properties notebook for properties that you cannot change.

About this task

If you reinitialize a Q Capture program, this action will also reinitialize any publications that are defined within the Q Capture schema. Reinitializing a Q Capture program also requires more system resources.

If you only need to change the properties of one or two unidirectional Q subscriptions, it is less costly in terms of resources to reinitialize one Q subscription at a time.

Procedure

To change the properties of unidirectional Q subscriptions without stopping replication:

1. Change one or more Q subscriptions. Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the ALTER QSUB command. For example, the following commands set the environment and change the unidirectional EMPLOYEE0001 Q subscription at the SAMPLE server:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET RUN SCRIPT NOW STOP ON SQL ERROR ON;`
	`SET SERVER CAPTURE TO SAMPLE;`
	`SET SERVER TARGET TO TARGET;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`SET APPLY SCHEMA ASN1;`
	`ALTER QSUB EMPLOYEE0001 REPLQMAP SAMPLE_ASN1_TO_TARGETDB_ASN1`
	`USING OPTIONS ALL CHANGED ROWS N HAS LOAD PHASE I`
	`SUPPRESS DELETES N CONFLICT ACTION F;`

Replication Center	Use the Q Subscription Properties notebook. To open the notebook, right-click a Q subscription and select **Properties**.

2. If you are changing a single Q subscription, reinitialize the Q subscription. Use one of the following methods:

Method	Description
Replication Center	Use the Manage Q Subscriptions window. To open the window, right-click the Q Capture server where the source table for the Q subscription is located and select **Manage** > **Q Subscriptions**.
SQL	Insert a REINIT_SUB signal into the IBMQREP_SIGNAL table at the Q Capture server: `insert into schema.IBMQREP_SIGNAL( SIGNAL_TIME, SIGNAL_TYPE, SIGNAL_SUBTYPE, SIGNAL_INPUT_IN, SIGNAL_STATE ) values ( CURRENT TIMESTAMP, 'CMD', 'REINIT_SUB', 'subname', 'P' );` Where *schema* identifies the Q Capture program that is processing the Q subscription, and *subname* is the name of the Q subscription that you want to reinitialize.

3. If you are changing multiple Q subscriptions, reinitialize the Q Capture program. Use one of the following methods:

Method	Description

`reinit` parameter	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to reinitialize all of the Q subscriptions for one Q Capture schema:
	 MODIFY
	 `f myqcap,reinit`
	 Where `myqcap` is the Q Capture job name.
	 asnqccmd
	 `asnqccmd capture_server=server_name capture_schema=schema reinit`
	 Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program for which you want to reinitialize all Q subscriptions.

Replication Center	Use the Reinitialize Q Capture window. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to reinitialize and select **Reinitialize Q Capture Program**.

Adding existing columns to a Q subscription (unidirectional replication)

You can add existing columns from the source table to a unidirectional Q subscription while the replication programs are running. If the columns are not existing and your servers are at Version 10.1 or later, you use a different procedure.

Before you begin

- The Q subscription that the columns are being added to must be in A (active) state.
- Linux UNIX Windows If the data type of the column is LONG VARCHAR or GRAPHIC, the source database or subsystem must be configured with DATA CAPTURE CHANGES INCLUDE VARCHAR COLUMNS.

Restrictions

- The columns that you are adding must be nullable, or defined as NOT NULL WITH DEFAULT.
- If you add columns with default values, you must run the REORG utility on the source table before you begin replicating the new column. For more detail, see Avoiding CHECKFAILED errors when adding columns to DB2 for z/OS target tables.
- You cannot add more than 20 columns within one Q Capture commit interval as specified by the **commit_interval** parameter.
- **Federated targets:** To add columns to an existing Q subscription, you can use the ADDCOL signal but you must drop the Q subscription and recreate it after you alter the target table because you cannot add columns to a nickname.

About this task

For Version 10.1 or later: You do not need to use this procedure when you are adding new columns to the source table if the participating servers are at Version 10.1 or later on both z/OS and Linux, UNIX, and Windows. If you set the value of the REPL_ADDCOL column in the IBMQREP_SUBS table to Y (yes), when you add new columns to a table, the columns are automatically added to the Q subscription, and added to the target table if they do not already exist. To make this setting, specify REPLICATE ADD COLUMN YES in the ASNCLP CREATE QSUB command or click the **Automatically replicate new columns added to the source table** check box when you are creating or changing the properties of a Q subscriptions in the Replication Center. For more detail, see these topics:

- z/OS Enabling replication of ADD COLUMN and SET DATA TYPE operations
- "Enabling automatic replication of newly added columns from the source table" on page 130

When you insert the signal at the Q Capture server, the column is automatically added to the target table if you did not already add it. If you want to add multiple columns to a Q subscription, you insert one signal for each new column. You can add multiple columns in a single transaction. The Q Capture program can be stopped when you insert the signals and it will read them from the log when it restarts.

If you let the replication programs automatically add new columns to the target table it helps ensure that they match the columns at the source. Columns are added to the target table with the same data type, null characteristic, and default value as the matching columns in the source table. You can specify a different name for the target column if you use the ALTER ADD COLUMN command in the ASNCLP command-line program or an ADDCOL signal.

Procedure

To add columns to a unidirectional Q subscription, use one of the following methods:

Method	Description

ASNCLP command-line program.	Use the ALTER ADD COLUMN command. For example, the following command adds the column BONUS to the DEPARTMENT0001 Q subscription:
	`ALTER ADD COLUMN USING SIGNAL (BONUS)`
	`QSUB DEPARTMENT0001`
	`USING REPQMAP SAMPLE_ASN_TO_TARGET_ASN;`

Q Replication Dashboard	On the Subscriptions tab, select a Q subscriptions from the table and click **Actions** > **Add Columns**.

SQL	Use a command prompt or one of the DB2 command-line tools to insert an ADDCOL signal into the IBMQREP_SIGNAL table at the Q Capture server. For example:
	`insert into schema.IBMQREP_SIGNAL(`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` ) values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'ADDCOL',`
	`     'subname;column_name;before_column_name;`
	`target_column_name',`
	`     'P' );`
	schema
	Identifies the Q Capture program that is processing the Q subscription that you are adding a column to.
	subname;column_name;before_column_name;target_column_name
	The name of the Q subscription that you want to add the column to and the name of the column that you are adding, separated by a semicolon. These names are case-sensitive and do not require double quotation marks to preserve case. Follow these examples:
	Add column in source table to Q subscription and to target table `QSUB1;COL10`
	Add column and before image of the column (for CCD target tables)
	`QSUB1;COL10;XCOL10`
	Add column without before image but with different target column name
	`QSUB1;COL10;;TRGCOL10` (Use the double semicolon (;;) to indicate that you are omitting the before-image column.)

After processing the signal, the Q Capture program begins capturing changes to the new column when it reads log data that includes the column. Changes to the column that are committed after the commit of the ADDCOL signal insert will be replicated to the new column in the target table. Rows that existed in the target table before the new column is added will have a NULL or default value for the new column.

Adding existing columns to a Q subscription (bidirectional or peer-to-peer replication)

You can add existing columns from the source table to a bidirectional or peer-to-peer Q subscription while the replication programs are running. If the columns are not existing and your servers are at Version 10.1 or later, you use a different procedure.

Before you begin

- The Q subscriptions that specify the table must be in A (active) state at all servers.
- Linux UNIX Windows If the data type of the column is LONG VARCHAR or GRAPHIC, the source database or subsystem must be configured with DATA CAPTURE CHANGES INCLUDE VARCHAR COLUMNS.

Restrictions

- Any columns that you add must be nullable, or defined as NOT NULL WITH DEFAULT.
- If you add columns with default values, you must run the REORG utility on the source table before you begin replicating the new column. For more detail, see Avoiding CHECKFAILED errors when adding columns to DB2 for z/OS target tables.
- You cannot alter the default value of a newly added column until the ADDCOL signal for that column is processed.
- You cannot add more than 20 columns within one Q Capture commit interval as specified by the **`commit_interval`** parameter.

About this task

For Version 10.1 or later: You do not need to use this procedure when you are adding new columns to the source table if the participating servers are at Version 10.1 or later on both z/OS and Linux, UNIX, and Windows. If you set the value of the REPL_ADDCOL column in the IBMQREP_SUBS table to Y (yes), when you add new columns to a table, the columns are automatically added to the Q subscription, and added to the target table if they do not already exist. To make this setting, specify REPLICATE ADD COLUMN YES in the ASNCLP CREATE QSUB command or click the **Automatically replicate new columns added to the source table** check box when you are creating or changing the properties of a Q subscriptions in the Replication Center. For more detail, see these topics:

- z/OS Enabling replication of ADD COLUMN and SET DATA TYPE operations
- "Enabling automatic replication of newly added columns from the source table" on page 130

To use this procedure, first you alter a table at one server to add a column. Then you insert an SQL signal at the server. When the signal is processed, the versions of the table at the other servers are automatically altered to add the column, unless you added it manually. The signal also adds the column to the Q subscription definitions at all servers.

You can add any number of columns to the source table at a time. You can do this while the Q Capture and Q Apply programs are running or stopped.

Recommendation: Insert one ADDCOL signal at a time and issue a COMMIT before inserting the next ADDCOL signal or doing any other transactions.

Procedure

To add columns to replicate in bidirectional or peer-to-peer replication:

1. Alter the logical table at one of the servers to add the column.

   If the ALTER TABLE operation that adds the column to the source table fails, all the Q subscriptions in the peer-to-peer group will be deactivated.
2. Use one of the following methods to signal the Q Capture program that you want to add the column to the Q subscription for the source table.

Method	Description
ASNCLP command-line program	Use the ALTER ADD COLUMN command. For example, the following command adds the column BONUS to the DEPARTMENT0001 Q subscription: `ALTER ADD COLUMN USING SIGNAL (BONUS)` `QSUB DEPARTMENT0001` `USING REPQMAP SAMPLE_ASN_TO_TARGET_ASN;`
Q Replication Dashboard	On the Subscriptions tab, select a Q subscriptions from the table and click **Actions** > **Add Columns**.

Method	Description

SQL	Use a command prompt or one of the DB2 command-line tools to insert an ADDCOL signal into the IBMQREP_SIGNAL table at the Q Capture server. For example:
	`insert into schema.IBMQREP_SIGNAL(`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` ) values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'ADDCOL',`
	`     'subname;column_name',`
	`     'P' );`
	schema
	Identifies the Q Capture program at the server where you altered the table.
	subname;column_name
	The name of a Q subscription that originates at the Q Capture server where you altered the table, and the name of the column that you are adding, separated by a semicolon. These names are case-sensitive and do not require double quotation marks to preserve case.

Consider the following example. A peer-to-peer configuration has three servers: ServerA, ServerB, and ServerC, and six Q subscriptions: subA2B, subB2A, subA2C, subC2A, subB2C, and subC2B for the EMPLOYEE table.

You add a column, ADDRESS, to the EMPLOYEE table on ServerA. Then you insert an ADDCOL signal for the Q subscription that handles transactions from ServerA to ServerB, and specify `subA2B;ADDRESS` for the Q subscription name and column name. Only one ADDCOL signal is required. The replication programs automatically add the ADDRESS column to the EMPLOYEE tables at ServerB and ServerC, and add the column definition to all six Q subscriptions.

How Q Capture handles DDL operations at the source database

The Q Capture program automatically replicates some Data Definition Language (DDL) operations at the source database. Other DDL changes require you to take action at the target database.

The following table describes how Q Capture handles different types of DDL operations and what you need to do for any affected Q subscriptions.

Table 25. How Q Capture handles DDL changes to the source database and what you need to do

DDL operation	How it is handled	What you need to do
CREATE TABLE	**Version 10.1 on Linux, UNIX, and Windows** Automatically replicated if the new source table matches the schema- and table-naming pattern in a schema-level subscription. When the Q Capture program detects a CREATE TABLE operation in the log that matches a schema-level subscription, it informs the Q Apply program to create a matching target table. A table-level Q subscription is also created that maps the new source and target tables. **z/OS or earlier versions on Linux, UNIX, and Windows** No automatic replication of CREATE TABLE operations.	**Version 10.1 on Linux, UNIX, and Windows** Ensure that newly created source table matches the schema- and table-naming pattern in a schema-level subscription. **z/OS or earlier versions on Linux, UNIX, and Windows** Create a table-level Q subscription for the new source table and use the replication administration tools to create a matching target table, or use an existing target table.

Table 25. How Q Capture handles DDL changes to the source database and what you need to do (continued)

DDL operation	How it is handled	What you need to do
DROP TABLE	**Version 10.1 on Linux, UNIX, and Windows** Automatically replicated if the source table is part of a schema-level subscription. When the Q Capture program detects a DROP TABLE operation in the log that matches a schema-level subscription, the associated table-level Q subscriptions for all queues are also dropped. **z/OS or earlier versions on Linux, UNIX, and Windows** Q Capture leaves the Q subscription active, but there are no log records to read for the source table. On z/OS, the ASN0197W warning message is issued.	**Version 10.1 on Linux, UNIX, and Windows** Ensure that source table is included in a schema-level subscription. **z/OS or earlier versions on Linux, UNIX, and Windows** When you drop a table the Q subscription for the table still exists. To remove, stop the Q subscription and then delete the Q subscription.

Table 25. How Q Capture handles DDL changes to the source database and what you need to do (continued)

DDL operation	How it is handled	What you need to do
ALTER TABLE ADD (COLUMN)	**Version 10.1 on z/OS and Linux, UNIX, or Windows or later** If you set the value of the REPL_ADDCOL column in the IBMQREP_SUBS table to Y (yes), when you add new columns to a table the columns are automatically added to the Q subscription and added to the target table if they do not already exist. **Earlier versions** Q Capture leaves the Q subscription active, but does not replicate the added column until it receives an ADDCOL signal.	**Version 10.1 on z/OS and Linux, UNIX, or Windows or later** Specify REPLICATE ADD COLUMN YES in the ASNCLP CREATE QSUB command or click the **Automatically replicate new columns added to the source table** check box when you are creating or changing the properties of a Q subscriptions in the Replication Center. For more detail, see these topics: • z/OS Enabling replication of ADD COLUMN and SET DATA TYPE operations • "Enabling automatic replication of newly added columns from the source table" on page 130 **Earlier versions** Use the Q Replication Dashboard or ASNCLP ALTER ADD COLUMN command, or manually insert an ADDCOL signal to indicate that you want to replicate the new column.
TRUNCATE TABLE	z/OS A TRUNCATE operation is logged similarly to a mass delete, so the operation is replicated as a series of single row deletes. Linux UNIX Windows Replication of TRUNCATE operations is not supported.	z/OS No action is required. If the target table has rows that are not in the source table, those rows are not deleted. Linux UNIX Windows If you need to perform TRUNCATE on a target table in addition to its source, you must issue the TRUNCATE statement directly against the target table.
ALTER TABLE ALTER COLUMN SET DATA TYPE	Automatically replicated for Version 10.1 and later. The data type of the corresponding target table column is changed and replication continues normally.	z/OS See Enabling replication of ADD COLUMN and SET DATA TYPE operations. Linux UNIX Windows See "Automatic replication of ALTER TABLE ALTER COLUMN SET DATA TYPE operations" on page 135.
Other DDL that alters the structure of a table	Q Capture leaves the Q subscription unchanged.	1. Stop the Q subscription. 2. Alter the source and target tables. 3. Start the Q subscription.

Table 25. How Q Capture handles DDL changes to the source database and what you need to do (continued)

DDL operation	How it is handled	What you need to do
DDL that does not alter the table structure Examples: • CREATE INDEX • ALTER FOREIGN KEY • ADD CONSTRAINT	Q Capture leaves the Q subscription active.	Ensure that unique constraints, primary keys, and referential integrity constraints match between the source and target tables. If you change any of these properties at the source, make a corresponding change to the target to avoid unexpected behavior. Also, restart the Q Apply program so that it recognizes the change.

Changing properties of replication queue maps

A replication queue map includes options for how Q subscriptions that use a paired send queue and receive queue are processed. By updating a queue map and then reinitializing the send queue, receive queue, or both, you can change some of these settings without stopping replication.

About this task

Properties for replication queue maps are saved in the control tables. When you reinitialize a send queue, the Q Capture program obtains the latest settings from the IBMQREP_SENDQUEUES table. When you reinitialize a receive queue, the Q Apply program obtains the latest settings from the IBMQREP_RECVQUEUES table. The new settings affect all Q subscriptions that use the replication queue map.

If the Q Capture program is stopped when you make changes to the queue map, you do not need to reinitialize the send queue because Q Capture reads the changes when it starts. If the Q Apply program is stopped when you make changes, you do not need to reinitialize the receive queue.

Procedure

To update a replication queue map and prompt the Q Capture program or Q Apply program to recognize the changes:

1. Change the properties of the queue map by using one of the following methods:

Method	Description

ASNCLP command-line program	Use the **ALTER REPLQMAP** command. For example, the following commands set the environment and change the replication queue map SAMPLE_ASN1_TO_TARGETDB_ASN1 by setting the number of Q Apply agent threads to four and specifying that the Q Capture program should stop if an error occurs on a queue:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`SET SERVER TARGET TO DB TARGETDB;`
	`SET APPLY SCHEMA ASN1;`
	`SET RUN SCRIPT LATER;`
	`ALTER REPLQMAP SAMPLE_ASN1_TO_TARGET_ASN1`
	`USING NUM APPLY AGENTS 4 ERROR ACTION S;`

Replication Center	Use the Replication Queue Map Properties window. To open the window, right-click a replication queue map and select **Properties**.

Method	Description
Q Replication Dashboard	Use the Queues page. Select the Q Capture-Q Apply program pair, and then on the Properties tab make your changes and click **Save**.

2. If you change the WebSphere MQ queues in the queue map, use the replication administration tools to validate any new queues and to test the message flow between the queues.
3. Reinitialize the send queue, receive queue, or both, depending on which properties you changed. If you need to reinitialize both queues, you can do so in any order. Use one of the following methods:

Method	Description
Replication Center (send queue)	Use the Manage Send Queues window. To open the window, right-click the Q Capture server where the send queue is located and select **Manage** > **Send Queues**.
Replication Center (receive queue)	Use the Manage Receive Queues window. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage** > **Receive Queues**.
`reinitq` parameter (send queue)	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to reinitialize a send queue whose properties you changed in the queue map: **MODIFY** `f myqcap,reinitq=`*`queue_name`* Where `myqcap` is the Q Capture job name. **asnqccmd** `asnqccmd capture_server=`*`server_name`* *`capture_schema`*`=schema reinitq=`*`queue_name`* Where *server_name* is the name of the database or subsystem where the Q Capture program is running, *capture_schema* identifies the Q Capture program that uses the send queue, and *queue_name* is the name of the send queue that you want to reinitialize.
`reinitq` parameter (receive queue)	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqacmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to reinitialize a receive queue whose properties you changed in the queue map: **MODIFY** `f myqapp,reinitq=`*`queue_name`* Where `myqapp` is the Q Apply job name. **asnqacmd** `asnqacmd apply_server=`*`server_name`* `apply_schema=`*`schema_name`* `reinitq=`*`queue_name`* Where *server_name* is the name of the database or subsystem where the Q Apply program is running, *schema_name* identifies the Q Apply program that uses the receive queue, and *queue_name* is the name of the receive queue that you want to reinitialize.

Changing queue names when the queue map is used by Q subscriptions

If a replication queue map is already part of a Q subscription, you must manually update the control tables if you want to change the name of the send queue or receive queue in the queue map.

About this task

If a queue map is not being used for any Q subscriptions, you can always use the Replication Center or ASNCLP program to change queue names in the queue map.

Procedure

1. Stop the Q Capture program and let the Q Apply program process all messages in the receive queue.
2. Optional: If any of the Q subscriptions that use the queue map are active, stop the Q Apply program. This step is not required if you want to change the name of only the send queue.
3. Optional: If you created the Q Capture control tables by using the replication administration tools prior to Version 9.7 Fix Pack 2, use the following command to drop a foreign key constraint from the IBMQREP_SUBS table that refers to the IBMQREP_SENDQUEUES table:

   ```
   ALTER TABLE schema.IBMQREP_SUBS DROP CONSTRAINT FKSENDQ
   ```

   This constraint is no longer part of control tables that are created starting with Version 9.7 Fix Pack 2.
4. Use SQL to change the send queue name in the IBMQREP_SENDQUEUES table. Typically the send queue and corresponding receive queue have the same name. If this is the case, make sure to also change the receive queue name.

   For example, the following statement changes the send queue and receive queue name from Q2 to Q3 in the IBMQREP_SENDQUEUES table (in a later step the name of the receive queue is changed in the Q Apply control tables):

   ```
   UPDATE schema.IBMQREP_SENDQUEUES SET SENDQ='Q3',RECVQ='Q3' WHERE SENDQ='Q2'
   ```
5. Update the send queue name in the IBMQREP_SUBS table. For example:

   ```
   UPDATE schema.IBMQREP_SUBS SET SENDQ='Q3' WHERE SUBNAME='EMPLOYEE0001';
   ```
6. On the Q Apply server, update the queue names in three control tables: IBMQREP_RECVQUEUES, IBMQREP_TARGETS, and IBMQREP_TRG_COLS. For example:

   ```
   UPDATE schema.IBMQREP_RECVQUEUES SET RECVQ='Q3',SENDQ='Q3' WHERE RECVQ='Q2'
   UPDATE schema.IBMQREP_TARGETS SET RECVQ='Q3' WHERE RECVQ='Q2'
   UPDATE schema.IBMQREP_TRG_COLS SET RECVQ='Q3' WHERE RECVQ='Q2'
   ```
7. Optional: If you dropped the foreign key in Step 2, restore it by using the following command:

   ```
   ALTER TABLE schema.IBMQREP_SUBS ADD CONSTRAINT FKSENDQ FOREIGN KEY(SENDQ)
   REFERENCES schema.IBMQREP_SENDQUEUES(SENDQ)
   ```
8. z/OS Optional: If you dropped and restored the foreign key, run the CHECK DATA utility to bring the table space that contains the IBMQREP_SUBS table out of CHECK-pending status.
9. Optional: If you stopped the Q Apply program in Step 2, start the Q Apply program.
10. Start the Q Capture program in warm mode.

Restarting failed Q subscriptions without dropping and recreating them

When Q subscriptions fail to start and leave the control tables in an inconsistent state, you can reset the Q subscription state manually to avoid having to drop and recreate the Q subscriptions.

About this task

This procedure is especially helpful for bidirectional or peer-to-peer replication.

For example, Q subscriptions might fail to start because a WebSphere MQ channel was not started or the WebSphere MQ configuration was defined incorrectly. In these cases, the Q subscription is correctly defined, but the subscription schema message from Q Capture to Q Apply is not delivered.

This failure can cause inconsistent Q subscription states between the source Q Capture and target Q Apply control tables. You might receive an error message for which the "User response" section recommends that you drop and recreate the failed Q subscriptions after you fix the WebSphere MQ problem.

You can avoid dropping and recreating the Q subscriptions by resetting their state in the control tables. The procedure below ensures that the Q subscriptions states are cleared (back to inactive or `I`) before they are restarted, just as when they were first created.

Procedure

To restart failed Q subscriptions without dropping and recreating them:

1. Address the problem that prevented the Q subscriptions from starting. You might need to upgrade software or make small changes to source tables, target tables, or both, or change associated database elements. Use any reason codes or response information in error messages to help diagnose the problem.
2. Reset the status markers within the control tables by running the following SQL statements:

   Q Capture server
   ```
   UPDATE schema.IBMQREP_SUBS
   SET STATE='I', STATE_TRANSITION=NULL,
   GROUP_MEMBERS=NULL WHERE SUBNAME='Q_subscription_name'
   ```

   Q Apply server
   ```
   UPDATE schema.IBMQREP_TARGETS
   SET STATE='I' WHERE SUBNAME='Q_subscription_name'
   ```
3. Start the Q subscriptions.

Putting Q subscriptions into a temporary spill mode for maintenance

You can place Q subscriptions into a temporary spill mode to perform maintenance on target tables without stopping the capture of changes at the source or affecting replication to other tables.

About this task

When you place a Q subscription in spill mode, changes are not applied to the target table but instead are spilled to a dynamically created spill queue. Changes continue to be captured and applied for other Q subscriptions. When maintenance is complete, you can send a resume command to the Q Apply program

The temporary spill queue is created based on the definition of the model queue that is used for load operations. Ensure that the maximum depth of the queue is large enough to hold the spilled rows until the rows can be applied after resuming operations.

Note about transactional consistency: If you choose to spill replicated data from one table, be aware that any transactions that involve that table and other tables can become inconsistent in spill mode. For example, assume that you wanted to perform maintenance on target table T1, and issued a **spillsub** command. Consider a source transaction that updates tables T1 and T2:

```
UPDATE T1;
UPDATE T2;
COMMIT;
```

The `UPDATE T1` operation is spilled, but the `UPDATE T2` operation is applied to the target table, and a database commit and WebSphere MQ commit are issued. If you look at the target tables, only the update to T2 exists. So, a partial transaction was applied and committed to the database, and transactional consistency was not respected. If you expect transactional consistency at all times, you should not use the **spillsub** feature.

Restrictions

Spill mode is not allowed if the Q subscription involves a table that has referential integrity dependencies. You must drop and restore RI constraints on the target table.

Recommendation: Use the same receive queue for all Q subscriptions that involve RI-related tables. If you need to perform maintenance, stop message processing on the receive queue instead of spilling an individual Q subscription.

Procedure

1. Place the Q subscription into spill mode by using the **spillsub** parameter with the **MODIFY** command or **asnqacmd** command:
   ```
   F Q_Apply_job_name,spillsub="receive_queue_name:Q_subscription_name"
   ```
2. To resume normal operations, use the **resumesub** parameter:
   ```
   F Q_Apply_job_name,resumesub="receive_queue_name:Q_subscription_name"
   ```

When all rows in the spill queue are applied, the Q Apply program places the Q subscription back in active (A) state.

Deleting Q subscriptions

You can delete a Q subscription that is not being actively processed.

Before you begin

The Q subscription that you want to delete must be in I (inactive) or N (new) state.

About this task

When you delete a Q subscription, you have the option of dropping the target table that it refers to. You can also drop the table space for the target table.

If the target table is in a non-DB2 database, you can specify whether to drop the nickname for the target table when the Q subscription is deleted.

Note: Deleting a Q subscription does not delete the replication queue map that it uses.

Procedure

To delete one or more Q subscriptions, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the DROP QSUB command. For example, the following commands set the environment and drop the Q subscription for unidirectional replication EMPLOYEE0001:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET SERVER TARGET TO DB TARGET;`
	
	`DROP QSUB (SUBNAME EMPLOYEE0001`
	`USING REPLQMAP SAMPLE_ASN1_TO_TARGETDB_ASN1);`
	
	Use the SET DROP command to specify whether to drop the target table and its table space.

Replication Center	Use the Delete Q Subscriptions window. To open the window, right-click a Q subscription and select **Delete**. You can optionally use this window to drop the target tables for Q subscriptions that you delete. If you drop a target table, you can optionally use the window to drop the associated table space if no other tables are using the table space.

Deleting replication queue maps

You can delete a replication queue map that is no longer needed by any Q subscriptions.

Procedure

To delete a replication queue map:

1. Make sure that no Q subscriptions are using the replication queue map.
   a. Optional: Use the Show Related window in the Replication Center to see if any Q subscriptions are using the replication queue map. To open the window, right-click the queue map and select **Show Related**.
   b. If any Q subscriptions are using the queue map, delete the Q subscriptions.
2. Delete the queue map. Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the **DROP REPLQMAP** command. For example, the following commands set the environment and drop the queue map SAMPLE_ASN1_TO_TARGET_ASN1:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	`SET SERVER TARGET TO DB TARGET;`
	`SET APPLY SCHEMA ASN1;`
	`SET RUN SCRIPT LATER;`
	
	`DROP REPLQMAP`
	`SAMPLE_ASN1_TO_TARGETDB_ASN1;`

Replication Center	Use the Delete Replication Queue Maps window. To open the window, right-click the replication queue map and select **Delete**.

Dropping Q Capture or Q Apply control tables

When you drop Q Capture or Q Apply control tables, the action also removes the associated Q Capture or Q Apply schema from the Replication Center object tree. Dropping control tables also removes the associated Q Capture or Q Apply program instances.

About this task

Dropping control tables also deletes the following objects:

Q Capture control tables
Both Q subscriptions and publications are removed because definitions for these two objects are stored in the same control tables. Dropping control tables also removes replication queue maps and publishing queue maps.

Q Apply control tables
Q subscriptions and replication queue maps are removed. All nicknames that were created for non-DB2 target tables are also removed.

Q subscriptions and replication queue maps are defined on both the Q Capture server and Q Apply server. If you are dropping only Q Capture control tables or Q Apply control tables, any Q subscriptions or replication queue maps that are also defined in control tables on the opposite server must be removed before you drop the control tables.

Procedure

To drop Q Capture or Q Apply control tables:

1. Stop the Q Capture program or Q Apply program that uses the control tables that you want to drop.
2. If you are dropping only Q Capture control tables or only Q Apply control tables, take the following actions:
   - Deactivate any Q subscriptions that are also defined on the opposite server.
   - Delete the Q subscriptions.
   - Delete any replication queue maps that are also defined on the opposite server.
3. Use one of the following methods to drop the control tables:

Method	Description

ASNCLP command-line program	Use the DROP CONTROL TABLES ON command. For example, the following commands set the environment and drop the control tables on the Q Apply server TARGET that are identified by the schema ASN1:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER APPLY TO DB TARGET;`
	`SET APPLY SCHEMA ASN1;`
	
	`DROP CONTROL TABLES ON APPLY SERVER;`

Replication Center	Use the Drop Q Capture Schema or Drop Q Apply Schema window. To open the windows, right-click the schema and select **Drop**. If you are dropping the last schema on a Q Capture server or Q Apply server, that server no longer contains a set of control tables and is removed from the Replication Center object tree.

Chapter 18. Changing an event publishing environment

You can change the properties of event publishing objects such as publications and publication queue maps after you define them, in most cases without stopping publishing.

Changing properties of publications

You can change the properties of publications without stopping publishing. First you make your changes. Then you reinitialize either a single publication or reinitialize the Q Capture program if you are changing multiple publications.

Restrictions

Not all properties of publications can be changed.

- If you are using the ASNCLP command-line program to make changes, check the syntax of the **ALTER PUB** command to see which properties you can change.
- In the Replication Center, controls are disabled on the Publication Properties notebook for properties that you cannot change.

About this task

If you reinitialize a Q Capture program, this action will also reinitialize any Q subscriptions that are defined within the Q Capture schema. Reinitializing a Q Capture program also requires more system resources. If you only need to change the attributes of one or two publications, it is less costly in terms of resources to reinitialize one publication at a time.

Procedure

To change the properties of publications without stopping publishing:

1. Use one of the following methods to change the publication:

Method	Description

ASNCLP command-line program	Use the **ALTER PUB** command. For example, the following commands set the environment and change the publication EMPLOYEE0001 by sending all columns in a row that are part of the publication whenever any of them have changed, and to not send deleted rows:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	`SET RUN SCRIPT LATER;`
	
	`ALTER PUB EMPLOYEE0001`
	`FOR JK.EMPLOYEE OPTIONS`
	`CHANGED COLS ONLY N SUPPRESS DELETES Y;`

Replication Center	Use the Publication Properties notebook. To open the notebook, right-click a publication and select **Properties**.

2. If you are changing a single publication, reinitialize the publication by using one of the following methods:

Option	Description
Replication Center	Use the Manage Publications window. To open the window, right-click the Q Capture server where the source table for the publication is located and select **Manage** > **Publications** .

SQL	Insert a REINIT_SUB signal into the IBMQREP_SIGNAL table at the Q Capture server:
	`insert into schema.IBMQREP_SIGNAL(`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` ) values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'REINIT_SUB',`
	`     'pubname',`
	`     'P' );`
	
	Where *schema* identifies the Q Capture program that is processing the publication, and *pubname* is the name of the publication that you want to reinitialize.

3. If you are changing multiple publications, reinitialize the Q Capture program by using one of the following methods:

Method	Description

`reinit` parameter	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to reinitialize all of the publications for one Q Capture schema:
	MODIFY
	`f myqcap,reinit`
	
	Where `myqcap` is the Q Capture job name.
	asnqccmd
	`asnqccmd capture_server=server_name`
	`capture_schema=schema reinit`
	
	Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program for which you want to reinitialize all publications.

Replication Center	Use the Reinitialize Q Capture window. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to reinitialize and select **Reinitialize Q Capture Program**.

Adding columns to existing publications

You can add columns dynamically to an existing publication while continuing to publish changes from the source table.

Before you begin

- The publication that the columns are being added to must be in A (active) state when the signal is inserted.

- If the data type of the column is LONG VARCHAR or GRAPHIC, the source database or subsystem must be configured with DATA CAPTURE CHANGES INCLUDE VARCHAR COLUMNS.

Restrictions

- The columns that you are adding must be nullable, or defined as NOT NULL WITH DEFAULT.
- You cannot add more than 20 columns during a single WebSphere MQ commit interval, which is set by the Q Capture COMMIT_INTERVAL parameter.

About this task

The columns can already exist at the source table, or you can add the columns to the table and then add the columns to the publication in the same transaction.

To add a new column, you insert an SQL signal at the Q Capture server. The SQL signal contains details about the column. If you want to add multiple columns to a publication, you insert one signal for each new column. You can add multiple columns in a single transaction.

You can add columns to all publications that subscribe to a source table by specifying the **`SOURCE`** option instead of the **`PUB`** option.

Tip: If you plan to quiesce the source database or instance before adding columns, you can use the Replication Center or **`asnccmd chgparms`** command to set the TERM parameter for the Q Capture program to N (no). Setting TERM=N prompts the program to continue running while the database or instance is in quiesce mode. When DB2 is taken out of quiesce mode, the Q Capture program goes back to capturing changes from the last restart point in the log without requiring you to restart the program.

Procedure

To add columns to an existing publication, use one of the following methods:

Method	Description

ASNCLP command-line program.	Use the **`ALTER ADD COLUMN`** command. For example, the following command adds the column BONUS to the DEPARTMENT0001 publication:
	`ALTER ADD COLUMN USING SIGNAL (BONUS)`
	`PUB DEPARTMENT0001;`

Method	Description

SQL	Use a command prompt or one of the DB2 command-line tools to insert an ADDCOL signal into the IBMQREP_SIGNAL table at the Q Capture server. For example:
	`insert into schema.IBMQREP_SIGNAL(`
	`     SIGNAL_TIME,`
	`     SIGNAL_TYPE,`
	`     SIGNAL_SUBTYPE,`
	`     SIGNAL_INPUT_IN,`
	`     SIGNAL_STATE`
	` ) values (`
	`      CURRENT TIMESTAMP,`
	`     'CMD',`
	`     'ADDCOL',`
	`     'pubname;column_name',`
	`     'P' );`
	schema Identifies the Q Capture program that is processing the publication that you are adding a column to.
	pubname;column_name
	The name of the publication that you want to add the column to and the name of the column that you are adding, separated by a semicolon. These names are case-sensitive and do not require double quotation marks to preserve case.

After processing the signal, the Q Capture program sends an `add column` message to the user application and begins capturing changes to the new column when the Q Capture program reads log data that includes the column. Changes to the column that are committed after the commit of the ADDCOL signal insert will be published.

Deleting publications

You can delete a publication that is not being actively processed by a Q Capture program.

Before you begin

The publication that you want to delete must be in I (inactive) or N (new) state.

About this task

Deleting a publication does not delete the publishing queue map that it uses.

Procedure

To delete one or more publications, use one of the following methods:

Method	Description

ASNCLP command-line program	Use the DROP PUB command. For example, the following commands set the environment and drop the publication EMPLOYEE0001:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	`SET RUN SCRIPT LATER;`
	
	`DROP PUB (PUBNAME EMPLOYEE0001);`

Method	Description
Replication Center	Use the Delete Publications window. To open the window, right-click a publication and select **Delete**.

Changing properties of publishing queue maps

A publishing queue map points to one send queue, and includes settings for how a Q Capture program handles the publications that use the send queue. By updating a queue map and then reinitializing the send queue, you can change some of these settings without having to stop publishing of changes from the source table.

About this task

Properties for publishing queue maps are saved in the IBMQREP_SENDQUEUES control table. When you reinitialize a send queue, the Q Capture program obtains the latest settings from this table. The new settings affect all of the publications that use the send queue.

Procedure

To update a publishing queue map without stopping publishing:

1. Change the properties of the queue map by using one of the following methods:

Method	Description

ASNCLP command-line program	Use the ALTER PUBQMAP command. For example, the following commands set the environment and alter the publishing queue map SAMPLE_EP1_TO_SUBSCRIBER by changing the message type from row to transaction and specifying six seconds between heartbeat messages:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	`SET RUN SCRIPT LATER;`
	
	`ALTER PUBQMAP SAMPLE_EP1_TO_SUBSCRIBER`
	`USING MESSAGE CONTENT TYPE T`
	`HEARTBEAT INTERVAL 6;`

Replication Center	Use the Publishing Queue Map Properties window. To open the window, right-click a publishing queue map and click **Properties**.
Q Replication Dashboard	Use the Queues page. Select the Q Capture program, and then on the Properties tab make your changes and click **Save**.

2. If you change the WebSphere MQ queue that is used as the send queue, use the replication administration tools to validate the new queue.
3. Use one of the following methods to prompt a Q Capture program to recognize the changes that you made without stopping the Q Capture program:

Method	Description
`reinitq` **parameter**	Use this parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to reinitialize a send queue whose properties you changed: **MODIFY** `f myqcap,reinitq=`*`queue_name`* Where `myqcap` is the Q Capture job name. **asnqccmd** `asnqccmd capture_server=`*`server_name`* *`capture_schema`*`=schema reinitq=`*`queue_name`* Where *server_name* is the name of the database or subsystem where the Q Capture program is running, *capture_schema* identifies the Q Capture program that uses the send queue, and *queue_name* is the name of the send queue that you want to reinitialize.
Replication Center	Use the Manage Send Queues window. To open the window, right-click the Q Capture server where the send queue that you want to reinitialize is located and select **Manage** > **Send Queues**.

Deleting publishing queue maps

You can delete a publishing queue map that is no longer needed by any publications.

Procedure

To delete a publishing queue map:

1. Ensure that no publications are using the publishing queue map.
   a. Optional: Use the Show Related window in the Replication Center to see if any publications are using the publishing queue map. To open the window, right-click the publishing queue map and select **Show Related**.
   b. If any publications are using the queue map, delete the publications.
2. Delete the publishing queue map. Use one of the following methods:

Method	Description

ASNCLP command-line program	Use the DROP PUBQMAP command. For example, the following commands set the environment and drop the publishing queue map SAMPLE_EP1_TO_SUBSCRIBER:
	
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE EP1;`
	`SET RUN SCRIPT LATER;`
	
	`DROP PUBQMAP SAMPLE_EP1_TO_SUBSCRIBER;`

Replication Center	Use the Delete Publishing Queue Maps window. To open the window, right-click the publishing queue map and select **Delete**.

Dropping Q Capture control tables

When you drop Q Capture control tables, you also remove any publications, Q subscriptions, publishing queue maps, or replication queue maps that are defined in the control tables.

Before you begin

- The Q Capture program that uses the control tables must be stopped.
- Q subscriptions or replication queue maps in the Q Capture control tables are also defined in the Q Apply control tables. You must drop the Q subscriptions or queue maps before you drop the Q Capture the control tables.

About this task

Dropping Q Capture control tables also removes the associated Q Capture schema from the Replication Center object tree and removes the associated Q Capture program instance.

Procedure

To drop Q Capture control tables:

1. Stop the Q Capture program that uses the control tables that you want to drop.
2. Use one of the following methods to drop the control tables:

Method	Description

ASNCLP command-line program	Use the **DROP CONTROL TABLES ON** command. For example, the following commands set the environment and drop the control tables on the Q Capture server SAMPLE that are identified by the schema ASN1:
	`ASNCLP SESSION SET TO Q REPLICATION;`
	`SET SERVER CAPTURE TO DB SAMPLE;`
	`SET CAPTURE SCHEMA SOURCE ASN1;`
	
	`DROP CONTROL TABLES ON CAPTURE SERVER;`

Replication Center	Use the Drop Q Capture Schema window. To open the window, right-click the schema and select **Drop**. If you are dropping the last schema on a Q Capture server, that server no longer contains a set of control tables and is removed from the Replication Center object tree.

Chapter 19. Checking the status of the Q Replication and Event Publishing programs

The following topics provide information about how you can check the status of your Q Replication and Event Publishing environment.

Checking the status of the Q Replication and Event Publishing programs

You can use the Q Replication Dashboard or system commands to check the current status of the Q Capture program and Q Apply program. You can use a system command to check the current status of the Replication Alert Monitor.

About this task

The Q Replication Dashboard provides live visual cues so you can detect status changes for the Q Capture and Q Apply program, and you can also configure email alerts to notify you if either of the programs stops. With system commands, you can view messages about the state of program threads. These messages can help you determine whether the programs are working correctly. For the Q Capture and Q Apply programs, you can optionally use commands to view more detailed reports about current program operations.

Procedure

To check the status of the Q Replication and Event Publishing programs:

Use one of the following methods:

Q Replication Dashboard
You can view the status of the Q Capture and Q Apply programs from the Summary tab, or you can click the Programs tab to get more detailed status information. From the Alert Manager tab, you can enable email alerts for Q Capture status and Q Apply status, and the dashboard sends an email to one or more specified contacts if the programs stop.

Command line

Q Capture
Use the **`status`** parameter with the **`MODIFY`** command on z/OS or **`asnqccmd`** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to view messages that indicate the state of each Q Capture thread.

To view more detailed status about the program and its operations, use the **`show details`** parameter.

MODIFY
```
f myqcap,status show details
```

Where `myqcap` is the Q Capture job name.

asnqccmd

```
asnqccmd capture_server=server
capture_schema=schema
status show details
```

Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program that you want to check.

Q Apply

Use the **status** parameter with the **MODIFY** command on z/OS or **asnqacmd** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to view messages that indicate the state of each Q Apply thread.

To view more detailed status about the program and its operations, use the **show details** parameter.

MODIFY

```
f myqapp,status show details
```

Where myqapp is the Q Apply job name.

asnqacmd

```
asnqacmd apply_server=server
apply_schema=schema
status show details
```

Where *server_name* is the name of the database or subsystem where the Q Apply program is running, and *schema* identifies the Q Apply program that you want to check.

Replication Alert Monitor

Use the **status** parameter with the **MODIFY** command on z/OS or **asnqmcmd** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to view messages that indicate the state of each monitor program thread.

MODIFY

```
f mymon,status
```

Where mymon is the monitor job name.

asnmcmd

```
asnmcmd monitor_server=server
monitor_qual=qualifier
status show details
```

Where *server_name* is the name of the database or subsystem where the monitor program is running, and *qualifier* identifies the monitor program that you want to check.

Threads of the Q Capture, Q Apply, and Replication Alert Monitor programs

When you check the status of the Q Replication and Event Publishing programs, you can tell whether the programs are working correctly by the messages that you receive about the program threads.

The following list shows the threads for the Q Capture, Q Apply, and Replication Alert Monitor programs. After each thread name are abbreviations that denote whether the thread interacts with the database ("DB") or WebSphere MQ ("MQ"):

Q Capture program

A Q Capture program has the following threads:

Logreader (DB)
For DB2, reads the database recovery log and captures changes for subscribed tables. For Oracle sources, starts the Oracle LogMiner utility, reads from the V$LOGMNR_CONTENTS view to find changes for subscribed tables. Rebuilds log records into transactions in memory before passing them to the worker thread when the commit for a transaction is processed.

Worker (DB; MQ)
Receives completed transactions from the log reader thread, turns transactions into WebSphere MQ messages, and puts the messages on send queues. Maintains program restart information.

Administration (DB; MQ)
Handles control messages that are put by the Q Apply program or a user application on the administration queue, and writes statistics to the monitor tables.

Prune (DB)
Deletes old data from some of the Q Capture control tables.

Holdlock (DB on all platforms except z/OS)
Handles signals sent to the Q Capture program. Prevents two Q Capture programs with the same schema from running on a server.

Worker, logreader, administration, and prune threads are typically in running or resting states. Holdlock thread is always in a waiting state. If the worker thread is in a running state but data is not being captured, you can use the Q Replication Dashboard, Q Capture Messages window in the Replication Center, or IBMQREP_CAPTRACE table to look for messages that might explain why data is not being captured. Alternatively look in the IBMQREP_CAPMON or IBMQREP_CAPQMON tables to verify that log records are being processed.

Q Apply program

A Q Apply program has the following threads:

Browser (DB; MQ)
Reads transaction messages from a receive queue, maintains dependencies between transactions, and launches one or more agent threads. The Q Apply program launches one browser thread for each receive queue. This thread is identified by the characters BR*xxxxx*, where *xxxxx* specifies the number of browser threads for a Q Apply program.

Agent (DB; MQ)
: Rebuilds transactions in memory and applies them to targets. You set the number of agent threads that will be used for parallel processing of transactions when you create a replication queue map.

Spill agent (DB; MQ)
: Created on demand when changes for a Q subscription need to be applied from a spill queue. Situations where this thread is created include when a table has finished being loaded or when a **`resumesub`** command is received. Rebuilds transactions that were held in a spill queue and applies them to targets. Spill agents terminate after the spill queue is emptied and the Q subscription becomes active.

Administration (DB; MQ)
: Maintains the Q Apply control tables. Responsible for starting new browsers when a **`startq`** command is received.

Monitor (DB; MQ)
: Logs information about the Q Apply program's performance in the IBMQREP_APPLYMON control table.

Prune (DB; MQ)
: Deletes rows from the IBMQREP_DONEMSG control table after the messages to which the rows pertain are deleted from the receive queue and the corresponding data is applied. This thread is used when the **`prune_method`** parameter is set to a value of 2, the default.

Holdlock (DB on all platforms except z/OS)
: Prevents two Q Apply programs with the same schema from running on a server.

Browser, agent, and admin threads are typically in a running state. If agent threads are in a running state but data is not being applied, you can use the Q Replication Dashboard, Q Apply Messages window, or IBMQREP_APPLYTRACE table to look for messages that might explain why data is not being applied.

Replication Alert Monitor

A Replication Alert Monitor program has the following threads:

Worker
: Monitors the control tables to see whether user-defined alert conditions have been met and sends alert notifications.

Administration
: Handles internal messages and error messages.

Serialization
: Handles signals between the monitor program threads.

Worker and administration threads are typically in working or resting states. Serialization threads are typically in a waiting state. If the worker thread is in a working state but you are not receiving expected alert notifications, you can use the IBMSNAP_MONTRACE and IBMSNAP_MONTRAIL control tables to look for messages and other information that might explain why alerts are not being sent.

When you check the status of the Q Capture, Q Apply, and Replication Alert Monitor programs, if the messages do not indicate typical states for your threads, you might need to take further action as described in Table 26.

Table 26. Descriptions of thread states and suggested actions

Thread state	Description	Suggested action
Running	The thread is actively processing.	No action.
Exists	The thread exists but cannot start.	If the thread does not leave this state, stop the program and restart it.
Started	The thread started but cannot initialize.	Investigate potential system resource problems, such as too many threads or not enough processing power.
Initializing	The thread is initialized but cannot work.	If the thread does not leave this state, stop the program and restart it.
Resting	The thread is sleeping and will wake up when there is work to do.	No action.
Stopped	The thread is not running.	Use the Q Capture Messages window, Q Apply Messages window, or Monitor Messages window in the Replication Center to search for messages that explain why the thread stopped. Or look for similar messages in the IBMQREP_CAPTRACE, IBMQREP_APPLYTRACE, or IBMSNAP_MONTRACE control tables.

Latency

Latency is a measure of the time it takes for transactions to replicate from the Q Capture server to the Q Apply server. The Q Capture and Q Apply programs save performance data that lets you track latency between various stages of the replication process. These statistics can help you pinpoint problems and tune your environment.

By using the Latency window in the Replication Center, you can see how long it takes for transactions to move between the DB2 recovery log and the send queue, the send queue and the receive queue, and the receive queue and the target table. You can use the Q Capture Latency window to view an approximate measure of how a Q Capture program is keeping up with changes to the log.

Recommendation: Any latency measure that involves transactions that are replicated between remote Q Capture and Q Apply servers can be affected by clock differences between the source and target systems. To get a true measure, ensure that the clocks are synchronized.

The following sections provide more detail about each measure of latency:

Q Capture latency
Q Capture latency measures the difference between a given point in time and the timestamp of the last committed transaction. This measure uses the

value of the MONITOR_TIME and CURRENT_LOG_TIME columns in the IBMQREP_CAPMON control table. When examined in aggregate, these latency measurements can help you determine how well a Q Capture program is keeping up with the database log.

For example, if a Q Capture program inserted a row of performance data into the IBMQREP_CAPMON table (MONITOR_TIME) at 10 a.m. and the timestamp of the last committed transaction (CURRENT_LOG_TIME) is 9:59 a.m., then the Q Capture latency is one minute. To improve log-reading performance, consider creating an additional Q Capture schema and moving some Q subscriptions or publications to the new schema. Each additional schema adds another worker thread to read the log.

Q Capture transaction latency
Q Capture transaction latency measures the time between the Q Capture program reading the commit statement for a transaction in the DB2 recovery log, and the message containing the transaction being put on a send queue. This statistic can give you an idea of how long it takes the Q Capture program to reassemble transactions in memory, filter out rows and columns based on settings for the Q subscription or publication, and then put the transaction messages on a queue. To reduce Q Capture transaction latency, consider making the following adjustments:

- Increasing the value of the **memory_limit** parameter, which sets the total amount of memory allocated by a Q Capture program.
- Raising the **max_message_size** parameter, which is defined when you create or change a replication queue map or publication queue map. This parameter sets the amount of memory that a Q Capture program allocates for building transaction messages for each send queue. If the maximum message size is too small, the Q Capture program divides transactions into multiple messages, requiring more processing time and increasing latency.

Queue latency
Queue latency measures the time between the Q Capture program putting a transaction on a send queue and the Q Apply program getting the transaction from the receive queue. This statistic might give you an idea of WebSphere MQ performance, although in distributed environments network performance might also be a factor. Also, a longer commit interval prompts the Q Capture program to wait before committing some transactions that have been put on the send queue. This delay can contribute to queue latency.

Q Apply latency
Q Apply latency measures the time it takes for a transaction to be applied to a target table after the Q Apply program gets the transaction from a receive queue. The more agent threads that you have specified for a receive queue, the smaller this number should be. Note, however, that the Q Apply program delays applying a transaction involving dependent tables until all previous transactions that it depends on have been applied. This delay can increase Q Apply latency.

End-to-end latency
End-to-end latency measures the time between the Q Capture program reading the log record for a transaction commit statement and the Q Apply program committing the transaction at the target. This statistic is an overall measure of Q replication latency.

To open the Latency window, right-click the Q Apply server that contains the Q Apply program that you want to view statistics for and select **Reports** > **Latency**.

To open the Q Capture Latency window, right-click the Q Capture server that contains the Q Capture program that you want to view statistics for and select **Reports** > **Q Capture Latency**.

Exceptions

Exceptions are rows that a Q Apply program did not apply to targets or reworked because of conflicts or SQL errors. The Q Apply program saves these rows, along with details about the conflict or error, in the IBMQREP_EXCEPTIONS table.

Conflicts can occur when an application other than the Q Apply program is updating the target. For example, the Q Apply program might try to insert a row that already exists at the target. If the conflict action for the Q subscription is F (force), Q Apply turns the INSERT into an UPDATE, and the original INSERT statement is logged in the IBMQREP_EXCEPTIONS table.

SQL errors cover a broad range of potential problems. For example, if the recovery log at the target server filled up, SQL errors would be generated when the Q Apply program tried to apply rows.

You can use the Exceptions window in the Replication Center to view details about problem rows. You can also use the data that the Q Apply program saves in its control table to recover rows that were not applied, and to learn more about possible problems in your replication environment.

The following list shows the types of problems that can prevent the Q Apply program from applying rows:

Unexpected SQL errors
Rows that caused SQL errors that were not defined as acceptable for the Q subscription.

Acceptable SQL errors
Rows that caused SQL errors that were defined as acceptable. You define acceptable SQL errors when you create a Q subscription.

Value-based conflicts
Rows that caused conflicts such as an attempt to delete or update a row that did not exist at the target, or to insert a row that already existed. The Q Apply program records the type of conflict detection that was used (check only key values, check changed non-key columns as well as key columns, check all non-key columns as well as key columns), and whether the row was applied despite being saved. When you create a Q subscription, you can specify that the Q Apply program should apply some rows even when they cause conflicts.

Version-based conflicts
Rows that caused conflicts in peer-to-peer replication, such as an attempt to update a row with the values from older row, or an attempt to insert a row when a newer row with the same key already existed at the target.

When the Q Apply program saves rows in the IBMQREP_EXCEPTIONS table, it records the time when the error or conflict occurred, the reason for the error, and the type of SQL operation that resulted in the problem. The Q Apply program also saves the names of the receive queue and Q subscription, and source commit information for the transaction.

For unexpected SQL errors and acceptable SQL errors, the Q Apply program saves the SQL code and SQL state code returned by DB2 for the transaction, as well as the error message tokens from the SQLCA structure that is used for executing the transaction. For value-based conflicts and version-based conflicts, the Q Apply program records the reason that the row could not be applied. Rows that are not applied are saved in SQL format in the TEXT column of the IBMQREP_EXCEPTIONS table.

In peer-to-peer replication, the Q Apply program saves rows that it did not apply at the server where the conflict or SQL error occurred.

To open the Exceptions window, right-click the Q Apply server that contains the Q Apply program that you want to view exceptions for and select **Reports** > **Exceptions**.

Chapter 20. Maintaining a Q Replication and Event Publishing environment

Q Replication and Event Publishing work with your database system and require limited changes to your existing database activities. Maintenance of source systems, control tables, and target tables can ensure that your entire system continues to run smoothly.

Considerations for maintaining Q Replication and Event Publishing source systems

You need to consider the availability of source tables and log data to Q Replication and Event Publishing so that the Q Capture and Q Apply programs are always able to proceed.

The replication source system contains the following objects:

- The source tables from which you want to replicate or publish data.
- The log data that the Q Capture program reads to capture changes that are made to source tables.

Maintaining source tables in a Q Replication and Event Publishing environment

Replication sources are database tables. Source tables for Q Replication and Event Publishing require the same maintenance as other database tables on your system.

Q Replication and Event Publishing do not require direct access to source tables during most processing. However, Q Replication and Event Publishing must access your source tables directly in the following situations:

- The Q Apply program performs a load.
- The Q Capture program captures LOB data.

To maintain source tables for Q Replication and Event Publishing:

- Continue to run your existing utilities and maintenance routines on these tables.
- Make sure that read access is available to your source tables to avoid disrupting the Q Apply program during a load.

Retaining log files for Q Replication and Event Publishing

If you need to know which log files are required by the Q Capture program, you can use a Q Capture command on Linux, UNIX, and Windows, or you can find the oldest required log manually by reading a Q Capture control table and using a DB2 utility on z/OS.

Why you must retain log data for Q Replication and Event Publishing

You need to retain log data for both DB2 recovery and for Q Replication or event publishing. Also, be absolutely certain that the Q Capture programs and DB2 are completely finished with a set of logs before you delete them.

Your DB2 recovery logs:

- Provide DB2 recovery capabilities
- Provide information to your running Q Capture programs

Log data resides in log buffers, active logs, or archive logs. Each time the Q Capture program warm starts it requires all the DB2 logs that were created since it stopped and any DB2 logs that it did not completely process.

If you run the Q Capture program continuously, it is typically up to date with the DB2 recovery log. If you also retain log files for a week or longer, you can continue to use your existing log retention procedures. However, you should change your log retention procedures to accommodate Q Replication and Event Publishing if:

- You typically delete log records as soon as DB2 completes a backup, and these log records are no longer needed for forward recovery.
- You face storage constraints and need to delete your archived recovery logs frequently.
- You run the Q Capture program periodically instead of continuously.

Determining the oldest log file that Q Capture needs (z/OS)

You can read a Q Capture control table and use a DB2 utility to determine the log sequence number for the oldest log record that the Q Capture program needs on z/OS operating systems.

About this task

Use DB2 to reference the log sequence number with the log file that contains the oldest log record. The Q Capture program needs this log file and more recent ones.

Procedure

To determine the oldest log file that the Q Capture program requires:

1. Run the following SQL statement to obtain the log sequence number for the most recent transaction the Q Capture program has seen, processed, and recorded in its control tables:
   ```
   SELECT max(RESTART_SEQ)
   FROM schema.IBMQREP_CAPMON
   WITH UR;
   ```
   This is an example of a log sequence number:
   ```
   0000555551F031230000
   ```
   Ignore the first four characters of the log sequence number, which are always 0000. The next 12 characters correspond to the active log sequence number. (This 12–character value is the relative byte address (RBA) in non-data sharing environments and is the log record sequence number (LRSN) in data sharing environments.) The last four characters are 0000 in non-data sharing environments; these last four characters correspond to the member ID in data sharing environments.
2. Use the DSNJU004 utility to invoke the Print Log Map utility. This utility displays information about the bootstrap data sets (BSDS). For example:
   ```
   #  ACTIVE LOG COPY 1 DATA SETS
   #   START RBA/TIME    END RBA/TIME         DATE   LTIME    DATA SET INFORMATION
   #------------------  --------------      -------- ------  -------------------------
   # 555551F03000        555551F05FFF       1998.321 12:48   DSN=DSNC710.LOGCOPY1.DS02
   # 2001.57 15:46:32.2  2001.057 15:47:03.9 PASSWORD=(NULL) STATUS=TRUNCATED,REUSABLE
   # 555551F06000        555551F09FFF       1998.321 12:49   DSN=DSNC710.LOGCOPY1.DS03
   # 2001.57 15:47:32.2  2001.057 15:48:12.9 PASSWORD=(NULL) STATUS=TRUNCATED,REUSABLE
   ```

3. Compare your 12–character active log number of the RESTART_SEQ value to the Start RBA and corresponding End RBA range in each displayed row.
4. Find the row for which the 12–character active log number from the IBMQREP_CAPMON table falls within the start RBA and end RBA. In the example:
   ```
   # 555551F03000        555551F05FFF      1998.321 12:48  DSN=DSNC710.LOGCOPY1.DS02
   # 2001.57 15:46:32.2  2001.057 15:47:03.9 PASSWORD=(NULL)STATUS=TRUNCATED,REUSABLE
   ```
5. Note the corresponding Data Set Information for that active log number. In the example:
   ```
   DSNC710.LOGCOPY1.DS02
   ```
6. Note the date and time of this data set. The Q Capture program needs this data set and more recent data sets to restart.

Determining the oldest log file that Q Capture needs (Linux, UNIX, Windows)

You can use a command to determine the oldest log file that the Q Capture program needs and which log files you can safely remove on Linux, UNIX, and Windows operating systems.

Before you begin

The Q Capture program must be running for you to issue the command.

About this task

The command uses the db2flsn utility to determine the oldest DB2 log file that is needed. The Q Capture program needs this log file and more recent log files to perform a restart at any particular time. You must retain this log file and more recent log files to ensure continuous operation of the Q Capture programs. You can delete any older logs.

Procedure

To determine the oldest log file that the Q Capture program needs:

Use the **asnqccmd** command with the following parameters while the Q Capture program is running:

```
asnqccmd capture_server=server_name capture_schema=schema status show details
```

The command returns a report on Q Capture program status, including the following details:

- Path to DB2 log files
- Oldest DB2 log file needed for Q Capture restart
- Current DB2 log file captured

Here is sample output for these details:

```
Path to DB2 log files                              (DB2LOG_PATH) = /home2/szp/szp/
  NODE0000/SQL00002/SQLOGDIR/
Oldest DB2 log file needed for Q Capture restart   (OLDEST_DB2LOG) = S0000043.LOG
Current DB2 log file captured                    (CURRENT_DB2LOG) = S0000046.LOG
```

Recommendation: Run the Q Capture program whenever DB2 is up. This should keep the Q Capture program reading as close as possible to the end of the DB2 log, where the most recent log records are. Reading near the end of the log minimizes the number of older log files that the Q Capture program needs.

Considerations for managing compression dictionaries in Q replication and event publishing (z/OS)

If you are using DB2 compression dictionaries, you must coordinate the use of utilities with your Q Capture programs.

Updating DB2 compression dictionaries

When the Q Capture program requests log records, DB2 must decompress the log records of any table that is stored in a compressed table space. DB2 uses the current compression dictionary for decompression. In some cases the compression dictionary might be unavailable. The Q Capture program takes different actions in each case:

If the compression dictionary is temporarily unavailable
DB2 returns an error to the Q Capture program. The Capture program makes several attempts to continue processing. If the dictionary remains unavailable, the Q Capture program issues an ASN0011E message and terminates.

If the compression dictionary is permanently unavailable
A compression dictionary might be lost if you use the REORG utility without specifying KEEPDICTIONARY=YES. In this case, the Q Capture program issues an ASN0011E message, deactivates the Q subscription, and terminates.

With APAR PK19539 (DB2 for z/OS Version 8), DB2 will keep one backup of the compression dictionary in memory when you use the REORG utility without specifying KEEPDICTIONARY=YES. So you do not need to specify KEEPDICTIONARY=YES unless:

- You restart DB2.
- You use the REORG utility twice for the same table space before the Q Capture program reads all of the old log records for that table.

To avoid these situations in DB2 for z/OS Version 7, let the Q Capture program process all log records for a table before performing any activity that affects the compression dictionary for that table. Some of the following activities can affect compression dictionaries:

- Altering a table space to change its compression setting
- Using DSN1COPY to copy compressed table spaces from one subsystem to another, including from data sharing to non-data-sharing environments
- Running the REORG utility on the table space

Latching DB2 compression dictionaries

You should also consider the availability of your compression dictionary. When the Q Capture program reads compressed log records, DB2 takes a latch on the source compressed table space to access the dictionary. The Q Capture program stops if the compressed table space on the source system is in the STOPPED state when the DB2 Log Read Interface needs this latch. Conversely, a utility that requires complete access to the source table space or that requires the table space to be in a STOPPED state can be locked out by the latch held by the Q Capture program while it is reading the dictionary.

To prevent any temporary lockout due to an unavailable latch, suspend the Q Capture program when a source compressed table space needs to be used exclusively by a DB2 (or vendor) utility.

Maintaining control tables in Q Replication and Event Publishing

Control tables store object definitions and other replication-specific control information. Although the size of some control tables remains static, other control tables can grow and shrink depending on the size of your database and your replication requirements.

Pruning control tables in Q Replication and Event Publishing

Some Q Replication and Event Publishing control tables grow regularly. The replication programs prune most of these control tables, and you can also use a system command or the Replication Center to do additional pruning.

About this task

Table 27 lists control tables that the Q Capture program prunes that can grow regularly.

Table 27. Control tables that the Q Capture program prunes

Control table	Parameter that specifies which rows are eligible for pruning
IBMQREP_CAPMON	**monitor_limit**
IBMQREP_CAPQMON	**monitor_limit**
IBMQREP_CAPTRACE	**trace_limit**
IBMQREP_SIGNAL	**signal_limit**

The **prune_interval** parameter specifies how often the Q Capture program checks for rows that are eligible for pruning.

Table 28 lists control tables that the Q Apply program prunes that can grow regularly.

Table 28. Control tables that the Q Apply program prunes

Control table	Parameter that specifies which rows are eligible for pruning
IBMQREP_APPLYMON	**monitor_limit**
IBMQREP_APPLYTRACE	**trace_limit**

The IBMQREP_EXCEPTIONS table can also grow regularly. Use SQL to manually prune rows that contain exceptions that you have already processed.

The **prune_interval** parameter specifies how often the Q Apply program checks for rows that are eligible for pruning.

The Replication Alert Monitor prunes the IBMSNAP_ALERTS table, which can grow regularly. The **alert_prune_limit** parameter specifies how much data is kept in the table. The rate of growth depends on your replication configuration and parameters.

Procedure

To do additional pruning of control tables:

Use one of the following methods:

Method	Description
System command	Use the **`prune`** parameter with the **`MODIFY`** command on z/OS or one of the following commands on Linux, UNIX, Windows, and UNIX System Services on z/OS: **Q Capture control tables** **`asnqccmd`** **Q Apply control tables** **`asnqacmd`** **Monitor control tables** **`asnmcmd`**
Replication Center	Use one of the following windows: **Q Capture control tables** Use the Prune Q Capture Control Tables window. To open the window, right-click a Q Capture server in the contents pane and select **Prune Q Capture Control Tables**. **Q Apply control tables** Use the Prune Q Apply Control Tables window. To open the window, right-click a Q Apply server in the contents pane and select **Prune Q Apply Control Tables**. **Monitor control tables** Use the Prune Monitor Control Tables window. To open the window, right-click a Monitor qualifier in the contents pane and select **Prune Monitor Control Tables**.

Considerations for using the RUNSTATS utility on control tables for Q Replication and Event Publishing

The optimizer can improve access to your Q Replication and Event Publishing control tables.

The RUNSTATS utility updates statistics about the physical characteristics of your tables and associated indexes. Continue to run the RUNSTATS utility on your existing tables at the same frequency as before you used Q Replication. However, run the RUNSTATS utility on control tables that grow regularly (and are pruned regularly) only one time when these tables contain substantial amounts of data. RUNSTATS reports meaningful information about these dynamic tables when these tables are at their maximum production-level size, and the optimizer gains the necessary statistics to determine the best strategy for accessing data.

Reorganizing control tables

Regularly reorganize any control tables that frequently change size to eliminate fragmented data and reclaim space.

About this task

Reorganize the following control tables once a week:
- IBMQREP_APPLYMON
- IBMQREP_CAPMON
- IBMQREP_CAPQMON
- IBMQREP_APPLYTRACE

- IBMQREP_CAPTRACE
- IBMSNAP_MONTRAIL
- IBMSNAP_MONTRACE
- IBMQREP_SIGNAL

Depending on your replication environment and configurations, you might also need to reorganize the following tables:
- IBMQREP_DELTOMB (peer-to-peer configurations)
- IBMQREP_DONEMSG
- IBMQREP_EXCEPTIONS
- IBMSNAP_ALERTS (replication alert monitoring)

Note: Use caution when reorganizing the IBMQREP_SIGNAL table while the replication programs are active. If the state of a Q subscription or publication is changed while the REORG utility is running, contention at the IBMQREP_SIGNAL table could cause problems.

Procedure

To reorganize control tables:
- z/OS Use the REORG utility with the PREFORMAT option. The PREFORMAT option speeds up the insert processing of the Q Capture program.
- Linux UNIX Windows Use the **`REORG`** command.

When replication programs cannot connect to their DB2 server

To run correctly, the Q Capture program, Q Apply program, and Replication Alert Monitor must be able to connect to the DB2 server that contains their control tables. When a replication program cannot access its control tables, it issues an appropriate error message and shuts down.

Connectivity issues typically require you to restart the program when connectivity returns. For example, if a Q Apply program shuts down because the DB2 server that contains its control tables has been shut down or quiesced, simply restart the Q Apply program when the DB2 server is running.

If the program can connect to the DB2 server but receives an SQL error when the program tries to access its control tables, take the appropriate corrective action for that SQL error and then restart the program. For example, if the SQL error indicates that a control table needs to be recovered, use a standard DB2 recovery procedure to forward recover the table and then restart the program.

Maintaining target tables

Maintaining target tables is similar to maintaining other tables. However, you need to consider the operations of the Q Apply program in your maintenance routine for these tables.

About this task

When you need to perform maintenance on a target table such as reorganizing a table, you want to prevent applications such as the Q Apply program from using the table until the maintenance is finished. The Q Apply program might stop because of an error if it tries to make an update to a table during maintenance.

You can choose from several options that allow you to perform maintenance on the target tables and meet your requirements. If you need to reorganize a single table at a time, you can set the Q subscription for that table to use a temporary spill mode. Your configuration and requirements determine which option you choose.

Procedure

To maintain target tables:
- Maintain the tables on the target server in the same way that you maintain other tables on your database system.
- Use your current backup and maintenance routines on these target tables, whether your target tables are existing database tables or tables that you specified to be automatically generated by Q Replication.
- Choose the option that best meets your requirements. Some methods affect more tables than other methods.

Option	Description
Place a Q subscription into spill mode	This option places a Q subscription for a single target table into the spill state. The Q Apply program holds changes from the source table in a temporary spill queue. While the changes are spilled, you can maintain the target table. The spill queue is created based on your model queue definition. You might need to adjust the maximum depth of your model queue to ensure the queue is large enough to hold the spilled rows. Use the **spillsub** and **resumesub** parameters to place the Q subscription into the spill state and to later resume normal operations. You can use these parameters with the **MODIFY** command on z/OS or **asnqacmd** command on Linux, UNIX, Windows, and UNIX System Services on z/OS. The following example uses **MODIFY**: `f myqapp,spillsub=`*`receive_queue_name`*`:`*`q_subscription_name`* **Restriction:** If your tables have referential integrity constraints, you must use another method such as stopping message processing on the receive queue.
Stop processing messages on a receive queue	You can use the **stopq** parameter with the **MODIFY** command on z/OS or **asnqacmd** command on Linux, UNIX, Windows, and UNIX System Services on z/OS to instruct the Q Apply program to stop processing messages for the receive queue. The receive queue continues to receive transactions, which are held until the receive queue is reactivated. This method affects all of the tables that use the receive queue. If you need to maintain tables that share a receive queue or your tables have referential integrity constraints, this method is recommended. Use the **stopq** parameter to stop message processing and the **startq** parameter to resume message processing when maintenance is finished.
Stop the Q Apply program	Your configuration might require that you stop the Q Apply program to be able to maintain your target tables. This affects all of the target tables for that Q Apply server. The messages for Q subscriptions are placed on their receive queues and will be processed when the Q Apply program is started again.

Considerations for rebinding packages and plans for Q Replication and Event Publishing

Packages for Q Replication and Event Publishing must remain bound with isolation UR (uncommitted reads) to maintain optimal system performance.

Many of the packages and plans for Q Replication and Event Publishing are bound using isolation UR. Your internal maintenance programs that are used for automatic rebinding of these packages and plans can cause contention problems between the Q Capture program and Q Apply program. If your internal maintenance programs rebind the replication packages with standard options such as CS (cursor stability), they will interfere with the Q Capture program and the Q Apply program.

z/OS

The Q Capture program, Q Apply program, and Common packages are bound automatically. You can use the z/OS sample ASNQBNDL to bind ASNCOMMON, ASNQCAPTURE, ASNQAPPLY, and ASNMON packages at a DB2 subsystem.

Q Replication and Event Publishing specify the VERSION AUTO setting when packages are precompiled. DB2 for z/OS automatically frees any packages that are older than the two most current versions.

Linux UNIX Windows

The packages for the Q Capture program, the Q Apply program, and the Replication Alert Monitor, are bound automatically the first time that the program connects to its control tables.

Chapter 21. Comparing and repairing tables

The **asntdiff** and **asntrep** commands detect and repair differences between tables. In Q Replication and SQL Replication, the commands enable you to find differences quickly and synchronize tables without performing a full refresh, or load, of the target table.

About this task

Source and target tables can lose synchronization, for example if a target table is unexpectedly changed by a user or application, or if you experienced an extended network or target system outage.

The **asntdiff** and **asntrep** commands run independently of the Q Capture, Q Apply, Capture, and Apply programs. They use DB2 SQL to fetch data from the source table and the target table and do not use WebSphere MQ queues. The compare and repair utilities do not depend on logs, triggers, or isolation level.

Procedure

To compare and repair tables, run the **asntdiff** command, and then run the **asntrep** command.

Table compare utility (asntdiff)

The **asntdiff** command compares the columns in one table to their corresponding columns in another table and generates a list of differences between the two in the form of a DB2 table.

To use the compare utility, you run the **asntdiff** command and specify the name of a Q subscription (Q Replication) or subscription set member (SQL Replication) that contains the source and target tables that you want to compare. You can also use SQL statements in an input file to specify the tables to compare.

The following sections explain how to use the asntdiff command:
- "Overview of the asntdiff command"
- "When to use the compare utility" on page 318
- "Where differences are stored" on page 318
- "Required authorizations" on page 319
- "Restrictions for key columns at source and target" on page 320
- "Data type considerations" on page 320
- "Effects of filtering" on page 321
- "Comparisons based on queries instead of subscriptions" on page 322
- "Comparing a subset of table rows" on page 322

Overview of the asntdiff command

You can run the **asntdiff** command on Linux, UNIX, Windows, and z/OS operating systems. The command compares tables on Linux, UNIX, Windows, z/OS, or System i® operating systems. The **asntdiff** command can be used with federated sources and targets if the corresponding columns in the two tables have the same data types.

z/OS The ASNTDIFF sample job in the SASNSAMP data set provides further information that is specific to the z/OS platform.

For Q Replication, the target must be a user copy table or a consistent-change-data (CCD) table that is condensed and complete. Stored procedure targets are not supported. For SQL Replication, the target must be a user table, point-in-time table, replica table, user-copy table, or consistent-change-data (CCD) table that is condensed and complete.

When you run the command, you specify an SQL WHERE clause that uniquely identifies the Q subscription or subscription set member:

Q Replication
The WHERE clause identifies a row in the IBMQREP_SUBS control table at the Q Capture server, based on the value of the SUBNAME column. For example:

```
where="subname = 'my_qsub'"
```

SQL Replication
The WHERE clause identifies a row in the IBMSNAP_SUBS_MEMBR table at the Apply control server, based on the value of the SET_NAME column. For example:

```
where="set_name = 'my_set' and source_table='EMPLOYEE'"
```

You might need to use more predicates in the WHERE clause to uniquely identify the subscription set member. For example, you might need to add the APPLY_QUAL, the SOURCE_OWNER, the TARGET_OWNER, or the TARGET_TABLE column from the IBMSNAP_SUBS_MEMBR table to the clause.

When to use the compare utility

The best time to use the utility is when the source and target tables are stable. You might want to run the utility when the Q Capture and Q Apply programs or Capture and Apply programs are idle. For example, you could run the utility when the Q Capture program reached the end of the DB2 recovery log and all changes are applied at the target. If applications are still updating the source, the comparison might not be accurate.

If the replication programs are running, you might need to run the **asntdiff** command more than once to get a complete picture of evolving differences between the source and target tables.

Where differences are stored

The **asntdiff** command creates a difference table in the source database or subsystem to store differences that it finds.

The difference table is named *schema*.ASNTDIFF, where *schema* is the value specified in the DIFF_SCHEMA parameter. If the schema is not specified, it defaults to ASN. You can also use the DIFF parameter to specify a table name.

By default, the difference table is created in the default DB2 user table space. You can specify a different, existing table space by using the DIFF_TABLESPACE parameter.

The difference table has two or more columns. One column is named DIFF, with a blank space at the end on Linux, UNIX, and Windows. The value in the DIFF column is a character that indicates an insert, update, or delete operation followed by a numeric value that indicates which table contains a row with differences. The other columns contain the value of replication key columns. There is one row in the difference table for each unmatched row in the target table.

The difference table uses three identifiers that indicate the operation that is needed to change the target table so that it matches the source table:

`D` (delete)
: Indicates that a row with the key value exists only at the target and not at the source.

`U` (update)
: Indicates that rows with the same key value exist at both the source and target, but at least one non-key column is different at the target.

`I` (insert)
: Indicates that a row with the key value exists only at the source and not at the target.

A value of ? 1 indicates that there is an invalid character in one or more source columns.

A value of ? 2 indicates that there is an invalid character in one or more target columns.

Example:

The following list of values is returned by comparing an EMPLOYEE table at the source with a target copy of the same table. The key column for replication is the employee number, EMPNO:

```
DIFF  EMPNO
U 2   000010
I 2   000020
I 2   000040
D 2   000045
I 2   000050
D 2   000055
```

The first row in the example shows that a row with the key value `000010` exists at both the source and target tables, but at least one non-key column at the target has a different value. The next two rows show that rows with the key values `000020` and `000040` exist only at the source. The fourth row shows that a row with the key value `000045` exists only at the target.

The values `? 1` and `? 2` are not shown in the example.

Required authorizations

These database authorizations are required for the compare utility:

- Access privileges to the tables that are being compared, and to the replication control tables unless the -f (file) option is used

- Linux UNIX Windows Read privileges for the password file if the PWDFILE keyword is used
- WRITE privileges for the directory that is specified by the DIFF_PATH keyword
- To create the difference table, CREATETAB authority on the source database and USE privilege on the table space. In addition, one of the following privileges is needed:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on an existing schema if the table is created in this schema

  On z/OS, if the user ID that runs asntdiff does not have authority to create tables, you can use the SQLID keyword to specify an authorization ID that can be used to create the difference table.
- DROPIN privilege on the schema to drop the difference table unless DIFF_DROP=N is used
- SELECT, DELETE, and INSERT privileges on the difference table (at the source). The default schema name is ASN and the default table name is ASNTDIFF.

Restrictions for key columns at source and target

The asntdiff utility supports multiple-byte character sets when the database is defined with SYSTEM or IDENTITY. However, the columns that are used as keys for replication at the source and target tables must use single-byte characters for the utility to compare the tables.

In a Linux, UNIX, or Windows database that uses Unicode, the characters in key data cannot be greater than the base U.S. English ASCII subset (first 256 ASCII characters) or the asntdiff utility cannot compare the tables.

Data type considerations

You need to consider the data types of the tables that you are comparing when using asntdiff.

Different data types in sources and targets
The compare utility can build two SELECT SQL statements that are based on the description of a subscription. To obtain the differences between the source and target tables, the utility compares the data that result from executing both statements. The data types and lengths of the columns for both SQL statements must be the same.

SQL Replication
The utility builds the SQL statement for the source by using the EXPRESSION column in the IBMSNAP_SUBS_COLS table.

Q Replication
The data types for both the source and the target must be the same.

Unsupported data types
The compare utility does not support comparisons between the following data types:

Nonkey columns
DECFLOAT, BLOB_FILE, CLOB_FILE, DBCLOB_FILE

Key columns
DECFLOAT, BLOB, CLOB, DBCLOB, VARGRAPHIC, GRAPHIC, LONG_VARGRAPHIC, BLOB_FILE, CLOB_FILE, DBCLOB_FILE, XML

Comparing the GRAPHIC data type
Columns with the GRAPHIC data type at the source and target might not match when you use the utility to compare the source and target tables. DB2 columns with the GRAPHIC data type have blank padding after the graphic data. This padding might be single-byte or double-byte spaces, depending on the code page that the database was created in. This padding might cause data to not match between the source and the target tables, especially if the source and target tables are in different code pages. This padding applies only to GRAPHIC data types and not other graphic data types such as VARGRAPHIC or LONG VARGRAPHIC.

To compare columns with GRAPHIC data types, you must remove the blank padding in the data before you compare the source and target tables by using the DB2 scalar function `rtrim(<column>`. This function eliminates the code page differences for single-byte or double-byte spaces and ensures that the utility compares the GRAPHIC data in a consistent manner.

TIMESTAMP WITH TIMEZONE restriction
The compare utility does not support comparisons that involved the TIMESTAMP WITH TIMEZONE data type that was introduced in DB2 for z/OS Version 10.

Effects of filtering

In some cases, differences between source and target tables are intentional, for example, if you use a search condition in Q Replication to filter which rows are replicated. The utility will not show differences between source and target tables that are a result of predicates or suppressed deletes.

Row filtering
The compare utility uses information from the replication control tables to avoid showing intentional differences:

SQL Replication
The utility uses the PREDICATES column in the IBMSNAP_SUBS_MEMBR table to select rows from the source tables. The value of the UOW_CD_PREDICATES column is ignored (asntdiff looks directly at the source table, where the Apply program looks at the CD table).

Q Replication
The utility uses the value of the SEARCH_CONDITION column in the IBMQREP_SUBS table to build the WHERE clause for the SELECT statement.

Suppressed delete operations
In Q Replication, you can choose to suppress replication of delete operations from the source table. If you do not replicate delete operations, rows that exist in the target table might not exist in the source table. When the SUPPRESS_DELETES value for a Q subscription is Y, the asntdiff utility ignores the rows that are unique to the target and reports no differences. A warning is issued to indicate how many rows were suppressed.

The asntdiff -f (input file) option does not support SUPPRESS_DELETES because it bases the table comparison on a SQL SELECT statement rather than the Q subscription definition.

Comparisons based on queries instead of subscriptions

The **asntdiff -f** command option enables you to do differencing by using SQL SELECT statements that are read from an input file. This option provides greater flexibility to do differencing between two generic tables. The **asntdiff -f** option does not use replication definitions to determine which tables and rows to compare as the standard asntdiff command does.

The **asntdiff -f** option works for all tables on Linux, UNIX, Windows, and z/OS. For details on this option, see "asntdiff –f (input file) command option" on page 433.

In addition to the SELECT statements, the input file contains the source and target database information, the difference table information, and optional parameters that specify methods for processing the differences. You can use a password file that is created by the **asnpwd** command to specify a user ID and password for connecting to the source and target databases.

Note: To compare DB2 XML columns by using the **asntdiff -f** option, you need to serialize the XML column as a character large-object (CLOB) data type by using the XMLSERIALIZE scalar function. For example, this SELECT statement in the input file compares the XMLColumn column in the source table Table 1 to the same column in another database table (the TARGET_SELECT would use the same function):

```
SOURCE_SELECT="select ID, XMLSERIALIZE(XMLColumn AS CLOB) AS XMLColumn
from Table1 order by 1"
```

Comparing a subset of table rows

You can use the **asntdiff** RANGECOL parameter to compare only some of the rows in the two tables. This parameter specifies a range of rows from the source table that are bounded by two timestamps. You provide the name of a DATE, TIME, or TIMESTAMP column in the source table, and then use one of three different clauses for specifying the range. When you compare tables that are involved in peer-to-peer replication, you can use the IBM-generated IBMQREPVERTIME column for the source column in the range clause.

The RANGECOL parameter is not valid for the **asntdiff -f** (input file) option. You can use a SQL WHERE clause in the input file to achieve similar results.

Running the asntdiff utility in parallel mode (z/OS)

By using the PARALLEL=Y option with the asntdiff command, you can run the table compare utility in a parallel mode that provides optimal performance when comparing very large tables.

In parallel mode, the asntdiff utility uses as many as 21 threads to compare the data in two specified tables, significantly decreasing processing time while maintaining the accuracy of the comparison. With this mode:

- The utility internally partitions the two tables and compares these partitions in parallel.

- Row retrieval for each partition pair occurs in parallel.
- The differences that are found in each partition pair are then combined to obtain the overall result.

Using this method reduces processing time and reduces memory use because it avoids materializing large intermediate results. The parallel mode also minimizes network traffic because the checksum calculations are pushed down to each database.

To use asntdiff in parallel mode, it is recommended but optional that the two tables have date, time, or timestamp columns and a unique index or primary key. Both tables must be on DB2 for z/OS and must use the same code page and collation sequence.

The following sections provide more detail about the use of parallel mode.
- "Installation requirements"
- "Required authorizations"
- "Restrictions" on page 324
- "Usage tips" on page 324

Installation requirements

To use the asntdiff utility in parallel mode, you must install a stored procedure (ASNTDSP) at the systems that contain any table to be compared. The ASNTDSP sample job is included in the SASNSAMP dataset. For best results, use an application environment with NUMTCB = 8 - 15.

The following code defines ASNTDSP:

```
CREATE PROCEDURE ASN.TDIFF
 ( IN     SELECTSTMT          VARCHAR(32700),
   IN     GTTNAME             VARCHAR(128),
   IN     OPTIONS             VARCHAR(1331),
   OUT    RETURN_BLOCK_CRC    VARCHAR(21),
   OUT    RETURN_NUM_ROWS     INTEGER,
   OUT    RETURN_CODE         INTEGER,
   OUT    RETURN_MESSAGE      VARCHAR(30000))
 PARAMETER CCSID EBCDIC
 EXTERNAL NAME ASNTDSP
 LANGUAGE C
 PARAMETER STYLE GENERAL WITH NULLS
 COLLID ASNTDIFF
 WLM ENVIRONMENT !!WLMENV4!!
 MODIFIES SQL DATA
 ASUTIME NO LIMIT
 STAY RESIDENT YES
 PROGRAM TYPE MAIN
 SECURITY USER
 RUN OPTIONS 'TRAP(OFF),STACK(,,ANY,),POSIX(ON)'
 COMMIT ON RETURN NO;
```

Required authorizations

These database authorizations and privileges or higher are required to run asntdiff in parallel mode:
- SELECT on the tables that are being compared.
- These privileges for the difference table:

  - CREATEIN and DROPIN on an existing schema if the table is created in this schema
  - If the parameter DIFF_TABLESPACE is not specified, CREATETAB and CREATETS authority on the default database DSNDB04. Note that on DB2 for z/OS Version 10, if the IN clause is not specified with CREATE TABLE, CREATETAB privilege on database DSNDB04 is required.
  - If the parameter DIFF_TABLESPACE is explicitly specified, CREATETAB authority on the database that contains the DIFF_TABLESPACE and USE privilege on the table space that is specified by DIFF_TABLESPACE
- To create the created global temporary tables in the work file databases, CREATETMTAB privileges at any databases that are involved
- SELECT, DELETE, and INSERT privileges on the database where the difference table is created. The default schema name is ASN and the default table name is ASNTDIFF.
- SELECT privileges on the catalog tables SYSIBM.SYSDUMMY1, SYSIBM.SYSTABLES, SYSIBM.SYSKEYS, and SYSIBM.SYSINDEXES
- EXECUTE privileges on procedure ASN.TDIFF
- EXECUTE privileges on package ASNTDIFF.ASNTDSP
- EXECUTE privileges on plan ASNRD101

Make sure that the necessary DB2 authorizations are granted to the user ID that runs asntdiff before you start the utility. You can use the TARGET_SQLID and SOURCE_SQLID parameters to change the value of CURRENT SQLID to an authorization ID that has sufficient authorities.

Restrictions

- The two tables that are being compared must have the same code page and collation sequence. Otherwise, use the PARALLEL=N option (the default) to compare the tables.
- When used in parallel mode, the asntdiff utility should be run from z/OS as a batch job that uses JCL.
- The tables that are being compared must reside on z/OS.
- Only the -F PARM option is supported when asntdiff runs in parallel mode.
- The supported SELECT statements that you use with the SOURCE_SELECT and TARGET_SELECT parameters must use this strucure:

  ```
  SELECT xxx FROM yyy (WHERE zzz) ORDER BY aaa
  ```

  The WHERE clause is optional.
- Supported data types for nonkey columns are DATE, TIME, TIMESTAMP, VARCHAR, CHAR, LONG VARCHAR, FLOAT, REAL, DECIMAL, NUMERIC, BIGINT, INTEGER, SMALLINT, ROWID, VARBINARY, BINARY, VARGRAPH, GRAPHIC, LONGRAPH.
- Supported data types for key columns are DATE, TIME, TIMESTAMP, VARCHAR, CHAR, LONG VARCHAR, FLOAT, REAL, DECIMAL, NUMERIC, BIGINT, INTEGER, SMALLINT, ROWID, VARBINARY, BINARY.

For other restrictions, see "asntdiff –f (input file) command option" on page 433.

Usage tips

- Columns that are used in WHERE clauses and ORDER BY clauses should use an index. The columns that you specify in the ORDER BY clause must follow the same index column order and ascending/descending attributes. Use the RUNSTATS and REORG utilities to keep table access information current.

- In parallel mode, the asntdiff utility does support a mix of ascending and descending order in the ORDER BY clause. The mix should be same as in the index. However, the utility might not give you optimal performance when the index uses this mixture. Results will still be correct.
- For optimal performance:
  - Increase the system resource limit for application threads and set NUMTHREADS to 21.
  - Do not use column alias and expressions against the key columns in SOURCE_SELECT and TARGET_SELECT.

Table repair utility (asntrep)

The **asntrep** command repairs differences between source and target tables on all DB2 servers by deleting, inserting, and updating rows. The command runs on Linux, UNIX, or Windows operating systems.

The **asntrep** command uses the difference table that is generated by the **asntdiff** command to take the following actions:

- Delete rows from the target table that have no matching key in the source table
- Insert rows that are in the source table but have no matching key in the target table
- Update target rows that have matching keys in the source but different non-key data

For Q Replication, the target must be a table; it cannot be a stored procedure. For SQL Replication, the target must be a user table, a point-in-time table, a replica table, or a user-copy table. If you use the asntrep utility with a Q subscription for peer-to-peer replication, you must repair all of the copies of a logical table two copies at a time.

You run the **asntrep** command after you run the **asntdiff** command. The **asntrep** command copies the difference table from the source database or subsystem to the target, and then uses the copy to repair the target table.

To use the **asntrep** command, you provide the same WHERE clause that you used for the **asntdiff** command to identify the Q subscription or subscription set member that contains the source and target tables that you want to synchronize. The repair utility does not support the use of an input file as does the compare utility.

During the repair process, referential integrity constraints on the target table are not dropped. An attempt to insert or delete a row from a target table can fail if the insert or delete operation violates a referential integrity constraint. Also, a duplicate source row might be impossible to repair at the target if the target has a unique index.

How the compare utility handles DB2 SQL compatibility features

DB2 for Linux, UNIX, and Windows Version 9.7 introduced SQL compatibility enhancements such as variable-length timestamps, the VARCHAR2 data type with special character string processing, and DATE-TIMESTAMP compatibility. Some considerations are required to use the **asntdiff** command with these new features.

The following sections describe these considerations:
- "Comparing TIMESTAMP non-key columns with different precisions "
- "Comparing TIMESTAMP key columns with different precision"
- "Considerations when using the DATE data type as TIMESTAMP(0)" on page 328
- "Behavior when using the rangecol parameter " on page 327
- "Compatibility option for text based strings " on page 328
- "asntdiff file option (asntdiff –f) " on page 329

Comparing TIMESTAMP non-key columns with different precisions

When asntdiff compares two tables that have TIMESTAMP columns of different precision, it creates a truncated version of the longer TIMESTAMP column and then compares the two equal-length values.

In the following example, Table A and Table B have TIMESTAMP columns of different lengths:

Table A	Table B

`Col1 - TIMESTAMP(6)`
`2009-02-05-12.46.01.126412`
	`Col1 - TIMESTAMP(12)`
	`2009-02-05-12.46.01.126412000000`

In this situation, asntdiff compares 2009-02-05-12.46.01.126412 from Table A with the truncated value of 2009-02-05-12.46.01.126412 from Table B, and reports matching values.

In the next example, Table A has a longer TIMESTAMP column than Table B because the target value was truncated as a result of replication (typically this occurs when the target database is pre-Version 9.7 and only supports the default TIMESTAMP precision of six-digits):

Table A	Table B

`Col1 - TIMESTAMP(12)`
`2009-02-05-12.46.01.126412123456`
	`Col1 - TIMESTAMP(6)`
	`2009-02-05-12.46.01.126412`

Here, asntdiff compares a truncated version of the source value, 2009-02-05-12.46.01.126412, with the target value of 2009-02-05-12.46.01.126412 and reports a match. Whenever asntdiff truncates a TIMESTAMP column, the utility issues warning message ASN4034W.

Comparing TIMESTAMP key columns with different precision

When asntdiff compares key columns with different TIMESTAMP precision, the same basic concepts hold: A version of the longer timestamp column is truncated to the length of the shorter timestamp column for the purpose of comparing.

In the following example, Table A has a TIMESTAMP(6) key column and a character column, and Table B has a TIMESTAMP(12) key column and a character column.

Table A	Table B
`KEYCol1 - TIMESTAMP(6)` `2009-02-05-12.46.01.126412`	`KEYCol1 - TIMESTAMP(12)` `2009-02-05-12.46.01.126412000000`
`Col2 CHAR(12)` `"test String"`	`Col2 CHAR(12)` `"String test"`

The utility compares the Table A key value of 2009-02-05-12.46.01.126412 with a truncated version of the Table B key value, 2009 -02-05-12.46.01.126412, and reports a match. It then compares the nonkey column values "test String" and "String test" and reports a "U 2" (update needed) in the difference table to signify that rows with the same key value exist at both the source and target, but at least one non-key column is different at the target:

DIFF	Col1 TIMESTAMP(6)
`U 2`	`2009-02-05-12.46.01.126412`

The second column in the difference table always contains the key value. Because the difference table DDL is based on the source table, asntdiff uses the source TIMESTAMP(6) value. If the source table had the longer TIMESTAMP column, for example a TIMESTAMP(12), the utility would truncate the TIMESTAMP(12) to a TIMESTAMP(6) in order to compare the keys. However, it would use the source table's TIMESTAMP(12) definition to create the difference table. The key value that is written to the difference table is, however, the key value that has been used during comparison: TIMESTAMP(6). This value is then padded to a TIMESTAMP(12).

In this situation, when you use the asntrep utility to repair differences between the source and target tables, asntrep assumes that the target key-column value is a result of replication, and thus if DB2 pads with 0s, a matching key on the target side is found and can be updated.

Behavior when using the rangecol parameter

The asntdiff **`rangecol`** invocation parameter, which enables you to compare a subset of rows in two tables based on a specified timestamp column, also requires special attention when the timestamp column is variable length and also a key column.

Table A

KEYCol1 - TIMESTAMP(12)	Col2 CHAR(12)
`2009-02-05-12.46.01.126412123456`	`"test String"`
`2009-02-05-12.46.02.126412123456`	`"test String"`
`2009-02-05-12.46.03.126412123456`	`"test String"`

Table B

KEYCol1 - TIMESTAMP(6)	Col2 CHAR(12)
`2009-02-05-12.46.01.126412`	`"String test"`
`2009-02-05-12.46.02.126412`	`"String test"`

KEYCol1 - TIMESTAMP(6)	**Col2 CHAR(12)**
`2009-02-05-12.46.03.126412`	`"String test"`

Using Table A and Table B above as examples, consider the following rangecol portion of an asntdiff invocation in which the TIMESTAMP(6) is used to specify which rows to compare:

```
RANGECOL="'KEYCol1' FROM: '2009-02-05-12.46.01.126412'
TO: '2009-02-05-12.46.03.126412'"
```

The range clause is rewritten by asntdiff into a SQL SELECT statement with a BETWEEN clause:

```
WHERE ("KEYCol1" BETWEEN '2009-02-05-12.46.01.126412'
AND '2009-02-05-12.46.03.126412')
```

To include all rows in the above scenario, use the source key values in the range clause. As a general rule, always use the longer TIMESTAMP column value in the range clause. For example, the following statement considers all six rows on both target and source side:

```
RANGECOL="'KEYCol1' FROM: '2009-02-05-12.46.01.126412123456'
TO: '2009-02-05-12.46.03.126412123456'"
```

Note: The scenarios described are only valid when the target table content has exclusively been populated by the Q Apply or Apply program. Any manual interaction with the target table could result in unexpected asntdiff results. As always, a thorough analysis of the results in the differencing table is required before you use the **`asntrep`** command to repair differences.

Considerations when using the DATE data type as TIMESTAMP(0)

The asntdiff utility does not support comparison of DATE and TIMESTAMP(0) data types. If the DATE data type compatibility feature is not enabled for the database that features the table with the DATE column, asntdiff gives the following message and terminates abnormally: "ASN4003E The data type or the length ... are not compatible."

The following example shows two databases, the second of which is enabled to use TIMESTAMP(0) columns for dates:

Database 1, Table A	**Database 2 (compatibility vector 0x40), Table B**

`DATE`
`02/05/2009`
	`TIMESTAMP(0)`
	`2009-02-05-12.46.01`

To compare these two tables, you must use the asntdiff file option and manually cast either of the two data types to a compatible data type.

Compatibility option for text based strings

With the compatibility option for character data enabled, an insert of an empty string into a text-based column results in a null value.

Database 1, Table A KEYCol1 - TIMESTAMP(6)	Database 1, Table A Col2 VARCHAR(12)
2009-02-05-12.46.01.126412	""

Database 2 (compatibility vector 0x20), Table B KEYCol1 - TIMESTAMP(6)	Database 2 (compatibility vector 0x20), Table B Col2 VARCHAR(12)
2009-02-05-12.46.01.126412	NULL

By default asntdiff flags a difference in Col2 and reports an update needed in the difference table. If you do not want asntdiff to report this as a difference, you can use the asntdiff file option with the following SQL statement in the SOURCE_SELECT parameter:

```
SELECT Col2 CASE WHEN Col2 = \'\' THEN NULL ELSE Col2 END FROM Database1
```

In any case, the warning message ASN4035W is issued once to make you aware of this scenario.

asntdiff file option (asntdiff –f)

To override any of the default behaviors mentioned above, it is recommended to employ the asntdiff file option that was introduced in Version 9.7.

The option allows you to use any SQL expression, for example you could use a CAST statement to avoid the truncation when comparing different length timestamp columns.

The following example pads the TIMESTAMP(6) to a TIMESTAMP(12):

```
SOURCE_SELECT= "SELECT CAST(KEYCol1 AS TIMESTAMP(12) ) AS KEYCol1, Col2
FROM TABLE_A ORDER BY 1"
TARGET_SELECT= "SELECT KEYCol1, Col2 FROM TABLE_B ORDER BY 1"
```

Chapter 22. Using system services to operate the Q replication and event publishing programs

You can operate the replication programs for Q replication and event publishing by using system services that are designed for each operating system.

The z/OS operating system can use the job control language (JCL), system-started tasks, or the automatic restart manager (ARM), to operate the replication programs. The Windows operating system can operate the replication programs by using a system service. You can schedule replication programs on the Linux operating system, the UNIX operating system, the Windows operating system, and the z/OS operating system.

Using z/OS system services to run the Q replication and event publishing programs

On z/OS, you can start the replication programs by using JCL or system-started tasks. You can use the Automatic Restart Manager (ARM) to restart failed replication programs.

Running the Q replication and event publishing programs by using JCL

When replicating data on z/OS, you can use JCL to operate the replication programs.

Specifying the CAPTURE_PATH parameter (z/OS)

Before you can run the Q Capture program using JCL, you must specify the **CAPTURE_PATH** parameter, which contains the path that references the data set where the transaction log is stored.

About this task

If you do not specify the parameter, the Q Capture program writes log files to the home directory of the user who submits the JCL.

Procedure

To specify the **CAPTURE_PATH** parameter on z/OS, use one of the following methods:

JCL Use the PARM field of the JCL statement that will start the Q Capture program. For example:

```
//  PARM='/CAPTURE_SERVER=DSN7 CAPTURE_PATH=//JAYQC // LOGSTDOUT
capture_schema=JAY'
```

In this example, the Q Capture program writes its log files to the USER1.JAYQC.D7DP.JAY.QCAP.LOG file. USER1 is the user who submits the JCL.

If you want the data set for the log to have a specific high level qualifier, use this example:

```
// PARM='/capture_server=DSN7 capture_schema=JAY //
CAPTURE_PATH=//''OEUSR01'
```

Now the Q Capture program writes its log files to the OEUSR01.DSN7.JAY.QCAP.LOG file.

If you want to specify the path to SYSADM.XYZ, use one of the following examples:

```
//  PARM='/CAPTURE_server=DSN7 Capture_path=//''SYSADM.XYZ //
capture_schema=JAY'
```

```
//  PARM='/CAPTURE_server=DSN7 capture_schema=JAY //
capture_PATH=//''SYSADM.XYZ'
```

Ensure that the path name does not exceed the 44 character limit for MVS data sets. The user ID that runs this JCL must be authorized to write to the above data set.

SQL Issue an insert statement to the IBMQREP_CAPPARMS table. For example:

```
INSERT INTO JAY.IBMQREP_CAPPARMS (qmgr, restartq, adminq, startmode,
memory_limit, commit_interval, autostop, monitor_interval,monitor_limit,
trace_limit, signal_limit, prune_interval, sleep_interval, logreuse,
logstdout, term, capture_path, arch_level)
   VALUES
   ( 'CSQ1', 'IBMQREP.ASN.RESTARTQ','IBMQREP.ASN.ADMINQ',
     'WARMSI', 32, 500, 'N', 300000, 10080, 10080, 10080, 300, 5000,
     'N', 'N', 'Y', '//JAYQC', '901');
```

If you want the data set for the log to have a specific high level qualifier, use this example:

```
INSERT INTO JAY.IBMQREP_CAPPARMS (qmgr, restartq, adminq, startmode,
memory_limit, commit_interval, autostop, monitor_interval,monitor_limit,
trace_limit, signal_limit, prune_interval, sleep_interval, logreuse,
logstdout, term, capture_path, arch_level)
  VALUES
  ( 'CSQ1', 'IBMQREP.ASN.RESTARTQ', 'IBMQREP.ASN.ADMINQ',
    'WARMSI', 32, 500, 'N',300000, 10080, 10080, 10080, 300, 5000, 'N',
    'N', 'Y', '//''OEUSR01','901');
```

To specify the path to SYSADM.XYZ, use this example:

```
INSERT INTO JAY.IBMQREP_CAPPARMS  (qmgr, restartq, adminq, startmode,
memory_limit, commit_interval, autostop, monitor_interval,monitor_limit,
trace_limit, signal_limit, prune_interval, sleep_interval, logreuse,
logstdout, term, capture_path, arch_level)
   VALUES
   ( 'CSQ1', 'IBMQREP.ASN.RESTARTQ','IBMQREP.ASN.ADMINQ',
     'WARMSI', 32, 500,   'N',300, 10080, 10080, 10080, 300, 5000,
     'N', 'N', 'Y', '//''SYSADM.XYZ','901');
```

Starting the Q Capture program with JCL

The WebSphere Replication Server for z/OS Version 9 samples library contains sample JCL and scripts that you can modify and use to start the Q Capture program.

Recommendation: Copy the jobs from the SASNSAMP library to a different library before making changes. See the Program Directory for a complete list of the sample jobs found in the library.

Procedure

To start a Q Capture program by using JCL:

1. Specify the appropriate optional invocation parameters in the PARM field of the Q Capture job.

You must set the TZ (time zone) and LANG (language) environment variables in the JCL if you did not set them in the system-wide `/etc/profile` file or in the profile file in the home directory of the running replication program. For more information about setting these variables, see Replication installation and customization for z/OS. The following example from the invocation JCL includes setting the TZ and LANG variables:

```
//CAPJFA EXEC PGM=ASNQCAP,
// PARM='ENVAR('TZ=PST8PDT','LANG=en_US')/CAPTURE_SERVER=DQRG
// capture_schema=JFA'
```

2. Specify a directory path with the TMPDIR environment variable if you want the replication programs to write temporary files to a directory other than the `/tmp` directory

Starting the Q Apply program with JCL

The WebSphere Replication Server for z/OS Version 9 samples library contains sample JCL and scripts that you can modify and use to start the Q Apply program.

Recommendation: Copy the jobs from the SASNSAMP library to a different library before making changes. See the Program Directory for a complete list of the sample jobs found in the library.

Procedure

To start a Q Apply program by using JCL:

Specify the appropriate optional invocation parameters in the PARM field of the Q Apply job. The following example shows the invocation JCL for the Q Apply program:

```
//PLS EXEC PGM=ASNQAPP, // PARM='APPLY_SERVER=DQRG APPLY_SCHEMA=JAY'
```

Starting the Replication Alert Monitor by using JCL

The samples library contains sample JCL and scripts that you can modify and use to start the Replication Alert Monitor.

Recommendation: Copy the jobs from the SASNSAMP library to a different library before making changes. See the Program Directory for a complete list of the sample jobs found in the library.

Procedure

To start the Replication Alert Monitor by using JCL:

Specify the appropriate optional invocation parameters in the PARM field of the Replication Alert Monitor job. The following example shows the invocation JCL for the Replication Alert Monitor:

```
//monasn EXEC PGM=ASNMON,PARM='monitor_server=DSN
monitor_qual=monqual'
```

Running the Q replication and event publishing programs with JCL in batch mode

To run the Q Capture, Q Apply, and Replication Alert Monitor programs on with JCL in batch mode, you customize the JCL in library SASNSAMP for the appropriate program.

Procedure

To run replication programs in batch mode:

1. Customize the JCL in library SASNSAMP for the appropriate program. Table 29 shows which sample job to use to start each program:

Table 29. Sample jobs to start the replication programs in JCL

Sample	Program
ASNQSTRA	Q Apply
ASNQSTRC	Q Capture
ASNSTRM	Replication Alert Monitor
ASNQTON	Trace (for the Q Capture program or the Q Apply program)

2. Prepare the JCL for z/OS by specifying the appropriate optional invocation parameters in the PARM field of the DPROPR jobs (Q Capture, Q Apply, Replication Alert Monitor and asntrc).

Working with running Q replication and event publishing programs by using the MVS MODIFY command

After you start the Q Capture program, the Q Apply program, or the Replication Alert Monitor, you can use the **MODIFY** command to stop the program or to perform related tasks.

About this task

For descriptions of the parameters that you can use with MODIFY, see asnqccmd: Working with a running Q Capture program, asnqacmd: Working with a running Q Apply program, and asnmcmd: Working with a running Replication Alert Monitor.

Procedure

To work with running programs on z/OS:

Run the **MODIFY** command from the z/OS console. You can use the abbreviation f, as shown in the following syntax example:

```
►►──f──jobname──,──┤ Parameters ├──────────────────────────────────────────►◄
```

f *jobname* , replaces the actual command name: **asnqccmd**, **asnqacmd**, or **asnmcmd**. The operational parameters that apply to each of the commands can be used with the f keyword.

For example, to stop a running Q Apply program that uses the PLS job name, you would use the following command:

```
F PLS,stop
```

Table 30 shows the Q Capture commands that you can run with the f keyword. In all examples, the job name is myqcap.

Table 30. Sample MODIFY commands for the Q Capture program

Parameter	Sample command that uses f keyword
prune	`f myqcap,prune`
qryparms	`f myqcap,qryparms`

Table 30. Sample MODIFY commands for the Q Capture program (continued)

Parameter	Sample command that uses f keyword
reinit	`f myqcap,reinit`
reinitq	`f myqcap,reinitq=`*send_queue_name*
startq	`f myqcap,startq=`*send_queue_name*
startq all	`f myqcap,startq all`
stopq	`f myqcap,stopq=`*send_queue_name*
status	`f myqcap,status`
status show details	`f myqcap,status show details`
stop	`f myqcap,stop`
chgparms	`f myqcap,chgparms autostop=`*y/n* `f myqcap,chgparms commit_interval=`*n* `f myqcap,chgparms logreuse=`*y/n* `f myqcap,chgparms logstdout=`*y/n* `f myqcap,chgparms monitor_interval=`*n* `f myqcap,chgparms monitor_limit=`*n* `f myqcap,chgparms prune_interval=`*n* `f myqcap,chgparms qfull_num_retries=`*n* `f myqcap,chgparms qfull_retry_delay=`*n* `f myqcap,chgparms sleep_interval=`*n* `f myqcap,chgparms signal_limit=`*n* `f myqcap,chgparms term=`*y/n* `f myqcap,chgparms trace_limit=`*n*

Table 31 shows the Q Apply commands that you can run with the f keyword. In all examples, the job name is myqapp.

Table 31. Sample MODIFY commands for the Q Apply program

Parameter	Sample command that uses f keyword
loaddonesub	`f myqapp,loaddonesub=`*receive_queue_name*`:`*q_sub_name*
prune	`f myqapp,prune`
qryparms	`f myqapp,qryparms`
stopq	`f myqapp,stopq=`*receive_queue_name*
startq	`f myqapp,startq=`*receive_queue_name*
startq;**skiptrans**	`f myqapp,startq="`*receive_queue_name*`;skiptrans=`*transaction_ID*`"`
reinitq	`f myqapp,reinit=`*receive_queue_name*
stop	`f myqapp,stop`
status	`f myqapp,status`
status show details	`f myqapp,status show details`
spillsub	`f myqapp,spillsub=`*receive_queue_name*`:`*q_sub_name*
resumesub	`f myqapp,resumesub=`*receive_queue_name*`:`*q_sub_name*

Table 31. Sample MODIFY commands for the Q Apply program (continued)

Parameter	Sample command that uses f keyword
chgparms	`f myqapp,chgparms autostop=`*y/n* `f myqapp,chgparms logreuse=`*y/n* `f myqapp,chgparms logstdout=`*y/n* `f myqapp,chgparms monitor_interval=`*n* `f myqapp,chgparms monitor_limit=`*n* `f myqapp,chgparms prune_interval=`*n* `f myqapp,chgparms term=`*y/n* `f myqapp,chgparms trace_limit=`*n* `f myqapp,chgparms deadlock_retries=`*n* `f myqapp,chgparms discardconflicts=`*y/n* `f myqapp,chgparms dropris=`*y/n*

Table 32 shows asntrc program commands that you can run with the f keyword. In all examples, the job name is myqcap.

Table 32. Sample MODIFY commands for the asntrc program

Task	Sample command that uses f keyword

Start a program trace with the **asntrc** command	`f myqcap,asntrc on`
	`f myqcap,asntrc statlong`

Format an asntrc fmt report and direct the output to a z/OS data set	`F myqcap, asntrc fmt -ofn //'USRT001.TRCFMT'`
Format an asntrc flw report and direct the output to a z/OS data set	`F myqcap, asntrc flw -ofn //'USRT001.TRCFLW'`
Stop a program trace	`F myqcap, asntrc off`

Recommendation: Preallocate asntrc flw and fmt output files so that they are large enough to contain the asntrc reports. Use these attributes:
- **Data set name:** `USRT001.TRCFMT` or `USRT001.TRCFLW`
- **Primary allocated cylinders:** 2
- **Normal allocated extents:** 1
- **Data class:** None (Current utilization)
- **Used cylinders:** 2
- **Record format:** VB used extents: 1
- **Record length:** 1028
- **Block size:** 6144
- **1st extent cylinders:** 2
- **Secondary cylinders:** 1
- **SMS compressible:** NO

Table 33 shows the Replication Alert Monitor commands that you can run with the f keyword. In all examples, the job name is mymon.

Table 33. Sample MODIFY commands for the Replication Alert Monitor program

Parameter	Sample command that uses f keyword
reinit	`f mymon,reinit`
status	`f mymon,status`
qryparms	`f mymon,qryparms`

Table 33. Sample MODIFY commands for the Replication Alert Monitor program (continued)

Parameter	Sample command that uses f keyword
suspend	`f mymon,suspend`
resume	`f mymon,resume`
stop	`f mymon,stop`
chgparms	`f mymon,chgparms monitor_interval=`*n* `f mymon,chgparms autoprune=y` `f mymon,chgparms trace_limit=`*n* `f mymon,chgparms alert_prune_limit=`*n* `f mymon,chgparms max_notifications_per_alert=`*n* `f mymon,chgparms max_notifications_minutes=`*n*

For information about **MODIFY**, see *z/OS MVS System Commands*.

Running the Q replication and event publishing programs with system-started tasks

You can use system-started tasks to operate the Q Capture program, Q Apply program, and Replication Alert Monitor.

Procedure

To start a program as a system-started task for the z/OS operating system:

1. Create a procedure (*procname*) in your PROCLIB.
2. Create an entry in the RACF STARTED class for *procname*. This entry associates *procname* with the RACF user ID to be used to start the Q Capture program. Make sure that the necessary DB2 authorization is granted to this user ID before you start the program.
3. From the z/OS console, run the command **start** *procname*. The following sample procedure is for the Q Capture program:

```
// PARM='CAPTURE_SERVER=DSN7 capture_schema=ASN startmode=cold'
//STEPLIB DD DSN=qrhlqual.SASNLOAD,DISP=SHR
// DD DSN=dsnhlqual.SDSNLOAD,DISP=SHR
//* DD DSN=mqhlqual.SCSQANLE,DISP=SHR
//* DD DSN=mqhlqual.SCSQLOAD,DISP=SHR
//* DD DSN=xmlhlqual.SIXMMOD1,DISP=SHR
//CAPSPILL DD DSN=&&CAPSPILL,DISP=(NEW,DELETE,DELETE),
// UNIT=VIO,SPACE=(CYL,(50,70)),
// DCB=(RECFM=VB,BLKSIZE=6404)
//MSGS DD PATH='/usr/lpp/db2repl_10_01/msg/En_US/db2asn.cat'
//CEEDUMP DD SYSOUT=* //SYSPRINT DD SYSOUT=* //SYSUDUMP DD DUMMY
```

qrhlqual
   The Q Replication target library high-level qualifier

dsnhlqual
   The DB2 target library high-level qualifier

mqhlqual
   The WebSphere MQ target library high-level qualifier

xmlhlqual
   The XML Toolkit library high-level qualifier

JCL that executes the Q Capture program must add the WebSphere MQ libraries to the STEPLIB if they are not installed in the LNKLST. You can also add the XML Toolkit libraries if you want replication to use ICU instead of UCS for code page conversions.

Using Automatic Restart Manager (ARM) to automatically restart replication and publishing (z/OS)

You can use the Automatic Restart Manager (ARM) recovery system on z/OS to restart the Q Capture, Q Apply, Capture, Apply, and Replication Alert Monitor programs.

Before you begin

Ensure that ARM is installed and that the replication programs are set up correctly. To use ARM with a replication program, ensure that the program is APF authorized. For example, to use ARM with the Q Apply, Apply, or Replication Alert Monitor program, you must copy the appropriate load module into an APF authorized library. (The Q Capture and Capture programs must be APF authorized regardless of whether or not you are using ARM.)

About this task

ARM is a z/OS recovery function that can improve the availability of specific batch jobs or started tasks. When a job or task fails, or the system on which it is running fails, ARM can restart the job or task without operator intervention.

ARM uses element names to identify the applications with which it works. Each ARM-enabled application generates a unique element name for itself that it uses in all communication with ARM. ARM tracks the element name and has its restart policy defined in terms of element names. For details about setting up ARM, see *z/OS MVS Sysplex Services Guide*.

Procedure

To use ARM to automatically restart replication and publishing programs:

1. Specify one of the following element names when you configure ARM:

Program	**Element name**
Q Capture	ASNQC*xxxxyyyy*
Q Apply	ASNQA*xxxxyyyy*
Capture	ASNTC *xxxxyyyy*
Apply	ASNTA *xxxxyyyy*
Replication Alert Monitor	ASNAM *xxxxyyyy*

   Where *xxxx* is the DB2 subsystem name and *yyyy* is the data-sharing member name (the latter is needed only for data-sharing configurations). The element name is always 16 characters long, padded with blanks.

2. Optional: If you have more than one instance of a replication or publishing program running within a data-sharing member, specify the **arm** parameter when you start the programs to create a unique ARM element name for each program instance.

   The **arm** parameter takes a three-character value that is appended to the element names that are listed in the previous table. The syntax is **arm**=*zzz*, where *zzz* can be any length of alphanumeric string. The replication program, however, will concatenate only up to three characters to the current name and pad with blanks, if necessary, to make a unique 16-byte name.

The replication programs use the element name to register with ARM during initialization. They do not provide ARM with an event exit when they register. The event exit is not needed because the replication programs do not run as a z/OS subsystem. ARM restarts registered programs if they terminate abnormally (for example, if a segment violation occurs). A registered replication program de-registers if it terminates normally (for example, due to a STOP command) or if it encounters an invalid registration.

Tip: If you start the Q Capture, Q Apply, Capture, Apply, or Replication Alert Monitor program with the parameter `term=n`, the program does not stop when DB2 is quiesced or stopped. In this case, the program does not de-register from ARM. It continues to run but does not perform its actual work until DB2 is unquiesced or started.

Replication services (Windows)

You can run the replication programs as a system service on the Windows operating system by using the Windows Service Control Manager (SCM).

Description of Windows services for replication

On the Windows operating system, a replication service is a program that starts and stops the Q Capture, Q Apply, Capture, Apply, or Replication Alert Monitor programs.

When you create a replication service, it is added to the SCM in Automatic mode and the service is started. Windows registers the service under a unique service name and display name.

The following terms describe naming rules for replication services:

Replication service name

The replication service name uniquely identifies each service and is used to stop and start a service. It has the following format:

```
DB2.instance.alias.program.qualifier_or_schema
```

Table 34 describes the inputs for the replication service name.

Table 34. Inputs for the replication service name

Input	Description
instance	The name of the DB2 instance.
alias	The database alias of the Q Capture server, Q Apply server, Capture control server, Apply control server, or Monitor control server.
program	One of the following values: QCAP (for Q Capture program), QAPP (for Q Apply program), CAP (for Capture program), APP (for Apply program), or MON (for Replication Alert Monitor program).
qualifier_or_schema	One of the following identifiers: Q Capture schema, Q Apply schema, Capture schema, Apply qualifier, or Monitor qualifier.

Example: The following service name is for a Q Apply program that has the schema ASN and is working with database DB1 under the instance called INST1:

```
DB2.INST1.DB1.QAPP.ASN
```

Display name for the replication service

The display name is a text string that you see in the Services window and it is a more readable form of the service name. For example:

```
DB2 - INST1 DB1 QAPPLY ASN
```

If you want to add a description for the service, use the Service Control Manager (SCM) after you create a replication service. You can also use the SCM to specify a user name and a password for a service.

Creating a replication service

You can create a replication service to start a Q Capture program, Q Apply program, Capture program, Apply program, and the Replication Alert Monitor program on Windows operating systems.

Before you begin

- Before you create a replication service, make sure that the DB2 instance service is running. If the DB2 instance service is not running when you create the replication service, the replication service is created but it is not started automatically.
- After you install DB2, you must restart your Windows server before you start a replication service.

About this task

When you create a service, you must specify the account name that you use to log on to Windows and the password for that account name.

You can add more than one replication service to your system. You can add one service for each schema on every Q Capture, Q Apply, or Capture control server, and for each qualifier on every Apply control server and Monitor control server, respectively. For example, if you have five databases and each database is an Q Apply control server and a Monitor control server, you can create ten replication services. If you have multiple schemas or qualifiers on each server, you could create more services.

Procedure

To create a replication service:

Use the **`asnscrt`** command.
When you create a service, you must specify the account name that you use to log on to Windows and the password for that account name.

Tip: If your replication service is set up correctly, the service name is sent to stdout after the service is started successfully. If the service does not start, check the log files for the program that you were trying to start. By default, the log files are in the directory specified by the DB2PATH environment variable. You can override this default by specifying the path parameter (**capture_path**,**apply_path**,**monitor_path**) for the program that is started as a service. Also, you can use the Windows Service Control Manager (SCM) to view the status of the service.

Starting a replication service

After you create a replication service, you can stop it and start it again.

About this task

Important: If you started a replication program from a service, you will get an error if you try to start the program by using the same schema or qualifier.

Procedure

To start a replication service, use one of the following methods.
- The Windows Service Control Manager (SCM)
- **net stop** command

Stopping a replication service

After you create a replication service, you can stop it and start it again.

About this task

When you stop a replication service, the program associated with that service stops automatically. However, if you stop a program by using a replication system command (**asnqacmd**, **asnqccmd**, **asnccmd**, **asnacmd**, or **asnmcmd**), the service that started the program continues to run. You must stop it explicitly.

Procedure

To stop a replication service, use one of the following methods.
- The Windows Service Control Manager (SCM)
- **net stop** command

Viewing a list of replication services

You can view a list of all your replication services and their properties by using the **asnlist** command.

Procedure

To view a list of replication services, use the **asnlist** command. You can optionally use the **details** parameter to view a list of replication services and descriptions of each service.

Dropping a replication service

If you no longer need a replication service you can drop it so that it is removed from the Windows Service Control Manager (SCM).

About this task

If you want to change the start-up parameters for a program that is started by a service, you must drop the service and then create a new one using new start-up parameters.

Procedure

To drop a service for replication commands, use the **asnsdrop** command.

Scheduling the replication programs

You can schedule the Q Capture program, the Q Apply program or the Replication Alert Monitor program to start at prescribed times.

Scheduling the replication and event publishing programs (Linux, UNIX)

To start a replication program at a specific time on a Linux or UNIX operating system, use the **at** command.

About this task

Table 35 shows commands that are used to start the replication programs at 3:00 p.m. on Friday.

Table 35. Scheduling commands for the replication programs (Linux, UNIX)

Replication program	Linux or UNIX command
Q Capture	`at 3pm Friday asnqcap autoprune=n`
Q Apply	`at 3pm Friday asnqapply applyqual=myqual`
Replication Alert Monitor	`at 3pm Friday asnmon monitor_server=db2srv1 monitor_qualifier=mymon`

Scheduling the replication programs (Windows)

You can use the Windows Service Control Manager, the Windows Task Manager, or the **at** command to start the replication programs at a scheduled time on Windows operating systems. This topic describes the use of the **at** command.

Procedure

To start a replication program at a specific time on a Windows operating system:

1. Start the Windows Schedule Service.
2. Create a password file in the directory of the replication program (CAPTURE_PATH, APPLY_PATH, or MONITOR_PATH). The password file must contain entries for the servers where the replication program that you are starting is running.
3. Issue the **at** command. Send the output to a file to check for errors.

   Table 36 shows commands that are used to start the replication programs at 3:00 p.m. on Friday. Note the "^" character when redirecting to a file.

Table 36. Scheduling commands for the replication programs (Windows)

Replication program	Windows command
Q Capture	`c:\>at 15:00 db2cmd asnqcap capture_server=qcapdb capture_schema=schema capture_path=c:\capture ^> c:\capture\asnqcap.out`

Table 36. Scheduling commands for the replication programs (Windows) (continued)

Replication program	Windows command
Q Apply	`c:\>at 15:00 db2cmd asnqapp apply_server=qappdb apply_schema=applyqual apply_path=c:\apply ^> c:\apply\asnqapp.out`
Replication Alert Monitor	`c:\>at 15:00 db2cmd asnmon monitor_server=mondb monitor_qual=monqual  monitor_path=c:\ monitor ^> c:\monitor\asnmon.out`

Scheduling the replication and event publishing programs (z/OS)

You can use either the **$TA JES2** command or the **AT NetView** command to start the Q Capture and Q Apply programs at a specific time on z/OS.

Procedure

To schedule replication and event publishing programs on the z/OS operating system:

1. Create a procedure that calls the program for z/OS in the PROCLIB.
2. Modify the Resource Access Control Facility (RACF) module (or appropriate definitions for your MVS security package) to associate the procedure with a user ID.
3. Link-edit the module in SYS1.LPALIB.
4. Use either the **$TA JES2** command or the **AT NetView** command to start the Q Capture program or the Q Apply program at a specific time. See *MVS/ESA JES2 Commands* for more information about using the **$TA JES2** command. See the *NetView for MVS Command Reference* for more information about using the **AT NetView** command.

Chapter 23. Naming rules and guidelines for Q Replication and Event Publishing—Overview

When you create objects for Q Replication and Event Publishing, you must observe certain restrictions for the types of characters and length of each object's name.

You should also be aware of how lowercase and uppercase characters are handled.

Naming rules for Q Replication and Event Publishing objects

The name for each Q Replication and Event Publishing object must conform to naming rules.

Table 37 lists the limits for names of objects in Q Replication and Event Publishing.

Table 37. Name limits for objects in Q Replication and Event Publishing

Object	Name limits	Length limit
Source and target tables	**DB2**: The names of DB2 source tables and target tables must follow the naming rules for DB2 table names. **non-DB2**: The names of non-DB2 target tables must follow the table naming rules that are required by the DB2 federated server to set up nicknames.	z/OS Both short and long schema names are supported for tables on DB2 for z/OS. Table names can include up to: • 18 bytes for subsystems that are running DB2 for z/OS Version 8 compatibility mode or earlier • 128 bytes for subsystems that are running DB2 for z/OS Version 8 new-function mode Linux UNIX Windows 30 or fewer characters
Source and target columns	**DB2**: The names of the DB2 source and target columns must follow the naming rules for DB2 column names. **non-DB2**: The names of non-DB2 target columns must follow the column naming rules that are required by the DB2 federated server to set up nicknames.	**CCD targets:** Before-image columns have a one-character prefix added to them. To avoid ambiguous before-image column names, ensure that source column names are unique to 127 characters and that the before-image column names will not conflict with existing column names when the before-image character prefix is added to the column name.

Table 37. Name limits for objects in Q Replication and Event Publishing (continued)

Object	Name limits	Length limit
Table owner	Both short and long schema names are supported for table owner.	Table owner names can be up to 128 bytes.
Send queue	The name of the send queue can include any characters that DB2 and WebSphere MQ allow for VARCHAR data types. Send queue names cannot contain spaces.	48 or fewer characters
Receive queue	The name of the receive queue can include any characters that DB2 and WebSphere MQ allow for VARCHAR data types. Receive queue names cannot contain spaces.	48 or fewer characters
Restart queue	The name of the restart queue can include any characters that DB2 and WebSphere MQ allow for VARCHAR data types. Restart queue names cannot contain spaces.	48 or fewer characters
Q subscription	The name of a Q subscription can include any characters that DB2 allows for VARCHAR data type columns. All Q subscription names must be unique. Because the name of the Q subscription is stored at both the source and target server, be sure that the name is compatible with the code pages for both the source and target servers. Q subscription names cannot contain spaces or semicolons (;).	30 or fewer characters
SUBGROUP	The name of the SUBGROUP for bidirectional and peer-to-peer replication can include any characters that DB2 allows for VARCHAR data type columns. **Recommendation:** Use a unique group name for the set of Q subscriptions for a logical table.	30 or fewer characters
Publication	The name of a publication can include any characters that DB2 allows for VARCHAR data type columns. For each Q Capture program, all publication names must be unique. Be sure that the name of the publication is compatible with the code page for the subscribing application. Publication names cannot contain spaces or semicolons (;).	30 or fewer characters

Table 37. Name limits for objects in Q Replication and Event Publishing (continued)

Object	Name limits	Length limit
Q Capture schema	The name of the Q Capture schema can include only the following valid characters: • A through Z (uppercase letters) • a through z (lowercase letters) • Numerals (0 through 9) • The underscore character (_)	The Q Capture schema can be a string of up to 128 characters. z/OS **Subsystems that are running Version 8 compatibility mode or earlier:** 18 or fewer characters
Q Apply schema	The name of the Q Apply schema can include only the following valid characters: • A through Z (uppercase letters) • a through z (lowercase letters) • Numerals (0 through 9) • The underscore character (_)	The Q Apply schema can be a string of up to 128 characters. z/OS **Subsystems that are running Version 8 compatibility mode or earlier:** 18 or fewer characters
Monitor qualifier	The name of the monitor qualifier can include only the following valid characters: • A through Z (uppercase letters) • a through z (lowercase letters) • Numerals (0 through 9) • The underscore character (_)	The name of the monitor qualifier can be a string of 18 or fewer characters.

How lowercase object names are handled for Q replication and publishing

The system commands for Q Replication and Event Publishing and the Replication Center, by default, convert all names that you provide to uppercase. Enclose a mixed-case character name in double quotation marks (or whatever character the target system is configured to use) to preserve the case and save the name exactly as you typed it.

For example, if you type `myqual` or `MyQual` or `MYQUAL`, the name is saved as `MYQUAL`. If you type those same names and enclose them in double quotation marks, they are saved as `myqual` or `MyQual` or `MYQUAL`, respectively. Some operating systems do not recognize double quotation marks and you might need to use an escape character, typically a backslash (\).

Windows **Important:** When setting up Windows services for the Q Capture program, the Q Apply program, or the Replication Alert Monitor, you must use unique names for the Q Capture schema, Q Apply schema, and Monitor qualifier. You cannot use case to differentiate names. You must use a unique path to differentiate between names that are otherwise identical. For example, assume that you have three Q Apply schemas: `myschema` , `MySchema`, and `MYSCHEMA`. The three names use the same characters but different case. If these three qualifiers are in the same directory on the Q Apply server, they will cause name conflicts.

For WebSphere MQ objects, all naming rules are the same as specified by WebSphere MQ.

Chapter 24. System commands for Q Replication and Event Publishing

You can use system commands on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS to start, operate, and modify the replication programs.

You can specify parameters in any order as a **name**=*value* pair. Parameters and their arguments are not case sensitive. Use double quotation marks ("") if you want to preserve case.

Specifying yes/no (Boolean) parameters without an argument is supported, but not recommended. For example, specifying **logreuse** is the same as **logreuse**=**y**. But to specify no logreuse, you must use **logreuse**=**n**.

Invoking commands with a question mark (for example, `asnoqcap ?`), displays a help message that shows the command syntax.

Table 38 helps you match common tasks with the system commands.

Table 38. Q Replication and Event Publishing tasks and their corresponding system commands

If you want to ...	Use this system command
Start a Q Capture program and specify startup parameters (Linux, UNIX, Windows, z/OS)	"asnqcap: Starting a Q Capture program" on page 350
Start a Q Capture program for Oracle sources and specify startup parameters (Linux, UNIX, Windows)	"asnoqcap: Starting a Q Capture program for an Oracle database" on page 372
Work with a running Q Capture program (Linux, UNIX, Windows, z/OS) • Check parameter values • Change parameters • Prune the control tables • Check Q Capture status • Stop Q Capture • Reinitialize all Q subscriptions or publications • Reinitialize one send queue	"asnqccmd: Working with a running Q Capture program" on page 380
Work with a running Q Capture program for Oracle sources (Linux, UNIX, Windows) • Check parameter values • Change parameters • Prune the control tables • Check Q Capture status • Stop Q Capture • Reinitialize all Q subscriptions or publications • Reinitialize one send queue	"asnoqccmd: Working with a running Q Capture program on Oracle databases" on page 386
Start a Q Apply program and specify startup parameters (Linux, UNIX, Windows, z/OS)	"asnqapp: Starting a Q Apply program" on page 389

Table 38. Q Replication and Event Publishing tasks and their corresponding system commands (continued)

If you want to ...	Use this system command
Work with a running Q Apply program (Linux, UNIX, Windows, z/OS) • Check parameter values • Change parameters • Check Q Apply status • Prune the control tables • Stop Q Apply • Stop Q Apply reading from a queue • Start Q Apply reading from a queue • Reinitialize one receive queue	"asnqacmd: Working with a running Q Apply program" on page 408

asnqcap: Starting a Q Capture program

Use the **asnqcap** command to start a Q Capture program on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS. Run this command at an operating system prompt or in a shell script. Any startup parameters that you specify will apply to this session only.

After you start the Q Capture program, it runs continuously until you stop it or it detects an unrecoverable error.

Syntax

```
►►──asnqcap──capture_server=db_name──┬─────────────────────────┬──────────────►
                                     └─capture_schema=schema─┘

►──┬─────────────────────┬──┬──────────────────────┬──┬─────────────────┬───────►
   └─capture_path=path─┘  │              ┌─n─┐     │  └─arm=identifier─┘
                          └─add_partition=─┴─y─┴─┘

►──┬────────────────────────────────┬──┬──────────────────────┬──┬───────────┬──►
   │           ┌─n─┐                │  └─commit_interval=n─┘  │     ┌─n─┐  │
   └─autostop=─┴─y─┴──┬──────────┬──┘                          └─hs=─┴─y─┴──┘
                      │     ┌─n─┐ │
                      └─caf=─┴─y─┴─┘

►──┬──────────────────┬──┬────────────────────────────────┬──┬──────────────────┬──►
   │           ┌─n─┐  │  └─ignore_transid=transaction_ID─┘  │          ┌─n─┐  │
   └─igncasdel=─┴─y─┴─┘                                     └─igntrig=─┴─y─┴──┘

►──┬──────────────────────┬──┬────────────────┬──┬────────────────────────┬──►
   │                 ┌─I─┐ │  └─logrdbufsz=n─┘  └─logread_prefetch=─┬─n─┬─┘
   └─lob_send_option=─┴─S─┴─┘                                       └─y─┘

►──┬─────────────────┬──┬──────────────────┬──┬─────────────────────┬──────────►
   │          ┌─n─┐  │  │           ┌─n─┐  │  └─lsn=formatted_lsn─┘
   └─logreuse=─┴─y─┴─┘  └─logstdout=─┴─y─┴─┘
```

```
►─┬─────────────────────────┬─┬─────────────────┬─┬───────────────────────────┬──────►
  └─maxcmtseq=formatted_lsn─┘ └─memory_limit=n─┘ │                  ┌─y─┐    │
                                                 └─msg_persistence=─┴─n─┴────┘

►─┬──────────────────┬─┬─────────────────────┬─┬──────────────────┬───────────────────►
  │        ┌─y─┐     │ └─monitor_interval=n─┘ └─monitor_limit=n─┘
  └─migrate=─┴─n─┴───┘

►─┬─────────────────────┬─┬───────────────────┬─┬─────────────────────────────┬───────►
  │           ┌─n─┐     │ └─nmi_socket_name=n─┘ │                   ┌─n─┐     │
  └─nmi_enable=─┴─y─┴───┘                       └─override_restartq=─┴─y─┴───┘

►─┬───────────────────┬─┬─────────────────────────┬─┬───────────────────┬─────────────►
  └─part_hist_limit=n─┘ │         ┌─asnpwd.aut─┐  │ └─prune_interval=n─┘
                        └─pwdfile=─┴─filename───┴──┘

►─┬────────────────────┬─┬────────────────────┬─┬────────────────┬────────────────────►
  └─qfull_num_retries=n─┘ └─qfull_retry_delay=n─┘ └─signal_limit=n─┘

►─┬──────────────────┬─┬─────────┬─┬────────────────────┬─────────────────────────────►
  └─sleep_interval=n─┘ └─stale=n─┘ │           ┌─y─┐    │
                                   └─startallq=─┴─n─┴───┘

►─┬─────────────────────────┬─┬───────────────┬─┬───────────────┬─────────────────────►
  │            ┌─warmsi─┐   │ │     ┌─y─┐     │ └─trace_limit=n─┘
  └─startmode=─┼─warmns─┼───┘ └─term=─┴─n─┴───┘
               └─cold───┘

►─┬──────────────────┬─┬──────────────┬─┬────────────┬────────────────────────────────►◄
  └─trans_batch_sz=n─┘ └─warnlogapi=n─┘ └─warntxsz=n─┘
```

Descriptions of asnqcap parameters

These descriptions provide detail on the asnqcap parameters, their defaults, and why you might want to change the default in your environment.

- “add_partition (Linux, UNIX, Windows)”
- “arm (z/OS)”
- “autostop” on page 353
- “caf (z/OS)” on page 353
- “capture_path” on page 353
- “capture_schema” on page 354
- “capture_server” on page 354
- “commit_interval” on page 355
- “hs (z/OS)” on page 356
- “ignore_transid” on page 356
- “igncasdel” on page 356
- “ignsetnull” on page 357
- “igntrig” on page 357
- “lob_send_option” on page 358
- “logrdbufsz” on page 359
- “logread_prefetch (Linux, UNIX, Windows)” on page 359
- “logreuse” on page 359
- “logstdout” on page 360
- “lsn” on page 360
- “maxcmtseq” on page 361
- “memory_limit” on page 362
- “migrate (Linux, UNIX, Windows)” on page 362
- “monitor_interval” on page 363
- “msg_persistence” on page 364
- “override_restartq” on page 364
- “part_hist_limit (Linux, UNIX, Windows)” on page 365
- “prune_interval” on page 365
- “qfull_num_ retries” on page 366
- “qfull_retry_delay” on page 366
- “signal_limit” on page 367
- “sleep_interval” on page 367
- “stale” on page 368
- “startallq” on page 368
- “startmode” on page 368
- “term” on page 369
- “trace_limit” on page 370
- “trans_batch_sz” on page 370
- “warnlogapi (z/OS)” on page 370
- “warntxsz” on page 371

add_partition (Linux, UNIX, Windows)

Default: **add_partition**=n

Method of changing: When Q Capture starts

The **add_partition** parameter specifies whether a Q Capture program starts reading the DB2 recovery log for partitions that were added since the last time the Q Capture program was restarted.

Specify **add_partition**=y when starting a Q Capture program to have the Q Capture program read the log. On each new partition, when the Q Capture program is started in warm start mode, Q Capture will read the log file starting from the first log sequence number (LSN) that DB2 used after the first database CONNECT statement is issued for the DB2 instance.

Oracle sources: The **add_partition** parameter does not apply to Q Capture on Oracle sources, and has no effect if specified.

arm (z/OS)

Default: None

Method of changing: When Q Capture starts

Specifies a three-character alphanumeric string that is used to identify a single instance of the Q Capture program to the Automatic Restart Manager. The value that you supply is appended to the ARM element name that Q Capture generates for itself: ASNQC*xxxxyyyy* (where *xxxx* is the data-sharing group attach name, and *yyyy* is the DB2 member name). You can specify any length of string for the **arm** parameter, but the Q Capture program will concatenate only up to three characters to the current name. If necessary, the Q Capture program will pad the name with blanks to make a unique 16-byte name.

autostop

Default: **autostop**=n

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **autostop** parameter controls whether a Q Capture program terminates when it reaches the end of the active DB2 or Oracle redo log. By default, a Q Capture program does not terminate after reaching the end of the log.

Typically, the Q Capture program is run as a continuous process whenever the source database is active, so in most cases you would keep the default (**autostop**=n). Set **autostop**=y only for scenarios where the Q Capture program is run at set intervals, such as when you synchronize infrequently connected systems, or in test scenarios.

If you set **autostop**=y, the Q Capture program retrieves all eligible transactions and stops when it reaches the end of the log. You need to start the Q Capture program again to retrieve more transactions.

caf (z/OS)

Default: **caf** =n

Method of changing: When Q Capture starts

The Q Capture program runs with the default of Recoverable Resource Manager Services (RRS) connect (**caf** =n). You can override this default and prompt the Q Capture program to use the Call Attach Facility (CAF) by specifying the **caf** =y option. This option specifies that the Q Capture program overrides the default RRS connect and runs with CAF connect.

If RRS is not available you receive a message and the Q Capture program switches to CAF. The message warns that the program was not able to connect because RRS is not started. The program attempts to use CAF instead. The program runs correctly with CAF connect.

capture_path

Default: None

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **capture_path** parameter specifies the directory where a Q Capture program stores its work files and log file. By default, the path is the directory where you start the program. You can change this path.

z/OS

Because the Q Capture program is a POSIX application, the default path depends on how you start the program:

- If you start a Q Capture program from a USS command line prompt, the path is the directory where you started the program.
- If you start a Q Capture program using a started task or through JCL, the default path is the home directory in the USS file system of the user ID that is associated with the started task or job.

To change the path, you can specify either a path name or a high-level qualifier (HLQ), such as //QCAPV9. When you use an HLQ, sequential files are created that conform to the file naming conventions for z/OS sequential data set file names. The sequential data sets are relative to the user ID that is running the program. Otherwise these file names are similar to the names that are stored in an explicitly named directory path, with the HLQ concatenated as the first part of the file name. For example, sysadm.QCAPV9.filename. Using an HLQ might be convenient if you want to have the Q Capture log and LOADMSG files be system-managed (SMS).

If you want the Q Capture started task to write to a .log data set with a user ID other than the ID that is executing the task (for example TSOUSER), you must specify a single quotation mark (‘) as an escape character when using the SYSIN format for input parameters to the started task. For example, if you wanted to use the high-level qualifier JOESMITH, then the user ID TSOUSER that is running the Q Capture program must have RACF authority to write data sets by using the high-level qualifier JOESMITH, as in the following example:

```
//SYSIN    DD  *
 CAPTURE_PATH=//'JOESMITH
/*
```

Windows

If you start a Q Capture program as a Windows service, by default the program starts in the sqllib\bin subdirectory under the installation directory.

capture_schema

Default: capture_schema=ASN

The **capture_schema** parameter lets you distinguish between multiple instances of the Q Capture program on a Q Capture server.

The schema identifies one Q Capture program and its control tables. Two Q Capture programs with the same schema cannot run on a server.

Creating more than one copy of a Q Capture program on a Q Capture server allows you to improve throughput by dividing data flow into parallel streams, or meet different replication requirements while using the same source.

capture_server

z/OS **Default:** None

Linux UNIX Windows **Default: capture_server**=value of DB2DBDFT environment variable, if it is set

The **capture_server** parameter identifies the database or subsystem where a Q Capture program runs, and where its control tables are stored. The control tables contain information about sources, Q subscriptions, WebSphere MQ queues, and user preferences. Because a Q Capture program reads the source database log, it must run at the source database or subsystem.

Oracle sources: If you do not specify a Q Capture server, this parameter defaults to the value of the ORACLE_SID environment variable.

z/OS You must specify the **`capture_server`** parameter. For data sharing you can provide the group attach name instead of a subsystem name so that you can run the replication job in any LPAR.

commit_interval

Default: `commit_interval`=500 milliseconds (a half second) for DB2 sources; 1000 milliseconds (1 second) for Oracle sources

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **`commit_interval`** parameter specifies how often, in milliseconds, a Q Capture program commits transactions to WebSphere MQ. By default, a Q Capture program waits 500 milliseconds (a half second) between commits. At each interval, the Q Capture program issues an MQCMIT call. This signals the WebSphere MQ queue manager to make messages that were placed on send queues available to the Q Apply program or other user applications.

All of the transactions that are grouped within an MQCMIT call are considered to be a WebSphere MQ unit of work,'or transaction. Typically, each WebSphere MQ transaction contains several database transactions. If the database transaction is large, the Q Capture program will not issue an MQCMIT call even if the commit interval is reached. The Q Capture program will commit only after the entire large database transaction is put on the send queue.

When the number of committed database transactions that are read by a Q Capture program reaches 128, the program issues an MQCMIT call regardless of your setting for **`commit_interval`**.

Finding the best commit interval is a compromise between latency (the delay between the time transactions are committed at the source and target databases) and CPU overhead associated with the commit process:

To reduce latency, shorten the commit interval
Transactions will be pushed through with less delay. This is especially important if changes to the source database are used to trigger events. If the number of transactions published per commit interval is high, you might want to have the Q Capture program commit fewer transactions at a time to WebSphere MQ. See Determining the number of transactions published per commit interval for more detail.

To reduce CPU overhead, lengthen the commit interval
A longer commit interval lets you send as many database transactions as possible for each WebSphere MQ transaction. A longer commit interval also reduces I/O that is caused by logging of messages. If you lengthen the commit interval, you might be limited by the memory allocated for a Q Capture program, the maximum depth (number of messages) for send queues, and the queue manager's maximum uncommitted messages

(MAXUMSGS) attribute. If a Q Capture program waits longer between commits, the publication of some transactions might be delayed, which could increase latency.

hs (z/OS)

Default: **hs**=n

Method of changing: When Q Capture starts

The **hs** parameter specifies whether the Q Capture program creates one or more spill files in hiperspace (high performance data space) if Q Capture exceeds its memory limit during an attempt to write a row in memory. By default (**hs**=n) Q Capture creates the spill file on disk or virtual input/output (VIO).

Recommendation: Allocate enough memory to the Q Capture job to avoid the need for spill files.

ignore_transid

Default: None

Method of changing: When Q Capture starts

The **ignore_transid**=*transaction_ID* parameter specifies that the Q Capture program ignores the transaction that is identified by *transaction_ID*. The transactions are not replicated or published. You can use this parameter if you want to ignore a very large transaction that does not need to be replicated, for example a large batch job. The value for *transaction_ID* is a 10-byte hexadecimal identifier in the following format:

z/OS

0000:*xxxx*:*xxxx*:*xxxx*:*mmmm*

Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *mmmm* is the data-sharing member ID. You can find the member ID in the last 2 bytes of the log record header in the LOGP output. The member ID is 0000 if data-sharing is not enabled.

Linux UNIX Windows

nnnn:0000:*xxxx*:*xxxx*:*xxxx*

Where *xxxx*:*xxxx*:*xxxx* is the transaction ID, and *nnnn* is the partition identifier for partitioned databases (this value is 0000 if for non-partitioned databases).

Tip: The shortened version **transid** is also acceptable for this parameter.

igncasdel

Default: **igncasdel**=n

Method of changing: When Q Capture starts; at the Q subscription level using replication administration tools (value stored in IBMQREP_SUBS table)

The **igncasdel** parameter specifies whether the Q Capture program replicates delete operations that result from the delete of parent rows on tables with referential integrity relationships (cascading deletes). You can use this option to reduce the amount of data that needs to be propagated when the delete of the parent row will be cascaded at the target.

By default, when a parent row is deleted Q Capture replicates the delete operations from child rows. The Q Apply program reorders transactions to ensure that no child row is deleted at the target before its parent row is deleted. If you specify **igncasdel**=y, Q Capture replicates only the delete of the parent row. Use this option to avoid redundant delete operations by the Q Apply program when replication of the parent row delete would cause cascading deletes at the target table.

You can also specify this option at the Q subscription level by using the replication administration tools to change the value of the IGNCASDEL column in the IBMQREP_SUBS table from the default of N to Y. If you specify Y in the IBMQREP_SUBS table, the setting for the **igncasdel** parameter is overridden for the individual Q subscription.

ignsetnull

Default: **igntrig**=n

Method of changing: When Q Capture starts; at the Q subscription level using replication administration tools (value stored in IBMQREP_SUBS table)

The **ignsetnull** parameter specifies that the Q Capture program should not replicate UPDATE operations that result from the deletion of parent rows in tables with referential integrity relationships when the ON DELETE SET NULL rule is in effect.

By default, when a parent row is deleted and the ON DELETE SET NULL rule is in effect, Q Capture replicates these UPDATE operations in which one or more column values are set to NULL. If ON DELETE SET NULL is in effect at the target, you can set **ignsetnull**=y and Q Capture ignores these UPDATE operations.

You can also specify this option at the Q subscription level by using the replication administration tools to change the value of the IGNSETNULL column in the IBMQREP_SUBS table from the default of N to Y. If you specify Y in the IBMQREP_SUBS table, the setting for the **ignsetnull** parameter is overridden for the individual Q subscription.

igntrig

Default: **igntrig**=n

Method of changing: When Q Capture starts; at the Q subscription level using replication administration tools (value stored in IBMQREP_SUBS table)

The **igntrig** parameter specifies that the Q Capture program should discard trigger-generated rows. When a trigger on the source table generates a secondary SQL statement after an SQL operation, both the initial and secondary SQL statements are replicated to the target. These secondary statements create conflicts because the trigger on the target table generates the same rows when source changes are applied. Setting **igntrig**=y prompts the Q Capture program to not capture any trigger-generated SQL statements.

You can also specify the option at the Q subscription level by changing the value of the IGNTRIG column in the IBMQREP_SUBS table from the default of n to y. If you specify Y in the IBMQREP_SUBS table, the setting for the **igntrig** parameter is overridden for the individual Q subscription.

If you use **igntrig**=n, be sure to define triggers on target tables as identical to triggers on the source table. If not, conflicts can occur when Q Apply updates a target table that has a non-matching trigger.

You can also specify this option at the Q subscription level by using the replication administration tools to change the value of the IGNTRIG column in the IBMQREP_SUBS table from the default of N to Y. If you specify Y in the IBMQREP_SUBS table, the setting for the **igntrig** parameter is overridden for the individual Q subscription.

Note about INSTEAD OF triggers: If the source table is updated by INSTEAD OF triggers on a view and the **igntrig** parameter is set to y, the Q Capture program does not replicate the change to the source table.

lob_send_option

Default: lob_send_option=I

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **lob_send_option** parameter specifies whether the Q Capture program sends LOB values inline (I) within a transaction message or in a separate message (S). By default, large object (LOB) values are sent within the transaction message. The Q Capture program manages the amount of memory that it consumes when it replicates or publishes LOB data types. If you are replicating LOB data, the value of **max_message_size** determines how often the Q Capture program accesses the source table to fetch the LOB data in multiple chunks (one chunk per message). A low maximum message size can impede Q Capture performance in replicating or publishing LOB data.

Linux, UNIX, Windows: Starting with DB2 Version 10.1, the Q Capture program can fetch LOB data directly from the DB2 recovery log even if the data exceeds the value of the INLINE LENGTH option in the definition of a LOB column in a CREATE TABLE or ALTER TABLE statement. In previous versions, Q Capture connected to the source database to fetch LOB data if the data was larger than INLINE LENGTH. See Improving performance when replicating LOB data for more details.

If you set **lob_send_option**=S to have LOB values sent in a separate LOB message, the LOB values might not fit into a single LOB message. If the size of the LOB value exceeds the maximum message size for the Q Capture program, then the LOB message is divided into two or more smaller messages. If you expect to replicate or publish many LOB values or BLOB values, allocate sufficient memory and storage, and set the queue depth accordingly.

Use the following guidelines for setting **max_message_size**:
- One typical large transaction should fit into one message, so set the value of **max_message_size** to be slightly higher than the maximum size of a typical transaction.

- For very large transactions that exceed the value of **`max_message_size`** , ensure that you set **`max_message_size`** so that at least one row of the transaction fits into one message.
- The value of **`max_message_size`** must be less than or equal to the parameter MAXMSGL, which sets the maximum message size for a queue.

When the Q Capture program is using the separate LOB mode, LOB values for all LOB columns that are part of a Q subscription or publication are sent for every row in a transaction. This behavior results in more WebSphere MQ messages if a LOB value is updated multiple times in a transaction or if the CHANGED_COLS_ONLY option in the Q subscription or publication is set to N.

LOB values for all LOB columns that are part of a Q subscription or publication are sent for key updates regardless of the CHANGED_COLS_ONLY setting.

logrdbufsz

Default: **`logrdbufsz`**=66KB for z/OS; 256KB for Linux, UNIX, and Windows

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **`logrdbufsz`** parameter specifies the size of the buffer in KB that the Q Capture program passes to DB2 when Q Capture retrieves log records. DB2 fills the buffer with available log records that Q Capture has not retrieved. For partitioned databases, Q Capture allocates a buffer of the size that is specified by **`logrdbufsz`** for each partition.

The default values should be optimal for most situations. However, you may want to increase this value if you have a high volume of data changes and sufficient memory available.

logread_prefetch (Linux, UNIX, Windows)

Default: **`logread_prefetch`**=y for partitioned databases; n for nonpartitioned databases

Method of changing: When Q Capture starts

The **`logread_prefetch`** parameter specifies whether the Q Capture program uses separate threads to prefetch log records from each partition in a partitioned database. By default (**`logread_prefetch`**=y), Q Capture uses separate log-reader threads to connect to each partition. Using separate threads can increase Q Capture throughput but might increase CPU usage.

If you specify **`logread_prefetch`**=n, a single Q Capture log reader thread connects to all partitions.

logreuse

Default: **`logreuse`**=N

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

Each Q Capture program keeps a diagnostic log file that tracks its work history, such as start and stop times, parameter changes, errors, pruning, and the points where it left off while reading the database log.

By default, the Q Capture program adds to the existing log file when the program restarts. This default lets you keep a history of the program's actions. If you don't want this history or want to save space, set **logreuse**=y. The Q Capture program clears the log file when it starts, then writes to the blank file.

The log is stored by default in the directory where the Q Capture program is started, or in a different location that you set using the **capture_path** parameter.

z/OS The log file name is *capture_server.capture_schema*.QCAP.log. For example, SAMPLE.ASN.QCAP.log. Also, if **capture_path** is specified with slashes (//) to use a High Level Qualifier (HLQ), the file naming conventions of z/OS sequential data set files apply, and **capture_schema** is truncated to eight characters.

Linux UNIX Windows The log file name is *db2instance.capture_server.capture_schema*.QCAP.log. For example, DB2.SAMPLE.ASN.QCAP.log.

Oracle sources: The log file name is *capture_server.capture_schema*.QCAP.log. For example, ORASAMPLE.ASN.QCAP.log.

logstdout

Default: logstdout=n

By default, a Q Capture program writes its work history only to the log. You can change the **logstdout** parameter if you want to see the program's history on the standard output (stdout) in addition to the log.

Error messages and some log messages (initialization, stop, subscription activation, and subscription deactivation) go to both the standard output and the log file regardless of the setting for this parameter.

lsn

Default: None

Method of changing: When Q Capture starts

The **lsn** parameter specifies the log sequence number at which the Q Capture program starts during a warm restart. You specify both the **lsn** and **maxcmtseq** parameters to start Q Capture from a known point in the DB2 log. When specifying the **lsn** parameter, you also must specify the **maxcmtseq** parameter in the same command invocation, and you cannot use these parameters if the value of **startmode** is cold.

This value represents the earliest log sequence number that the Q Capture program found for which a commit or abort record has not yet been found. You can obtain the value for **lsn** from the restart message by using the **asnqmfmt** command. You can also use the value in the RESTART_SEQ column of the IBMQREP_CAPMON table. If you use the latter method, choose an entry in the monitor table that is older than the time that Q Capture stopped to ensure that any lost messages are recaptured.

To start from the end of the log without triggering a load (full refresh) of the target table, specify one of the following values, depending on your DB2 version:

Version 9.7 and below
: **`lsn`**=FFFF:FFFF:FFFF:FFFF:FFFF and **`maxcmtseq`**=FFFF:FFFF:FFFF:FFFF:FFFF.

Version 10.1 or higher with `compatibility` of 1001 or higher, or Version 9.8
: **`lsn`**=FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF and **`maxcmtseq`**=FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF.

You can also specify **`lsn`** and **`maxcmtseq`** without colons to save space.

maxcmtseq

Default: None

Method of changing: When Q Capture starts

The **`maxcmtseq`** parameter is used to specify the commit log record position of the last transaction that was successfully sent by the Q Capture program before shutdown. You can specify both the **`maxcmtseq`** and **`lsn`** parameters to start Q Capture from a known point in the DB2 log. When specifying the **`maxcmtseq`** parameter, you also must specify the **`lsn`** parameter in the same command invocation, and you cannot use these parameters if the value of **startmode** is cold.

The value of **`maxcmtseq`** is an internal log marker that is different for each type of database system. The marker is encoded as a 10-character string:

z/OS
: On z/OS, the value is the LSN of the commit log record, to which Q Capture might append a sequence number because on z/OS with data sharing several log records might have the same LSN.

Linux UNIX Windows
: On Linux, UNIX, and Windows, the value is a timestamp with nanosecond precision that uniquely identifies a transaction. The value is encoded as two integers, seconds, and nanoseconds.

You can find the value for **`maxcmtseq`** in one of these places:

- From the restart message, by using the **`asnqmfmt`** command
- From the Q Capture output log file, messages ASN7108I and ASN7109
- From the IBMQREP_APPLYMON table (OLDEST_COMMIT_LSN for z/OS sources and OLDEST_COMMIT_SEQ for Linux, UNIX, and Windows sources)

To start from the end of the log without triggering a load (full refresh) of the target table, specify one of the following values, depending on your DB2 version:

Version 9.7 and below
: **`lsn`**=FFFF:FFFF:FFFF:FFFF:FFFF and **`maxcmtseq`**=FFFF:FFFF:FFFF:FFFF:FFFF.

Version 10.1 or higher with `compatibility` of 1001 or higher, or Version 9.8
: **`lsn`**=FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF and **`maxcmtseq`**=FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF.

You can also specify **lsn** and **maxcmtseq** without colons to save space.

memory_limit

Default: **memory_limit**=0 on z/OS; 500 MB on Linux, UNIX, and Windows

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **memory_limit** parameter specifies the amount of memory that a Q Capture program can use to build database transactions in memory. By default, the **memory_limit** is set to 0 on z/OS and the Q Capture program calculates a memory allocation that is based on the Q Capture region size in the JCL or started task. On Linux, UNIX, and Windows, a Q Capture program uses a maximum of 500 MB by default. When the memory amount allocated by this parameter is used, a Q Capture program spills in-memory transactions to a file that is located in the **capture_path** directory. On z/OS, the Q Capture program spills to VIO or to the file that is specified in the CAPSPILL DD card.

The maximum allowed value for this parameter is 100 GB.

You can adjust the memory limit based on your needs:

To improve the performance of a Q Capture program, increase the memory limit
If your goal is higher throughput, maximize the memory limit whenever possible.

To conserve system resources, lower the memory limit
A lower memory limit reduces competition with other system operations. However, setting the memory limit too low will use more space on your system for the spill file and prompt more I/O that can slow your system.

You can use data in the IBMQREP_CAPMON table to find the best memory limit for your needs. For example, check the value for CURRENT_MEMORY to see how much memory a Q Capture program is using to reconstruct transactions from the log. Or, check the value for TRANS_SPILLED to find out how many transactions a Q Capture program spilled to a file when it exceeded the memory limit. You can use the Q Capture Throughput window in the Replication Center to check these values. See the Replication Center online help for details.

migrate (Linux, UNIX, Windows)

Default: **migrate**=y

Method of changing: When Q Capture starts

The **migrate** parameter specifies that the Q Capture program starts from the beginning of the log after DB2 for Linux, UNIX, and Windows is upgraded. Use this option after you upgrade DB2 to a new release, such as Version 9.5 or Version 9.7. The **migrate** parameter is not required after you upgrade DB2 with a fix pack.

Use the **migrate** parameter only the first time that Q Capture is started and specify **startmode**=warmns.

monitor_interval

Default: **`monitor_interval`**=60000 milliseconds (1 minute) on z/OS and 30000 milliseconds (30 seconds) on Linux, UNIX, and Windows

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **`monitor_interval`** parameter tells a Q Capture program how often to insert performance statistics into two of its control tables. The IBMQREP_CAPMON table shows statistics for overall Q Capture program performance, and the IBMQREP_CAPQMON table shows Q Capture program statistics for each send queue.

By default, rows are inserted into these tables every 300000 milliseconds (5 minutes). Typically, a Q Capture program commits WebSphere MQ transactions at a much shorter interval (the default commit interval is a half second). Thus, if you use shipped defaults for the monitor interval and commit interval, each insert into the monitor tables contains totals for 120 commits. If you want to monitor Q Capture activity at a more granular level, use a monitor interval that is closer to the commit interval.

Important for Q Replication Dashboard users: When possible, you should synchronize the Q Capture **`monitor_interval`** parameter with the dashboard refresh interval (how often the dashboard retrieves performance information from the Q Capture and Q Apply monitor tables). The default refresh interval for the dashboard is 10 seconds (10000 milliseconds). If the value of **`monitor_interval`** is higher than the dashboard refresh interval, the dashboard refreshes when no new monitor data is available.

monitor_limit

Default: **`monitor_limit`**=10080 minutes (7 days)

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **`monitor_limit`** parameter specifies how old the rows must be in the IBMQREP_CAPMON and IBMQREP_CAPQMON tables before they are eligible for pruning.

By default, rows that are older than 10080 minutes (7 days) are pruned. The IBMQREP_CAPMON and IBMQREP_CAPQMON tables contain statistics about a Q Capture program's activity. A row is inserted at each monitor interval. You can adjust the monitor limit based on your needs:

Increase the monitor limit to keep statistics
: If you want to keep records of the Q Capture program's activity beyond one week, set a higher monitor limit.

Lower the monitor limit if you look at statistics frequently
: If you monitor the Q Capture program's activity on a regular basis, you probably do not need to keep one week of statistics and can set a lower monitor limit, which prompts more frequent pruning.

msg_persistence

Default: **msg_persistence**=y

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **msg_persistence** parameter specifies whether a Q Capture program writes persistent (logged) or nonpersistent (unlogged) data messages to WebSphere MQ queues. (Data messages contain replicated data from source tables.) By default, Q Capture uses persistent data messages. The queue manager logs persistent messages and can recover the messages after a system failure or restart.

In some cases you might want to avoid the CPU and storage overhead of persistent messages. In that case, specify **msg_persistence**=n.

The Q Capture and Q Apply program always put persistent messages onto their administration queues. Q Capture always puts a persistent message onto its restart queue. Therefore logging of messages on these queues is not affected by the setting for **msg_persistence**.

If you created a send queue with the DEFPSIST(Y) option so that the queue carries persistent messages by default, you can still specify **msg_persistence**=n and Q Capture sends nonpersistent messages, which overrides the queue default.

If you choose nonpersistent messages and data messages are lost because of a problem that forces the queue manager to restart, you need to do one of the following things to keep the source and target tables synchronized:
- Stop and start the Q Capture program and specify a known point in the recovery log so that Q Capture rereads the log at a point before the messages were lost.
- Stop and start the Q subscription to prompt a new load (full refresh) of the target table.

override_restartq

Default: **override_restartq**=n

Method of changing: When Q Capture starts

The **override_restartq** parameter specifies whether the Q Capture program gets its warm start information from a data set or file rather than from the restart message. By default, Q Capture gets its restart information from the restart message. If you specify **override_restartq**=y, Q Capture reads the restart file when it starts, and gets the starting point for each send queue and data partition from the file. The file is saved in the directory that is specified by the **capture_path** parameter, or in the directory from which Q Capture was started if nothing is specified for **capture_path**. The data set name differs by platform:

z/OS

capture_server.*capture_schema*.QCAP.QRESTART

Linux UNIX Windows

db2_instance.*capture_server*.*capture_schema*
.QCAP.QRESTART

To use this parameter, the user ID that starts Q Capture must have the authorization to open and read from the restart file.

part_hist_limit (Linux, UNIX, Windows)

Default: **`part_hist_limit`**=10080 minutes (seven days)

Method of changing: When Q Capture starts;

The **`part_hist_limit`** parameter specifies how long you want old data to remain in the IBMQREP_PART_HIST table before the data becomes eligible for pruning. This parameter also controls how far back in the log you can restart the Q Capture program because Q Capture uses IBMQREP_PART_HIST to determine what log records to read for a partitioned source table.

The Q Capture program uses the partition history that is stored in the IBMQREP_PART_HIST table to help handle partition changes such as add, attach, or detach. One row, identified by a log sequence number (LSN), is inserted for each partition in the source table on two occasions:

- The first Q subscriptions or subscription-set member for the table is activated.
- The table is altered to add, attach, or detach a partition.

If you are replicating partitioned tables on Linux, UNIX, or Windows, ensure that the **`part_hist_limit`** value is large enough to allow for possible recapture of past log records in the case of a system failure or other outage.

prune_interval

Default: **`prune_interval`**=300 seconds (5 minutes)

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **`prune_interval`** parameter determines how often a Q Capture program looks for eligible rows to prune from the IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_SIGNAL, and IBMQREP_CAPTRACE tables. By default, a Q Capture program looks for rows to prune every 300 seconds (5 minutes).

Your pruning frequency depends on how quickly these control tables grow, and what you intend to use them for:

Shorten the prune interval to manage monitor tables
A shorter prune interval might be necessary if the IBMQREP_CAPMON and IBMQREP_CAPQMON tables are growing too quickly because of a shortened monitor interval. If these and other control tables are not pruned often enough, they can exceed their table space limits, which forces a Q Capture program to stop. However, if the tables are pruned too often or during peak times, pruning can interfere with application programs that run on the same system.

Lengthen the prune interval for record keeping
You might want to keep a longer history of a Q Capture program's performance by pruning the IBMQREP_CAPTRACE and other tables less frequently.

The prune interval works in conjunction with the **trace_limit**, **monitor_limit**, and **signal_limit** parameters, which determine when data is old enough to prune. For example, if the **prune_interval** is 300 seconds and the **trace_limit** is 10080 seconds, a Q Capture program will try to prune every 300 seconds. If the Q Capture program finds any rows in the IBMQREP_CAPTRACE table that are older than 10080 minutes (7 days), it prunes them.

pwdfile

Default: **pwdfile**=*capture_path*/asnpwd.aut

Method of changing: When Q Capture starts

The **pwdfile** parameter specifies the name of the password file that is used to connect to multiple-partition databases. If you do not use the **pwdfile** parameter to specify the name of a password file, the Q Capture program looks for a file with the name of asnpwd.aut in the directory that is specified by the **capture_path** parameter. If no **capture_path** parameter is specified, this command searches for the password file in the directory from which the command was invoked.

You can create a password file by using the **asnpwd** command. Use the following example to create a password file with the default name of asnpwd.aut in the current directory: asnpwd INIT.

qfull_num_ retries

Default: **qfull_num_ retries**=30

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

You can use the **qfull_num_ retries** parameter to specify the number of times that a Q Capture program tries to put a message on a queue when the initial MQPUT operation fails because the queue is full. The default is 30 retries, and the maximum is 1,000 retries. A value of 0 instructs the Q Capture program to stop whenever an MQPUT operation fails. The alternate syntax for the **qfull_num_retries** parameter is **RN**.

For example' the following command specifies that a Q Capture program retry the MQPUT operation 100 times before stopping:

```
asnqcap capture_server=SAMPLE capture_schema=ASN1 qfull_num_retries=100
```

qfull_retry_delay

Default: **qfull_retry_delay**=250 milliseconds

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

You can use the **qfull_retry_delay** parameter to specify how long in milliseconds the Q Capture program waits between MQPUT attempts when the initial MQPUT operation fails because the queue is full. The allowed value range is 10 milliseconds to 3600000 milliseconds (1 hour). The default delay is 250 milliseconds or the value of the **commit_interval** parameter, whichever is less. (The default for **commit_interval** is 500 milliseconds.) If the specified value is larger than the

commit_interval value, it will be set lower to the **commit_interval** value. The alternate syntax for the **qfull_retry_delay** parameter is **RD**.

For example, the following command specifies that a Q Capture program retry the MQPUT operation 50 times before stopping, with a delay of 10000 milliseconds (10 seconds):

```
asnqcap capture_server=SAMPLE capture_schema=ASN1 qfull_num_retries=50
qfull_retry_delay=10000
```

signal_limit

Default: **signal_limit**=10080 minutes (7 days)

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **signal_limit** parameter specifies how long rows remain in the IBMQREP_SIGNAL table before they can be pruned.

By default, a Q Capture program prunes rows that are older than 10080 minutes (7 days) at each pruning interval.

The IBMQREP_SIGNAL table contains signals inserted by a user or a user application. It also contains corresponding signals t'at are inserted by a Q Capture program after it receives control messages from the Q Apply program or a user application. The Q Capture program sees the signals when it reads the log record for the insert into the IBMQREP_SIGNAL table.

These signals tell a Q Capture program to stop running, to activate or deactivate a Q subscription or publication, to ignore a database transaction in the log, or to invalidate a send queue. In addition, the LOADDONE signal tells a Q Capture program that a target table is loaded.

You can adjust the signal limit depending on your environment. Shorten the limit to manage the size of the IBMQREP_SIGNAL table. For bidirectional Q Replication, the Q Apply program inserts a signal into the IBMQREP_SIGNAL table for every transaction that it receives and applies to make sure that the Q Capture program does not recapture the transaction. If you have a large number of bidirectional Q subscriptions, the table might grow large and you might want to lower the default signal limit so that it can be pruned more frequently. Lengthen the limit to use this table for record-keeping purposes.

sleep_interval

Default: **sleep_interval**=500 milliseconds (0.5 seconds) for DB2 sources; 2000 milliseconds (2 seconds) for Oracle sources

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **sleep_interval** parameter specifies the number of milliseconds that a Q Capture program waits after reaching the end of the active log and assembling any transactions that remain in memory.

By default, a Q Capture program sleeps for 5000 milliseconds (5 seconds). After this interval, the program starts reading the log again. You can adjust the sleep interval based on your environment:

Lower the sleep interval to reduce latency
A smaller sleep interval can improve performance by lowering latency (the time that it takes for a transaction to go from source to target), reducing idle time, and increasing throughput in a high-volume transaction environment.

Increase the sleep interval to save resources
A larger sleep interval gives you potential CPU savings in an environment where the source database has low traffic, or where targets do not need frequent updates.

stale

Default: **`stale`**=3600

Method of changing: When Q Capture starts;

The **`stale`** parameter specifies the number of seconds that the Q Capture program waits to issue a warning message or take other action after it detects a long-running transaction with no commit or rollback log record. The program behavior depends on the platform of the source. On z/OS, Q Capture issues warning messages if has not seen a commit or rollback record for one hour (**`stale`**=3600). On both z/OS and Linux, UNIX, and Windows, if a transaction has been running for the number of seconds that are specified by **`stale`** and Q Capture did not see any row operations in the log for the transaction, it issues warning messages and increments the log sequence number that it considers to be the oldest "in-flight" transaction that was not committed or rolled back. If some rows were captured for the transaction, only warning messages are issued.

startallq

Default: **`startallq`**=y

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **`startallq`** parameter specifies whether the Q Capture program activates all send queues during startup. You can use this parameter to keep a disabled send queue inactive.

By default, when Q Capture starts it activates all send queues that are not already in active (`A`) state. If you specify **`startallq`**=n, when Q Capture starts it does not activate send queues that are in inactive (`I`) state.

startmode

Default: **`startmode`**=warmsi

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **`startmode`** parameter specifies the steps that a Q Capture program takes when it starts. The program starts in either warm or cold mode. With a warm start, the Q Capture program continues capturing changes where it left off after its last run (there are three types of warm start). If you choose cold start, the program starts reading at the end of the log. Choose from one of the four start modes, depending on your environment:

cold
: The Q Capture program clears the restart queue and administration queue, and starts processing all Q subscriptions or publications that are in N (new) or A (active) state. With a cold start, the Q Capture program starts reading the DB2 recovery log or Oracle redo log at the end.

  Generally, use a cold start only the first time that you start a Q Capture program. Warmsi is the recommended start mode. You can use a cold start if this is not the first time that you started a Q Capture program, but you want to begin capturing changes from the end of the active log instead of from the last restart point. On a cold start, Q Capture stops and then starts unidirectional Q subscriptions that have a load phase specified. For bidirectional and peer-to-peer replication, Q Capture only stops the Q subscriptions on a cold start. To start these Q subscriptions, you must use the replication administration tools or insert a CAPSTART signal for each Q subscription that you want to start.

  Important: To avoid unwanted cold starts, be sure that this start mode is not specified in the IBMQREP_CAPPARMS table.

warmsi (warm start; switch first time to cold start)
: The Q Capture program starts reading the log at the point where it left off, except if this is the first time that you are starting it. In that case the Q Capture program switches to a cold start. The warmsi start mode ensures that a Q Capture program cold starts only when it initially starts.

warmns (warm start; never switch to cold start)
: The Q Capture program starts reading the log at the point where it left off. If it cannot warm start, it does not switch to cold start. Use this start mode to prevent a Q Capture program from cold starting unexpectedly. This start mode allows you to repair problems (such as unavailable databases or table spaces) that are preventing a warm start. With warmns, if a Q Capture program cannot warm start, it shuts down and leaves all tables intact.

During warm starts, the Q Capture program will only load those Q subscriptions or publications that are in not in I (inactive) state.

term

Default: `term`=y

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **`term`** parameter controls whether a Q Capture program keeps running when DB2 or the queue manager are unavailable.

By default (**`term`**=y), the Q Capture program terminates when DB2 or the queue manager are unavailable. You can change the default (**`term`**=n) if you want a Q Capture program to keep running while DB2 or the queue manager are unavailable. When DB2 or the queue manager are available, Q Capture begins sending transactions where it left off without requiring you to restart the program.

Oracle sources: The default value for **`term`** is `n` (the Q Capture program continues to run when the Oracle database is unavailable). Setting the value to `y` has no effect.

Note: Regardless of the setting for **`term`**, if the WebSphere MQ sender or receiver channels stop, the Q Capture program keeps running because it cannot detect channel status. This situation causes replication to stop because the two queue managers cannot communicate. If you find that replication has stopped without any messages from the Q Replication programs, check the channel status by using the WebSphere MQ DISPLAY CHSTATUS command.

trace_limit

Default: **`trace_limit`**=10080 minutes (7 days)

Methods of changing: When Q Capture starts; while Q Capture is running; IBMQREP_CAPPARMS table

The **`trace_limit`** parameter specifies how long rows remain in the IBMQREP_CAPTRACE table before they can be pruned.

The Q Capture program inserts all informational, warning, and error messages into the IBMQREP_CAPTRACE table. By default, rows that are older than 10080 minutes (7 days) are pruned at each pruning interval. Modify the trace limit depending on your need for audit information.

trans_batch_sz

Default: **`trans_batch_sz`**=1

Methods of changing: When Q Capture starts;

The **`trans_batch_sz`** parameter specifies the number of source database transactions that Q Capture groups together in a WebSphere MQ message. You can use this parameter to reduce CPU consumption at the source when the replication workload typically consists of small transactions, for example one or two rows.

The default is 1 (no batching) and the maximum batch size is 128. Q Capture respects the value of **`trans_batch_sz`** unless the Q Capture commit interval is reached before the number of transactions reaches this value, or unless the total size of transactions in the batch exceeds the value of the **`max_message_size`** parameter for the send queue or the value of the WebSphere MQ maximum message length (MAXMSGL) for the send queue.

Note: Using this parameter in combination with other parameters or consistent-change data (CCD) targets might product undesired results. For details, see trans_batch_sz parameter.

warnlogapi (z/OS)

Default: **`warnlogapi`**=0

Methods of changing: When Q Capture starts; IBMQREP_CAPPARMS table

The **`warnlogapi`** parameter specifies the number of milliseconds that the Q Capture program waits for the DB2 for z/OS instrumentation facility interface (IFI) or DB2 for Linux, UNIX, and Windows log read API to return log records before Q Capture prints a warning to the standard output.

warntxsz

Default: **`warntxsz`**=0 MB

Methods of changing: When Q Capture starts

You can use the **`warntxsz`** parameter to detect very large transactions. This parameter prompts the Q Capture program to issue a warning message when it encounters transactions that are larger than a specified size. You provide a threshold value in megabytes, and transactions that exceed the threshold prompt a warning message. Q Capture issues multiple warning messages if the transaction size is a multiple of the **`warntxsz`** value. For example, if you set **`warntxsz`** to 10 MB and Q Capture encounters a 30 MB transaction, three warnings are issued (one for 10 MB, one for 20 MB, and one for 30 MB).

Q Capture issues the ASN0664W message when the size of a transaction exceeds the **`warntxsz`** value. The default value of 0 MB means warnings are never issued.

Examples of asnqcap usage

These examples illustrate how to use the **`asnqcap`** command.

Overriding the default start mode and log parameters

To start a Q Capture program on a server called sourcedb with a schema of alpha, to start the program using the cold start mode, and to temporarily override the default settings for **`logreuse`** and **`logstdout`**, issue the following command:

```
asnqcap capture_server=sourcedb capture_schema="alpha" startmode=cold
logreuse=y logstdout=y
```

Overriding the default commit interval

To start a Q Capture program and instruct it to commit messages on queues more frequently than the default, issue the following command:

```
asnqcap capture_server=sourcedb capture_schema="alpha" commit_interval=250
```

Changing the memory allocation

To start a Q Capture program and temporarily increase the default amount of memory that it uses to build transactions, issue the following command:

```
asnqcap capture_server=sourcedb capture_schema="alpha" memory_limit=64
```

Specifying the log and work file location

To start a Q Capture program and direct its work files to the `/home/files/qcapture` directory, issue the following command:

```
asnqcap capture_server=sourcedb capture_schema="alpha"
capture_path="/home/files/qcapture"
```

Specifying lsn and maxcmtseq

To start a Q Capture program on a server named testdb, and to specify that after a warm restart the program start at a log sequence number (**lsn**) of 0000:0000:0000:115b:7704 and **maxcmtseq** of 41c2:2264:0000:0004:0000, issue the following command:

```
asnqcap capture_server=testdb lsn=0000:0000:0000:115b:7704
maxcmtseq=41c2:2264:0000:0004:0000:
```

Starting from the end of the log

To start a Q Capture program from the end of the log without triggering a load (full refresh) of the target table, issue the following command:

```
asnqcap capture_server=testdb lsn=FFFF:FFFF:FFFF:FFFF:FFFF
maxcmtseq=FFFF:FFFF:FFFF:FFFF:FFFF
```

Ignoring a specific transaction

To start a Q Capture program on z/OS data sharing and specify that the program ignore a transaction with a transaction ID of BD71:1E23:B089 (the data-sharing member ID is 0001), issue the following command:

```
asnqcap capture_server=sample capture_schema=ASN
ignore_transid=0000:BD71:1E23:B089:0001
```

asnoqcap: Starting a Q Capture program for an Oracle database

Use the **asnoqcap** command to start a Q Capture program for Oracle databases on Linux, UNIX, or Windows systems. Run this command at an operating system prompt or in a shell script. Any startup parameters that you specify will apply to this session only.

After you start the Q Capture program, it runs continuously until you stop it or it detects an unrecoverable error.

Syntax

```
►►──asnoqcap──capture_server=db_name──┬───────────────────────────┬──────────────►
                                      └─capture_schema=schema─┘

►──┬────────────────────┬──┬─────────────────┬──┬─────────────────────┬──────────►
   └─capture_path=path─┘  │           ┌─n─┐  │  └─commit_interval=n─┘
                           └─autostop=─┴─y─┴─┘

►──┬────────────────────────────────┬──┬──────────────────────────┬───────────────►
   └─ignore_transid=transaction_ID─┘  │                   ┌─I─┐  │
                                      └─lob_send_option=─┴─S─┴─┘

►──┬───────────────────┬──┬────────────────────┬──┬───────┬──┬─────────────┬─────►
   │           ┌─n─┐   │  │            ┌─n─┐   │  └─lsn=n─┘  └─maxcmtseq=n─┘
   └─logreuse=─┴─y─┴───┘  └─logstdout=─┴─y─┴───┘

►──┬──────────────────┬──┬──────────────────────┬──┬───────────────────┬─────────►
   └─memory_limit=n─┘  └─monitor_interval=n─┘  └─monitor_limit=n─┘
```

```
►─┬───────────────────────────┬──┬────────────────────┬──┬──────────────────────┬──►
  │            ┌─asnpwd.aut─┐ │  └─prune_interval=n───┘  └─qfull_num_retries=n──┘
  └─pwdfile=───┴─filename───┴─┘

►─┬──────────────────────┬──┬────────────────┬──┬──────────────────┬────────────────►
  └─qfull_retry_delay=n──┘  └─signal_limit=n─┘  └─sleep_interval=n─┘

►─┬─────────┬──┬──────────────────┬──┬──────────────────────────┬───────────────────►
  └─stale=n─┘  │            ┌─y─┐ │  │              ┌─warmsi─┐  │
               └─startallq=─┴─n─┴─┘  └─startmode=───┼─warmns─┼──┘
                                                    └─cold───┘

►─┬─────────────┬──┬───────────────┬──────────────────────────────────────────────►◄
  │       ┌─y─┐ │  └─trace_limit=n─┘
  └─term=─┴─n─┴─┘
```

Parameters

The following table defines the invocation parameters for the **asnoqcap** command.

Table 39. Definitions for asnoqcap invocation parameters

Parameter	Definition
capture_server=*dbname*	Specifies the name of the database that contains the Q Capture control tables. If you do not specify a Q Capture server, this parameter defaults to the value of the ORACLE_SID environment variable.
capture_schema=*schema*	Specifies a name that identifies the Q Capture program that you want to start. The default schema is ASN.
capture_path=*path*	Specifies the location where you want a Q Capture program to write its log and work files. The default is the directory where you invoked the **asnoqcap** command. This location is an absolute path name, and you must enclose it in double quotation marks to preserve case.
autostop=**y/n**	Specifies whether a Q Capture program stops after reaching the end of the active Oracle redo log. **y** The Q Capture program stops when it reaches the end of the active Oracle redo log. **n (default)** The Q Capture program does not stop after reaching the end of the active Oracle redo log.
commit_interval=*n*	Specifies how often, in milliseconds, a Q Capture program issues an MQCMIT call. This call signals the WebSphere MQ queue manager to make data messages and informational messages that have been placed on queues available to a Q Apply program or subscribing application. The default is 1000 milliseconds (1 second).

Table 39. Definitions for asnoqcap invocation parameters (continued)

Parameter	Definition
ignore_transid= *transaction_ID*	Specifies that the Q Capture program ignores the transaction that is identified by *transaction_ID*. The transactions are not replicated or published. The transaction ID is the value of XID from the V$CONTENTS view of the Oracle LogMiner utility. The ID is a RAW(8) value. When displayed in text, the transaction ID is formatted as a hexadecimal string (16 digits total). **Tip:** The shortened version **transid** is also acceptable for this parameter.
lob_send_option=I/S	Specifies whether the Q Capture program sends LOB values inlined (I) within a transaction message, or the Q Capture program sends LOB values in a separate message (S). By default, LOB values are sent within the transaction message.
logreuse=**y/n**	Specifies whether a Q Capture program reuses or appends messages to its diagnostic log file. The log file name is *capture_server*.*capture_schema*.QCAP.log. **y** On restart, the Q Capture program reuses its log file by clearing the file and then writing to the blank file. **n (default)** The Q Capture program appends messages to the log file, even after the Q Capture program is restarted.
logstdout=**y/n**	Specifies whether a Q Capture program sends log messages to both its diagnostic log file and the console. **y** The Q Capture program sends log messages to both its log file and the console (stdout). **n (default)** The Q Capture program directs most log messages to the log file only. Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file regardless of the setting for this parameter.

Table 39. Definitions for asnoqcap invocation parameters (continued)

Parameter	Definition

lsn=*n*	Specifies the log record from which the Q Capture program starts reading during a warm restart. This value represents the earliest log record that the Q Capture program found for which a commit or abort record was not yet found.
	
	You can find the value for the **lsn** parameter in the ASN7108I and ASN7109I messages. For example, in the following message the value for **lsn** is "the lowest log sequence number of a transaction still to be committed" or 0000:0000:0F1B:8629:0000:
	
	`2010-03-08-13.34.36.913067 ASN7108I "Q Capture" : "ASN3" : "WorkerThread" : At program initialization, the highest log sequence number of a successfully processed transaction is "0000:0000:0F1B:8486:0000" and the lowest log sequence number of a transaction still to be committed is "0000:0000:0F1B:8629:0000".`
	
	The **lsn** value consists of five sets of four hexadecimal characters, with the first four sets representing the Oracle system change number (SCN) and the last set a sequence number that is generated by Q Capture. For example, the following **lsn** value combines the hex version of SCN 123456 (1E240) with a sequence number of 0000:
	
	`Hex of SCN 123456 Sequence number`
	`0000:0000:0001:E240: 0000`
	
	Q Capture ignores the leading zeros and colons (:). You could also specify the above **lsn** value in the following ways:
	
	`lsn=1E2400000`
	`lsn=1E240:0000`
	`lsn=1:E240:0000`
	`lsn=000000000001E2400000`
	`lsn=0000:0000:0001:E240:0000`
	
	If you specify 0 for the **lsn** and **maxcmtseq** parameters, Q Capture starts reading the log at the current SCN. You can specifiy the **lsn** value from the ASN7108I or ASN7109I message for both **lsn** and **maxcmtseq**, or specify the value from the message for **lsn** and **maxcmtseq**=0, and Q Capture starts reading at the given log sequence number.
	
	Specify **lsn**=FFFF:FFFF:FFFF:FFFF:FFFF to start from the end of the log without triggering a load (full refresh) of the target table.

Table 39. Definitions for asnoqcap invocation parameters (continued)

Parameter	Definition
maxcmtseq=*n*	Specifies the newest log record from a successfully committed transaction that the Q Capture program read during its last run. You can use the **maxcmtseq** value along with the value for the **lsn** parameter to start Q Capture from a known point in the log. You can find this value in the ASN7108I and ASN7109I messages. For example, in the following message the value for **maxcmtseq** is "the highest log sequence number of a successfully processed transaction" or 0000:0000:0F1B:8486:0000: `2010-03-08-13.34.36.913067 ASN7108I "Q Capture" : "ASN3" : "WorkerThread" : At program initialization, the highest log sequence number of a successfully processed transaction is "0000:0000:0F1B:8486:0000" and the lowest log sequence number of a transaction still to be committed is "0000:0000:0F1B:8629:0000".` The format of the **maxcmtseq** value is the same as that of **lsn** (see above). If you build the **maxcmtseq** value manually, you can specify the sequence number as 0000 because Q Capture starts reading the first log record from a given SCN. If you use the value in the ASN7108I and ASN7109I messages, the sequence number might be other than 0000. If you specify 0 for the **lsn** and **maxcmtseq** parameters, Q Capture starts reading the log at the current SCN.
memory_limit=*n*	Specifies the amount of memory, in megabytes, that a Q Capture program can use to build transactions. After this allocation is used, in-memory transactions spill to a file. The default is 500 MB. The maximum allowed value for this parameter is 100 GB.
msg_persistence=y/n	Specifies whether a Q Capture program writes persistent (logged) messages to WebSphere MQ queues. **y (default)** Q Capture write persistent messages to all queues. The messages are logged by the queue manager and can be recovered. **n** Q Capture write nonpersistent messages to all queues. The messages are not logged and cannot be recovered. **Note:** If you specify **msg_persistence**=n, ensure that queues are empty before stopping the queue manager by first stopping Q Capture and then verifying that Q Apply has emptied all queues.

Table 39. Definitions for asnoqcap invocation parameters (continued)

Parameter	Definition
monitor_interval=*n*	Specifies how often, in milliseconds, a Q Capture program adds a row to the IBMQREP_CAPMON and IBMQREP_CAPQMON tables. The default is 30000 milliseconds (30 seconds). **Important for Q Replication Dashboard users:** When possible, you should synchronize the Q Capture **monitor_interval** parameter with the dashboard refresh interval (how often the dashboard retrieves performance information from the Q Capture and Q Apply monitor tables). The default refresh interval for the dashboard is 10 seconds (10000 milliseconds). If the value of **monitor_interval** is higher than the dashboard refresh interval, the dashboard refreshes when no new monitor data is available.
monitor_limit=*n*	Specifies the number of minutes that rows remain in the IBMQREP_CAPMON and the IBMQREP_CAPQMON tables before they can be pruned. At each pruning interval, the Q Capture program prunes rows in these tables if they are older than this limit based on the current timestamp. The default is 10080 minutes (seven days).
pwdfile=*filename*	Specifies the name of the password file that is used to connect to multiple partition databases. If you do not specify a password file, the default is `asnpwd.aut`. This command searches for the password file in the directory specified by the **capture_path** parameter. If no **capture_path** parameter is specified, this command searches for the password file in the directory where the command was invoked. asnpwd: Creating and maintaining password files describes how to create and change a password file.
prune_interval=*n*	Specifies how often, in seconds, a Q Capture program looks for rows that are old enough to prune in the IBMQREP_SIGNAL, IBMQREP_CAPTRACE, IBMQREP_CAPMON and IBMQREP_CAPQMON tables. The default is 300 seconds (five minutes).
qfull_num_ retries=*n*	Specifies the number of retries to attempt. The default is 30 retries, and the maximum is 1,000 retries. A value of 0 instructs the Q Capture program to stop whenever an MQPUT operation fails. The alternate syntax for the **qfull_num_retries** parameter is **RN**.
qfull_retry_delay=*n*	Specifies how long in milliseconds the Q Capture program waits between MQPUT attempts. The allowed value range is 10 milliseconds to 3600000 milliseconds (1 hour). The default delay is 250 milliseconds or the value of the **commit_interval** parameter, whichever is less. (The default for **commit_interval** is 500 milliseconds.) The alternate syntax for the **qfull_retry_delay** parameter is **RD**.
signal_limit=*n*	Specifies the number of minutes that rows remain in the IBMQREP_SIGNAL table before they can be pruned. At each pruning interval, the Q Capture program prunes rows in the IBMQREP_SIGNAL table if they are older than the signal limit based on the current timestamp. The default is 10080 minutes (seven days).

Table 39. Definitions for asnoqcap invocation parameters (continued)

Parameter	Definition
sleep_interval=*n*	Specifies the number of milliseconds that a Q Capture program is idle after processing the active log and any transactions that remain in memory. The default is 2000 milliseconds (2 seconds).
stale=*n*	Specifies the number of seconds that the Q Capture program waits to issue a warning message or take other action after it detects a long-running transaction with no commit or rollback log record. The program behavior depends on the platform of the source. If a transaction has been running for the number of seconds that are specified by **stale** and Q Capture did not see any row operations in the log for the transaction, it issues warning messages, does not replicate the transaction, and increments the log sequence number that it considers to be the oldest "in-flight" transaction that was not committed or rolled back. If some rows were captured for the transaction, only warning messages are issued.
startallq=y/n	Specifies whether the Q Capture program activates all send queues during startup. You can use this parameter to keep a disabled send queue inactive. **y (default)** When the Q Capture program starts, it activates all send queues that are not already in active (A) state. **n** When the Q Capture program starts, it does not activate send queues that are in inactive (I) state.
startmode=*mode*	Specifies the actions that a Q Capture program takes when it starts. **warmsi (default)** The Q Capture program starts reading the log at the point where it left off, except when you start the program for the first time. When you start the program for the first time, the Q Capture program switches to a cold start. The warmsi start mode ensures that the Q Capture program cold starts only when it initially starts. **warmns** The Q Capture program starts reading the log at the point where it left off. If the Q Capture program cannot warm start, it does not switch to cold start. The warmns start mode prevents the Q Capture program from cold starting unexpectedly. When the Q Capture program warm starts, it resumes processing where it ended. If errors occur after the Q Capture program starts, the program terminates and leaves all tables intact. **cold** The Q Capture program clears the restart queue and administration queue, and starts processing all Q subscriptions or publications that are in N (new) or A (active) state. With a cold start, the Q Capture program starts reading the Oracle active redo log at the end. During warm starts, the Q Capture program will load only those Q subscriptions or publications that are not in I (inactive) state.

Table 39. Definitions for asnoqcap invocation parameters (continued)

Parameter	Definition
term=*y/n*	Specifies whether the Q Capture program terminates if the source database is quiesced. The **term** parameter has no effect on Oracle sources. The default value is N, which indicates that the capture program continues running if Oracle is quiesced or stopped. When Oracle is taken out of quiesce mode, the Q Capture program continues processing at the point where it left off when Oracle was quiesced. If the Oracle source is shut down and restarted, the Q Capture program must be stopped and restarted.
trace_limit=*n*	Specifies the number of minutes that rows remain in the IBMQREP_CAPTRACE table before they can be pruned. At each pruning interval, the Q Capture program prunes rows in this table if they are older than the trace limit based on the current timestamp. The default is 10080 minutes (seven days).

Examples for asnoqcap

The following examples illustrate how to use the **asnoqcap** command.

Example - Overriding the default start mode and log parameters

To start a Q Capture program on a server called sourcedb with a schema of alpha, and to start the program in the cold start mode, and to temporarily override the default settings for **logreuse** and **logstdout**, issue the following command:

```
asnoqcap capture_server=sourcedb capture_schema="alpha" startmode=cold
logreuse=y logstdout=y
```

Example - Overriding the default commit message interval

To start a Q Capture program and instruct it to commit messages on queues more frequently than the default, issue the following command:

```
asnoqcap capture_server=sourcedb capture_schema="alpha" commit_interval=250
```

Example - Changing the memory allocation

To start a Q Capture program and temporarily increase the default amount of memory that it uses to build transactions, issue the following command:

```
asnoqcap capture_server=sourcedb capture_schema="alpha" memory_limit=64
```

Example - Specifying the log and work file location

To start a Q Capture program and direct its work files to the `/home/files/qcapture` directory, issue the following command:

```
asnoqcap capture_server=sourcedb capture_schema="alpha"
capture_path="/home/files/qcapture"
```

Example - Specifying `lsn` and `maxcmtseq`

To start a Q Capture program on a server named testdb, and to specify that after a warm restart that the program start at an Oracle system change number (specified by the **lsn** parameter) of 0000:0000:0F1B:8629:0000 and **maxcmtseq** of 0000:0000:0F1B:8486:0000, issue the following command:

```
asnoqcap capture_server=testdb lsn=0000:0000:0F1B:8629:0000
maxcmtseq=0000:0000:0F1B:8486:0000
```

Example - Starting from the end of the log

To start a Q Capture program from the end of the log without triggering a load (full refresh) of the target table, issue the following command:

```
asnoqcap capture_server=testdb lsn=FFFF:FFFF:FFFF:FFFF:FFFF
maxcmtseq=FFFF:FFFF:FFFF:FFFF:FFFF
```

Example - Ignoring a specific transaction

To start a Q Capture program on an Oracle server and specify that the program ignore a transaction with a transaction ID of BD71:1E23:B089:0001, issue the following command:

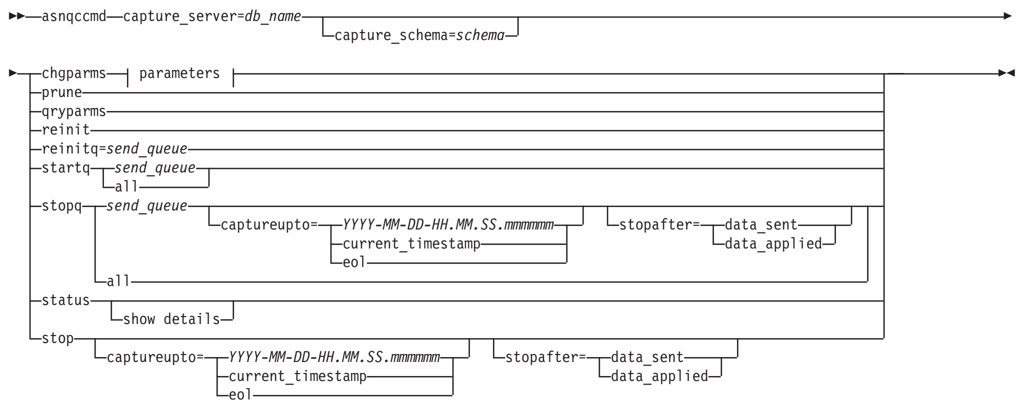
```
asnoqcap capture_server=ora_sample capture_schema=ASN
ignore_transid=BD71:1E23:B089:0001
```

asnqccmd: Working with a running Q Capture program

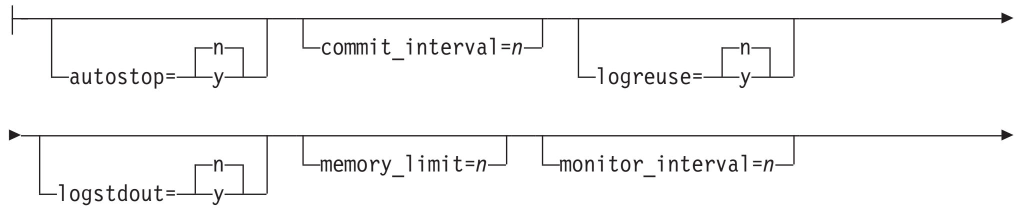
Use **asnqccmd** to send a command to a running Q Capture program on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS. Run this command at an operating system prompt or in a shell script.

For information on using the MVS™ MODIFY command to send commands to a running Q Capture program on z/OS, see Working with running Q Replication and Event Publishing programs by using the MVS MODIFY command.

Syntax



Parameters:



```
►─┬─────────────────┬──┬───────────────────┬──┬────────────────┬──────────────────►
  └─monitor_limit=n─┘  └─prune_interval=n──┘  └─signal_limit=n─┘

►─┬──────────────────┬──┬──────────────┬──┬───────────────┬───────────────────────┤
  └─sleep_interval=n─┘  │       ┌─y─┐  │  └─trace_limit=n─┘
                        └─term=─┴─n─┴──┘
```

Parameters

Table 40 defines the invocation parameters for the **asnqccmd** command.

Table 40. Definitions for asnqccmd invocation parameters

Parameter	Definition
capture_server=*db_name*	Specifies the name of the database or subsystem that contains the Q Capture control tables. z/OS Specifies the name of the DB2 subsystem where the Q Capture program is running. For data sharing, the value of **capture_server** might be the group attach name that was used to run Q Capture in any LPAR without changing the JCL for the started task. Linux UNIX Windows If you do not specify a Q Capture server, this parameter defaults to the value from the *DB2DBDFT* environment variable.
capture_schema=*schema*	Specifies a name that identifies the Q Capture program that you want to work with.

Table 40. Definitions for asnqccmd invocation parameters (continued)

Parameter	Definition
chgparms	Specify to change one or more of the following operational parameters of a Q Capture program while it is running: • **autostop** • **commit_interval** • **logreuse** • **logstdout** • **memory_limit** • **monitor_interval** • **monitor_limit** • **prune_interval** • **qfull_num_retries** • **qfull_retry_delay** • **signal_limit** • **sleep_interval** • **term** • **trace_limit** **Important:** The parameter that you are changing must immediately follow the **chgparms** parameter. z/OS **Restriction:** The value of **memory_limit** cannot be altered while the Q Capture program is running. To change the value you must stop the Q Capture program. You can specify multiple parameters in one **asnqccmd chgparms** command, and you can change these parameter values as often as you want. The changes temporarily override the values in the IBMQREP_CAPPARMS table, but they are not written to the table. When you stop and restart the Q Capture program, it uses the values in IBMQREP_CAPPARMS. "Descriptions of asnqcap parameters" on page 351 includes details about the parameters that you can override with this command.
prune	Specify to instruct a Q Capture program to prune the IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_CAPTRACE, and IBMQREP_SIGNAL tables once. This pruning is in addition to any regularly scheduled pruning that is specified by the **prune_interval** parameter.
qryparms	Specify if you want the current operational parameter values for a Q Capture program written to the standard output (stdout).
reinit	Specify to have a Q Capture program deactivate and then activate all Q subscriptions and publications using the latest values in the IBMQREP_SUBS, IBMQREP_SRC_COLS, and IBMQREP_SENDQUEUES tables. This command allows you to change some attributes for multiple Q subscriptions or publications while a Q Capture program is running. This command will not prompt a new load of targets.

Table 40. Definitions for asnqccmd invocation parameters (continued)

Parameter	Definition
reinitq=*send_queue*	Specify to have the Q Capture program refresh one send queue using the latest attributes from the IBMQREP_SENDQUEUES table. This command affects all Q subscriptions or publications that use this send queue. Only the following attributes will be refreshed: ERROR_ACTION, HEARTBEAT_INTERVAL, MAX_MESSAGE_SIZE.
startq=*send_queue*/**all**	Specify to start putting messages on one or all disabled send queues. Q Capture sets the queue state to active (A) and resumes putting messages on a specified queue or all inactive queues. Q Capture restarts reading the log at the oldest restart point among all send queues, and catches up the queues that were stopped until all queues have the same restart point.
status	Specify to receive messages that indicate the state of each Q Capture thread (main, administration, prune, holdl, transaction, and worker).
show details	Specify after the **status** parameter to view a more detailed report about Q Capture program status, with the following information: • Whether the Q Capture program is running • Time since the program started • Location of the Q Capture diagnostic log • Number of active Q subscriptions • Value of the CURRENT_LOG_TIME and CURRENT_MEMORY. These values might be newer than what is inserted into the IBMQREP_CAPMON control table. • Logical log sequence number of the last transaction that the Q Capture program published to a send queue • Amount of memory in megabytes that Q Capture used during the latest monitor interval to build transactions from log records Linux UNIX Windows • Path to DB2 log files • Oldest DB2 log file needed for a Q Capture restart • Current® DB2 log file captured

Table 40. Definitions for asnqccmd invocation parameters (continued)

Parameter	Definition
stop	Specify to stop the Q Capture program in an orderly way and commit the messages that it processed up to that point. You can specify the **captureupto** parameter, **stopafter** parameter, or both to stop Q Capture in a controlled manner from a specified point: **captureupto** Specify with a full or partial timestamp to instruct Q Capture to stop reading the log at a specified point and then stop. Alternatively, you can specify CURRENT_TIMESTAMP, or specify EOL to prompt Q Capture to stop after it reaches the end of the active log. **stopafter** Specifies that Q Capture stop after one of the following conditions is true: **data_applied** Q Capture stops after all changes up to the specified stopping point are applied at the target. **data_sent** Q Capture stops after all messages are removed from the transmission queue that points to the target system, or after the Q Apply program has processed all messages in the case of a shared local send queue-receive queue.

Table 40. Definitions for asnqccmd invocation parameters (continued)

Parameter	Definition
stopq=*send_queue*/**all**	Specify to stop putting messages on one or all send queues. Q Capture sets the queue state to inactive (I) and stops putting messages on a specified queue or all queues. The Q Capture program continues to publish changes for Q subscriptions that are associated with active send queues. If all send queues are stopped, Q Capture continues reading the log for signals such as CAPSTART, continues to insert into its monitor tables, and waits for commands. When you specify to stop an individual queue, you can also specify the **captureupto** parameter, **stopafter** parameter, or both to instruct Q Capture to stop putting messages on the queue in a controlled manner from a specified point: **captureupto** Specify with a full or partial timestamp to instruct Q Capture to publish up to the specified point and then stop putting messages on the queue. Alternatively, you can specify CURRENT_TIMESTAMP, or specify EOL to prompt Q Capture to stop putting messages on the queue after it reaches the end of the active log. **stopafter** Specifies that Q Capture stop putting messages on the queue after one of the following conditions is true: **data_applied** All changes up to the specified stopping point are applied at the target. **data_sent** All messages are removed from the transmission queue that points to the target system, or the Q Apply program has processed all messages in the case of a shared local send queue-receive queue.

Example 1

To instruct a Q Capture program to refresh all Q subscriptions and publications using the latest values in the Q Capture control tables:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" reinit
```

This command will start all inactive queues if the Q Capture program was started with **startallq**=y (the default setting).

Example 2

To instruct a Q Capture program to refresh the ERROR_ACTION, HEARTBEAT_INTERVAL, and MAX_MESSAGE_SIZE attributes for all Q subscriptions and publications that use a send queue called Q1:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" reinitq="Q1"
```

Example 3

To temporarily shorten the default pruning interval for a running Q Capture program to one minute, and temporarily lengthen the default amount of time that the Q Capture program sleeps after processing Q subscriptions and publications:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" chgparms
prune_interval=60 sleep_interval=10000
```

Example 4

To receive detailed messages about the status of the Q Capture program:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" status show details
```

Example 5

To prompt a Q Capture program to stop reading the recovery log October 15, 2010 at 3:30 p.m., wait until all transactions that were committed to the target database up to that time are applied, and then stop:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" stop
captureupto="2010-10-15-15.30" stopafter=data_applied
```

Example 6

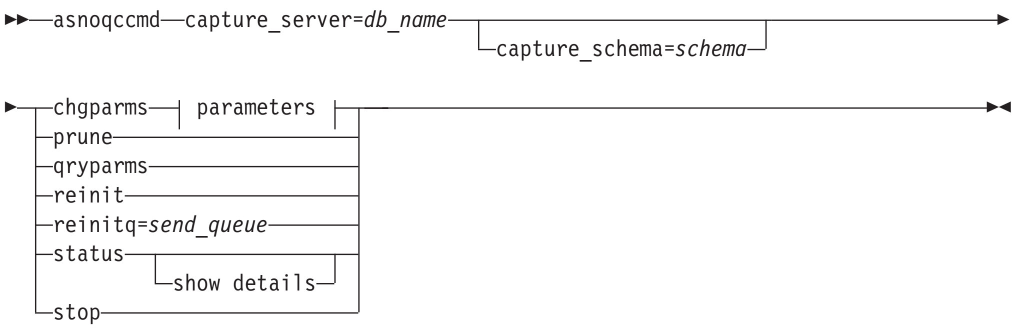
To prompt a Q Capture program to stop reading the recovery log immediately and then stop after all messages are drained from the transmission queue (or the shared local send-receive queue):

```
asnqccmd capture_server=sourcedb capture_schema="alpha" stop stopafter=data_sent
```

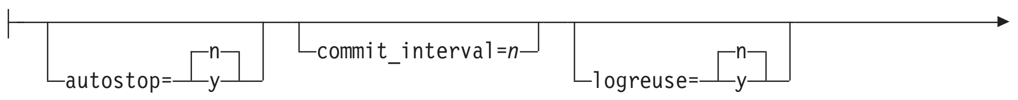
asnoqccmd: Working with a running Q Capture program on Oracle databases

Use the **asnoqccmd** command to send a command to a Q Capture program that is running on an Oracle database on a Linux, UNIX, or Windows system. Run this command at an operating system prompt or in a shell script.

Syntax



Parameters:



```
►─┬────────────────────┬─┬──────────────────┬─┬────────────────────┬──────────────►
  │              ┌─n─┐ │ └─memory_limit=n─┘ └─monitor_interval=n─┘
  └─logstdout=───┴─y─┴─┘

►─┬─────────────────┬─┬──────────────────┬─┬────────────────┬──────────────────────►
  └─monitor_limit=n─┘ └─prune_interval=n─┘ └─signal_limit=n─┘

►─┬──────────────────┬─┬───────────────┬─┬───────────────┬─────────────────────────┤
  └─sleep_interval=n─┘ │         ┌─y─┐ │ └─trace_limit=n─┘
                       └─term=───┴─n─┴─┘
```

Parameters

The following table defines the invocation parameters for the **asnoqccmd** command.

Table 41. Definitions for asnoqccmd invocation parameters

Parameter	Definition
capture_server=*db_name*	Specifies the name of the Oracle database that contains the Q Capture control tables. If you do not specify a Q Capture server, this parameter defaults to the value from the *ORACLE_SID* environment variable.
capture_schema=*schema*	Specifies a name that identifies the Q Capture program that you want to work with. The default schema is ASN.
chgparms	Specify this parameter to change one or more of the following operational parameters of a Q Capture program while it is running on an Oracle database: • **autostop** • **commit_interval** • **logreuse** • **logstdout** • **memory_limit** • **monitor_interval** • **monitor_limit** • **prune_interval** • **qfull_num_retries** • **qfull_retry_delay** • **signal_limit** • **sleep_interval** • **trace_limit** You can specify multiple parameters in one **asnoqccmd chgparms** command, and you can change these parameter values as often as you want. The changes temporarily override the values in the IBMQREP_CAPPARMS table, but they are not written to the table. When you stop and restart the Q Capture program, it uses the values in IBMQREP_CAPPARMS. “asnoqcap: Starting a Q Capture program for an Oracle database” on page 372 includes descriptions of the parameters that you can override with this command. **Requirement:** The parameter that you are changing must immediately follow the **chgparms** parameter.

Table 41. Definitions for asnoqccmd invocation parameters (continued)

Parameter	Definition
prune	Specify this parameter to instruct a Q Capture program to prune the IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_CAPTRACE, and IBMQREP_SIGNAL tables once. This pruning is in addition to any regularly scheduled pruning that is specified by the **prune_interval** parameter.
qryparms	Specify this parameter if you want the current operational parameter values for a Q Capture program to be written to the standard output (stdout).
reinit	Specify this parameter to have a Q Capture program deactivate and then activate all Q subscriptions and publications using the latest values in the IBMQREP_SUBS, IBMQREP_SRC_COLS, and IBMQREP_SENDQUEUES tables. This command allows you to change some attributes for multiple Q subscriptions or publications while a Q Capture program is running. This command will not prompt a new load of targets.
reinitq=*send_queue*	Specify this parameter to have the Q Capture program refresh one send queue using the latest attributes from the IBMQREP_SENDQUEUES table. This command affects all Q subscriptions or publications that use this send queue. Only the following attributes will be refreshed: ERROR_ACTION, HEARTBEAT_INTERVAL, MAX_MESSAGE_SIZE.
status	Specify this parameter to receive messages that indicate the state of each Q Capture thread (main, administration, prune, holdl, worker, and transaction).
show details	Specify this parameter after the **status** parameter to view a more detailed report about Q Capture program status, with the following information: • Whether the Q Capture program is running • Time since the program started • Location of the Q Capture diagnostic log • Number of active Q subscriptions • Value of the CURRENT_LOG_TIME and CURRENT_MEMORY. These values might be newer than the values that are inserted into the IBMQREP_CAPMON control table. • Logical log sequence number of the last transaction that the Q Capture program published to a send queue. This number is shown as a hexadecimal translation of the original decimal Oracle system change number (SCN). • Amount of memory in megabytes that Q Capture used during the latest monitor interval to build transactions from log records • Path to Oracle redo log files • Oldest Oracle redo log file that is needed for a Q Capture restart • Current Oracle redo log file captured
stop	Specify this parameter to stop the Q Capture program in an orderly way and commit the messages that it processed up to that point.

Example - Refresh all Q subscriptions and publications

To instruct a Q Capture program to refresh all Q subscriptions and publications with the latest values in the Q Capture control tables, issue the following command:

```
asnoqccmd capture_server=sourcedb capture_schema="alpha" reinit
```

Example - Refresh send queue attributes

To instruct a Q Capture program to refresh the ERROR_ACTION, HEARTBEAT_INTERVAL, and MAX_MESSAGE_SIZE attributes for all Q subscriptions and publications that use a send queue called Q1, issue the following command:

```
asnoqccmd capture_server=sourcedb capture_schema="alpha" reinitq="Q1"
```

Example - Change specified parameters

To temporarily shorten the default pruning interval for a running Q Capture program to one minute, and to temporarily lengthen the default amount of time that the Q Capture program sleeps after processing Q subscriptions and publications, issue the following command:

```
asnoqccmd capture_server=sourcedb capture_schema="alpha" chgparms
prune_interval=60 sleep_interval=10000
```

Example - Show Q Capture status details

To receive detailed messages about the status of the Q Capture program, issue the following command:

```
asnoqccmd capture_server=sourcedb capture_schema="alpha" status show details
```

asnqapp: Starting a Q Apply program

Use the **asnqapp** command on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS to start a Q Apply program. Run this command at an operating system prompt or in a shell script. Any startup parameters that you specify will apply to this session only.

After you start the Q Apply program, it runs continuously until you stop it or it detects an unrecoverable error.

Syntax

```
►►──asnqapp──apply_server=db_name──┬──────────────────────┬──────────►
                                   └─apply_schema=schema─┘

►─┬──────────────────┬──┬─────────────┬──────────────────────────────►
  └─apply_path=path─┘  └─applydelay=n─┘
```

```
►──┬─────────────────────────────────────────────────────┬──┬──────────────────┬──────►
   │                                   ┌─WAIT───┐        │  └─arm=identifier─┘
   ├─applyupto=─┬─gmt_timestamp──,────┴─NOWAIT─┴──────┬──┤
   │            │                          ┌─WAIT───┐   │  │
   │            └─CURRENT_TIMESTAMP──,────┴─NOWAIT─┴──┘  │
   │             ┌─y─┐                                    │
   └─autostop=──┴─n─┴────────────────────────────────────┘

►──┬───────────────────────┬──┬──────────┬──┬────────────────────────┬──────►
   │                 ┌─n─┐ │  │    ┌─n─┐ │  └─classic_load_file_sz=n─┘
   └─buffered_inserts=──┴─y─┴─┘  └─caf=──┴─y─┴─┘

►──┬────────────────┬──┬───────────────────┬──────►
   └─commit_count=n─┘  └─deadlock_retries=n─┘

►──┬───────────────────────────────────┬──┬────────────────────┬──────►
   └─dftmodelq=model_spill_queue_name─┘  │              ┌─n─┐ │
                                          └─ignbaddata=──┴─y─┴─┘

►──┬─────────────────────────────────────────────────────────┬──────►
   │                    ┌─y─┐                                │
   └─insert_bidi_signal=──┴─n─┴──┬───────────────────────────┬─┘
                                 └─loadcopy_path=filepath─┘

►──┬─────────────────────┬──┬──────────────────────────┬──┬──────────────────┬──────►
   └─load_data_buff_sz=n─┘  │               ┌─gmt───┐  │  │            ┌─n─┐ │
                            └─logmarkertz=──┴─local─┴──┘  └─logreuse=──┴─y─┴─┘

►──┬────────────────────────────────────────────────┬──┬────────────────────┬──────►
   │            ┌─n─┐                               │  └─monitor_interval=n─┘
   └─logstdout=──┴─y─┴──┬─────────────────────────┬─┘
                        └─max_parallel_loads=n─┘

►──┬─────────────────┬──┬─────────────────────┬──┬─────────────────────┬──────►
   └─monitor_limit=n─┘  └─nickname_commit_ct=n─┘  │              ┌─n─┐ │
                                                  └─nmi_enable=──┴─y─┴─┘

►──┬───────────────────┬──┬──────────────────────────┬──┬──────────────────┬──────►
   └─nmi_socket_name=n─┘  │                   ┌─y─┐  │  │             ┌─n─┐ │
                          └─oracle_empty_str=──┴─n─┴──┘  └─p2p_2nodes=──┴─y─┴─┘

►──┬────────────────────┬──┬──────────────────┬──┬────────────────────────┬──────►
   └─prune_batch_size=n─┘  └─prune_interval=n─┘  │               ┌─1─┐    │
                                                 └─prune_method=──┴─2─┴────┘

►──┬─────────────────────────────┬──┬───────────────────────────┬──┬────────────┬──────►
   │          ┌─asnpwd.aut─┐     │  │                   ┌─y─┐   │  └─richklvl=n─┘
   └─pwdfile=──┴─filename───┴────┘  └─report_exception=──┴─n─┴───┘

►──┬───────────────┬──┬──────────────────────────────────┬──────►
   │       ┌─y─┐   │  └─skiptrans=─┤ skiptrans-clause ├─┘
   └─term=──┴─n─┴──┘

►──┬─────────────────────┬──┬─────────────────┬──┬────────────────────┬──────►
   └─spill_commit_count=n─┘  └─startallq=──┬─y─┬┘  │             ┌─n─┐  │
                                           └─n─┘   └─trace_ddl=──┴─y─┴──┘

►──┬───────────────┬──────────────────────────────────────────────────────►◄
   └─trace_limit=n─┘
```

skiptrans-clause::

```
   ┌─,──────────────────────────────────────────────────────────┐
   ▼                                                            │
├──┴─receive_queue─;─┬─transaction_ID─────────────────────────┬─┴──────────┤
                     └─begin_transaction_ID-end_transaction_ID─┘
```

Descriptions of asnqapp parameters

These descriptions provide detail on the asnqapp parameters, their defaults, and why you might want to change the default in your environment.

- "apply_server"
- "apply_schema"
- "apply_path" on page 392
- "applydelay" on page 393
- "applyupto" on page 393
- "arm" on page 395
- "autostop" on page 395
- "buffered_inserts" on page 395
- "caf" on page 396
- "commit_count" on page 396
- "deadlock_retries" on page 396
- "dftmodelq" on page 397
- "ignbaddata" on page 397
- "insert_bidi_signal" on page 398
- "loadcopy_path" on page 398
- "load_data_buff_sz" on page 399
- "logmarkertz" on page 399
- "logreuse" on page 399
- "logstdout" on page 400
- "max_parallel_loads" on page 400
- "monitor_interval" on page 401
- "monitor_limit" on page 401
- "nickname_commit_ct" on page 402
- "p2p_2nodes" on page 402
- "pwdfile" on page 404
- "prune_interval" on page 403
- "report_exception" on page 404
- "richklvl" on page 404
- "skiptrans" on page 405
- "spill_commit_count" on page 405
- "startallq" on page 405
- "term" on page 406
- "trace_limit" on page 406
- "trace_ddl" on page 406

apply_server

z/OS **Default:** None

Linux UNIX Windows **Default:** **apply_server**=value of DB2DBDFT environment variable, if it is set

The **apply_server** parameter identifies the database or subsystem where a Q Apply program runs, and where its control tables are stored. The control tables contain information about targets, Q subscriptions, WebSphere MQ queues, and user preferences. The Q Apply server must be the same database or subsystem that contains the targets.

apply_schema

Default: apply_schema=ASN

The **apply_schema** parameter lets you distinguish between multiple instances of the Q Apply program on a Q Apply server.

The schema identifies one Q Apply program and its control tables. Two Q Apply programs with the same schema cannot run on a server.

A single Q Apply program can create multiple browser threads. Each browser thread reads messages from a single receive queue. Because of this, you do not need to create multiple instances of the Q Apply program on a server to divide the flow of data that is being applied to targets.

z/OS On z/OS, no special characters are allowed in the Q Apply schema except for the underscore (_).

apply_path

Default: None

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **apply_path** parameter specifies the directory where a Q Apply program stores its work files and log file. By default, the path is the directory where you start the program. You can change this path.

Windows

If you start a Q Apply program as a Windows service, by default the program starts in the \SQLLIB\bin directory.

z/OS

Because the Q Apply program is a POSIX application, the default path depends on how you start the program:

- If you start a Q Apply program from a USS command line prompt, the path is the directory where you started the program.
- If you start a Q Apply program using a started task or through JCL, the default path is the home directory in the USS file system of the user ID that is associated with the started task or job.

To change the path, you can specify either a path name or a high-level Qualifier (HLQ), such as //QAPP. When you use an HLQ, sequential files are created that conform to the file naming conventions for z/OS sequential data set file names. The sequential data sets are relative to the user ID that is running the program. Otherwise these file names are similar to the names that are stored in an explicitly named directory path, with the HLQ concatenated as the first part of the file name. For example, sysadm.QAPPV9.filename. Using an HLQ might be convenient if you want to have the Q Apply log and LOADMSG files be system-managed (SMS).

If you want the Q Apply started task to write to a .log data set with a user ID other than the ID that is executing the task (for example TSOUSER), you must specify a single quotation mark (‘) as an escape character when using the SYSIN format for input parameters to the started task. For example, if you wanted to use the high-level qualifier JOESMITH, then the user ID TSOUSER that is running the Q Apply program must have RACF authority to write data sets by using the high-level qualifier JOESMITH, as in the following example:

```
//SYSIN    DD  *
 APPLY_PATH=//'JOESMITH
/*
```

You can set the **apply_path** parameter when you start the Q Apply program, or you can change its saved value in the IBMQREP_APPLYPARMS table. You cannot alter this parameter while the Q Apply program is running.

applydelay

Default: **applydelay**=0 seconds

Method of changing: When Q Apply starts

The **applydelay** parameter controls the amount of time in seconds that the Q Apply program waits before replaying each transaction at the target. The delay is based on the source commit time of the transaction. Q Apply delays applying transactions until the current time reaches or exceeds the source transaction commit time plus the value of **applydelay**. Changes at the source database are captured and sent to the receive queue, where they wait during the delay period.

This parameter can be used, for example, to maintain multiple copies of a source database at different points in time for failover in case of problems at the source system. For example, if a user accidentally deletes data at the primary system, a copy of the database exists where the data is still available.

The **applydelay** parameter has no effect on the **applyupto** or **autostop** parameters.

Important: If you plan to use the **applydelay** parameter, ensure that the receive queue has enough space to hold messages that accumulate during the delay period.

applyupto

Default: None

Method of changing: When Q Apply starts

The **applyupto** parameter identifies a timestamp that instructs the Q Apply program to stop after processing transactions that were committed at the source on or before one of the following times:

- A specific timestamp that you provide
- The CURRENT_TIMESTAMP keyword, which signifies the time that the Q Apply program started

You can optionally specify the WAIT or NOWAIT keywords to control when Q Apply stops:

WAIT (default)
: Q Apply does not stop until it receives and processes all transactions up to the specified GMT timestamp or the value of CURRENT_TIMESTAMP, even if the receive queue becomes empty.

NOWAIT
: Q Apply stops after it processes all transactions on the receive queue, even if it has not seen a transaction with a commit timestamp that matches or exceeds the specified GMT timestamp or the value of CURRENT_TIMESTAMP.

The **applyupto** parameter applies to all browser threads of a Q Apply instance. Each browser thread stops when it reads a message on its receive queue with a source commit timestamp that matches or exceeds the specified time. The Q Apply program stops when all of its browser threads determine that all transactions with a source commit timestamp prior to and including the **applyupto** timestamp have been applied. All transactions with a source commit time greater than the specified GMT timestamp stay on the receive queue and are processed the next time the Q Apply program runs.

The timestamp must be specified in Greenwich mean time (GMT) in a full or partial timestamp format. The full timestamp uses the following format: `YYYY-MM-DD-HH.MM.SS.mmmmmm`. For example, `2007-04-10-10.35.30.555555` is the GMT timestamp for April 10th, 2007, 10:35 AM, 30 seconds, and 555555 microseconds.

You can specify the partial timestamp in one of the following formats:

YYYY-MM-DD-HH.MM.SS
: For example, `2007-04-10-23.35.30` is the partial GMT timestamp for April 10th, 2007, 11:35 PM, 30 seconds.

YYYY-MM-DD-HH.MM
: For example, `2007-04-10-14.30` is the partial GMT timestamp for April 10th, 2007, 1:30 PM.

YYYY-MM-DD-HH
: For example, `2007-04-10-01` is the partial GMT timestamp for April 10th, 2007, 1:00 AM.

HH.MM For example, `14:55` is the partial GMT timestamp for today at 2:55 PM.

HH For example, `14` is the partial GMT timestamp for today at 2 PM.

The partial timestamp could be used to specify a time in the format `HH.MM`. This format could be helpful if you schedule a task to start the Q Apply program every day at 1 AM Pacific Standard Time (PST) and you want to stop the program after processing the transactions that were committed at the source with a GMT timestamp on or before 4 AM PST. For example, run the following task at 1 AM PST and set the **applyupto** parameter to end the task at 4 AM PST:

```
asnqapp apply_server=MYTESTSERVER apply_schema=ASN applyupto=12.00
```

During daylight saving time, the difference between GMT and local time might change depending on your location. For example, the Pacific timezone is GMT-8 hours during the fall and winter. During the daylight saving time in the spring and summer, the Pacific timezone is GMT-7 hours.

Restriction: You cannot specify both the **autostop** parameter and the **applyupto** parameter.

You might want to set the heartbeat interval to a value greater than zero so that the Q Apply program can tell if the time value specified in the **applyupto** parameter has passed.

z/OS

arm

Default: None

Method of changing: When Q Apply starts

You can use the **arm**=*identifier* parameter to specify a unique identifier for the Q Apply program that the Automatic Restart Manager uses to automatically start a stopped Q Apply instance. The three-character alphanumeric value that you supply is appended to the ARM element name that Q Apply generates for itself: ASNQA*xxxxyyyy* (where *xxxx* is the data-sharing group attach name, and *yyyy* is the DB2 member name). You can specify any length of string for the **arm** parameter, but the Q Apply program will concatenate only up to three characters to the current name. If necessary, the Q Apply program pads the name with blanks to make a unique 16-byte name.

autostop

Default: **autostop**=n

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **autostop** parameter lets you direct a Q Apply program to automatically stop when there are no transactions to apply. By default (**autostop**=n), a Q Apply program keeps running when queues are empty and waits for transactions to arrive.

Typically, the Q Apply program is run as a continuous process whenever the target database is active, so in most cases you would keep the default (**autostop**=n). Set **autostop**=y only for scenarios where the Q Apply program is run at set intervals, such as when you synchronize infrequently connected systems, or in test scenarios.

If you set **autostop**=y, the Q Apply program shuts down after all receive queues are emptied once. When the browser thread for each receive queue detects that the queue has no messages, the thread stops reading from the queue. After all threads stop, the Q Apply program stops. Messages might continue to arrive on queues for which the browser thread has stopped, but the messages will collect until you start the Q Apply program again.

Restriction: You cannot specify both the **autostop** parameter and the **applyupto** parameter.

Linux UNIX Windows

buffered_inserts

Default: **buffered_inserts**=n

Method of changing: When Q Apply starts

The **buffered_inserts** parameter specifies whether the Q Apply program uses buffered inserts, which can improve performance in some partitioned databases that are dominated by INSERT operations. If you specify **buffered_inserts**=y, Q Apply internally binds appropriate files with the INSERT BUF option. This bind option enables the coordinator node in a partitioned database to accumulate inserted rows in buffers rather than forwarding them immediately to their destination partitions. When a buffer is filled, or when another SQL statement such as an UPDATE, DELETE, or INSERT to a different table, or COMMIT/ROLLBACK are encountered, all the rows in the buffer are sent together to the destination partition.

You might see additional performance gains by combining the use of buffered inserts with the **commit_count** parameter.

When buffered inserts are enabled, Q Apply does not perform exception handling. Any conflict or error prompts Q Apply to stop reading from the queue. To recover past the point of an exception, you must start message processing on the queue and start Q Apply with **buffered_inserts**=n.

z/OS

caf

Default: caf=n

Method of changing: When Q Apply starts

The Q Apply program runs with the default of Recoverable Resource Manager Services (RRS) connect. You can override this default and prompt the Q Apply program to use the Call Attach Facility (CAF) by specifying the **caf**=y option.

If RRS is not available you receive a message and the Q Apply program switches to CAF. The message warns that the program was not able to initialize a connection because RRS is not started. The program attempts to use CAF instead. The program runs correctly with CAF connect.

commit_count

Default: commit_count=1

Method of changing: When Q Apply starts

The **commit_count** parameter specifies the number of transactions that each Q Apply agent thread applies to the target table within a commit scope. By default, the agent threads commit after each transaction that they apply.

By increasing **commit_count** and grouping more transactions within the commit scope, you might see improved performance.

Recommendation: Use a higher value for **commit_count** only with row-level locking. This parameter requires careful tuning when used with a large number of agent threads because it could cause lock escalation resulting in lock timeouts and deadlock retries.

deadlock_retries

Default: deadlock_retries=3

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **deadlock_retries** parameter specifies how many times the Q Apply program tries to reapply changes to target tables when it encounters an SQL deadlock or lock timeout. The default is three tries. This parameter also controls the number of times that the Q Apply program tries to insert, update, or delete rows from its control tables after an SQL deadlock.

After the limit is reached, if deadlocks persist the browser thread stops. You might want to set a higher value for **deadlock_retries** if applications are updating the target database frequently and you are experiencing a high level of contention. Or, if you have a large number of receive queues and corresponding browser threads, a higher value for **deadlock_retries** might help resolve possible contention in peer-to-peer and other multidirectional replication environments, as well as at control tables such as the IBMQREP_DONEMSG table.

Restriction: You cannot lower the default value for **deadlock_retries**.

dftmodelq

Default: None

Method of changing: When Q Apply starts

By default, the Q Apply program uses IBMQREP.SPILL.MODELQ as the name for the model queue that it uses to create spill queues for the loading process. To specify a different default model queue name, specify the **dftmodelq** parameter. The following list summarizes the behavior of the parameter:

If you specify dftmodelq when you start Q Apply
For each Q subscription, Q Apply will check to see if you specified a model queue name for the Q subscription by looking at the value of the MODELQ column in the IBMQREP_TARGETS control table:
- If the value is NULL or IBMQREP.SPILL.MODELQ, then Q Apply will use the value that you specify for the **dftmodelq** parameter.
- If the column contains any other non-NULL value, then Q Apply will use the value in the MODELQ column and will ignore the value that you specify for the **dftmodelq** parameter.

If you do not specify dftmodelq when you start Q Apply
Q Apply will use the value of the MODELQ column in the IBMQREP_TARGETS table. If the value is NULL, Q Apply will default to IBMQREP.SPILL.MODELQ.

ignbaddata

Default: None

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

Note: This parameter applies only if the Q Apply program uses International Components for Unicode (ICU) for code page conversion (if the code page of the source database and the code page that Q Apply uses are different).

The **ignbaddata** parameter specifies whether the Q Apply program checks for illegal characters in data from the source and continues processing even if it finds illegal characters.

If you specify **ignbaddata**=y, Q Apply checks for illegal characters and takes the following actions if any are found:

- Does not apply the row with the illegal characters.
- Inserts a row into the IBMQREP_EXCEPTIONS table that contains a hexadecimal representation of the illegal characters.
- Continues processing the next row and does not follow the error action that is specified for the Q subscription.

A value of n prompts Q Apply to not check for illegal characters and not report exceptions for illegal characters. With this option, the row might be applied to the target table if DB2 does not reject the data. If the row is applied, Q Apply continues processing the next row. If the bad data prompts an SQL error, Q Apply follows the error action that is specified for the Q subscription and reports an exception.

insert_bidi_signal

Default: **insert_bidi_signal**=y

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **insert_bidi_signal** parameter specifies whether the Q Capture and Q Apply programs use signal inserts to prevent recapture of transactions in bidirectional replication.

By default, the Q Apply program inserts P2PNORECAPTURE signals into the IBMQREP_SIGNAL table to instruct the Q Capture program at its same server not to recapture applied transactions at this server.

When there are many bidirectional Q subscriptions, the number of signal inserts can affect replication performance. By specifying **insert_bidi_signal**=n, the Q Apply program does not insert P2PNORECAPTURE signals. Instead, you insert Q Apply's AUTHTKN information into the IBMQREP_IGNTRAN table, which instructs the Q Capture program at the same server to not capture any transactions that originated from the Q Apply program, except for inserts into the IBMQREP_SIGNAL table.

For improved performance when you use **insert_bidi_signal**=n, update the IBMQREP_IGNTRAN table to change the value of the IGNTRANTRC column to N (no tracing). This change prevents the Q Capture program from inserting a row into the IBMQREP_IGNTRANTRC table for each transaction that it does not recapture.

loadcopy_path

Default: **loadcopy_path**=*Value of apply_path parameter*

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

Use with the DB2 High Availability Disaster Recovery (HADR) feature: You can use the **loadcopy_path** parameter instead of the DB2_LOAD_COPY_NO_OVERRIDE registry variable when the Q Apply server is the primary server in a HADR configuration and tables on the primary server are loaded by the Q Apply program calling the DB2 LOAD utility. HADR sends log files to the standby site, but when a table on the primary server is loaded by the DB2 LOAD utility, the inserts are not logged. Setting LOADCOPY_PATH to an NFS directory that is accessible from both the primary and secondary servers prompts Q Apply to start the LOAD utility with the option to create a copy of the loaded data in the specified path. The secondary server in the HADR configuration then looks for the copied data in this path.

load_data_buff_sz

Default: `load_data_buff_size`=8

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

Use with multidimensional clustering (MDC) tables: Specifies the number of 4KB pages for the DB2 LOAD utility to use as buffered space for transferring data within the utility during the initial loading of the target table. This parameter applies only to automatic loads using the DB2 LOAD utility.

By default, the Q Apply program starts the utility with the option to use a buffer of 8 pages, which is also the minimum value for this parameter. Load performance for MDC targets can be significantly improved by specifying a much higher number of pages.

logmarkertz

Default: `logmarkertz`=gmt

Methods of changing: When Q Apply starts

The `logmarkertz` parameter controls the time zone that the Q Apply program uses when it inserts source commit data into the IBMSNAP_LOGMARKER column of consistent-change data (CCD) tables or point-in-time (PIT) tables. By default (`logmarkertz`=gmt), Q Apply inserts a timestamp in Greenwich mean time (GMT) to record when the data was committed at the source. If the Q Capture and Q Apply servers are in the same time zone with the same Daylight Savings Time or other time change in effect, you can specify `logmarkertz`=local and Q Apply inserts a timestamp in the local time of the Q Apply server.

Note: Because the value in the IBMSNAP_LOGMARKER column records the time that a row was committed at the source database, specifying `logmarkertz`=local is useful only when the Q Capture and Q Apply servers are in the same time zone with the same Daylight Savings Time or other time change in effect.

Existing rows in CCD or PIT targets that were generated before the use of `logmarkertz`=local are not converted by Q Apply and remain in GMT unless you manually convert them.

The `logmarkertz` parameter does not affect stored procedure targets. The `src_commit_timestamp IN` parameter for stored procedure targets always uses GMT-based timestamps.

logreuse

Default: `logreuse`=n

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

Each Q Apply program keeps a log file that tracks its work history, such as when it starts and stops reading from queues, changes parameter values, prunes control tables, or encounters errors.

By default, the Q Apply program adds to the existing log file when the program restarts. This default lets you keep a history of the program's actions. If you don't want this history or want to save space, set **logreuse**=y. The Q Apply program clears the log file when it starts, then writes to the blank file.

The log is stored by default in the directory where the Q Apply program is started, or in a different location that you set using the **apply_path** parameter.

z/OS The log file name is *apply_server*.*apply_schema*.QAPP.log. For example, SAMPLE.ASN.APP.log. Also, if **apply_path** is specified with slashes (//) to use a High Level Qualifier (HLQ), the file naming conventions of z/OS sequential data set files apply, and **apply_schema** is truncated to eight characters.

Linux UNIX Windows The log file name is *db2instance*.*apply_server*.*apply_schema*.QAPP.log. For example, DB2.SAMPLE.ASN.QAPP.log.

logstdout

Default: logstdout=n

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

By default, a Q Apply program writes its work history only to the log. You can change the **logstdout** parameter if you want to see program history on the standard output (stdout) in addition to the log.

Error messages and some log messages (initialization, stop, subscription activation, and subscription deactivation) go to both the standard output and the log file regardless of the setting for this parameter.

You can specify the **logstdout** parameter when you start a Q Apply program with the **asnqapp** command. If you use the Replication Center to start a Q Apply program, this parameter is not applicable.

max_parallel_loads

Default: max_parallel_loads=1 (z/OS); 15 (Linux, UNIX, Windows)

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **max_parallel_loads** parameter specifies the maximum number of automatic load operations of target tables that Q Apply can start at the same time for a given receive queue. The default for **max_parallel_loads** differs depending on the platform of the target server:

z/OS

On z/OS the default is one load at a time because of potential issues with the DSNUTILS stored procedure that Q Apply uses to call the DB2 LOAD utility. Depending on your environment you can experiment with values higher than **max_parallel_loads**=1. If errors occur, reset the value to 1.

Linux UNIX Windows

On Linux, UNIX, and Windows the default is 15 parallel loads.

monitor_interval

Default: **`monitor_interval`**=60000 milliseconds (1 minute) on z/OS; 30000 milliseconds (30 seconds) on Linux, UNIX, and Windows

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **`monitor_interval`** parameter tells a Q Apply program how often to insert performance statistics into the IBMQREP_APPLYMON table. You can view these statistics by using the Q Apply Throughput and Latency windows.

You can adjust the monitor_interval based on your needs:

If you want to monitor a Q Apply program's activity at a more granular level, shorten the monitor interval
For example, you might want to see the statistics for the number of messages on queues broken down by each 10 seconds rather than one-minute intervals.

Lengthen the monitor interval to view Q Apply performance statistics over longer periods
For example, if you view latency statistics for a large number of one-minute periods, you might want to average the results to get a broader view of performance. Seeing the results averaged for each half hour or hour might be more useful in your replication environment.

Important for Q Replication Dashboard users: When possible, you should synchronize the Q Apply **`monitor_interval`** parameter with the dashboard refresh interval (how often the dashboard retrieves performance information from the Q Capture and Q Apply monitor tables). The default refresh interval for the dashboard is 10 seconds (10000 milliseconds). If the value of **`monitor_interval`** is higher than the dashboard refresh interval, the dashboard refreshes when no new monitor data is available.

monitor_limit

Default: **`monitor_limit`**=10080 minutes (7 days)

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **`monitor_limit`** parameter specifies how old the rows must be in the IBMQREP_APPLYMON table before the rows are eligible for pruning.

By default, rows that are older than 10080 minutes (7 days) are pruned. The IBMQREP_APPLYMON table provides statistics about a Q Apply program's activity. A row is inserted at each monitor interval. You can adjust the monitor limit based on your needs:

Increase the monitor limit to keep statistics
If you want to keep records of the Q Apply program's activity beyond one week, set a higher monitor limit.

Lower the monitor limit if you look at statistics frequently.
If you monitor the Q Apply program's activity on a regular basis, you probably do not need to keep one week of statistics and can set a lower monitor limit.

You can set the **monitor_limit** parameter when you start the Q Apply program or while the program is running. You can also change its saved value in the IBMQREP_APPLYPARMS table.

nickname_commit_ct

Default: **nickname_commit_ct**=10

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

Federated targets: The **nickname_commit_ct** parameter specifies the number of rows after which the DB2 IMPORT utility commits changes to nicknames that reference a federated target table during the loading process. This parameter applies only to automatic loads for federated targets that use the IMPORT utility.

By default, Q Apply specifies that the IMPORT utility commits changes every 10 rows during the federated loading process. You might see improved load performance by raising the value of **nickname_commit_ct**. For example, a setting of **nickname_commit_ct**=100 would lower the CPU overhead by reducing interim commits. However, more frequent commits protect against problems that might occur during the load, enabling the IMPORT utility to roll back a smaller number of rows if a problem occurs.

The **nickname_commit_ct** parameter is a tuning parameter used to improve DB2 IMPORT performance by reducing the number of commits for federated targets.

p2p_2nodes

Default: **p2p_2nodes**=n

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **p2p_2nodes** parameter allows the Q Apply program to optimize for performance in a peer-to-peer configuration with only two active servers by not logging conflicting deletes in the IBMQREP_DELTOMB table. Only use the setting **p2p_2nodes**=y for peer-to-peer replication with two active servers.

By default, the Q Apply program records conflicting DELETE operations in the IBMQREP_DELTOMB table. With **p2p_2nodes**=y the Q Apply program does not use the IBMQREP_DELTOMB table. This avoids any unnecessary contention on the table or slowing of Q Apply without affecting the program's ability to correctly detect conflicts and ensure data convergence.

Important: The Q Apply program does not automatically detect whether a peer-to-peer configuration has only two active servers. Ensure that the option **p2p_2nodes**=y is used only for a two-server peer-to-peer configuration. Using the option for configurations with more than two active servers might result in incorrect conflict detection and data divergence.

prune_interval

Default: **prune_interval**=300 seconds (5 minutes)

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **prune_interval** parameter determines how often a Q Apply program looks for old rows to delete from the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables. By default, a Q Apply program looks for rows to prune every 300 seconds (5 minutes).

Your pruning frequency depends on how quickly these control tables grow, and what you intend to use them for:

Shorten the prune interval to manage monitor tables
A shorter prune interval might be necessary if the IBMQREP_APPLYMON table is growing too quickly because of a shortened monitor interval. If this table is not pruned often enough, it can exceed its table space limit, which forces a Q Apply program to stop. However, if the table is pruned too often or during peak times, pruning can interfere with application programs that run on the same system.

Lengthen the prune interval for record keeping
You might want to keep a longer history of a Q Apply program's performance by pruning the IBMQREP_APPLYTRACE and IBMQREP_APPLYMON tables less frequently.

The prune interval works in conjunction with the **trace_limit** and **monitor_limit** parameters, which determine when data is old enough to prune. For example, if the **prune_interval** is 300 seconds and the **trace_limit** is 10080 seconds, a Q Apply program will try to prune every 300 seconds. If the Q Apply program finds any rows in the IBMQREP_APPLYTRACE table that are older than 10080 minutes (7 days), it prunes them.

The **prune_interval** parameter does not affect pruning of the IBMQREP_DONEMSG table. Pruning of this table is controlled by the **prune_method** and **prune_batch_size** parameters.

z/OS

prune_method

Default: **prune_method**=2

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **prune_method** parameter specifies the method that the Q Apply program uses to delete unneeded rows from the IBMQREP_DONEMSG table. By default, (**prune_method**=2), Q Apply prunes groups of rows based on the **prune_batch_size** value. A separate prune thread records which messages were applied, and then issues a single range-based DELETE.

When you specify **prune_method**=1, Q Apply prunes rows from the IBMQREP_DONESG table one at a time. First Q Apply queries the table to see if data from a message was applied, then it deletes the message from the receive queue, and then prunes the corresponding row from IBMQREP_DONEMSG by issuing an individual SQL statement.

pwdfile

Default: `pwdfile`=*apply_path*/asnpwd.aut

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **pwdfile** parameter specifies the name of the encrypted password file that the Q Apply program uses to connect to the Q Capture server. This connection is required only when a Q subscription specifies an automatic load that uses the EXPORT utility. When you use the **asnpwd** command to create the password file, the default file name is `asnpwd.aut`. If you create the password file with a different name or change the name, you must change the **pwdfile** parameter to match. The Q Apply program looks for the password file in the directory specified by the **apply_path** parameter.

z/OS No password file is required.

You can set the **pwdfile** parameter when you start the Q Apply program, and you can change its saved value in the IBMQREP_APPLYPARMS table. You cannot change the value while the Q Apply program is running.

report_exception

Default: **report_exception**=y

Methods of changing: When Q Apply starts

The **report_exception** parameter controls whether the Q Apply program inserts data into the IBMQREP_EXCEPTIONS table when a conflict or SQL error occurs at the target table but the row is applied to the target anyway because the conflict action that was specified for the Q subscription was F (force). By default, (**report_exception**=y), Q Apply inserts details in the IBMQREP_EXCEPTIONS table for each row that causes a conflict or SQL error at the target, regardless of whether the row was applied or not. You can specify **report_exception**=n and Q Apply will not insert data into the IBMQREP_EXCEPTIONS table when a row causes a conflict but is applied. With **report_exception**=n, Q Apply continues to insert data about rows that were not applied.

When **report_exception**=n, the Q Apply program also tolerates codepage conversion errors when writing SQL text into the IBMQREP_EXCEPTIONS table and continues normal processing.

richklvl

Default: **richklvl**=2

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **richklvl** parameter specifies the level of referential integrity checking. By default (**richklvl**=2), the Q Apply program checks for RI-based dependencies between transactions to ensure that dependent rows are applied in the correct order.

If you specify **richklvl**=5, Q Apply checks for RI-based dependencies when a key value is updated in the parent table, a row is updated in the parent table, or a row is deleted from the parent table.

A value of 0 prompts Q Apply to not check for RI-based dependencies.

When a transaction cannot be applied because of a referential integrity violation, the Q Apply program automatically retries the transaction until it is applied in the same order that it was committed at the source table.

spill_commit_count

Default: **`spill_commit_count`**=10

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **`spill_commit_count`** parameter specifies how many rows are grouped together in a commit scope by the Q Apply spill agents that apply data that was replicated during a load operation. Increasing the number of rows that are applied before a COMMIT is issued can improve performance by reducing the I/O resources that are associated with frequent commits. Balance the potential for improvement with the possibility that fewer commits might cause lock contention at the target table and the IBMQREP_SPILLEDROW control table.

skiptrans

Default: None

Method of changing: When Q Apply starts

The **`skiptrans`** parameter specifies that the Q Apply program should not apply one or more transactions from one or more receive queues based on their transaction ID.

Stopping the program from applying transactions is useful in unplanned situations, for example:

- Q Apply receives an error while applying a row of a transaction and either stops or stops reading from the receive queue. On startup, you might want Q Apply to ignore the entire transaction in error.
- After the failover from a disaster recovery situation, you might want to ignore a range of transactions on the receive queue from the failover node to the fallback node.

You can also prompt the Q Capture program to ignore transactions. This action would be more typical when you can plan which transactions do not need to be replicated.

Note: Ignoring a transaction that was committed at the source server typically causes divergence between tables at the source and target. You might need use the **`asntdiff`** and **`asntrep`** utilities to synchronize the tables.

startallq

Default: **`startallq`**=n (z/OS); y (Linux, UNIX, Windows)

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **`startallq`** parameter specifies how Q Apply processes receive queues when it starts. With **`startallq`**=y, Q Apply puts all receive queues in active state and begins reading from them when it starts. When you specify **startallq**=n, Q Apply processes only the active receive queues when it starts.

You can use **startallq**=y to avoid having to issue the **startq** command for inactive receive queues after the Q Apply program starts. You can use **startallq**=n to keep disabled queues inactive when you start Q Apply.

term

Default: term=y

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **term** parameter controls whether a Q Apply program keeps running when DB2 or the queue manager are unavailable.

By default (**term**=y), the Q Apply program terminates when DB2 or the queue manager are unavailable. You can change the default (**term**=n) if you want a Q Apply program to keep running while DB2 or the queue manager are unavailable. When DB2 or the queue manager are available, Q Apply begins applying transactions where it left off without requiring you to restart the program.

Note: Regardless of the setting for **term**, if the WebSphere MQ sender or receiver channels stop, the Q Apply program keeps running because it cannot detect channel status. This situation causes replication to stop because the two queue managers cannot communicate. If you find that replication has stopped without any messages from the Q replication programs, check for WebSphere MQ errors. For example, check the channel status by using the WebSphere MQ DISPLAY CHSTATUS command.

trace_ddl

Default: trace_ddl=n

Methods of changing: When Q Apply starts; IBMQREP_APPLYPARMS table

The **trace_ddl** parameter specifies whether, when DDL operations at the source database are replicated, the SQL text of the operation that the Q Apply program performs at the target database is logged. By default (**trace_ddl**=n), Q Apply does not log the SQL text. If you specify **trace_ddl**=y, Q Apply issues an ASN message to the Q Apply log file, standard output, and IBMQREP_APPLYTRACE table with the text of the SQL statement. The SQL text is truncated to 1024 characters.

trace_limit

Default: trace_limit=10080 minutes (7 days)

Methods of changing: When Q Apply starts; while Q Apply is running; IBMQREP_APPLYPARMS table

The **trace_limit** parameter specifies how long rows remain in the IBMQREP_APPLYTRACE table before the rows can be pruned.

The Q Apply program inserts all informational, warning, and error messages into the IBMQREP_APPLYTRACE table. By default, rows that are older than 10080 minutes (7 days) are pruned at each pruning interval. Modify the trace limit depending on your need for audit information.

Examples of asnqapp usage

These examples illustrate how to use the **asnqcap** command.

Specifying a path to work files and password file

To start a Q Apply program on a server named targetdb, with a schema of alpha, with work files located in the `/home/files/qapply` directory, and using a password file called `pass1.txt`:

```
asnqapp apply_server=targetdb apply_schema="alpha"
apply_path="/home/files/qapply" pwdfile="pass1.txt"
```

Shortening the monitor interval to match Q Replication Dashboard

To start a Q Apply program and shorten the interval for inserting rows into the IBMQREP_APPLYMON table from the default of 60000 milliseconds (60 seconds) on z/OS to 10000 milliseconds (10 seconds) to match the default refresh interval for the Q Replication Dashboard:

```
asnqapp apply_server=targetdb apply_schema="alpha" monitor_interval=10000
```

Prompting Q Apply to stop after all queues have been emptied

To start a Q Apply program and instruct it to stop running after all queues have been emptied once:

```
asnqapp apply_server=targetdb apply_schema="alpha" autostop=y
```

Stopping Q Apply with at a specified timestamp

To start a Q Apply program and instruct it to stop running after it processes transactions that were committed at the source up to a specified timestamp:

```
asnqapp apply_server=targetdb apply_schema="alpha" applyupto="2011-04-02-20"
```

This example uses the partial timestamp format to stop the Q Apply program when it reads a transaction with a timestamp after 8 p.m. on April 2, 2011 Greenwich mean time.

Starting Q Apply and instructing the program when to stop

To start a Q Apply program and instruct it to stop running after it reads a transaction with a timestamp that matches the time that you started Q Apply, or after the receive queue is empty (whichever occurs first):

```
asnqapp apply_server=targetdb apply_schema="alpha"
applyupto=CURRENT_TIMESTAMP,NOWAIT
```

Prompting Q Apply to skip specified transactions

To start a Q Apply program and specify that it not apply one transaction on receive queue Q1 and a range of transactions on receive queue Q2:

```
asnqapp apply_server=targetdb apply_schema=ASN
skiptrans="Q1;0000:0000:0000:0000:51a1,
Q2;0000:0000:0000:0000:51b0-0000:0000:0000:0000:51c0"
```

asnqacmd: Working with a running Q Apply program

Use **asnqacmd** on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS to send a command to a running Q Apply program. Run this command at an operating system prompt or in a shell script.

For information on using the MVS MODIFY command to send commands to a running Q Apply program on z/OS, see Working with running Q Replication and Event Publishing programs by using the MVS MODIFY command.

Syntax

```
►►─asnqacmd─apply_server=db_name─┬──────────────────────┬─────────────────►
                                 └─apply_schema=schema──┘

►─┬─chgparms─┤ parameters ├──────────────────────────────┬───────────────►◄
  ├─loaddonesub=receive_queue:sub_name───────────────────┤
  ├─prune────────────────────────────────────────────────┤
  ├─qryparms─────────────────────────────────────────────┤
  ├─status─┬──────────────┬──────────────────────────────┤
  │        └─show details─┘                              │
  ├─stop─────────────────────────────────────────────────┤
  ├─stopq=receive_queue──────────────────────────────────┤
  ├─startq=receive_queue─────────────────────────────────┤
  ├─startq="receive_queue;skiptrans=┤ skiptrans-clause ├"┤
  ├─reinitq=receive_queue────────────────────────────────┤
  ├─resumesub=receive_queue:sub_name─────────────────────┤
  └─spillsub=receive_queue:sub_name──────────────────────┘
```

Parameters:

```
├─┬──────────────────┬─┬────────────────────┬─┬──────────────────┬─────►
  │           ┌─y─┐  │ └─deadlock_retries=n─┘ │           ┌─n─┐  │
  └─autostop=─┴─n─┴──┘                        └─logreuse=─┴─y─┴──┘

►─┬───────────────────┬─┬────────────────────┬─┬─────────────────┬─────►
  │            ┌─n─┐  │ └─monitor_interval=n─┘ └─monitor_limit=n─┘
  └─logstdout=─┴─y─┴──┘

►─┬──────────────────┬─┬──────────────┬─┬───────────────┬──────────────┤
  └─prune_interval=n─┘ │       ┌─y─┐  │ └─trace_limit=n─┘
                       └─term=─┴─n─┴──┘
```

skiptrans-clause::

```
├─┬────────────────────────────────────────┬───────────────────────────┤
  ├─transaction_ID─────────────────────────┤
  └─begin_transaction_ID-end_transaction_ID┘
```

Parameters

Table 42 defines the invocation parameters for the **asnqacmd** command.

Table 42. Definitions for the asnqacmd invocation parameters

Parameter	Definition
apply_server=*db_name*	Specifies the name of the database or subsystem that contains the Q Apply control tables. z/OS Specifies the name of the DB2 subsystem where the Q Apply program will run. For data sharing, use the group attach name to run Q Apply in any LPAR without changing the JCL for the started task. Linux UNIX Windows If you do not specify a Q Apply server, this parameter defaults to the value from the DB2DBDFT environment variable.
apply_schema=*schema*	Specifies a name that identifies the running Q Apply program that you want to work with.
chgparms	Specify to change one or more of the following operational parameters of the Q Apply program while it is running: • **autostop** • **deadlock_retries** • • • **logreuse** • **logstdout** • **monitor_interval** • **monitor_limit** • **prune_interval** • **term** • **trace_limit** **Important:** The parameters that you are changing must immediately follow the **chgparms** parameter. You can specify multiple parameters when you specify **chgparms**, and you can change these parameter values as often as you wish. The changes temporarily override the values in the IBMQREP_APPLYPARMS table, but they are not written to the table. When you stop and restart the Q Apply program, it uses the values in IBMQREP_APPLYPARMS."Descriptions of asnqapp parameters" on page 391 includes details about the parameters that you can override with this command.

Table 42. Definitions for the asnqacmd invocation parameters (continued)

Parameter	Definition
loaddonesub= *receive_queue*:*sub_name*	Specify to inform the Q Apply program that a manual load of the target table for the Q subscription is done. Use this parameter only if a manual load is specified for the Q subscription (HAS_LOADPHASE column in the IBMQREP_SUBS table has a value of E). When Q Apply receives the command, it starts processing messages in the spill queue for the Q subscription. When all spilled messages are processed, Q Apply changes the Q subscription state to A (active). **Bidirectional or peer-to-peer replication:** Issue this command for the same Q subscription that you started to begin the group activation process. This Q subscription is also the one whose source table was used as the load source.
prune	Specify to instruct the Q Apply program to prune the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables once. This pruning is in addition to any regularly scheduled pruning as specified by the **prune_interval** parameter.
qryparms	Specify if you want the current operational parameter values for the Q Apply program written to the standard output (stdout).
status	Specify to see a message about the state of each Q Apply thread (main, housekeeping, monitor, browsers, and agents).

Table 42. Definitions for the asnqacmd invocation parameters (continued)

Parameter	Definition
show details	Specify after the **status** parameter to view a more detailed report about Q Apply program status, with the following information about the Q Apply instance: • Whether the Q Apply program is running • Time since the program started • Location of the Q Apply diagnostic log • Number of active Q subscriptions • Time period used to calculate averages The following information is also displayed for each active receive queue: • Queue name • Number of active Q subscriptions • All transactions applied as of (time) (OLDEST_TRANS) • Restart point for Q Capture (MAXCMTSEQ) • All transactions applied as of (LSN) • Oldest in-progress transaction (OLDEST_INFLT_TRANS) • Average end-to-end latency • Average Q Capture latency • Average queue latency • Average Q Apply latency • Amount of memory in bytes that the browser thread used for reading transactions from the queue • Number of messages on the queue (queue depth) • Percent fullness of queue • Which agent threads are processing transactions • Which agent threads are waiting for transactions • Which agents are processing internal messages • Which agents are initializing
stop	Specify to stop the Q Apply program in an orderly way and commit the messages that it has processed up to that point.
stopq=*receive_queue*	Specify to instruct the Q Apply program to stop processing messages for a receive queue. All in-memory transactions are processed.
startq=*receive_queue*	Specify to instruct the Q Apply program to start processing messages on a receive queue.
startq= "*receive_queue*; **skiptrans**= *transaction_ID*"	Specify to instruct the Q Apply program to not apply one or a range of transactions from a receive queue when you start message processing on the receive queue. For details on how to specify the transaction identifier or range of identifiers, see the **skiptrans** parameter section in "asnqapp: Starting a Q Apply program" on page 389.
reinitq=*receive_queue*	Specify to have the Q Apply program update the attributes for NUM_APPLY_AGENTS and MEMORY_LIMIT from the IBMQREP_RECVQUEUES table for all Q subscriptions that use a particular receive queue. This command works only if the Q Apply program is reading from the receive queue that is named in the IBMQREP_RECVQUEUES table when you issue the command.

Table 42. Definitions for the asnqacmd invocation parameters (continued)

Parameter	Definition
resumesub= *recv_queue*:*sub_name*	Specify to resume applying spilled rows to the target table. Specify the name of the receive queue and Q subscription that identify the target table. The **spillsub** parameter is used to put the Q subscription in a spilling state (S) and the **resumesub** parameter is used to resume normal operations. Spill agents apply the rows from the temporary spill queue until the queue is emptied. While the spill agents are processing rows, incoming rows are added to the spill queue. When the spill queue is emptied, the state of the Q subscription is set to active (A) and normal operation resumes.
spillsub= *recv_queue*:*sub_name*	Specify to have all row changes for a Q subscription redirected to a temporary spill queue. You can perform maintenance on the target table such as reorganizing the table. Specify the name of the receive queue and Q subscription that identify the target table. The temporary spill queue is created based on the definition of your model queue. Ensure that the maximum depth of the queue is large enough to hold the spilled rows until the rows can be applied after resuming operations.

Example 1

To update all Q subscriptions that use a receive queue named Q1 with the latest values for number of Q Apply agents and memory limit from the IBMQREP_RECVQUEUES table:

```
asnqacmd apply_server=targetdb apply_schema="alpha" reinitq=Q1
```

Example 2

To instruct a running Q Apply program to stop after all queues are emptied once, and to shorten the monitor interval and the trace limit:

```
asnqacmd apply_server=targetdb apply_schema="alpha" chgparms autostop=y
monitor_interval=60000 trace_limit=5000
```

Example 3

To receive messages about the state of each Q Apply thread:

```
asnqacmd apply_server=targetdb apply_schema="alpha" status
```

Example 4

To receive detailed messages about the Q Apply program and active receive queues:

```
asnqacmd apply_server=targetdb apply_schema="alpha" status show details
```

Example 5

To prune the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables once:

```
asnqacmd apply_server=targetdb apply_schema="alpha" prune
```

Example 6

To place a Q subscription in spill mode for performing maintenance on the target table:

```
 asnqacmd apply_server=targetdb apply_schema="BSN"
    spillsub="BSN.QM1_TO_QM2.recvq:EmployeeSub"
```

When you are finished maintaining the target table, use the **resumesub** parameter to resume normal operation.

Example 7

To prompt the Q Apply program to skip a range of transactions when it starts processing receive queue Q2:

```
asnqacmd apply_server=targetdb apply_schema=ASN
   startq="Q2;skiptrans=0000:0000:0000:0000:51a1-0000:0000:0000:0000:51a8"
```

asnpwd: Creating and maintaining password files

Use the **asnpwd** command to create and change password files on Linux, UNIX, and Windows. Run this command at the command line or in a shell script.

Command help appears if you enter the **asnpwd** command without any parameters, followed by a ?, or followed by incorrect parameters.

Syntax

```
►►──asnpwd──┬─init──┤ Init parameters ├────┬──────────────────────────►◄
            ├─add──┤ Add parameters ├──────┤
            ├─modify──┤ Modify parameters ├┤
            ├─delete──┤ Delete parameters ├┤
            └─list──┤ List parameters ├────┘
```

Init parameters:

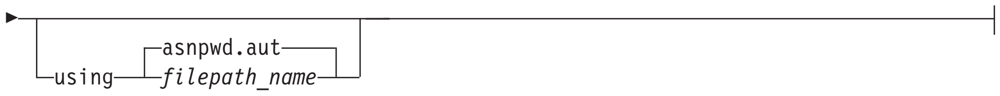
```
├──┬──────────────────────────┬──┬──────────────────────────────┬──────────────┤
   └─encrypt──┬─all──────┬─┘  └─using──┬─asnpwd.aut──────┬─┘
              └─password─┘             └─filepath_name───┘
```

Add parameters:

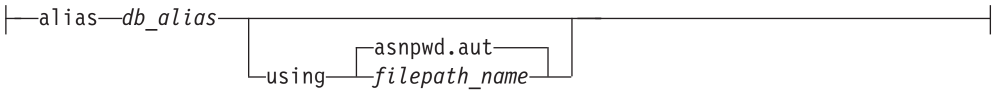
```
├──alias──db_alias──id──userid──password──password──────────────────────────►

►──┬──────────────────────────────┬─────────────────────────────────────────┤
   └─using──┬─asnpwd.aut──────┬─┘
            └─filepath_name───┘
```

Modify parameters:

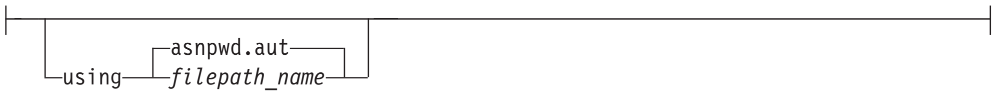
```
├──alias──db_alias──id──userid──password──password──────────────────────────►
```



Delete parameters:



List parameters:



Parameters

Table 43 defines the invocation parameters for the **asnpwd** command.

Important note about compatibility of password files: Password files that are created by the **asnpwd** command starting with Version 9.5 Fix Pack 2 use a new encryption method and cannot be read by older versions of the replication programs and utilities. If you share a password file among programs and utilities that are at mixed level, with some older than these fix packs, do not recreate the password file by using an **asnpwd** utility that is at these fix packs or newer. Replication programs and utilities at these fix packs or newer can continue to work with older password files. Also, you cannot change an older password file to use the new encryption method; you must create a new password file.

Usage note: On 64-bit Windows operating systems, the ADD, MODIFY, DELETE, and LIST options are not supported for password files that were created by using the **asnpwd** command before Version 9.5 Fix Pack 2.

Table 43. asnpwd invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
init	Specify to create an empty password file. This command will fail if you specify the **init** parameter with a password file that already exists.
add	Specify to add an entry to the password file. There can only be one entry in the password file per *db_alias*. This command will fail if you specify the **add** parameter with an entry that already exists in the password file. Use the **modify** parameter to change an existing entry in the password file.
modify	Specify to modify the password or user ID for an entry in the password file.
delete	Specify to delete an entry from the password file.
list	Specify to list the aliases and user ID entries in a password file. This parameter can be used only if the password file was created by using the **encrypt password** parameter. Passwords are never displayed by the **list** command.

Table 43. asnpwd invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
encrypt	Specifies which entries in a file to encrypt. **all (default)** Encrypt all entries in the specified file such that you cannot list the database aliases, user names, and passwords that are in the file. This option reduces the exposure of information in password files. **password** Encrypt the password entry in the specified file. This option allows users to list the database aliases and user names stored in their password file. Passwords can never be displayed.
using *filepath*	Specifies the path and name of the password file. Follow the file naming conventions of your operating system. An example of a valid password file on Windows is `C:\sqllib\mypwd.aut`. If you specify the path and name of the password file, the path and the password file must already exist. If you are using the **init** parameter and you specify the path and name of the password file, the path must already exist and the command will create the password file for you. If you do not specify this parameter, the default file name is `asnpwd.aut` and the default file path is the current directory.
alias *db_alias*	Specifies the alias of the database to which the user ID has access. The alias is always folded to uppercase, regardless of how it is entered.
id *userid*	Specifies the user ID that has access to the database.
password *password*	Specifies the password for the specified user ID. This password is case sensitive and is encrypted in the password file.

Return Codes

The **asnpwd** command returns a zero return code upon successful completion. A nonzero return code is returned if the command is unsuccessful.

Examples for asnpwd

The following examples illustrate how to use the **asnpwd** command.

Example 1

To create a password file with the default name of `asnpwd.aut` in the current directory:

```
asnpwd INIT
```

Example 2

To create a password file named `pass1.aut` in the `c:\myfiles` directory:

```
asnpwd INIT USING c:\myfiles\pass1.aut
```

Example 3

To create a password file named `mypwd.aut` with the **`encrypt all`** parameter:

```
asnpwd INIT ENCRYPT ALL USING mypwd.aut
```

Example 4

To create a password file named `mypwd.aut` with the **`encrypt password`** parameter:

```
asnpwd INIT ENCRYPT PASSWORD USING mypwd.aut
```

Example 5

To create a default password file with the **`encrypt password`** parameter:

```
asnpwd INIT ENCRYPT PASSWORD
```

Example 6

To add a user ID called oneuser and its password to the password file named `pass1.aut` in the `c:\myfiles` directory and to grant this user ID access to the db1 database:

```
asnpwd ADD ALIAS db1 ID oneuser PASSWORD mypwd using c:\myfiles\pass1.aut
```

Example 7

To modify the user ID or password of an entry in the password file named `pass1.aut` in the `c:\myfiles` directory:

```
asnpwd MODIFY AliaS sample ID chglocalid PASSWORD chgmajorpwd
  USING c:\myfiles\pass1.aut
```

Example 8

To delete the database alias called sample from the password file named `pass1.aut` in the `c:\myfiles` directory:

```
asnpwd delete alias sample USING c:\myfiles\pass1.aut
```

Example 9

To see command help:

```
asnpwd
```

Example 10

To list the entries in a default password file:

```
asnpwd LIST
```

Example 11

To list the entries in a password file named `pass1.aut`:

```
asnpwd LIST USING pass1.aut
```

The output from this command depends on how the password file was initialized:

- If it was initialized by using the **`encrypt all`** parameter, the following message is issued:

```
ASN1986E "Asnpwd" : "". The password file "pass1.aut" contains
encrypted information that cannot be listed.
```

- If it was not initialized by using the **encrypt all** parameter, the following details are listed:

```
asnpwd LIST USING pass1.aut
Alias: SAMPLE  ID: chglocalid
Number of Entries: 1
```

asnscrt: Creating a replication service

Use the **asnscrt** command to create a replication service in the Windows Service Control Manager (SCM) and invoke the **asnqcap**, **asnqapp**, **asnmon**, **asncap**, and **asnapply** commands. Run the **asnscrt** command on the Windows operating system.

Syntax

```
►►──asnscrt──┬─-QC─┬──db2_instance──account──password──┬─asnqcap_command──┬──────►◄
             ├─-QA─┤                                    ├─asnqapp_command──┤
             ├─-M──┤                                    ├─asnmon_command───┤
             ├─-C──┤                                    ├─asncap_command───┤
             └─-A──┘                                    └─asnapply_command─┘
```

Parameters

Table 44 defines the invocation parameters for the **asnscrt** command.

Table 44. asnscrt invocation parameter definitions for Windows operating systems

Parameter	Definition
-QC	Specifies that you are starting a Q Capture program.
-QA	Specifies that you are starting a Q Apply program.
-M	Specifies that you are starting a Replication Alert Monitor program.
-C	Specifies that you are starting a Capture program.
-A	Specifies that you are starting an Apply program.
db2_instance	Specifies the DB2 instance used to identify a unique DB2 replication service. The DB2 instance can be a maximum of eight characters.
account	Specifies the account name that you use to log on to Windows. If the account is local it must begin with a period and a backslash (.\). Otherwise the domain or machine name must be specified (for example, domain_name\account_name).
password	Specifies the password used with the account name. If the password contains special characters, type a backslash (\) before each special character.

Table 44. asnscrt invocation parameter definitions for Windows operating systems (continued)

Parameter	Definition
asnqcap_command	Specifies the complete **asnqcap** command to start a Q capture program. Use the documented **asnqcap** command syntax with the appropriate **asnqcap** parameters. If the DB2PATH environment variable is not defined, you must specify a location for the work files by including the **capture_path** parameter with the **asnqcap** command. If the DB2PATH variable is defined and you specify a **capture_path**, the **capture_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnqcap** parameters that you enter.
asnqapp_command	Specifies the complete **asnqapp** command to start a Q apply program. Use the documented **asnqapp** command syntax with the appropriate **asnqapp** parameters. If the DB2PATH environment variable is not defined, you must specify the location for the work files by including the **apply_path** parameter with the **asnqapp** command. If the DB2PATH variable is defined and you specify an **apply_path**, the **apply_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnqapp** parameters that you enter.
asnmon_command	Specifies the complete **asnmon** command to start a Replication Alert Monitor program. Use the documented **asnmon** command syntax with the appropriate **asnmon** parameters. If the DB2PATH environment variable is not defined, you must specify a location for the log files by including the **monitor_path** parameter with the **asnmon** command. If the DB2PATH variable is defined and you specify a **monitor_path**, the **monitor_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnmon** parameters that you enter.
asncap_command	Specifies the complete **asncap** command to start a Capture program. Use the documented **asncap** command syntax with the appropriate **asncap** parameters. If the DB2PATH environment variable is not defined, you must specify a location for the work files by including the **capture_path** parameter with the **asncap** command. If the DB2PATH variable is defined and you specify a **capture_path**, the **capture_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asncap** parameters that you enter.

Table 44. asnscrt invocation parameter definitions for Windows operating systems (continued)

Parameter	Definition
asnapply_command	Specifies the complete **asnapply** command to start an Apply program. Use the documented **asnapply** command syntax with the appropriate **asnapply** parameters. If the DB2PATH environment variable is not defined, you must specify the location for the work files by including the **apply_path** parameter with the **asnapply** command. If the DB2PATH variable is defined and you specify an **apply_path**, the **apply_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnapply** parameters that you enter.

Examples for asnscrt

The following examples illustrate how to use the **asnscrt** command.

Example 1

To create a DB2 replication service that invokes a Q Apply program under a DB2 instance called inst2 and uses a logon account of .\joesmith and a password of my$pwd:

```
asnscrt -QA inst2 .\joesmith my\$pwd asnqapp apply_server=mydb2 apply_schema =as2
  apply_path=X:\sqllib
```

Example 2

To create a DB2 replication service that invokes a Capture program under a DB2 instance called inst1:

```
asnscrt -C inst1 .\joesmith password asncap capture_server=sampledb
  capture_schema=ASN capture_path=X:\logfiles
```

Example 3

To create a DB2 replication service that invokes an Apply program under a DB2 instance called inst2 and uses a logon account of .\joesmith and a password of my$pwd:

```
asnscrt -A inst2 .\joesmith my\$pwd asnapply control_server=db2 apply_qual=aq2
  apply_path=X:\sqllib
```

Example 4

To create a DB2 replication service that invokes a Replication Alert Monitor program under a DB2 instance called inst3:

```
asnscrt -M inst3 .\joesmith password asnmon monitor_server=db3 monitor_qual=mq3
  monitor_path=X:\logfiles
```

Example 5

To create a DB2 replication service that invokes a Capture program under a DB2 instance called inst4 and overrides the default work file directory with a fully qualified **capture_path**:

```
asnscrt -C inst4 .\joesmith password X:\sqllib\bin\asncap capture_server=scdb
  capture_schema=ASN capture_path=X:\logfiles
```

Example 6

To create a DB2 replication service that invokes a Q capture program under a DB2 instance called inst1:

```
asnscrt -QC inst1 .\joesmith password asnqcap capture_server=mydb1
  capture_schema=QC1 capture_path=X:\logfiles
```

asnsdrop: Dropping a replication service

Use the **asnsdrop** command to drop replication services from the Windows Service Control Manager (SCM) on the Windows operating system.

Syntax

```
►►──asnsdrop──┬─service_name─┬──────────────────────────────────────────────►◄
              └─ALL──────────┘
```

Parameters

Table 45 defines the invocation parameters for the **asnsdrop** command.

Table 45. asnsdrop invocation parameter definitions for Windows operating systems

Parameter	Definition
service_name	Specifies the fully qualified name of the DB2 replication service. Enter the Windows SCM to obtain the DB2 replication service name. On Windows operating systems, you can obtain the service name by opening the Properties window of the DB2 replication service. If the DB2 replication service name contains spaces, enclose the entire service name in double quotation marks.
ALL	Specifies that you want to drop all DB2 replication services.

Examples for asnsdrop

The following examples illustrate how to use the **asnsdrop** command.

Example 1

To drop a DB2 replication service:

```
asnsdrop DB2.SAMPLEDB.SAMPLEDB.CAP.ASN
```

Example 2

To drop a DB2 replication service with a schema named A S N (with embedded blanks), use double quotation marks around the service name:

```
asnsdrop "DB2.SAMPLEDB.SAMPLEDB.CAP.A S N"
```

Example 3

To drop all DB2 replication services:

```
asnsdrop ALL
```

asnslist: Listing replication services

Use the **asnslist** command to list replication services in the Windows Service Control Manager (SCM). You can optionally use the command to list details about each service. Run the **asnslist** command on the Windows operating system.

Syntax

```
►►─asnslist──┬─────────┬──────────────────────────────────────────────►◄
             └─DETAILS─┘
```

Parameters

Table 46 defines the invocation parameter for the **asnslist** command.

Table 46. asnslist invocation parameter definition for Windows operating systems

Parameter	Definition
details	Specifies that you want to list detailed data about all DB2 replication services on a system.

Examples for asnlist

The following examples illustrate how to use the **asnslist** command.

Example 1

To list the names of DB2 replication services on a system:

```
asnslist
```

Here is an example of the command output:

```
DB2.DB2.SAMPLE.QAPP.ASN
DB2.DB4.SAMPLE.QCAP.ASN
```

Example 2

To list details about all services on a system:

```
asnslist details
```

Here is an example of the command output:

```
DB2.DB2.SAMPLE.QAPP.ASN
Display Name: DB2 DB2 SAMPLE QAPPLY ASN
Image Path:   ASNSERV DB2.DB2.SAMPLE.APP.AQ1 -ASNQAPPLY QAPPLY_SERVER=SAMPLE AP
              PLY_SCHEMA=ASN QAPPLY_PATH=C:\PROGRA~1\SQLLIB
Dependency:   DB2-0

DB2.DB4.SAMPLE.QCAP.ASN
Display Name: DB2 DB4 SAMPLE QAPPLY ASN
Image Path:   ASNSERV DB2.DB4.SAMPLE.APP.AQ1 -ASNQCAP QCAPTURE_SERVER=SAMPLE CA
              PTURE_SCHEMA=ASN QCAPTURE_PATH=C:\PROGRA~1\SQLLIB
Dependency:   DB4-0
```

asntdiff: Comparing data in source and target tables (Linux, UNIX, Windows)

Use the **asntdiff** command to compare two relational tables and generate a list of differences between the two. Run the **asntdiff** command at an operating system prompt or in a shell script.

This topic describes usage on Linux, UNIX, or Windows. For details on running **asntdiff** on z/OS, see “asntdiff: Comparing data in source and target tables (z/OS)” on page 426. For information on the asntdiff –f command option, which enables you to compare tables whether or not they are involved in replication by using an input file, see “asntdiff –f (input file) command option” on page 433.

The tables that you compare can reside on DB2 for Linux, UNIX, Windows, DB2 for z/OS, or DB2 for System i.

Syntax

```
►►──asntdiff──DB=server──┬────────────────┬────────────────────────────────────►
                         └─SCHEMA=schema──┘

►──┬──────────────────────────────────────────┬──┬──────────────────────────────┬──►
   └─DIFF_SCHEMA=difference_table_schema──────┘  └─DIFF_TABLESPACE=tablespace───┘

►──┬─────────────────────┬──┬───────────────────────────────┬──WHERE=WHERE_clause──────►
   │             ┌─n─┐   │  └─MAXDIFF=difference_limit──────┘
   └─DIFF_DROP=──┴─y─┴───┘

►──┬──────────────────────┬──┬───────────────────┬──┬─────────────────────┬──────────►
   └─DIFF_PATH=log_path───┘  └─PWDFILE=filename──┘  └─DIFF=table_name─────┘

►──┬────────────────────────────────────────┬───────────────────────────────────►◄
   └─RANGECOL=──┤ range_clause_option ├─────┘
```

range_clause_option:

```
├──┬─src_colname─FROM:date-time_lower-bound─TO:date-time_upper-bound─┬──────────┤
   ├─src_colname─FROM:date-time──────────────────────────────────────┤
   └─src_colname─TO:date-time────────────────────────────────────────┘
```

Parameters

Table 47 defines the invocation parameters for the **asntdiff** command.

Table 47. asntdiff invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
DB=*server*	Specifies the DB2 alias of the database that stores information about the source and target tables to be compared. The value differs depending on whether you are using Q Replication or SQL Replication: **Q Replication** The name of the Q Capture server, which contains the IBMQREP_SUBS table. **SQL Replication** The name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table.
SCHEMA=*schema*	Specifies the schema of the Q Capture control tables for Q Replication, or the schema of the Apply control tables for SQL Replication. The default is ASN.
DIFF_SCHEMA=*difference_table_schema*	Specifies the schema of the difference table. The default is ASN.
DIFF_TABLESPACE=*tablespace*	Specifies the table space of the difference table. If this parameter is not specified, the table is created in the default table space in the database where the **asntdiff** command was run.
DIFF_DROP=y/n	Specifies whether an existing difference table will be dropped and recreated before it is reused to record differences. If the table does not exist, the **asntdiff** command creates it. **n (default)** The difference table will be used as is and the existing rows will be deleted. **y** The difference table will be dropped and recreated.
MAXDIFF=*difference_limit*	Specifies the maximum number of differences that you want the **asntdiff** command to process before it stops. The default value is 10000.

Table 47. asntdiff invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
WHERE=*WHERE_clause*	Specifies an SQL WHERE clause that uniquely identifies one row of the control table that stores information about the source and target tables that will be compared. The WHERE clause must be in double quotation marks. The value of this parameter differs depending on whether you are using Q Replication or SQL Replication: **Q Replication** The WHERE clause specifies a row in the IBMQREP_SUBS table and uses the SUBNAME column to identify the Q subscription that contains the source and target tables. **SQL Replication** The WHERE clause specifies a row in the IBMSNAP_SUBS_MEMBR table and uses the SET_NAME, APPLY_QUAL, TARGET_SCHEMA, and TARGET_TABLE columns to identify the subscription set member that contains the source and target tables.
DIFF_PATH=*log_path*	Specifies the location where you want the asntdiff command to write its log. The default value is the directory from which you ran the command. The value must be an absolute path name. Use double quotation marks ("") to preserve case.
PWDFILE=*filename*	Specifies the name of the password file that is used to connect to databases. If you do not specify a password file, the default value is `asnpwd.aut` (the name of the password file that is created by the **asnpwd** command). The **asntdiff** command searches for the password file in the directory that is specified by the DIFF_PATH parameter. If no value for the DIFF_PATH parameter is specified, the command searches for the password file in the directory where the command was run.
DIFF=*table_name*	Specifies the name of the table that is created in the source database to store differences between the source and target tables. The table has one row for each difference that is detected. If you do not include this parameter or the **DIFF_SCHEMA** parameter, the difference table is named ASN.ASNTDIFF.

Table 47. asntdiff invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition

RANGECOL clause	Specifies a range of rows from the source table that you want to compare. You provide the name of a DATE, TIME, or TIMESTAMP column in the source table, and then use one of three different clauses for specifying the range. The column name must be enclosed in single quotation marks. The clause must be enclosed in double quotation marks.
	
	The timestamp uses the following format: *YYYY-MM-DD-HH.MM.SS.mmmmm*. For example, 2010-03-10-10.35.30.55555 is the GMT timestamp for March 10, 2010, 10:35 AM, 30 seconds, and 55555 microseconds.
	
	Use one of the following clauses:
	
	src_colname **FROM:** *date-time_lower-bound* **TO:** *date-time_upper-bound*
	Specifies a lower and upper bound for the range of rows to compare.
	
	The following example uses a TIMESTAMP column:
	`"'SALETIME'`
	`FROM: 2008-02-08-03.00.00.00000`
	`TO: 2008-02-15-03.00.00.00000"`
	
	Remember: Both the **FROM:** and **TO:** keywords are required and both keywords must be followed by a colon (:).
	
	src_colname **FROM:** *date-time*
	Specifies that you want to compare all rows with timestamps that are greater than or equal to *date-time*.
	
	For example:
	`"'SALE_TIME'`
	`FROM: 2008-03-10-10.35.30.55555"`
	
	src_colname **TO:** *date-time*
	Specifies that you want to compare all rows with timestamps that are less than or equal to the *date-time*.
	
	For example:
	`"'SALETIME'`
	`TO: 2008-03-20-12.00.00.00000"`
	Recommendation: For better performance, ensure that you have an index on the source column that is specified in the range clause.When you compare tables that are involved in peer-to-peer replication, you can use the IBM-generated IBMQREPVERTIME column for the source column in the range clause.
	Restriction: The RANGECOL parameter is not valid for the **asntdiff -f** (input file) option. You can use a SQL WHERE clause in the input file to achieve similar results.

Examples for asntdiff

The following examples show how to use the **asntdiff** command.

Example 1

In Q Replication, to find the differences between a source and target table that are specified in a Q subscription named my_qsub, on a Q Capture server named source_db, with a Q Capture schema of asn:

```
asntdiff db=source_db schema=asn where="subname = 'my_qsub'"
```

Example 2

In SQL Replication, to find the differences between a source and target table that are specified in a subscription set called my_set, with a target table named trg_table, on an Apply control server named apply_db, with an Apply schema of asn, and to name the difference table diff_table:

```
asntdiff DB=apply_db schema=asn where="set_name = 'my_set'
 and target_table = 'trg_table'" diff=diff_table
```

Example 3

In Q Replication, to find the differences between a range of rows in the source and target tables that are specified in a peer-to-peer Q subscription named my_qsub, on a Q Capture server named source_db, with a Q Capture schema of asn:

```
asntdiff db=source_db schema=asn where="subname = 'my_qsub'"
 RANGECOL="'IBMQREPVERTIME' FROM: '2008-03-10-0.00.00.00000'
 TO: '2007-04-12-00.00.00.00000'"
```

Example 4

In SQL Replication, to find the differences between a range of rows in the source and target table that are specified in a subscription set called my_set, with a target table named trg_table, on an Apply control server named apply_db, with an Apply schema of asn, and to name the difference table diff_table:

```
asntdiff DB=apply_db schema=asn where="set_name = 'my_set'
 and target_table = 'trg_table'" diff=diff_table
 RANGECOL="'CREDIT_TIME' FROM:'2008-03-10-12.00.00.00000'
 TO: '2008-03-11-12.00.00.00000'"
```

asntdiff: Comparing data in source and target tables (z/OS)

Use the **asntdiff** command to compare two relational tables and generate a list of differences between the two. Run the **asntdiff** command with JCL or at a UNIX System Services (USS) command prompt or shell script.

This topic describes usage on z/OS:

- For details on running **asntdiff** on Linux, UNIX, and Windows, see "asntdiff: Comparing data in source and target tables (Linux, UNIX, Windows)" on page 422.
- For information on the asntdiff –f command option, which enables you to compare tables whether or not they are involved in replication, see "asntdiff –f (input file) command option" on page 433.
- For details on using asntdiff in parallel mode, see "Running the asntdiff utility in parallel mode (z/OS)" on page 322.

The tables that you compare can reside on DB2 for z/OS, DB2 for Linux, UNIX, Windows, or DB2 for System i.

Syntax

```
►►──asntdiff──DB=server──DB2_SUBSYSTEM=subsystem──┬──────────────────┬──────────────►
                                                  └─SCHEMA=schema────┘

►──┬─────────────────────────────────────────┬──┬──────────────────────────────────┬──►
   └─DIFF_SCHEMA=difference_table_schema─────┘  └─DIFF_TABLESPACE=tablespace───────┘

►──┬─────────────────────┬──┬──────────────────────────────┬──WHERE=WHERE_clause────►
   │            ┌─n─┐    │  └─MAXDIFF=difference_limit─────┘
   └─DIFF_DROP=─┴─y─┴────┘

►──┬──────────────────────┬──┬──────────────────────────────────────────┬──────────►
   └─DIFF=table_name──────┘  └─RANGECOL=─┤ range-clause-option ├────────┘

►──┬────────────────────────────────────────────────┬─────────────────────────────►
   │           ┌─n──────────────────────────────┐   │
   └─PARALLEL=─┴─y──┤ parallel-options ├────────┴───┘

►──┬─────────────────────────────────────────────┬────────────────────────────────►◄
   ├─SQLID=source_authorization_ID───────────────┤
   └─SOURCESQLID=source_authorization_ID─────────┘
```

range-clause-option:

```
├──┬─src_colname─FROM:date-time_lower-bound─TO:date-time_upper-bound─┬──────────┤
   ├─src_colname─FROM:date-time──────────────────────────────────────┤
   └─src_colname─TO:date-time────────────────────────────────────────┘
```

parallel-options:

```
├──┬──────────────────────────────────────┬──┬──────────────────────────────┬──┬──────────────────────────────────────┬──┤
   └─NUMTHREADS=level_of_parallelism──────┘  └─NUMBLOCKS==number_of_blocks──┘  └─TARGET_SQLID=target_authorization_ID─┘
```

Parameters

Table 48 defines the invocation parameters for the **asntdiff** command.

Table 48. asntdiff invocation parameter definitions for z/OS operating systems

Parameter	Definition
DB=*server*	Specifies the DB2 alias of the database that stores information about the source and target tables to be compared. The value differs depending on whether you are using Q Replication or SQL Replication: **Q Replication** The location name of the Q Capture server, which contains the IBMQREP_SUBS table. **SQL Replication** The location name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table. **Restriction:** This parameter is not valid for the asntdiff -f (input file) option. You can use the SOURCE_SERVER and TARGET_SERVER parameters with the -f option to specify the names of the source and target database. On z/OS, these are location names and you also must use the DB2_SUBSYSTEM parameter to specify the name of the subsystem where the asntdiff utility runs.
DB2_SUBSYSTEM=*subsystem*	Specifies the name of the subsystem where you run the asntdiff command.
SCHEMA=*schema*	Specifies the schema of the Q Capture control tables for Q Replication, or the schema of the Apply control tables for SQL Replication. The default is ASN.
DIFF_SCHEMA= *difference_table_schema*	Specifies the schema of the difference table. The default is ASN.
DIFF_TABLESPACE=*tablespace*	Specifies the table space of the difference table. If this parameter is not specified, the table is created in the default table space in the subsystem where the **asntdiff** command was run. This is a two-part name, *dbname.tablespace*, where *dbname* is the logical database name and *tablespace* is the table space name.
DIFF_DROP=y/n	Specifies whether an existing difference table will be dropped and recreated before it is reused to record differences. If the table does not exist, the **asntdiff** command creates it. **n (default)** The difference table will be used as is and the existing rows will be deleted. **y** The difference table will be dropped and recreated.
MAXDIFF=*difference_limit*	Specifies the maximum number of differences that you want the **asntdiff** command to process before it stops. The default value is 10000.

Table 48. asntdiff invocation parameter definitions for z/OS operating systems (continued)

Parameter	Definition
WHERE=*WHERE_clause*	Specifies an SQL WHERE clause that uniquely identifies one row of the control table that stores information about the source and target tables that will be compared. The WHERE clause must be in double quotation marks. The value of this parameter differs depending on whether you are using Q Replication or SQL Replication: **Q Replication** The WHERE clause specifies a row in the IBMQREP_SUBS table and uses the SUBNAME column to identify the Q subscription that contains the source and target tables. **SQL Replication** The WHERE clause specifies a row in the IBMSNAP_SUBS_MEMBR table and uses the SET_NAME, APPLY_QUAL, TARGET_SCHEMA, and TARGET_TABLE columns to identify the subscription set member that contains the source and target tables. **Restriction:** This parameter is not valid for the asntdiff -f (input file) option. You can use the SOURCE_SELECT and TARGET_SELECT parameters with the -f option to specify the tables to be compared, and can use a WHERE clause in the SQL query that is provided with these parameters.
DIFF=*table_name*	Specifies the name of the table that is created in the source subsystem to store differences between the source and target tables. The table has one row for each difference that is detected. If you do not include this parameter or the **DIFF_SCHEMA** parameter, the difference table is named ASN.ASNTDIFF.

Table 48. asntdiff invocation parameter definitions for z/OS operating systems (continued)

Parameter	Definition

RANGECOL clause	Specifies a range of rows from the source table that you want to compare. You provide the name of a DATE, TIME, or TIMESTAMP column in the source table, and then use one of three different clauses for specifying the range. The column name must be enclosed in single quotation marks. The clause must be enclosed in double quotation marks.
	
	The timestamp uses the following format: *YYYY-MM-DD-HH.MM.SS.mmmmm*. For example, 2010-03-10-10.35.30.55555 is the GMT timestamp for March 10, 2010, 10:35 AM, 30 seconds, and 55555 microseconds.
	
	Use one of the following clauses:
	
	src_colname **FROM:** *date-time_lower-bound* **TO:** *date-time_upper-bound*
	Specifies a lower and upper bound for the range of rows to compare.
	
	The following example uses a TIMESTAMP column:
	`"'SALETIME'`
	`FROM: 2008-02-08-03.00.00.00000`
	`TO: 2008-02-15-03.00.00.00000"`
	
	Remember: Both the **`FROM:`** and **`TO:`** keywords are required and both keywords must be followed by a colon (:).
	
	src_colname **FROM:** *date-time*
	Specifies that you want to compare all rows with timestamps that are greater than or equal to *date-time*.
	
	For example:
	`"'SALE_TIME'`
	`FROM: 2008-03-10-10.35.30.55555"`
	
	src_colname **TO:** *date-time*
	Specifies that you want to compare all rows with timestamps that are less than or equal to the *date-time*.
	
	For example:
	`"'SALETIME'`
	`TO: 2008-03-20-12.00.00.00000"`
	Recommendation: For better performance, ensure that you have an index on the source column that is specified in the range clause.When you compare tables that are involved in peer-to-peer replication, you can use the IBM-generated IBMQREPVERTIME column for the source column in the range clause.
	Restriction: The RANGECOL parameter is not valid for the **`asntdiff -f`** (input file) option. You can use a SQL WHERE clause in the input file to achieve similar results.

Table 48. asntdiff invocation parameter definitions for z/OS operating systems (continued)

Parameter	Definition
PARALLEL=y/n	Specifies whether the asntdiff utility uses parallel mode, in which multiple threads are used to compare the tables, or operates in serial mode with a single thread. **n (default)** The asntdiff utility uses serial mode. **y** The asntdiff utility uses parallel mode. For details on installation requirements, required authorizations, and restrictions, see "Running the asntdiff utility in parallel mode (z/OS)" on page 322.
NUMTHREADS=*number_of_threads*	Specifies the number of threads that the asntdiff utility is allowed to create. The minimum value is six. The recommended value is 21, which is also the maximum value and the default value. Ensure that the MAXTHREADS parameter value that is specified in BPXPRMXX is larger than the specified number of threads. Also, configure DB2 ZPARMS CTHREAD, IDFORE, and IDBACK to allow each of the created threads to connect to DB2.
NUMBLOCKS=*number_of_blocks*	Specifies the number of partitions into which the asntdiff utility divides the source and target tables (that is, the result sets of the SOURCE_SELECT and TARGET_SELECT parameters) for parallel compare. A value of 0 (the default) means that the utility automatically determines the number of blocks.
SQLID=*authorization_ID*	Use this parameter when asntdiff is running in non-parallel mode. The parameter specifies an authorization ID that can be used to create the difference table. Use this parameter if the ID that is used to run the **asntdiff** command does not have authorization to create tables. The value of the **SQLID** parameter is used as the schema for the difference table if you do not explicitly specify a schema by using the **DIFF_SCHEMA** parameter.
SOURCE_SQLID=*authorization_ID*	When you use asntdiff in parallel mode, this parameter specifies an authorization ID that can be used to execute stored procedures and packages and run DDL and DML on temporary tables at the source. Use this parameter if the ID that is used to run the **asntdiff** command does not have the necessary authorization.
TARGET_SQLID=*authorization_ID*	When you use asntdiff in parallel mode, this parameter specifies an authorization ID that can be used to execute stored procedures and packages and run DDL and DML on temporary tables at the target. Use this parameter if the ID that is used to run the **asntdiff** command does not have the necessary authorization.

Usage notes

The **asntdiff** command creates data sets (JCL) or temporary files (USS) for spilling data and for writing differences before inserting them into the difference table. You specify the location of the data sets or temporary files differently:

JCL

If you want ASNTDIFF to write to z/OS data sets, add these two DD statements to your ASNTDIFF JCL, modifying the size specifications to match the size of your source table:

```
//SPLFILE  DD  DSN=&&SPILL,DISP=(NEW,DELETE,DELETE),
//             UNIT=VIO,SPACE=(CYL,(11,7)),
//             DCB=(RECFM=VS,BLKSIZE=6404)
//DIFFFILE DD  DSN=&&DIFFLE,DISP=(NEW,DELETE,DELETE),
//             UNIT=VIO,SPACE=(CYL,(11,7)),
//             DCB=(RECFM=VS,BLKSIZE=6404)
```

USS On USS, temporary files are written by default to the hierarchical file system (HFS), in the home directory of the user ID that executes the **asntdiff** command. The default names are `DD:DIFFFILE` and `DD:SPILLFILE`. You can use a DIFFFILE DD statement to specify an alternative HFS path and file name for those files, as shown in this example:

```
//DIFFFILE DD PATH='/u/oeusr01/tdiffil2',
//         PATHDISP=(KEEP,KEEP),
//         PATHOPTS=(ORDWR,OCREAT),
//         PATHMODE=(SIRWXU,SIRGRP,SIROTH)
```

Redirecting the HFS requires you to create an empty file that can be written to or to use the above PATHDISP and PATHOPTS settings to create a new file if one does not exist.

Examples for asntdiff

The first four examples show how to use the **asntdiff** command on USS; the fifth example provides JCL. For more sample JCL, see the ASNTDIFF sample program in the SASNSAMP sample data set.

Example 1: Q Replication

In Q Replication, to find the differences between a source and target table that are specified in a Q subscription named my_qsub, on a Q Capture server named source_db, with a Q Capture schema of asn:

```
asntdiff db=source_db schema=asn where="subname = 'my_qsub'"
```

Example 2: SQL Replication

In SQL Replication, to find the differences between a source and target table that are specified in a subscription set called my_set, with a target table named trg_table, on an Apply control server named apply_db, with an Apply schema of asn, and to name the difference table diff_table:

```
asntdiff DB=apply_db schema=asn where="set_name = 'my_set'
 and target_table = 'trg_table'" diff=diff_table
```

Example 3: Comparing a range of rows in Q Replication

In Q Replication, to find the differences between a range of rows in the source and target tables that are specified in a peer-to-peer Q subscription named my_qsub, on a Q Capture server named source_db, with a Q Capture schema of asn:

```
asntdiff db=source_db schema=asn where="subname = 'my_qsub'"
 RANGECOL="'IBMQREPVERTIME' FROM: '2008-03-10-0.00.00.00000'
 TO: '2007-04-12-00.00.00.00000'"
```

Example 4: Comparing a range of rows in SQL Replication

In SQL Replication, to find the differences between a range of rows in the source and target table that are specified in a subscription set called my_set, with a target table named trg_table, on an Apply control server named apply_db, with an Apply schema of asn, and to name the difference table diff_table:

```
asntdiff DB=apply_db schema=asn where="set_name = 'my_set'
 and target_table = 'trg_table'" diff=diff_table
 RANGECOL="'CREDIT_TIME' FROM:'2008-03-10-12.00.00.00000'
 TO: '2008-03-11-12.00.00.00000'"
```

Example 5: Using asntdiff in parallel mode

To run the asntdiff utility in parallel mode to compare two tables with 21 parallel threads, you can use the following JCL after locating and changing all occurrences of the following strings:

- The subsystem name DSN! to the name of your DB2 subsystem
- DSN!!0 to the name if your DB2 target library
- ASNQ!!0 to the name of your Replication Server target library

```
//ASNTDIF1 EXEC PGM=ASNTDIFF,PARM='/-F'
//STEPLIB  DD DSN=ASNQ!!0.SASNLOAD,DISP=SHR
//            DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//MSGS     DD PATH='/usr/lpp/db2repl_10_01/msg/En_US/db2asn.cat'
//CEEDUMP  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  DUMMY
//SYSIN    DD   *
DB2_SUBSYSTEM=DSN!
SOURCE_SERVER=DQRG
SOURCE_SELECT="SELECT empno, department FROM employee
                            WHERE empno > 10000 ORDER BY 1"
TARGET_SERVER=D7DP
TARGET_SELECT="SELECT empno, department FROM employee
                            WHERE empno > 10000 ORDER BY 1"
PARALLEL=Y
NUMTHREADS=21
SOURCE_SQLID=SRCADM
TARGET_SQLID=TGTADM
DIFF_DROP=Y
MAXDIFF=20000
DEBUG=NO
/*
//
```

asntdiff –f (input file) command option

With the **asntdiff -f** command option, you use an input file to specify information about any two tables that you want to compare, whether or not they are being replicated.

The input file contains SQL SELECT statements for the source and target tables that specify the rows that you want to compare. The standard asntdiff command compares tables that are involved in replication by using subscription information from the replication control tables.

The **asntdiff -f** option can compare any tables on z/OS, Linux, UNIX, or Windows. You can run **asntdiff -f** from a Linux, UNIX, or Windows command prompt, from z/OS as a batch job that uses JCL, or from z/OS under the UNIX System Services (USS) environment.

In addition to the SELECT statements, the input file contains the source and target database information, the difference table information, and optional parameters that specify methods for processing the differences. You can use a password file that is created by the **asnpwd** command to specify a user ID and password for connecting to the source and target databases.

Note: The asntrep command for repairing table differences does not support the input file option.

The format of the input file contents is as follows:

```
* Optional comment line
# Optional comment line
SOURCE_SERVER=server_name
SOURCE_SELECT="SQL_SELECT_STATEMENT"
TARGET_SERVER=server_name
TARGET_SELECT="SQL_SELECT_STATEMENT"
PARAMETER=value
...
```

Follow these guidelines:

- Each parameter must follow the *parameter=value* format.
- Multiple parameter-value pairs can be specified on a single line, separated by a blank. The parameter-value pairs also can be specified on a new line.
- To preserve blanks, surround parameter values with double quotation marks ("). Double quotation marks are also required for the source and target SELECT statements.
- If you want to preserve mixed case or blanks in the names of single DB2 objects (column or table names, DIFF_SCHEMA, DIFF_TABLESPACE) mask them with `\" \"`, for example `\"MY NAME\"` or `\"ColumnName\"` or `\"name\"`.
- Comments must be prefixed with an asterisk (*) or pound sign (#). This line is ignored. Comments must be on their own line and cannot be added to a line that contains parameters.
- Surround the DIFF_PATH and PWDFILE parameters with double quotation marks ("). A final path delimiter for DIFF_PATH is not required.

Syntax

```
►►—asntdiff—-f—input_filename————————————————————————————————►◄
```

Parameters

Table 49 defines the mandatory parameters to include in the input file for the **asntdiff -f** command.

For descriptions of optional parameters that you can include in the input file (and which are shared by the standard asntdiff command) see “asntdiff: Comparing data in source and target tables (z/OS)” on page 426 or“asntdiff: Comparing data in source and target tables (Linux, UNIX, Windows)” on page 422.

Table 49. asntdiff -f invocation parameter definitions for Linux, UNIX, Windows, and z/OS

Parameter	Definition
input_filename	Specifies the name of the file that contains the source and target database information and SELECT statements. Specify a directory path if the file is located somewhere other than the directory from which you run the **asntdiff -f** command.

Table 49. asntdiff -f invocation parameter definitions for Linux, UNIX, Windows, and z/OS (continued)

Parameter	Definition
SOURCE_SERVER= *source_server_name*	Specifies the alias of the database where the source table exists.
TARGET_SERVER= *target_server_name*	Specifies the alias of the database where the target table exists.
SOURCE_SELECT= *source_select_statement* **TARGET_SELECT**= *target_select_statement*	Any valid SQL SELECT statement. The result sets from the SQL statement at each table must contain columns with matching data types and lengths. The **asntdiff** command describes the queries and compares the data from the two result sets. The command does not explicitly check the system catalog for type and length information. The SELECT can be an open select as in (*), or a SELECT statement that contains column names, SQL expressions, and WHERE clauses that are permitted. An ORDER BY clause is mandatory. The clause must contain the numeric values of the positions of the columns in the SQL statement. Ensure that the column or columns in the ORDER BY clause reference a unique key or unique composite key. Otherwise the results are incorrect. An index on the columns in the ORDER BY clause might improve performance by eliminating the need for a sort. The entire statement must be enclosed in double quotes to mark the beginning and the end.

The following examples show the mandatory parameters, SQL statements, and optional parameters that you put in the input file.

z/OS

Example 1

This example shows the use of an open SELECT statement on DB2 for z/OS. Note the use of the \" to preserve mixed case in the table owner, and the use of optional parameters in the input file. Also note the use of the DB2_SUBSYSTEM parameter.

```
SOURCE_SERVER=STPLEX4A_DSN7
SOURCE_SELECT="select * from CXAIMS.ALDEC order by 1"
TARGET_SERVER=STPLEX4A_DSN7
TARGET_SELECT="select * from \"Cxaims\".TARG_ALDEC order by 1"
DIFF_DROP=Y
DB2_SUBSYSTEM=DSN7
MAXDIFF=10000
DEBUG=YES
```

z/OS

Example 2

This example demonstrates the use of SUBSTR and CAST functions in the SELECT statements.

```
SOURCE_SERVER=D7DP
SOURCE_SELECT="select HIST_CHAR12,HIST_DATE,HIST_CHAR6,HIST_INT1,HIST_INT2,
HIST_INT3,SUBSTR(CHAR1,1,5) AS CHAR1,SUBSTR(CHAR2,1,10) AS CHAR2,HIST_INT3,
HIST_DEC1,HIST_DEC2,HIST_DEC3,CAST(INT1 AS SMALLINT) AS INT1
FROM BISVT.THIST17 ORDER BY 4"
TARGET_SERVER=STPLEX4A_DSN7
TARGET_SELECT="select HIST_CHAR12,HIST_DATE,HIST_CHAR6,HIST_INT1,HIST_INT2,
HIST_INT3,CHAR1,CHAR2,HIST_INT3,HIST_DEC1,HIST_DEC2,HIST_DEC3,SML1
FROM BISVT.THIST17 ORDER BY 4"
DB2_SUBSYSTEM=DSN7
DIFF_DROP=Y
DEBUG=YES
MAXDIFF=10000
```

Windows

Example 3

This example compares the EMPLOYEE tables on SOURCEDB and TARGETDB and includes several optional parameters.

```
SOURCE_SERVER=SOURCEDB
SOURCE_SELECT="select FIRSTNME, LASTNAME, substr(WORKDEPT,1,1)
as WORKDEPT, EMPNO from EMPLOYEE order by 4"
TARGET_SERVER=TARGETDB
TARGET_SELECT="select FIRSTNME, LASTNAME, substr(WORKDEPT,1,1)
as WORKDEPT, EMPNO from EMPLOYEE order by 4"
DIFF_DROP=Y
DIFF =\"diffTable\"
DEBUG=YES
MAXDIFF=10000
PWDFILE="asnpwd.aut"
DIFF_PATH="C:\utils\"
```

Linux UNIX

Example 4

This example compares the EMPLOYEE tables in a Linux or UNIX environment and uses a casting function.

```
SOURCE_SERVER=SOURCEDB
SOURCE_SELECT="select EMPNO, FIRSTNME, LASTNAME, cast(SALARY as INT)
as SALARY from EMPLOYEE order by 1"
TARGET_SERVER=TARGETDB
TARGET_SELECT="select EMPNO, FIRSTNME, LASTNAME, cast(SALARY as INT)
as SALARY from EMPLOYEE order by 1"
DIFF_DROP=Y
DIFF =\"diffTable\"
DEBUG=YES
MAXDIFF=10000
PWDFILE="asnpwd.aut"
DIFF_PATH="home/laxmi/utils"
```

asntrc: Operating the replication trace facility

Use the **asntrc** command to run the trace facility on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS. The trace facility logs program flow information from Q Capture, Q Apply, Capture, Apply, and Replication Alert Monitor programs. You can provide this trace information to IBM Software Support for troubleshooting assistance. Run this command at an operating system prompt or in a shell script.

You run this command at an operating system prompt or in a shell script.

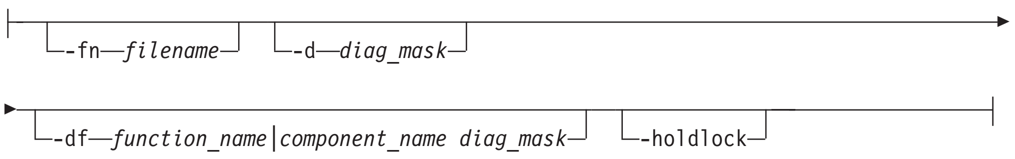
Syntax

```
►►──asntrc──────────────────────────────────────────────────────────────────────────────►

►──┬─on────-db──db_name────┬─-qcap─┬────────────────────────────┬─┬─┤ On parameters ├──────┬──►◄
   │                       │       └─-schema──qcapture_schema───┘ │                        │
   │                       ├─-qapp─┬──────────────────────────┬───┤                        │
   │                       │       └─-schema──qapply_schema───┘   │                        │
   │                       ├─-cap──┬──────────────────────────┬───┤                        │
   │                       │       └─-schema──capture_schema──┘   │                        │
   │                       ├─-app──┬────────────────────────────┬─┤                        │
   │                       │       └─-qualifier──apply_qualifier┘ │                        │
   │                       └─-mon──┬──────────────────────────────┴┘                       │
   │                               └─-qualifier──monitor_qualifier┘                        │
   ├─┬─off───────┬──-db──db_name──┬─-qcap─┬────────────────────────────┬─┬─────────────────┤
   │ ├─kill──────┤                │       └─-schema──qcapture_schema───┘ │                 │
   │ ├─clr───────┤                ├─-qapp─┬──────────────────────────┬───┤                 │
   │ ├─diag──────┤                │       └─-schema──qapply_schema───┘   │                 │
   │ └─resetlock─┘                ├─-cap──┬──────────────────────────┬───┤                 │
   │                              │       └─-schema──capture_schema──┘   │                 │
   │                              ├─-app──┬────────────────────────────┬─┤                 │
   │                              │       └─-qualifier──apply_qualifier┘ │                 │
   │                              └─-mon──┬──────────────────────────────┴┘                │
   │                                      └─-qualifier──monitor_qualifier┘                 │
   ├─dmp──filename────-db──db_name──┬─-qcap─┬────────────────────────────┬─┬─┬───────────┬─┤
   │                                │       └─-schema──qcapture_schema───┘ │ └─-holdlock─┘ │
   │                                ├─-qapp─┬──────────────────────────┬───┤               │
   │                                │       └─-schema──qapply_schema───┘   │               │
   │                                ├─-cap──┬──────────────────────────┬───┤               │
   │                                │       └─-schema──capture_schema──┘   │               │
   │                                ├─-app──┬────────────────────────────┬─┤               │
   │                                │       └─-qualifier──apply_qualifier┘ │               │
   │                                └─-mon──┬──────────────────────────────┴┘              │
   │                                        └─-qualifier──monitor_qualifier┘               │
   ├─┬─flw───┬─┬─────────────────────────────────────────────────────┬─┤ Format parameters ├─┤
   │ ├─fmt───┤ └─┬──────────────────┬─┬─-qcap─┬────────────────────────────┬─┬┘             │
   │ └─v7fmt─┘   └─-db──db_name─────┘ │       └─-schema──qcapture_schema───┘ │              │
   │                                  ├─-qapp─┬──────────────────────────┬───┤              │
   │                                  │       └─-schema──qapply_schema───┘   │              │
   │                                  ├─-cap──┬──────────────────────────┬───┤              │
   │                                  │       └─-schema──capture_schema──┘   │              │
   │                                  ├─-app──┬────────────────────────────┬─┤              │
   │                                  │       └─-qualifier──apply_qualifier┘ │              │
   │                                  └─-mon──┬──────────────────────────────┴┘             │
   │                                          └─-qualifier──monitor_qualifier┘              │
   ├─┬─stat─────┬─┬──────────────────────────────────────────────────────────────┬──────────┤
   │ └─statlong─┘ ├─┬──────────────────┬─┬─-qcap─┬────────────────────────────┬─┬─┤          │
   │              │ └─-db──db_name─────┘ │       └─-schema──qcapture_schema───┘ │ │          │
   │              │                      ├─-qapp─┬──────────────────────────┬───┤ │          │
   │              │                      │       └─-schema──qapply_schema───┘   │ │          │
   │              │                      ├─-cap──┬──────────────────────────┬───┤ │          │
   │              │                      │       └─-schema──capture_schema──┘   │ │          │
   │              │                      ├─-app──┬────────────────────────────┬─┤ │          │
   │              │                      │       └─-qualifier──apply_qualifier┘ │ │          │
   │              │                      └─-mon──┬──────────────────────────────┴┘│          │
   │              │                              └─-qualifier──monitor_qualifier┘ │          │
   │              └─-fn──filename─────────────────────────────────────────────────┘          │
   ├──-db──db_name────┬─-qcap─┬────────────────────────────┬─┬─┤ Change settings parameters ├─┤
   │                  │       └─-schema──qcapture_schema───┘ │                              │
   │                  ├─-qapp─┬──────────────────────────┬───┤                              │
   │                  │       └─-schema──qapply_schema───┘   │                              │
   │                  ├─-cap──┬──────────────────────────┬───┤                              │
   │                  │       └─-schema──capture_schema──┘   │                              │
   │                  ├─-app──┬────────────────────────────┬─┤                              │
   │                  │       └─-qualifier──apply_qualifier┘ │                              │
   │                  └─-mon──┬──────────────────────────────┴┘                             │
   │                          └─-qualifier──monitor_qualifier┘                              │
   ├─┬───────┬──────────────────────────────────────────────────────────────────────────────┤
   │ └─-help─┘                                                                              │
   └─-listsymbols───────────────────────────────────────────────────────────────────────────┘
```

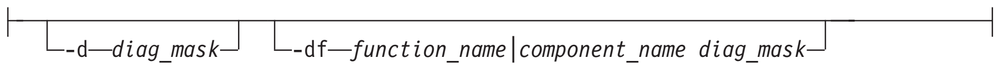
On parameters:

```
├──┬─────────────────┬──┬──────────────┬──┬──────────────┬───────────────────────────►
   └─-b──buffer_size─┘  └─-fn──filename─┘  └─-fs──filesize─┘

►──┬───────────────┬──┬──────────────────────────────────────────────────┬─────────┤
   └─-d──diag_mask─┘  └─-df──function_name|component_name diag_mask──────┘
```

Format parameters:



Change settings parameters:



Parameters

Table 50 defines the invocation parameters for the **asntrc** command.

Table 50. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems

Parameter	Definition
on	Specify to turn on the trace facility for a specific Q Capture, Q Apply, Capture, Apply, or Replication Alert Monitor program. The trace facility creates a shared memory segment used during the tracing process.
-db *db_name*	Specifies the name of the database to be traced: • Specifies the name of the Q Capture server for the Q Capture program to be traced. • Specifies the name of the Q Apply server for the Q Apply program to be traced. • Specifies the name of the Capture control server for the Capture program to be traced. • Specifies the name of the Apply control server for the Apply program to be traced. • Specifies the name of the Monitor control server for the Replication Alert Monitor program to be traced.
-qcap	Specifies that a Q Capture program is to be traced. The Q Capture program is identified by the **-schema** parameter.
-schema *qcapture_schema*	Specifies the name of the Q Capture program to be traced. The Q Capture program is identified by the Q Capture schema that you enter. Use this parameter with the **-qcap** parameter.
-qapp	Specifies that a Q Apply program is to be traced. The Q Apply program is identified by the **-schema** parameter.
-schema *qapply_schema*	Specifies the name of the Q Apply program to be traced. The Q Apply program is identified by the Q Apply schema that you enter. Use this parameter with the **-qapp** parameter.
-cap	Specifies that a Capture program is to be traced. The Capture program is identified by the **-schema** parameter.

Table 50. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
-schema *capture_schema*	Specifies the name of the Capture program to be traced. The Capture program is identified by the Capture schema that you enter. Use this parameter with the **-cap** parameter.
-app	Specifies that an Apply program is to be traced. The Apply program is identified by the **-qualifier** parameter.
-qualifier *apply_qualifier*	Specifies the name of Apply program to be traced. This Apply program is identified by the Apply qualifier that you enter. Use this parameter with the **-app** parameter.
-mon	Specifies that a Replication Alert Monitor program is to be traced. The Replication Alert Monitor program is identified by the **-qualifier** parameter.
-qualifier *monitor_qualifier*	Specifies the name of Replication Alert Monitor program to be traced. This Replication Alert Monitor program is identified by the monitor qualifier that you enter. Use this parameter with the **-mon** parameter.
off	Specify to turn off the trace facility for a specific Q Capture, Q Apply, Capture, Apply, or Replication Alert Monitor program and free the shared memory segment in use.
kill	Specify to force an abnormal termination of the trace facility. Use this parameter only if you encounter a problem and are unable to turn the trace facility off with the **off** parameter.
clr	Specify to clear a trace buffer. This parameter erases the contents of the trace buffer but leaves the buffer active.
diag	Specify to view the filter settings while the trace facility is running.
resetlock	Specify to release the buffer latch of a trace facility. This parameter enables the buffer latch to recover from an error condition in which the trace program terminated while holding the buffer latch.
dmp *filename*	Specify to write the current contents of the trace buffer to a file.
-holdlock	Specifies that the trace facility can complete a file dump or output command while holding a lock, even if the trace facility finds insufficient memory to copy the buffer.
flw	Specify to display summary information produced by the trace facility and stored in shared memory or in a file. This information includes the program flow and is displayed with indentations that show the function and call stack structures for each process and thread.

Table 50. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
fmt	Specify to display detailed information produced by the trace facility and stored in shared memory or in a file. This parameter displays the entire contents of the traced data structures in chronological order.
v7fmt	Specify to display information produced by the trace facility and stored in shared memory or in a file. This trace information appears in Version 7 format.
stat	Specify to display the status of a trace facility. This status information includes the trace version, application version, number of entries, buffer size, amount of buffer used, status code, and program timestamp.
statlong	Specify to display the status of a trace facility with additional z/OS version level information. This additional information includes the service levels of each module in the application and appears as long strings of text.
-fn *filename*	Specifies the file name containing the mirrored trace information, which includes all the output from the trace facility.
-help	Displays the valid command parameters with descriptions.
-listsymbols	Displays the valid function and component identifiers to use with the **-df** parameter.
-b *buffer_size*	Specifies the size of the trace buffer (in bytes). You can enter a K or an M after the number to indicate kilobytes or megabytes, respectively; these letters are not case sensitive.
-fs *filesize*	Specifies the size limit (in bytes) of the mirrored trace information file.

Table 50. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
-d *diag_mask*	Specifies the types of trace records to be recorded by the trace facility. Trace records are categorized by a diagnostic mask number: **1** Flow data, which includes the entry and exit points of functions. **2** Basic data, which includes all major events encountered by the trace facility. **3** Detailed data, which includes the major events with descriptions. **4** Performance data. **Important:** The higher diagnostic mask numbers are *not* inclusive of the lower diagnostic mask numbers. You can enter one or more of these numbers to construct a diagnostic mask that includes only the trace records that you need. For example, specify **-d 4** to record only performance data; specify **-d 1,4** to record only flow and performance data; specify **-d 1,2,3,4** (the default) to record all trace records. Separate the numbers with commas. Enter a diagnostic mask number of 0 (zero) to specify that no global trace records are to be recorded by the trace facility. Type **-d 0** to reset the diagnostic level before specifying new diagnostic mask numbers for a tracing facility.
-df *function_name*|*component_name diag_mask*	Specifies that a particular function or component identifier is to be traced. Type the diagnostic mask number (1,2,3,4) after the function or component identifier name. You can enter one or more of these numbers. Separate the numbers with commas.

Examples for asntrc

The following examples illustrate how to use the **asntrc** command. These examples can be run on Linux, UNIX, Windows, or z/OS operating systems.

Example 1

To trace a running Capture program:

1. Start the trace facility, specifying a trace file name with a maximum buffer and file size:
   ```
   asntrc on -db mydb -cap -schema myschema -b 256k -fn myfile.trc -fs 500m
   ```
2. Start the Capture program, and let it run for an appropriate length of time.
3. While the trace facility is on, display the data directly from shared memory.

   To display the summary process and thread information from the trace facility:
   ```
   asntrc flw -db mydb -cap -schema myschema
   ```

To view the flow, basic, detailed, and performance data records only from the Capture log reader:

```
asntrc fmt -db mydb -cap -schema myschema -d 0
  -df "Capture Log Read" 1,2,3,4
```

4. Stop the trace facility:

   ```
   asntrc off -db mydb -cap -schema myschema
   ```

   The trace file contains all of the Capture program trace data that was generated from the start of the Capture program until the trace facility was turned off.
5. After you stop the trace facility, format the data from the generated binary file:

   ```
   asntrc flw -fn myfile.trc
   ```

   and

   ```
   asntrc fmt -fn myfile.trc -d 0 -df "Capture Log Read" 1,2,3,4
   ```

Example 2

To start a trace facility of a Replication Alert Monitor program:

```
asntrc on -db mydb -mon -qualifier monq
```

Example 3

To trace only performance data of an Apply program:

```
asntrc on -db mydb -app -qualifier aq1 -b 256k -fn myfile.trc -d 4
```

Example 4

To trace all flow and performance data of a Capture program:

```
asntrc on dbserv1 -cap -schema myschema -b 256k
  -fn myfile.trc -d 1,4
```

Example 5

To trace all global performance data and the specific Capture log reader flow data of a Capture program:

```
asntrc on -db mydb -cap -schema myschema -b 256k -fn myfile.trc -d 4
  -df "Capture Log Read" 1
```

Example 6

To trace a running Capture program and then display and save a point-in-time image of the trace facility:

1. Start the trace command, specifying a buffer size large enough to hold the latest records:

   ```
   asntrc on -db mydb -cap -schema myschema -b 4m
   ```

2. Start the Capture program, and let it run for an appropriate length of time.
3. View the detailed point-in-time trace information that is stored in shared memory:

   ```
   asntrc fmt -db mydb -cap -schema myschema
   ```

4. Save the point-in-time trace information to a file:

   ```
   asntrc dmp myfile.trc -db mydb -cap -schema myschema
   ```

5. Stop the trace facility:

   ```
   asntrc off -db mydb -cap -schema myschema
   ```

Examples for asntrc with shared segments

The standalone trace facility, **asntrc**, uses a shared segment to communicate with the respective Q Capture, Q Apply, Capture, Apply or Replication Alert Monitor programs to be traced. The shared segment will also be used to hold the trace entries if a file is not specified. Otherwise, matching options must be specified for both the **asntrc** command and for the respective programs to be traced to match the correct shared segment to control traces. The following examples show the options that need to be specified when the trace facility is used in conjunction with Q Capture, Q Apply, Capture, Apply or Alert Monitor programs.

With the Q Capture program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **capture_server** parameter with the **asnqcap** command:

```
asntrc -db ASN6 -schema EMI -qcap
asnqcap capture_server=ASN6 capture_schema=EMI
```

With the Q Apply program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **apply_server** parameter with the **asnqapp** command:

```
asntrc -db TSN3 -schema ELB -qapp
asnqapp apply_server=TSN3 apply_schema=ELB
```

With the Capture program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **capture_server** parameter with the **asncap** command:

```
asntrc -db DSN6 -schema JAY -cap
asncap capture_server=DSN6 capture_schema=JAY
```

With the Apply program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **control_server** parameter with the **asnapply** command:

```
asntrc -db SVL_LAB_DSN6 -qualifier MYQUAL -app
asnapply control_server=SVL_LAB_DSN6 apply_qual=MYQUAL
```

With the Replication Alert Monitor program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **monitor_server** parameter with the **asnmon** command:

```
asntrc -db DSN6 -qualifier MONQUAL -mon
asnmon monitor_server=DSN6 monitor_qual=MONQUAL
```

asntrep: Repairing differences between source and target tables

Use the **asntrep** command to synchronize a source and target table by repairing differences between the two tables. Run the **asntrep** command on Linux, UNIX, and Windows at an operating system prompt or in a shell script.

Syntax

```
►►──asntrep──DB=server──DB2_SUBSYSTEM=subsystem──┬──────────────────┬──────────►
                                                 └─SCHEMA=schema─┘

►─┬──────────────────────────────────────┬─┬──────────────────────────────┬────►
  └─DIFF_SCHEMA=difference_table_schema─┘ └─DIFF_TABLESPACE=tablespace─┘
```

```
►─WHERE=WHERE_clause──┬──────────────────────┬──┬─────────────────────┬──────────►
                      └─DIFF_PATH=log_path───┘  └─PWDFILE=filename────┘

►──┬──────────────────────┬─────────────────────────────────────────────────────►◄
   └─DIFF=table_name──────┘
```

Parameters

Table 51 defines the invocation parameters for the **asntrep** command.

Table 51. asntrep invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
DB=*server*	Specifies the DB2 alias of the database that stores information about the source and target tables that you want to synchronize. The value differs depending on whether you are using Q Replication or SQL Replication: **Q Replication** The value is the name of the Q Capture server, which contains the IBMQREP_SUBS table. **SQL Replication** The value is the name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table. z/OS The value of this parameter is a location name.
DB2_SUBSYSTEM=*subsystem*	z/OS Specifies the name of the subsystem where you run the asntrep utility.
SCHEMA=*schema*	Specifies the schema of the Q Capture control tables for Q Replication, or the Apply control tables for SQL Replication.
DIFF_SCHEMA=*difference_table_schema*	Specifies the schema that qualifies the difference table. The default is ASN.
DIFF_TABLESPACE=*tablespace*	Specifies the table space where a copy of the difference table is placed in the target database or subsystem. The copy is then used to repair the target table. If this parameter is not specified, the table will be created in the default table space in the database or subsystem in which the **asntrep** command was run.

Table 51. asntrep invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
WHERE=*WHERE_clause*	Specifies a SQL WHERE clause that uniquely identifies one row of the control table that stores information about the source and target tables that you are synchronizing. The WHERE clause must be in double quotation marks. The value of this parameter differs depending on whether you are using Q Replication or SQL Replication: **Q Replication** The WHERE clause specifies a row in the IBMQREP_SUBS table and uses the SUBNAME column to identify the Q subscription that contains the source and target tables. **SQL Replication** The WHERE clause specifies a row in the IBMSNAP_SUBS_MEMBR table and uses the SET_NAME, APPLY_QUAL, TARGET_SCHEMA, and TARGET_TABLE columns to identify the subscription set member that contains the source and target tables.
DIFF_PATH=*log_path*	Specifies the location where you want the asntrep utility to write its log. The default value is the directory where you ran the command. The value must be an absolute path name. Use double quotation marks ("") to preserve case.
PWDFILE=*filename*	Specifies the name of the password file that is used to connect to databases. If you do not specify a password file, the default value is asnpwd.aut (the name of the password file that is created by the **asnpwd** command). The asntrep utility searches for the password file in the directory that is specified by the DIFF_PATH parameter. If no value for the DIFF_PATH parameter is specified, the command searches for the password file in the directory where the command was run.
DIFF=*table_name*	Specifies the name of the table that was created in the source database by the **asntdiff** command to store differences between the source and target tables. The information that is stored in this table is used to synchronize the source and target tables.

Examples for asntrep

The following examples illustrate how to use the **asntrep** command.

Example 1

In Q Replication, to synchronize a source and target table that are specified in a Q subscription named my_qsub, on a Q Capture server named source_db, with a Q Capture schema of asn, and whose differences are stored in a table called q_diff_table:

```
asntrep db=source_db schema=asn where="subname = 'my_qsub'" diff=q_diff_table
```

Example 2

In SQL Replication, to synchronize a source and target table that are specified in a subscription set called my_set, with a target table named trg_table, on an Apply control server named apply_db, with an Apply schema of asn, and whose differences are stored in a table called sql_diff_table:

```
asntrep DB=apply_db SCHEMA=asn WHERE="set_name = 'my_set'
 and target_table = 'trg_table'" diff=sql_diff_table
```

asnqanalyze: Operating the Q Replication Analyzer

Use the **asnqanalyze** command to gather information about the state of a Q replication or event publishing environment. You can also use the command to produce a formatted HTML report about Q Capture or Q Apply control tables, DB2 catalogs, diagnostic log files for the replication programs, and WebSphere MQ queue managers.

The command runs on Linux, UNIX, or Windows operating systems. However, it can connect to a locally cataloged DB2 subsystem on z/OS to analyze the control tables.

The **asnqanalyze** command has three parts:

- The GATHER option, which performs the analysis and stores the results in one or more XML files in the directory from where the command was run.
- The REPORT option, which produces an HTML report from the output of the GATHER option.
- The GENERATE HTML REPORT option, which generates an HTML report of the data in the Q Capture and Q Apply control tables without using the GATHER option.

You use GATHER once for each schema that is involved in the Q replication configuration. Then, you specify the output files from each GATHER invocation in the REPORT option. You can use the GENERATE HTML REPORT option to generate a raw HTML report rather than the detailed report that is generated by the GATHER plus REPORT options.

Run the **asnqanalyze** command at an operating system prompt.

Syntax

```
>>──asnqanalyze──┬─GATHER─┤ gather-options ├────────────────────────┬──────><
                 ├─REPORT─┤ report-options ├────────────────────────┤
                 └─GENERATE HTML REPORT─┤ generate-html-report-options ├─┘
```

gather options:

```
├──┬─┤ single-database-options ├───┬───────────────────────────────────┤
   └─┤ multiple-database-options ├─┘
```

single-database-options:

```
├──┬─DATABASE=dbname──────────────────────────┬──────────────────────────►
   └─CONFIGSERVER=srvrname──┬──────────────────┤
                            └─FILE=filename──┘

    ┌──────────────────────────────────────┐
    ▼                                      │
►──┬─INSTANCE=db2_instance───────────────┬─┴──┬─────────────────────┬────┤
   ├─SUBSYSTEM=db2_subsystem─────────────┤    └─ -o──output_file_name─┘
   ├─USERID=user_ID──────────────────────┤
   ├─PASSFILE=password_file──────────────┤
   ├─SCHEMA=schema───────────────────────┤
   ├─CAPLOGDIR=q_capture_log_directory───┤
   ├─APPLOGDIR=q_apply_log_directory─────┤
   ├─PORT=port───────────────────────────┤
   ├─CHANNEL=channel─────────────────────┤
   ├─HOSTNAME=hostname───────────────────┤
   │               ┌─OFF─┐               │
   ├─WARNERRSONLY=─┴─ON──┴───────────────┤
   │          ┌─ON──┐                    │
   ├─GETCOLS=─┴─OFF─┴────────────────────┤
   │             ┌─ON──┐                 │
   ├─GETMONITOR=─┴─OFF─┴─────────────────┤
   ├─LOGDAYS=number_of_days──────────────┤
   │      ┌─ON──┐                        │
   └─ZIP=─┴─OFF─┴────────────────────────┘
```

multiple-database-options:

```
             ┌────────────────────────┐
             ▼                        │                  ┌─OFF─┐
├──LIST─(────┤ database-options ├─────┴──)──┬─WARNERRSONLY=─┴─ON──┴──┬──────►
                                            │          ┌─ON──┐       │
                                            ├─GETCOLS=─┴─OFF─┴───────┤
                                            │             ┌─ON──┐    │
                                            ├─GETMONITOR=─┴─OFF─┴────┤
                                            ├─LOGDAYS=number_of_days─┤
                                            │      ┌─ON──┐           │
                                            └─ZIP=─┴─OFF─┴───────────┘

►──┬─────────────────────┬───────────────────────────────────────────────┤
   └─ -o──output_file_name─┘
```

database-options:

```
├──┬─DATABASE=dbname──────────────────────────┬──────────────────────────►
   └─CONFIGSERVER=srvrname──┬──────────────────┤
                            └─FILE=filename──┘

►─SCHEMA=schema──ID=user_ID──┬──────────────────────────┬────────────────┤
                             └─PASSFILE=password_file──┘
```

report-options:

```
                ┌──────────────────┐
                ▼                  │
├──REPORT──┬──XML_input_file──┴──┬────────────────────────────────────────┤
           └─-ZIP──zip_file_name──┘
```

generate-html-report-options:

```
├──GENERATE HTML REPORT──DATABASE=dbname──CONFIGSERVER=srvrname──────────────►

►──┤ optional parms ├──┬──────────────────────┬───────────────────────────┤
                       └─ -o──output_file_name─┘
```

optional parms

```
      ┌──────────────────────────────┐
      ▼                              │
├──────┬─USERID=user_ID──────────┬──┴───────────────────────────────────────┤
       ├─PASSFILE=password_file──┤
       ├─SCHEMA=schema───────────┤
       │          ┌─ON──┐        │
       ├─GETCOLS=─┴─OFF─┴────────┤
       │             ┌─ON──┐     │
       └─GETMONITOR=─┴─OFF─┴─────┘
```

Parameters for gather command, single database

Table 52 defines the invocation parameters for the **asnqanalyze gather** command for the single_database_options. Linux UNIX Windows

Table 52. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Description
DATABASE=*dbname*	The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed.
CONFIGSERVER=*srvrname*	The Q Capture server or Q Apply server to be analyzed when **asnqanalyze** uses a Type 4 connection to the server.
FILE=*file_name*	The configuration file where the port, hostname, and other connection information are located. This is required for databases that use a Type 4 connection.
INSTANCE=*db2_instance*	Linux UNIX Windows The DB2 instance of the database to be analyzed. Default: DB2
SUBSYSTEM= *db2_subsystem*	z/OS The DB2 subsystem to be analyzed. Default: DSN1
USERID=*user_ID*	The user ID to connect to the database.

Table 52. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Description
PASSFILE=*password_file*	The name of the file that stores the password for the user ID. To create a password file, create an ASCII file that contains only the password. Save this file in the directory where you are running the **gather** command. The password cannot be followed by any trailing blanks, special characters, or carriage returns. Q Analyzer cannot use encrypted password files that are created by the **asnpwd** command. You must specify a path to the password file if you run the **asnqanalyze** command from a directory other than where the file is stored. **Tip:** You might want to use a password file because commands that are issued at the prompt might be logged in the history and passwords are logged. A password file also saves you from typing your password each time that you run the Q Analyzer.
SCHEMA=*schema*	The schema of the Q Capture or Q Apply control tables to be analyzed. Default: ASN
CAPLOGDIR= *capture_log_directory*	The directory where the Q Capture program writes its diagnostic log file. If Q Capture is not running on the same system where you issue the **asnqanalyze** command, you must send the log file to a directory on the system where Q Capture runs, and this parameter specifies that directory. If you use FTP to send the log file, use ASCII mode. If no log files were transferred, the IBMQREP_CAPTRACE table is read for logging information. The data in this table is not as detailed as the data in the diagnostic log files.
APPLOGDIR= *apply_log_directory*	The directory where the Q Apply program writes its diagnostic log file. If Q Apply is not running on the same system where you issue the **asnqanalyze** command, you must send the log file to a directory on the system where Q Apply runs, and this parameter specifies that directory. If you use FTP to send the log file, use ASCII mode. If no log files were transferred, the IBMQREP_APPLYTRACE table is read for logging information. The data in this table is not as detailed as the data in the diagnostic log files.
PORT=*port*	The port number on which the listener program of the WebSphere MQ queue manager listens for incoming messages. You need to specify a port only if the server to be analyzed is remote from the computer on which you run the **asnqanalyze** command. Default: 1414 **Attention:** The queue manager must be running to be analyzed by the **gather** command.
CHANNEL=*channel*	The remote server connection channel that the queue manager is using. You need to specify a channel only if the server to be analyzed is remote from the computer on which you run the **asnqanalyze** command. Default: SYSTEM.DEF.SVRCONN
HOSTNAME=*hostname*	The host name or IP address of the remote system where the server to be analyzed resides. This parameter is required to connect to a remote WebSphere MQ queue manager.

Table 52. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Description
WARNERRSONLY=*ON/OFF*	The type of records to retrieve from the diagnostic log file. **OFF (default)** All records are retrieved, including error, warning, and informational messages and descriptions of program actions. **ON** Only error and warning messages are retrieved.
GETCOLS=*ON/OFF*	Specifies whether the Q Analyzer collects information about all columns or only the columns that are being replicated. **ON (default)** Information about all columns in the source and target tables is collected whether or not the columns are being replicated. **OFF** Information about replicated columns is collected.
GETMONITOR=*ON/OFF*	Specifies whether data in the IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables is retrieved. **ON (default)** Data is retrieved. **OFF** Data is not retrieved.
LOGDAYS=*integer*	Specifies how many days of records to retrieve from the log files and IBMQREP_CAPTRACE, IBMQREP_APPLYTRACE, IBMQREP_SIGNAL, IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables. Default: 3
ZIP=ON/OFF	Specifies whether to compress the generated XML files into one zip file (the default). If you specify **ZIP**=OFF the command generates individual XML files.
-o *output_file_name*	Specifies the name of the file that is produced by the **gather** option. If you do not specify a file name, the default name is as follows: **Compressed file** *database_name*.*schema*.zip **Uncompressed file** *database_name*.*schema*.xml

Parameters for gather command, multiple databases

Table 53 on page 451 defines the invocation parameters for the **asnqanalyze gather** command for the multiple_database_options. Use the **LIST**=*(database_options)* parameter to indicate that multiple databases are being provided.

Linux UNIX Windows

Table 53. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Description
LIST=*(database_options)*	Linux UNIX Windows Indicates that multiple databases are being provided. The database options *(database_options)* need to be specified within single parentheses following the parameter.
DATABASE=*dbname*	The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed.
CONFIGSERVER=*srvrname*	The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed with the Generate HTML Report option. Either the **DATABASE** or the **CONFIGSERVER** parameter should be specified. If **asnqanalyze** is used for a database that uses a Type 4 connection then the **CONFIGSERVER** parameter is used.
SCHEMA=*schema*	The schema of the Q Capture or Q Apply control tables to be analyzed. Default: ASN
USERID=*user_ID*	The user ID to connect to the database.
INIFILE=*file_name*	The configuration file where the port, host name are located. This is required for databases that use a Type 4 connection.
PASSFILE=*password_file*	The name of the file that stores the password for the user ID. To create a password file, create an ASCII file that contains only the password. Save this file in the directory where you are running the **gather** command. The password cannot be followed by any trailing blanks, special characters, or carriage returns. Q Analyzer cannot use encrypted password files that are created by the **asnpwd** command. You must specify a path to the password file if you run the **asnqanalyze** command from a directory other than where the file is stored. **Tip:** You might want to use a password file because commands that are issued at the prompt might be logged in the history and passwords are logged. A password file also saves you from typing your password each time that you run the Q Analyzer.
GETCOLS=*ON/OFF*	Specifies whether the Q Analyzer collects information about all columns or only the columns that are being replicated. **ON (default)** Information about all columns in the source and target tables is collected whether or not the columns are being replicated. **OFF** Information about replicated columns is collected.
GETMONITOR=*ON/OFF*	Specifies whether data in the IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables is retrieved. **ON (default)** Data is retrieved. **OFF** Data is not retrieved.
ZIP=ON/OFF	Specifies whether to compress the generated XML files into one zip file (the default). If you specify**ZIP**=OFF the command generates individual XML files.

Table 53. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Description
-o *output_file_name*	Specifies the name of the file that is produced by the **asnqanalyze gather** command. If you do not specify a file name, the default name is as follows: **Compressed file** `database_name.schema.zip` **Uncompressed file** `database_name.schema.xml`

Parameters for report command

Table 54 defines the invocation parameters for the **asnqanalyze report** command. Linux UNIX Windows

Table 54. asnqanalyze report invocation parameter definitions for Linux, UNIX, and Windows

Parameter	Description
XML_input_file	Specifies one or more space-separated XML files that were produced by the **asnqanalyze gather** command. Specify a path to the XML file if you run the **asnqanalyze report** command from a directory other than where the file is stored. The command prompts you for the database alias of the server where the file was created.
-ZIP*zip_file*	Specifies that a zipped file that was generated by the asnqanalyze gather command is provided as input. The zip file can have one or more XML files. The REPORT function unzips the file and analyzes each XML file.

Parameters for generate html report command

Table 55 defines the invocation parameters for the **asnqanalyze generate html report** command. Linux UNIX Windows

Table 55. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Description
DATABASE=*dbname*	The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed.
CONFIGSERVER=*srvrname*	Specifies the Q Capture server or Q Apply server to be analyzed when **asnqanalyze** uses a Type 4 connection to the server.
INIFILE=*file_name*	The configuration file where the port, host name are located. This is required for databases that use a Type 4 connection.
USERID=*user_ID*	The user ID to connect to the database.

Table 55. *asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)*

Parameter	Description
PASSFILE=*password_file*	The name of the file that stores the password for the user ID. To create a password file, create an ASCII file that contains only the password. Save this file in the directory where you are running the **gather** command. The password cannot be followed by any trailing blanks, special characters, or carriage returns. Q Analyzer cannot use encrypted password files that are created by the **asnpwd** command. You must specify a path to the password file if you run the **asnqanalyze** command from a directory other than where the file is stored. **Tip:** You might want to use a password file because commands that are issued at the prompt might be logged in the history and passwords are logged. A password file also saves you from typing your password each time that you run the Q Analyzer.
SCHEMA=*schema*	The schema of the Q Capture or Q Apply control tables to be analyzed. Default: ASN
GETCOLS=*ON/OFF*	Specifies whether the Q Analyzer collects information about all columns or only the columns that are being replicated. **ON (default)** Information about all columns in the source and target tables is collected whether or not the columns are being replicated. **OFF** Information about replicated columns is collected.
GETMONITOR=*ON/OFF*	Specifies whether data in the IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables is retrieved. **ON (default)** Data is retrieved. **OFF** Data is not retrieved.
-o *output_file_name*	Specifies the name of the file that is produced by the **asnqanalyze gather** command. If you do not specify a file name, the default name is as follows: **Compressed file** `database_name.schema.zip` **Uncompressed file** `database_name.schema.xml`

Example 1

The following command analyzes the Q Capture control tables, DB2 catalogs, and queue manager on the Q Capture server SAMPLE, for a user ID of db2admin with a password file `pwdfile.txt` and gathers only information about replicated columns.

In this example, the command is run from the SQLLIB\bin directory where the Q Capture program by default stores its log file, so there is no need to specify the CAPLOGDIR parameter. The password file is also saved in this directory, so you do not need to specify a directory path for this file.

Because the command is run from the same system as the queue manager, the PORT, CHANNEL, and HOSTNAME parameters also are not specified.

```
asnqanalyze gather DATABASE=SAMPLE USERID="db2admin"
PASSFILE="pwdfile.txt" GETCOLS=OFF
```

Example 2

The following command analyzes the Q Apply control tables, DB2 catalogs, and queue manager on the Q Apply server TARGET, which is a remote DB2 for z/OS subsystem DSN4 that is cataloged locally as DSN4_TGT. The Q Apply diagnostic log file was sent by FTP to the same directory from where the command is run and where the password file is stored.

This command also specifies that six days of data should be gathered from the IBMQREP_APPLYTRACE and IBMQREP_APPLYMON tables and that the output should be an uncompressed XML file named `target.zip`.

```
asnqanalyze gather DATABASE=TARGET SUBSYSTEM=DSN4
USERID=db2admin PASSFILE=pwdfile.txt HOSTNAME="Z145HG"
LOGDAYS=6 ZIP=OFF -o target.zip
```

Example 3

To produce an HTML report that uses the output of the **gather** command in Example 1:

```
asnqanalyze report SAMPLE.ASN.xml
```

Example 4

The following examples show two different ways to use the asnqanalyze **generate html report** command.

- Generate a raw HTML report for database V95DB, control tables under schema ASN. Get the information from the IBMQREP_SRC_COLS and IBMQREP_TRG_COLS tables and also the monitor tables:

  ```
  ASNQANALYZE GENERATE HTML REPORT DATABASE=V95DB
   GETCOLS=ON GETMONITOR=ON -o qanalyzer_report.html
  ```

- Generate a raw HTML report for the control tables in database V95DB, under schema ASNTEMP.

  ```
  ASNQANALYZE GENERATE HTML REPORT DATABASE=v95db SCHEMA=ASNTEMP
  ```

(The output is sent to the file V95DB.ASNTEMP.html)

asnqmfmt: Formatting and viewing Q replication and publishing messages

Use the **asnqmfmt** command to format and view messages that are used in Q Replication and Event Publishing. Run this command on Linux, UNIX, Windows, or UNIX System Services (USS) on z/OS at an operating system prompt or in a shell script.

Important: The format of Q Capture restart messages changed with z/OS APAR PK78112 or Linux, UNIX, and Windows Version 9.5 Fix Pack 3. You cannot use an asnqmfmt program that is older than this level to format messages that are in the new format. Also, a newer asnqmfmt program cannot format the older restart messages.

z/OS You can operate the message formatting program with JCL. You can find sample JCL in the sample qrhlqual.SASNSAMP(ASNQMFMT).

For Event Publishing, you must run the **asnqmfmt** command from the directory that contains the mqcap.xsd schema definition file. The default location for the file is `SQLLIB/samples/repl/q`.

Syntax

```
►►──asnqmfmt──queue_name──queue_manager_name──┬──────────────────────────┬────────►
                                              └─-o─┬───────────────┬─────┘
                                                   └─filepath_name─┘

►──┬──────┬──┬─────────────┬──┬─────────┬──┬───────┬──────────────────────────────►
   └─-hex─┘  └─-l──number──┘  └─-delmsg─┘  └─-mqmd─┘

►──┬──────────────────────────────┬──┬───────┬───────────────────────────────────►◄
   └─-oenc──output_encoding_name──┘  └─-help─┘
```

Parameters

Table 56 describes the invocation parameters for **asnqmfmt**:

Table 56. asnqmfmt invocation parameter definitions

Parameter	Definition
queue_name	Specifies the name of a WebSphere MQ queue whose messages you want to format, view, and optionally delete.
queue_manager_name	Specifies the name of a WebSphere MQ queue manager where the queue is defined.
-o *filepath_name*	Specifies the name of the file that contains the formatted output. If the **-o** parameter is not specified, the formatted messages will be written to the standard output (stdout). The output file will be written to a z/OS data set if its name starts with //. By default, the file is created in the directory from which the **asnqmfmt** command was invoked. You can change the directory by specifying a path with the file name.
-hex	Specifies that the messages are formatted in hexadecimal. If you do not specify this parameter, messages will be displayed according to their message format type, either compact, delimited, or XML.
-l *number*	Specifies the number of messages that you want to format.
-delmsg	Specifies that the messages will be deleted from the queue after they are formatted.
-mqmd	Specifies that you want to view the WebSphere MQ message descriptor for each message that is formatted.
-oenc *output_encoding_name*	Specifies a code page to be used for formatting the messages. If you do not specify this parameter, messages will be formatted in the default code page for the operating system where the command is invoked.
-help	Displays the valid command parameters with descriptions.

Examples for asnqmfmt

The following examples illustrate how to use the **asnqmfmt** command.

Example 1

To view on the standard output all messages that are on send queue Q1 that is defined in queue manager QMGR1:

```
asnqmfmt Q1 QMGR1
```

Example 2

To view all messages that are on send queue Q1 in a file called `Q1_messages` that is stored in the `C:\qrepl` directory (Windows):

```
asnqmfmt Q1 QMGR1 -o C:\qrepl\Q1_messages
```

Example 3

To view on the standard output a hexadecimal version of all messages that are on administration queue ADMNQ1 that is defined in the queue manager QMGR1:

```
asnqmfmt ADMNQ1 QMGR1 -hex
```

Example 4

To view on the standard output the message body and message descriptor of all messages that are on administration queue ADMNQ1 that is defined in the queue manager QMGR1, and then delete the messages from the queue:

```
asnqmfmt ADMNQ1 QMGR1 -delmsg -mqmd
```

Example 5

To view the first 100 messages that are on receive queue Q2 that is defined in the queue manager QMGR2 in a file called `Q2_messages` that is stored in the `C:\qrepl` directory (Windows):

```
asnqmfmt Q2 QMGR2 -l 100 -o C:\qrepl\Q2_messages
```

asnqxmfmt: Formatting and viewing Event Publishing messages (z/OS)

Use the **asnqxmfmt** command to format and view delimited or XML messages that are used in Event Publishing. This command runs on z/OS and UNIX System Services (USS) for z/OS only. The STEPLIB must include the WebSphere MQ and XML Toolkit libraries if they are not installed in the LNKLST. Currently **asnqxmfmt** requires XML4C 1.4 version (HXML14A.SIXMMOD1).

z/OS You can operate the message formatting program with JCL. You can find sample JCL in the sample qrhlqual.SASNSAMP(ASNQXMFMT).

Syntax

```
>>──asnqxmfmt──queue_name──queue_manager_name──┬──────────────────────┬──────────►
                                               └─-o─┬───────────────┬─┘
                                                    └─filepath_name─┘
```

```
►─┬──────┬──┬───────────┬──┬─────────┬──┬───────┬─────────────────────────────►
  └─-hex─┘  └─-l──number─┘  └─-delmsg─┘  └─-mqmd─┘

►─┬──────────────────────────────┬──┬───────┬─────────────────────────────────►◄
  └─-oenc──output_encoding_name──┘  └─-help─┘
```

Parameters

Table 57 describes the invocation parameters for **asnqxmfmt**:

Table 57. asnqxmfmt invocation parameter definitions

Parameter	Definition
queue_name	Specifies the name of a WebSphere MQ queue whose messages you want to format, view, and optionally delete.
queue_manager_name	Specifies the name of a WebSphere MQ queue manager where the queue is defined.
-o *filepath_name*	Specifies the name of the file that contains the formatted output. If the **-o** parameter is not specified, the formatted messages will be written to the standard output (stdout). The output file will be written to a z/OS data set if its name starts with //. By default, the file is created in the directory from which the **asnqxmfmt** command was invoked. You can change the directory by specifying a path with the file name.
-hex	Specifies that the messages are formatted in hexadecimal. If you do not specify this parameter, messages will be displayed according to their message format type, either delimited or XML.
-l *number*	Specifies the number of messages that you want to format.
-delmsg	Specifies that the messages will be deleted from the queue after they are formatted.
-mqmd	Specifies that you want to view the WebSphere MQ message descriptor for each message that is formatted.
-oenc *output_encoding_nam*	Specifies a code page to be used for formatting the messages. If you do not specify this parameter, messages will be formatted in the default code page for the operating system where the command is invoked.
-help	Displays the valid command parameters with descriptions.

Examples for asnqxmfmt

The following examples illustrate how to use the **asnqxmfmt** command.

Example 1

To view on the standard output all messages that are on send queue Q1 that is defined in queue manager QMGR1:

```
asnqxmfmt Q1 QMGR1
```

Example 2

To view on the standard output a hexadecimal version of all messages that are on administration queue ADMNQ1 that is defined in the queue manager QMGR1:

```
asnqxmfmt ADMNQ1 QMGR1 -hex
```

Example 3

To view on the standard output the message body and message descriptor of all messages that are on administration queue ADMNQ1 that is defined in the queue manager QMGR1, and then delete the messages from the queue:

```
asnqxmfmt ADMNQ1 QMGR1 -delmsg -mqmd
```

Chapter 25. Control tables for Q Replication and Event Publishing

Control tables are relational database tables that are used to store information for the Q Replication and Event Publishing programs. These control tables are stored at the Q Capture server, Q Apply server, and Monitor control server.

Control tables at the Q Capture server

The control tables at the Q Capture server contain information about data sources, options for Q subscriptions or publications, operating parameters for the Q Capture program, Q Capture performance statistics, and other metadata. These tables are built according to options that you specify in the replication administration tools.

Table 58 describes the control tables at the Q Capture server.

Table 58. Control tables at the Q Capture server

Table name	Description
“IBMQREP_ADMINMSG table” on page 460	An internal table that contains administrative messages received by a Q Capture program.
“IBMQREP_CAPENQ table” on page 461	Ensures that only one Q Capture program with a given schema is running per Q Capture server.
“IBMQREP_CAPENVINFO table” on page 461	Stores environment variables and other information that replication tools use to access remote programs.
“IBMQREP_CAPMON table” on page 462	Contains statistics about the performance of a Q Capture program.
“IBMQREP_CAPPARMS table” on page 464	Contains parameters that you can specify to control the operations of a Q Capture program.
“IBMQREP_CAPQMON table” on page 469	Contains statistics about the performance of a Q Capture program for each send queue.
“IBMQREP_CAPTRACE table” on page 471	Contains informational, warning, and error messages from a Q Capture program.
“IBMQREP_COLVERSION table” on page 472	Enables the Q Capture and Capture programs to keep track of different versions of a source table.
“IBMQREP_EOLFLUSH table” on page 473	An internal table that forces a flush of the log read buffer for Oracle sources.
“IBMQREP_EXCLSCHEMA table” on page 473	Contains information about tables that are excluded from schema-level Q subscriptions.
IBMQREP_IGNTRAN table	Can be used to inform the Q Capture program about transactions that you do not want to be captured from the DB2 recovery log or Oracle redo log.
IBMQREP_IGNTRANTRC table	Records information about transactions that were specified to be ignored.
“IBMQREP_PART_HIST table (Linux, UNIX, Windows)” on page 476	Maintains a history of changes to partitioned source tables on Linux, UNIX, and Windows systems. This table is used by Q Replication and SQL replication.

Table 58. Control tables at the Q Capture server (continued)

Table name	Description
"IBMQREP_SCHEMASUBS table" on page 477	Contains information about schema-level Q subscriptions, including the replication queue map that they use, the saved profile if one is used, and the state.
"IBMQREP_SENDQUEUES table" on page 479	Contains information about the WebSphere MQ queues that a Q Capture program uses to send transaction, row operation, large object, or informational messages.
"IBMQREP_SIGNAL table" on page 483	Contains signals that are used to prompt a Q Capture program. These signals are inserted by a user or subscribing application, or by a Q Capture program after it receives a control message from the Q Apply program or a subscribing application.
"IBMQREP_SRC_COLS table" on page 488	Identifies columns in the source table that are replicated or published for a Q subscription or publication.
"IBMQREP_SRCH_COND table" on page 489	An internal table that a Q Capture program uses to evaluate the search condition that you specified for a Q subscription or publication.
"IBMQREP_SUBS table" on page 490	Contains information about Q subscriptions and publications, including subscription type, source tables, search conditions, data sending options, target loading options, and states.
"IBMQREP_TABVERSION table" on page 497	Enables the Q Capture and Capture programs to keep track of different versions of a source table.

IBMQREP_ADMINMSG table

The IBMQREP_ADMINMSG table is an internal table that a Q Capture program uses to record the time and identifier of administrative messages that it receives.

Server: Q Capture server

Default schema: ASN

Primary key: MQMSGID

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 59 provides a brief description of the columns in the IBMQREP_ADMINMSG table.

Table 59. Columns in the IBMQREP_ADMINMSG table

Column name	Description
MQMSGID	**Data type:** CHAR(24) FOR BIT DATA; **Nullable:** No The WebSphere MQ message identifier of the message.
MSG_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Capture server when the message was inserted into this table. Default: Current timestamp.

IBMQREP_CAPENQ table

The IBMQREP_CAPENQ table ensures the uniqueness of the schema that is used to identify a Q Capture program and its control tables.

Server: Q Capture server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The IBMQREP_CAPENQ table ensures that:

- Linux UNIX Windows For DB2 for Linux, UNIX, and Windows, only one Q Capture program with a given schema is running per database.
- z/OS For non-data-sharing DB2 for z/OS, only one Q Capture program with a given schema is running per subsystem.
- z/OS For data-sharing DB2 for z/OS, only one Q Capture program with a given schema is running per data-sharing group.

When a Q Capture program is running, it exclusively locks this table. Starting the Q Capture program twice will place the second instance on a lock wait over this table. The table is created empty.

Table 60 provides a brief description of the column in the IBMQREP_CAPENQ table.

Table 60. Column in the IBMQREP_CAPENQ table

Column name	Description
LOCKNAME	**Data type:** INTEGER; **Nullable:** Yes This column contains no data.

IBMQREP_CAPENVINFO table

The IBMQREP_CAPENVINFO table contains eight rows that are used to store the value of runtime environment variables and other information that the replication administration tools use to access remote programs.

Server: Q Capture server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 61 on page 462 provides a brief description of the columns in the IBMQREP_CAPENVINFO table.

Table 61. Columns in the IBMQREP_CAPENVINFO table

Column name	Description
NAME	**Data type:** VARCHAR(30); **Nullable:** No **STARTTIME** The timestamp when the Q Capture program started. **HOSTNAME** The TCP/IP host name of the server where the Q Capture program is running. **LOGFILE** The path and file name of the Q Capture diagnostic log file. **TMPDIR** The path of the directory where the Inter-Process Communication (IPC) key of the Q Capture program is located. **ASNUSEMQCLIENT** The value of the Q Replication **ASNUSEMQCLIENT** environment variable. **MQSERVER** The value of the WebSphere MQ **MQSERVER** environment variable. **MQCHLLIB** The value of the WebSphere MQ **MQCHLLIB** environment variable. **MQCHLTAB** The value of the WebSphere MQ **MQCHLTAB** environment variable.
VALUE	**Data type:** VARCHAR(3800); **Nullable:** Yes For each row in the IBMQREP_CAPENVINFO table, the VALUE column contains the value that is associated with the corresponding NAME column.

IBMQREP_CAPMON table

The Q Capture program inserts a row in the IBMQREP_CAPMON table to record performance statistics during a given period of time. The value that you specify for MONITOR_INTERVAL in the IBMQREP_CAPPARMS table tells the Q Capture program how often to insert a row into the IBMQREP_CAPMON table. The MONITOR_LIMIT value indicates the number of minutes that rows remain in this table before they are eligible for pruning.

Server: Q Capture server

Default schema: ASN

Non-unique index: MONITOR_TIME DESC

Important: Do not use SQL to alter this table. Altering this table inappropriately can cause unexpected results and loss of data.

Table 62 on page 463 provides a brief description of the columns in the IBMQREP_CAPMON table.

Table 62. Columns in the IBMQREP_CAPMON table

Column name	Description
MONITOR_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server when the row was inserted into this table.
CURRENT_LOG_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server of the latest database commit that was seen by the Q Capture log reader.
CAPTURE_IDLE	This column is deprecated.
CURRENT_MEMORY	**Data type:** INTEGER; **Nullable:** No The amount of memory (in bytes) that the Q Capture program used to reconstruct transactions from the log.
ROWS_PROCESSED	**Data type:** INTEGER; **Nullable:** No The number of rows (individual insert, update, or delete operations) that the Q Capture program read from the log.
TRANS_SKIPPED	**Data type:** INTEGER; **Nullable:** No The number of transactions (containing changed rows) that were not put on queues because the changes were to columns that are not part of a Q subscription or publication (the ALL_CHANGED_ROWS parameter in the IBMQREP_SUBS table was set to No, the default).
TRANS_PROCESSED	**Data type:** INTEGER; **Nullable:** No The number of transactions that the Q Capture program processed.
TRANS_SPILLED	**Data type:** INTEGER; **Nullable:** No The number of transactions that the Q Capture program spilled to a file after exceeding the MEMORY_LIMIT threshold.
MAX_TRANS_SIZE	**Data type:** INTEGER; **Nullable:** No The largest transaction, in bytes, that the Q Capture program processed.
QUEUES_IN_ERROR	**Data type:** INTEGER; **Nullable:** No The number of queues that were not accepting messages.
RESTART_SEQ	**DB2 sources** **Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** No **Oracle sources** **Data type:** RAW(10); **Nullable:** No The log sequence number at which the Q Capture program starts during a warm restart. This value represents the earliest log sequence number that the Q Capture program found for which a commit or abort record has not yet been found.
CURRENT_SEQ	**DB2 sources** **Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** No **Oracle sources** **Data type:** RAW(10); **Nullable:** No The most recent log sequence number in the recovery log that the Q Capture program read.

Table 62. Columns in the IBMQREP_CAPMON table (continued)

Column name	Description
LAST_EOL_TIME	**Data type:** TIMESTAMP; **Nullable:** Yes The time at the Q Capture control server when the Q Capture program reached the end of the log.
LOGREAD_API_TIME	**Data type:** INTEGER; **Nullable:** Yes The number of milliseconds that the Q Capture program spent using the DB2 log read application program interface (API) to retrieve log records.
NUM_LOGREAD_CALLS	**Data type:** INTEGER; **Nullable:** Yes The number of log read API calls that Q Capture made. For Oracle sources, this counter shows the number of times that Q Capture fetched log records from the LogMiner utility.
NUM_END_OF_LOGS	**Data type:** INTEGER; **Nullable:** Yes The number of times that Q Capture reached the end of the log.
LOGRDR_SLEEPTIME	**Data type:** INTEGER; **Nullable:** Yes The number of milliseconds that the Q Capture log reader thread slept because there were no changes to capture or because Q Capture is operating at its memory limit.
MQCMIT_TIME	**Data type:** INTEGER; **Nullable:** No, with default The number of milliseconds during the monitor interval that the Q Capture program spent interacting with the WebSphere MQ API for committing messages on all send queues.

IBMQREP_CAPPARMS table

The IBMQREP_CAPPARMS table contains a single row that you can modify to control the operations of the Q Capture program. For example, you can set the processing method that the Q Capture program uses when it starts. Or, you can set the amount of time that the Q Capture program waits before committing messages that are on send queues. The Q Capture program reads changes to this table only during startup.

Server: Q Capture server

Default schema: ASN

Unique index: QMGR

This table contains information that you can update by using SQL.

Important: If this table has no row, or more than one row, the Q Capture program will not run.

Table 63 on page 465 provides a brief description of the columns in the IBMQREP_CAPPARMS table.

Table 63. Columns in the IBMQREP_CAPPARMS table

Column name	Description
QMGR	**Data type:** VARCHAR(48); **Nullable:** No The name of the WebSphere MQ queue manager that the Q Capture program uses.
REMOTE_SRC_SERVER	**Data type:** VARCHAR(18); **Nullable:** Yes Reserved for future use.
RESTARTQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the queue that stores restart messages for the Q Capture log reader. The name must be unique for each Q Capture program.
ADMINQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the queue that receives control messages from the Q Apply program or a user application. The name must be unique for each Q Capture program.
STARTMODE	**Data type:** VARCHAR(6); **Nullable:** No, with default The steps that the Q Capture program takes when it starts. **cold** The Q Capture program clears the restart queue and administration queue, and starts processing all Q subscriptions or publications that are in N (new) or A (active) state. With a cold start, the Q Capture program starts reading the DB2 recovery log or Oracle active redo log at the end. **warmsi (default)** The Q Capture program starts reading the log at the point where it left off, except if this is the first time you are starting it. In that case the Q Capture program switches to a cold start. The warmsi start mode ensures that the Q Capture program cold starts only when it initially starts. **warmns** The Q Capture program starts reading the log at the point where it left off. If it cannot warm start, it does not switch to cold start. The warmns start mode prevents the Q Capture program from cold starting unexpectedly. When the Q Capture program warm starts, it resumes processing where it ended. If errors occur after the Q Capture program starts, the program terminates and leaves all tables intact. During warm starts, the Q Capture program will process only Q subscriptions or publications that are not in I (inactive) state.
MEMORY_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The amount of memory, in megabytes, that the Q Capture program can use to build transactions. After this allocation is used, in-memory transactions spill to a file. Default: 0 for z/OS, 500 MB for Linux, UNIX, and Windows z/OS A value of 0 tells the Q Capture program to calculate a memory allocation from the Q Capture region size in the JCL or started task.
COMMIT_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in milliseconds, the Q Capture program issues an MQCMIT call. This call signals the WebSphere MQ queue manager to make data messages and informational messages that have been placed on queues available to the Q Apply program or user applications. Default: 500

Table 63. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
AUTOSTOP	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program stops when it reaches the end of the active DB2 log. **Y** The Q Capture program stops when it reaches the end of the active DB2 log. **N (default)** The Q Capture program continues running when it reaches the end of the active DB2 log.
MONITOR_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in milliseconds, the Q Capture program adds rows to the IBMQREP_CAPMON and IBMQREP_CAPQMON tables. Default: 60000 milliseconds (1 minute) on z/OS; 30000 milliseconds (30 seconds) on Linux, UNIX, and Windows
MONITOR_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_CAPMON and the IBMQREP_CAPQMON tables before they are eligible for pruning. At each pruning interval, rows in these tables are pruned if they are older than this limit based on the current timestamp. Default: 10080
TRACE_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_CAPTRACE table before they can be pruned. At each pruning interval, rows in the IBMQREP_CAPTRACE table are pruned if they are older than this limit based on the current timestamp. Default: 10080
SIGNAL_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_SIGNAL table before they can be pruned. At each pruning interval, rows in the IBMQREP_SIGNAL table are pruned if they are older than this limit based on the current timestamp. Default: 10080
PRUNE_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Capture program automatically prunes rows in the IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_SIGNAL, and IBMQREP_CAPTRACE tables that are no longer needed. Default: 300
SLEEP_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default The number of milliseconds that the Q Capture program is idle after processing the active log and any transactions that remain in memory. Default: 500 milliseconds (0.5 seconds) for DB2 sources; 2000 milliseconds (2 seconds) for Oracle sources
LOGREUSE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program reuses the Q Capture log file or appends to it. **Y** On restart, the Q Capture program reuses its log file by clearing the file then writing to the blank file. **N (default)** The Q Capture program appends new information to an existing Q Capture log file when it restarts.

Table 63. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
LOGSTDOUT	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program sends log messages to outputs other than its log file. **Y** The Q Capture program sends log messages to both the log file and console (stdout). **N (default)** The Q Capture program directs most log file messages to the log file only. Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file.
TERM	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program stops if the source DB2 or queue manager is unavailable. **Y (default)** The Q Capture program stops if DB2 or the queue manager are unavailable. **N** The Q Capture program keeps running if DB2 or the queue manager are unavailable. When DB2 or the queue manager are available, Q Capture begins sending transactions where it left off without requiring you to restart the program. **Oracle sources:** The **`term`** parameter has no effect on Oracle sources. The default value for Oracle capture is N, which indicates that the Q Capture program continues to run if Oracle is unavailable.
CAPTURE_PATH	**Data type:** VARCHAR(1040); **Nullable:** Yes The path where the files that are created by the Q Capture program are stored.
ARCH_LEVEL	**Data type:** CHAR(4); **Nullable:** No, with default The version of the control tables. For Version 9.7 Fix Pack 3, the value is 0973. Other ARCH_LEVEL values are 0802, 0901, 0905, and 0907. **Attention:** When updating the IBMQREP_CAPPARMS table, do not change the value in this column.
COMPATIBILITY	**Data type:** CHAR(4); **Nullable:** No, with default The version of messages that the Q Capture program sends to a Q Apply program. **0907 (default)** Version 9.7 messages are sent. This compatibility value is also used when the source and target servers are at Replication Server Version 10 on z/OS. **0905** Version 9.5 messages are sent. **0901** Version 9.1 messages are sent. **0802** Version 8 messages are sent.

Table 63. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
LOB_SEND_OPTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates how the Q Capture program sends LOB data. **I (default)** Inline. The LOB values are sent within the transaction message. The inlined LOB values can improve performance. **S** Separate. The LOB values are sent in one or more separate LOB messages that follow the transaction message.
QFULL_NUM_RETRIES	**Data type:** INTEGER; **Nullable:** No, with default This column specifies the number of times that the Q Capture program will retry an MQPUT for the send queue. Default: 30; Maximum: 1000
QFULL_RETRY_DELAY	**Data type:** INTEGER; **Nullable:** No, with default This column specifies the amount of time that the Q Capture program will sleep before the next retry. The value is specified in milliseconds. Default: 250 (milliseconds); Minimum: 20; Maximum: 3600000
MSG_PERSISTENCE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that specifies whether a Q Capture program writes persistent (logged) or nonpersistent (unlogged) messages to WebSphere MQ queues. **Y (default)** Q Capture write persistent messages to all queues. The messages are logged by the queue manager and can be recovered. **N** Q Capture write nonpersistent messages to all queues. The messages are not logged and cannot be recovered.
LOGRDBUFSZ	**Data type:** INTEGER; **Nullable:** No, with default The size of the buffer in KB that the Q Capture program passes to DB2 when Q Capture retrieves log records. DB2 fills the buffer with available log records that Q Capture has not retrieved. Default: DB2 for z/OS 66KB; DB2 for Linux, UNIX, and Windows 256KB.
CAPTURE_ALIAS	**Data type:** VARCHAR(8); **Nullable:** Yes The alias name for the database or subsystem that is used as the Q Capture server. This is the alias as cataloged on the system where the replication administration tools run and used to connect to the source database or subsystem to create Q Capture control tables. This column is populated by the Replication Center or ASNCLP command-line program when control tables are created.
STARTALLQ	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Capture program whether to activate all send queues that are not already in active state when Q Capture starts. Send queues that are already active are always processed when Q Capture starts. **Y (default)** When the Q Capture program starts, it activates all send queues that are not already in active (A) state. **N** When the Q Capture program starts, it does not activate send queues that are in inactive (I) state.

Table 63. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
WARNTXSZ	**Data type:** INTEGER; **Nullable:** No Specifies whether the Q Capture program issues warning messages when it encounters transactions that are larger than a specified size. You provide a threshold value in megabytes, and transactions that exceed the threshold prompt a warning message. Q Capture issues multiple warning messages if the transaction size is a multiple of the WARNTXSZ value. For example, if you set WARNTXSZ to 10 MB and Q Capture encounters a 30 MB transaction, three warnings are issued (one for 10 MB, one for 20 MB, and one for 30 MB). The default value of 0 MB means warnings are never issued.
LOGRD_ERROR_ACTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that specifies the action that the Q Capture program takes when it encounters a compression dictionary error while reading the recovery log. A "permanent" error means that the table space was reorganized twice with KEEPDICTIONARY=NO. A "transient" error means all other compression dictionary errors. **D (default)** For a permanent dictionary error, deactivate the Q subscription. For a transient dictionary error, stop the Q Capture program. **E** For either type of error, deactivate the Q subscription. **S** For either type of error, stop the Q Capture program.

IBMQREP_CAPQMON table

Server: Q Capture server

Default schema: ASN

Non-unique index: MONITOR_TIME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The Q Capture program inserts rows in the IBMQREP_CAPQMON table to record statistics about its performance for each send queue. The value that you specify for MONITOR_INTERVAL in the IBMQREP_CAPPARMS table indicates how frequently the Q Capture program makes these inserts. The MONITOR_LIMIT value sets the number of minutes that rows remain in the IBMQREP_CAPQMON table before they are eligible for pruning.

Table 64 provides a brief description of the columns in the IBMQREP_CAPQMON table.

Table 64. Columns in the IBMQREP_CAPQMON table

Column name	Description
MONITOR_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server when the row was inserted into this table.

Table 64. Columns in the IBMQREP_CAPQMON table (continued)

Column name	Description
SENDQ	**Data type:** VARCHAR(97); **Nullable:** No The name of the send queue that this row of monitor statistics tells you about.
ROWS_PUBLISHED	**Data type:** INTEGER; **Nullable:** No The number of rows (individual insert, update, or delete operations) that the Q Capture program put on this send queue.
TRANS_PUBLISHED	**Data type:** INTEGER; **Nullable:** No The number of transactions that the Q Capture program put on this send queue.
CHG_ROWS_SKIPPED	**Data type:** INTEGER; **Nullable:** No The number of changed rows that were not put on this send queue because the changes were to columns that are not part of a Q subscription or publication (the ALL_CHANGED_ROWS parameter in the IBMQREP_SUBS table was set to No, the default).
DELROWS_SUPPRESSED	**Data type:** INTEGER; **Nullable:** No The number of delete row operations that were not put on this send queue because the Q subscription or publication was created with the option to suppress deletes.
ROWS_SKIPPED	**Data type:** INTEGER; **Nullable:** No The number of rows that the Q Capture program did not transmit to this send queue because they did not meet the search condition defined in the Q subscription or publication.
LOBS_TOO_BIG	**Data type:** INTEGER; **Nullable:** No The number of LOB values that did not fit in a transaction message for a monitor interval. If the error action for the queue is set to E, an empty LOB value is sent. If the error action for the queue is set to S, the Q Capture program stops.
XMLDOCS_TOO_BIG	**Data type:** INTEGER; **Nullable:** No The number of XML documents that did not fit in a transaction message for a monitor interval. If the error action for the queue is set to E, a placeholder XML document is inserted instead. If the error action for the queue is set to S, the Q Capture program stops.
QFULL_ERROR_COUNT	**Data type:** INTEGER; **Nullable:** No The number of times that the Q Capture program retried putting messages (MQPUT) on the send queue that is specified in the SENDQ column.
MQ_BYTES	**Data type:** INTEGER; **Nullable:** Yes The number of bytes put on the send queue during the monitor interval, including data from the source table and the message header.
MQ_MESSAGES	**Data type:** INTEGER; **Nullable:** Yes The number of messages put on the send queue during the monitor interval.
CURRENT_SEQ	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** Yes The most recent log sequence number in the recovery log that the Q Capture program read for this send queue. Q Capture updates this column if the send queue is active.

Table 64. Columns in the IBMQREP_CAPQMON table (continued)

Column name	Description
RESTART_SEQ	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** Yes The log sequence number from which the Q Capture program starts putting messages on this send queue during a warm restart. This value represents the earliest log sequence number that the Q Capture program found that did not have a commit or abort record. Q Capture updates this column if the send queue is active.
MQPUT_TIME	**Data type:** INTEGER; **Nullable:** No, with default The number of milliseconds during the monitor interval that the Q Capture program spent interacting with the WebSphere MQ API for putting messages on this send queue.

IBMQREP_CAPTRACE table

The IBMQREP_CAPTRACE table contains informational, warning, and error messages from the Q Capture program.

Server: Q Capture server

Default schema: ASN

Non-unique index: TRACE_TIME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 65 provides a brief description of the columns in the IBMQREP_CAPTRACE table.

Table 65. Columns in the IBMQREP_CAPTRACE table

Column name	Description
OPERATION	**Data type:** CHAR(8); **Nullable:** No The type of message from the Q Capture program: **INFO** Describe actions that the Q Capture program takes. **WARNING** Describe conditions that could cause errors for the Q Capture program. **ERROR** Describe errors encountered by the Q Capture program.
TRACE_TIME	**Data type:** TIMESTAMP; **Nullable:** No The time at the Q Capture server that the message was put on a send queue.
DESCRIPTION	**Data type:** VARCHAR(1024); **Nullable:** No **Data type:** INTEGER; **Nullable:** Yes The reason code for the replication or Event Publishing error message. The ASN message ID followed by the message text. This column contains English-only text.
REASON_CODE	**Data type:** INTEGER; **Nullable:** Yes The reason code for the replication or publishing error message.

Table 65. Columns in the IBMQREP_CAPTRACE table (continued)

Column name	Description
MQ_CODE	**Data type:** INTEGER; **Nullable:** Yes The reason code for the WebSphere MQ error message.

IBMQREP_COLVERSION table

The IBMQREP_COLVERSION table is used by the Q Capture and Capture programs to keep track of different versions of a source table.

Server: Q Capture server

Default schema: ASN

Index: LSN, TABLEID1, TABLEID2, POSITION

Index: TABLEID1 ASC, TABLEID2 ASC

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The Q Capture or Capture program inserts rows into this table when the Q subscription or registration for a source table is first activated, and then each time the source table is altered.

Table 66 provides a brief description of the columns in the IBMQREP_COLVERSION table.

Table 66. Columns in the IBMQREP_COLVERSION table

Column name	Description
LSN	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** No The point in the DB2 recovery log where the Q Capture program or Capture program detected a new version of the source table.
TABLEID1	**Data type:** SMALLINT; **Nullable:** No The object identifier (OBID) in SYSIBM.SYSTABLES.
TABLEID2	**Data type:** SMALLINT; **Nullable:** No The database identifier (DBID) in SYSIBM.SYSTABLES.
POSITION	**Data type:** SMALLINT; **Nullable:** No The ordinal position of the column in the table, starting at 0 for the first column in the table.
NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the column.
TYPE	**Data type:** SMALLINT; **Nullable:** No An internal data type identifier for the column (SQLTYPE in SYSIBM.SYSCOLUMNS).
LENGTH	**Data type:** INTEGER; **Nullable:** No The maximum data length for this column.

Table 66. Columns in the IBMQREP_COLVERSION table (continued)

Column name	Description
NULLS	**Data type:** CHAR(1); **Nullable:** No A flag that identifies whether the column allows null values: **Y** The column allows null values. **N** The column does not allow null values.
DEFAULT	**Data type:** VARCHAR(1536); **Nullable:** Yes The default value of the column (DEFAULTVALUE) in SYSIBM.SYSCOLUMNS. This column is NULL if there is no default.
CODEPAGE	**Data type:** INTEGER; **Nullable:** Yes The code page that is used for data in this column. The value is 0 if the column is defined as FOR BIT DATA or is not a string type. Default: NULL
SCALE	**Data type:** INTEGER; **Nullable:** Yes The scale of decimal data in decimal columns. The value is 0 for non-decimal columns. Default: NULL

IBMQREP_EOLFLUSH table

IBMQREP_EOLFLUSH is an internal table that the Q Capture program writes to when Oracle LogMiner has not responded within the time that is specified by the **commit_interval** parameter. Writing to this table helps determine if any buffered but unreturned log records are available for Q Capture to process.

Server: Q Capture server for Oracle sources

Default schema: ASN

Important: Do not use SQL to alter this table. Altering this table inappropriately can cause unexpected results. The format of this table can change without notice.

The following table describes the column in the IBMQREP_EOLFLUSH table.

Table 67. Column in the IBMQREP_EOLFLUSH table

Column name	Description
EOL_TIMEOUT	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server when the row was inserted into this table.

IBMQREP_EXCLSCHEMA table

The IBMQREP_EXCLSCHEMA table stores information about tables that are excluded from schema-level Q subscriptions. Each row provides the table names for which SQL operations are not to be replicated even if a corresponding schema-level Q subscription exists in the IBMQREP_SCHEMASUBS table. By default, DB2 and replication catalog tables are excluded.

Server: Q Capture server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 68 provides a brief description of the columns in the IBMQREP_EXCLSCHEMA table.

Table 68. Columns in the IBMQREP_EXCLSCHEMA table

Column name	Description
SCHEMA_NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the schema within which certain tables are excluded from a schema-level Q subscription. Wild cards are not allowed in this column other than a single percentage sign (%) to indicate that the replication programs should review all schemas for tables to exclude from schema-level Q subscriptions. **z/OS:** The value in the CREATOR column of the SYSIBM.SYSTABLES system table for the source tables that are excluded from a schema-level Q subscription.
OBJECT_NAME	**Data type:** VARCHAR(128); **Nullable:** No, with default. Identifies the tables within a schema that are excluded from replication even if a corresponding schema-level Q subscription exists. You can use a percentage sign (%) as a wild card to identify multiple tables within one or all schemas. By default, DB2 and replication catalog tables are automatically excluded. Default: %
QMAPNAME	**Data type:** VARCHAR(128); **Nullable:** No The replication queue map that is used for the schema-level Q subscription.

IBMQREP_IGNTRAN table

The IBMQREP_IGNTRAN table can be used to inform the Q Capture or Capture program about transactions that you do not want to be captured from the DB2 recovery log. You use SQL to insert rows in the table that inform the programs to ignore transactions based on authorization ID, authorization token (z/OS only), or plan name (z/OS only).

Server: Q Capture server, Capture control server

Default schema: ASN

Unique index: AUTHID ASC, AUTHTOKEN ASC, PLANNAME ASC

Table 69 provides a brief description of the columns in the IBMQREP_IGNTRAN table.

Table 69. Columns in the IBMQREP_IGNTRAN table

Column name	Description
AUTHID	**Data type:** CHAR(128); **Nullable:** Yes The primary authorization ID for the transaction that you want to ignore.
AUTHTOKEN	**Data type:** CHAR(30); **Nullable:** Yes z/OS The authorization token (job name) for the transaction that you want to ignore.
PLANNAME	**Data type:** CHAR(8); **Nullable:** Yes z/OS The plan name for the transaction that you want to ignore.

Table 69. Columns in the IBMQREP_IGNTRAN table (continued)

Column name	Description
IGNTRANTRC	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Capture or Capture program whether to trace transactions that were ignored based on the AUTHID, AUTHTOKEN, or PLANNAME value that was specified in the IBMQREP_IGNTRAN table: **N (default)** Tracing is disabled. **Y** Tracing is enabled. Each time a transaction is ignored, a row is inserted into the IBMQREP_IGNTRANTRC table and a message is issued.

IBMQREP_IGNTRANTRC table

The IBMQREP_IGNTRANTRC table records information about transactions that were specified to be ignored.

Server: Q Capture server, Capture control server

Default schema: ASN

Index: IGNTRAN_TIME ASC

Important: Do not alter this table by using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

A row is inserted in the IBMQREP_IGNTRANTRC table when a transaction is ignored in the DB2 recovery log. This table is pruned according to the **`trace_limit`** parameter for the Q Capture or Capture program.

Table 70 provides a brief description of the columns in the IBMQREP_IGNTRANTRC table.

Table 70. Columns in the IBMQREP_IGNTRANTRC table

Column name	Description
IGNTRAN_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The time when the transaction was ignored. Default: Current timestamp
AUTHID	**Data type:** CHAR(128); **Nullable:** Yes The primary authorization ID of the transaction that was ignored.
AUTHTOKEN	**Data type:** CHAR(30); **Nullable:** Yes z/OS The authorization token (job name) for the transaction that was ignored.
PLANNAME	**Data type:** CHAR(8); **Nullable:** Yes z/OS The plan name for the transaction that was ignored.
TRANSID	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No The transaction identifier for the transaction that was ignored.

Table 70. Columns in the IBMQREP_IGNTRANTRC table (continued)

Column name	Description
COMMITLSN	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No The commit log sequence number or time sequence for the transaction that was ignored.

IBMQREP_PART_HIST table (Linux, UNIX, Windows)

The IBMQREP_PART_HIST table maintains a history of changes to partitioned source tables on Linux, UNIX, or Windows systems. This table is used by Q replication and SQL replication.

The Q Capture and Capture programs use the partition history to help them handle partition changes such as add, attach, or detach. One row, identified by a log sequence number (LSN), is inserted for each partition in the source table on two occasions:

- The first Q subscriptions or subscription-set member for the table is activated.
- The table is altered to add, attach, or detach a partition.

Older rows in this table are pruned based on the value of the **`part_hist_limit`** parameter.

Server: Q Capture server

Default schema: ASN

Primary key: LSN, TABSCHEMA, TABNAME, DATAPARTITIONID, TBSPACEID, PARTITIONOBJECTID

Index: TABSCHEMA, TABNAME, LSN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 71 provides a brief description of the columns in the IBMQREP_PART_HIST table.

Table 71. Columns in the IBMQREP_PART_HIST table

Column name	Description
LSN	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** No The point in the DB2 recovery log where the Q Capture program or Capture program detected a new version of the source table. This log sequence number (LSN) expresses one of the following values: • The LSN of the CAPSTART signal insert for the first Q subscription or subscription-set member for the table. • For Q subscriptions in N (new) state, the `restartLSN` value from the Q Capture restart message, the value of the Q Capture **`lsn`** parameter, or the LSN obtained from DB2 for the Q Capture **`migrate`**=Y parameter. Q subscriptions in N state are automatically started when Q Capture starts. • The LSN of an ADD, ATTACH, or DETACH operation on a table partition.

Table 71. Columns in the IBMQREP_PART_HIST table (continued)

Column name	Description
HISTORY_TIME	**Data type:** TIMESTAMP; **Nullable:** No The time that Q Capture or Capture detected new partition information for a table.
TABSCHEMA	**Data type:** VARCHAR(128); **Nullable:** No The schema name or high-level qualifier of the source table.
TABNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the source table.
DATAPARTITIONID	**Data type:** INTEGER; **Nullable:** No An identifier for the data partition. This value is read from the SYSCAT.DATAPARTITIONS system table.
TBSPACEID	**Data type:** INTEGER; **Nullable:** No An identifier for the table space in which the table data is stored. This value is read from the SYSCAT.DATAPARTITIONS system table.
PARTITIONOBJECTID	**Data type:** INTEGER; **Nullable:** No Identifier for the data partition within the table space. This value is read from the SYSCAT.DATAPARTITIONS system table.

IBMQREP_SCHEMASUBS table

The IBMQREP_SCHEMASUBS table contains information about schema-level Q subscriptions, including the replication queue map that they use, the saved profile if one is used, and the state.

Server: Q Capture server

Default schema: ASN

Primary key: SCHEMA_SUBNAME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 72 provides a brief description of the columns in the IBMQREP_SCHEMASUBS table.

Table 72. Columns in the IBMQREP_SCHEMASUBS table

Column name	Description
SCHEMA_SUBNAME	**Data type:** VARCHAR(64); **Nullable:** No An identifier for the schema-level Q subscription.
QMAPNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the replication queue map that is used by this schema-level Q subscription.

Table 72. Columns in the IBMQREP_SCHEMASUBS table (continued)

Column name	Description
SCHEMA_NAME	**Data type:** VARCHAR(128); **Nullable:** No An identifier for a set of schemas of which this schema-level Q subscription is one. This column can optionally contain the percentage sign (%) as a wild card suffix. (The single pattern "%" can be used to indicate all schemas.) **z/OS:** The value in the CREATOR column of the SYSIBM.SYSTABLES system table for the source tables that are to be included in the schema-level Q subscription.
OBJECT_NAME	**Data type:** VARCHAR(128); **Nullable:** No, with default An identifier for a set of tables that belong to this schema-level Q subscription. This column can optionally contain the percentage sign (%) as a wild card suffix. Default: %
SUBPROFNAME	**Data type:** VARCHAR(128); **Nullable:** No, with default An identifier for the saved profile that can be used to specify options for the table-level Q subscriptions within this schema-level Q subscription. Default: ASNBIDI
SUBGROUP	**Data type:** VARCHAR(30); **Nullable:** Yes An identifier that relates all Q subscriptions that are required to replicate a table in all directions in a multidirectional replication configuration. All schema-level Q subscriptions that are part of the same configuration (for example, a bidirectional replication configuration between two sites) must specify the same name for SUBGROUP at all servers that are involved in the configuration.
SOURCE_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the source server in a bidirectional or peer-to-peer Q subscription. Default: 0
TARGET_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the target server in a bidirectional or peer-to-peer Q subscription. Default: 0
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that is inserted by the Q Capture program to indicate the current state of the schema-level subscription. The initial state is N (new). The STATE_INFO field is initially set to ASN7247I (new schema-level subscription). **N (default)** The schema-level subscription is new. The Q Capture program automatically activates this subscription when the program is started or reinitialized. Activation means that SQL operations such as CREATE TABLE and DROP TABLE are replicated for tables that match the pattern for the subscription. **I** The schema-level subscription is inactive. The Q Capture program saw a STOP_SCHEMASUB signal in the log, or an error occurred and the schema-level subscription was deactivated. The Q Capture program stopped sending messages for this subscription but continued with others. **A** The schema-level subscription is active. The Q Capture program is sending messages based on the options that are defined for the subscription.

Table 72. Columns in the IBMQREP_SCHEMASUBS table (continued)

Column name	Description
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp of the last change in the state of the schema-level Q subscription. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:** Yes The identifier for the message that was issued about the state of the schema-level Q subscription. For details, see the IBMQREP_CAPTRACE table or the Q Capture diagnostic log.

IBMQREP_SENDQUEUES table

The IBMQREP_SENDQUEUES table contains information about the WebSphere MQ queues that are used by a Q Capture program to send data and informational messages. Each instance of the Q Capture program can work with multiple send queues. Each send queue is uniquely defined in the IBMQREP_SENDQUEUES table.

Server: Q Capture server

Default schema: ASN

Primary key: SENDQ

Unique index: PUBQMAPNAME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 73 provides a brief description of the columns in the IBMQREP_SENDQUEUES table.

Table 73. Columns in the IBMQREP_SENDQUEUES table

Column name	Description
PUBQMAPNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the publishing queue map that includes this send queue. For Q subscriptions, this name must match the value of REPQMAPNAME in the IBMQREP_RECVQUEUES table.
SENDQ	**Data type:** VARCHAR(48); **Nullable:** No The unique name for this send queue. The name can stand for the local definition of a remote queue, or for a local queue. Queue names cannot contain blanks.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** Yes The name of the receive queue for this Q subscription. This is a local queue on the Q Apply server. Queue names cannot contain blanks.
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the publishing queue map that contains this send queue.

Table 73. Columns in the IBMQREP_SENDQUEUES table (continued)

Column name	Description
MESSAGE_FORMAT	**Data type:** CHAR(1); **Nullable:** No, with default The format that is used to encode messages that are put on the send queue. **C (default)** The Q Capture program encodes transactions from the source database in a compact format that is designed to be read by the Q Apply program. **X** The Q Capture program encodes transactions from the source database in Extensible Markup Language (XML) format. **J** The Q Capture program encodes transactions from the source database in Extensible Markup Language (XML) format with an MQRFH2 header. **D** The Q Capture program encodes transactions from the source database in delimited format.
MSG_CONTENT_TYPE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether messages put on the queue will contain an entire database transaction or a row operation only. **T (default)** **R** Messages contain a single update, insert, or delete operation, and information about the transaction to which it belongs.
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that is inserted by the Q Capture program to show the status of the send queue. **A (default)** Active. Transactions are being written to this queue. **I** Inactive. A severe error was encountered on this queue.
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Capture server of the send queue's last state change. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:** Yes The number for the ASN message about the queue state. For details, see the IBMQREP_CAPTRACE table, or the Q Capture diagnostic log.

Table 73. Columns in the IBMQREP_SENDQUEUES table (continued)

Column name	Description
ERROR_ACTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Capture program what to do when the send queue is no longer accepting messages. For example, the queue might be full, or the queue manager might have reported a severe error for this queue. **S (default)** The Q Capture program stops. **I** This value is deprecated and is treated the same as a value of S. **Q** The Q Capture program continues to put messages on other send queues but stops putting messages on this queue and sets the state for this queue to inactive (`I`). Q Capture also leaves the state of all Q subscriptions or publications that specify this queue as active (`A`). After you fix the send queue error, you must take one of the following actions: • Issue a **`startq`** command to tell Q Capture to start putting messages on the queue (set its state to active). • Stop and start the Q Capture program. • Use the Q Capture **`reinit`** command to reload all Q subscriptions from the Q Capture control tables.
LOB_TOO_BIG_ACTION	A flag that tells the Q Capture program what to do when a single large object (LOB) value would exceed the maximum message size that is allowed for this send queue. **Q (default)** The Q Capture program follows the error action that is defined for the send queue. **E** The Q Capture program sends empty LOB values if the data does not fit in a single transaction message. If the substitute value does not fit into a message, Q Capture follows the error action for the queue.
XML_TOO_BIG_ACTION	A flag that tells the Q Capture program what to do when a single XML document would exceed the maximum message size allowed for this send queue. **Q (default)** The Q Capture program follows the error action that is defined for the send queue. **E** The Q Capture program sends an XML placeholder. If the XML placeholder does not fit into a message, Q Capture follows the error action for the queue.

Table 73. Columns in the IBMQREP_SENDQUEUES table (continued)

Column name	Description
HEARTBEAT_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Capture program sends messages on this queue to tell the Q Apply program or a user application that the Q Capture program is still running when there are no changes to publish. The value must be a multiple of the COMMIT_INTERVAL value, or it will be rounded to the nearest multiple (COMMIT_INTERVAL is set in the IBMQREP_CAPPARMS table). A value of 0 (the default) tells the Q Capture program not to send `heartbeat` messages. For delimited messages (MESSAGE_FORMAT=`D`), the value of HEARTBEAT_INTERVAL must be 0. **Note:** The HEARTBEAT_INTERVAL defined in the IBMQREP_SENDQUEUES table is different from the HBINT (heartbeat interval) parameter that you can define for a WebSphere MQ channel. For more information, see the *WebSphere MQ Script (MQSC) Command Reference*.
MAX_MESSAGE_SIZE	**Data type:** INTEGER; **Nullable:** No, with default The maximum size (in kilobytes) of the buffer that is used for sending messages over this send queue. The size of the buffer must not be larger than the WebSphere MQ maximum message length (MAXMSGL) attribute that is defined for any queues that will contain the message, or all Q subscriptions and publications that use this send queue will be invalidated. Default: 64 KB
APPLY_SERVER	**Data type:** VARCHAR(18); **Nullable:** Yes The name of the database where the Q Apply program runs and targets are defined. For z/OS, this is a location name.
APPLY_ALIAS	**Data type:** VARCHAR(8); **Nullable:** Yes The DB2 database alias that corresponds to the Q Apply server that is named in the APPLY_SERVER column.
APPLY_SCHEMA	**Data type:** VARCHAR(128); **Nullable:** Yes The schema of the Q Apply program that is applying transactions from this send queue.
MESSAGE_CODEPAGE	**Data type:** INTEGER; **Nullable:** Yes The code page that is used to encode messages for Event Publishing.
COLUMN_DELIMITER	**Data type:** CHAR(1); **Nullable:** Yes A single character that is used to separate header entries and column data within a delimited message. Default: comma (,)
STRING_DELIMITER	**Data type:** CHAR(1); **Nullable:** Yes A single character that is used to surround string data in a delimited message. Default: double quotation mark (")
RECORD_DELIMITER	**Data type:** CHAR(1); **Nullable:** Yes A single character that is used to separate change-data records in a delimited message. Default: new line character
DECIMAL_POINT	**Data type:** CHAR(1); **Nullable:** Yes A single character that is used to separate the fractional portion of numerical data in a delimited message. Default: period (.)

Table 73. Columns in the IBMQREP_SENDQUEUES table (continued)

Column name	Description
SENDRAW_IFERROR	**Data type:** CHAR(1); **Nullable:** No, with default **For Event Publishing:** Specifies whether character data is published when the data causes a code page conversion error. An example of invalid character data is an incomplete double-byte character and missing shift-in byte ('0F'x) in a string stored in a Japanese or Korean code page. The following string should have `'830F'` at the end, but does not: `x'0E4481448244'`. **N (default)** No data is sent for the character field that failed code page conversion. For delimited message format, none of the data from character columns in the row is sent when any single character field in the row fails code page conversion. Instead, all the character columns are sent as null values and an indicator at the start of the row is set to IBM-INVALID-COLUMN-####-@-NULL, where #### represents the first column in which the Q Capture program detected invalid data and @ is either B (before value) or A (after value). All large object (LOB) or XML columns are sent as null values. **Y** The character data is sent in hex string format. For delimited messages, the identifier field is set to IBM-INVALID-COLUMN-####-@-HEX, and all the character columns in the row are sent in hex string format. The #### field represents the first column in which the Q Capture program detected invalid data and @ is either B (before value) or A (after value). All LOB or XML columns are sent as null values.

IBMQREP_SIGNAL table

The IBMQREP_SIGNAL table allows a user, user application, or the Q Apply program to communicate with a Q Capture program.

Server: Q Capture server

Default schema: ASN

This table contains information that you can update by using SQL.

A user or user application can insert rows into the IBMQREP_SIGNAL table to request that the Q Capture program begin capturing changes from the log for a source table, or take other actions such as deactivate a Q subscription or ignore a transaction. The Q Apply program or a user application can make the same requests by sending control messages to the Q Capture program, which then inserts the corresponding signals into the IBMQREP_SIGNAL table. The Q Capture program receives the signals when it reads the log record for the insert into the IBMQREP_SIGNAL table.

Records in this table with a SIGNAL_STATE of C (complete) or records with a timestamp that is eligible for pruning are deleted when the Q Capture program prunes.

Table 74 on page 484 provides a brief description of the columns in the IBMQREP_SIGNAL table.

Table 74. Columns in the IBMQREP_SIGNAL table

Column name	Description
SIGNAL_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default A timestamp that is used to uniquely identify the row. The Q Capture program uses this value to find the correct row in the signal table to indicate when it completed processing the Q Capture signal. Default: Current timestamp
SIGNAL_TYPE	**Data type:** VARCHAR(30); **Nullable:** No A flag that indicates the type of signal that was posted: **CMD** A row that is inserted by the administrative commands, **asnqccmd**, the Replication Center, or another application. See the SIGNAL_SUBTYPE column for a list of the available signal subtypes. **USER** A signal posted by a user. The Q Capture program updates the SIGNAL_LSN column with the log sequence number of when the signal was inserted. The Q Capture program also updates the value in the SIGNAL_STATE column from pending (P) to received (R).
SIGNAL_SUBTYPE	**Data type:** VARCHAR(30); **Nullable:** Yes The type of action that a CMD-type signal is requesting that the Q Capture program perform. **CAPSTART** Start capturing changes for a Q subscription or publication. **CAPSTOP** Stop capturing changes for a Q subscription or publication. **QINERROR** Execute the error action defined for the send queue in the IBMQREP_SENDQUEUES table. **LOADDONE** Acknowledge receipt of this signal from the Q Apply program or user application. The LOADDONE signal notifies the Q Capture program that the target table is loaded. **STOP** Stop capturing changes and terminate. **IGNORETRANS** Ignore the DB2 transaction that contains this signal. **REINIT_SUB** Deactivate and then activate one Q subscription or publication using the latest values in the IBMQREP_SUBS, IBMQREP_SRC_COLS, and IBMQREP_SENDQUEUES tables. This signal will not prompt a new load of targets.

Table 74. Columns in the IBMQREP_SIGNAL table (continued)

Column name	Description
SIGNAL_SUBTYPE (Continued)	**ADDCOL** Add one column to an active, unidirectional Q subscription or to a publication. **STARTQ** Start putting messages on a specified queue or all inactive queues. **STOPQ** Stop putting messages on a specified queue or all inactive queues. **P2PNEW2MEMB** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at a new server and contains the number of active servers in the peer-to-peer configuration. **P2PMEMB2NEW** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at active servers. **P2PMEMB2INIT** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at active servers. **P2PSPOOLING** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at the server that initiated a new subscription. **P2PLOADDONE** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at a new server. **P2PSUBSTOP** An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the server that is being deactivated. **P2PSUBSTOPPING** An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the remaining active servers. **P2PREADYTOSTOP** An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the server that is being deactivated. **P2PNORECAPTURE** A signal that is inserted by the Q Apply program to prevent the Q Capture program from recapturing changes. Used in bidirectional replication. **Note:** P2PNORECAPTURE signals are pruned according to the **`prune_interval`** parameter, unlike other signal subtypes, which are pruned according to the **`signal_limit`** parameter. **REPLICATE_LOAD** An internal signal that the Q Capture and Q Apply programs use when they replicate load operations at the source table. This signal might also be used when Q Apply changes a Q subscription state, for example when it processes an ADDCOL signal or **`reinit`** command.

Table 74. Columns in the IBMQREP_SIGNAL table (continued)

Column name	Description
SIGNAL_SUBTYPE (Continued)	**P2PCREATESUB** An internal signal that is inserted by the Q Apply program when a CREATE TABLE operation is replicated in a multidirectional replication configuration. The signal instructs the paired Q Capture program at its server to create a Q subscription in the other direction. **P2PDROPSUB** An internal signal that is inserted by the Q Apply program when a DROP TABLE operation is replicated in a multidirectional replication configuration. The signal instructs the paired Q Capture program at its server to drop a Q subscription in the other direction. **START_SCHEMASUB** A signal that instructs the Q Capture program to start capturing SQL operations such as CREATE and DROP for tables in a specified schema. **STOP_SCHEMASUB** A signal that instructs the Q Capture program to stop capturing SQL operations such as CREATE and DROP for tables in a specified schema. **REINIT_SCHEMASUB** A signal that instructs the Q Capture program to reload any changes or additions in the IBMQREP_SUBS_PROF table to the profiles for schema-level Q subscriptions. These changes apply only to new tables (and their Q subscriptions) that are created in the specified schema.

Table 74. Columns in the IBMQREP_SIGNAL table (continued)

Column name	Description

SIGNAL_INPUT_IN	**Data type:** VARCHAR(500); **Nullable:** Yes
	
	If the SIGNAL_TYPE=USER, then this column contains user-defined input. If the SIGNAL_TYPE=CMD, then this value depends on the SIGNAL_SUBTYPE value:
	
	CMD + CAPSTART
	The Q subscription or publication name.
	
	CMD + CAPSTOP
	The Q subscription or publication name.
	
	CMD + LOADDONE
	The Q subscription or publication name.
	
	CMD + STOP
	NULL (no value is required). You can specify CAPTUREUPTO to prompt Q Capture to stop capturing changes that were committed at or before a provided timestamp and then shut down. You can also specify STOPAFTER to prompt Q Capture to stop capturing changes immediately and then shut down either when all messages are drained from the transmission queue (or a shared local send-receive queue), or after all changes up to the stopping point are applied at the target. For details, see "Stopping a Q Capture program at a specified point" on page 245.
	
	CMD + IGNORETRANS
	NULL (no value is required).
	
	CMD + QINERROR
	For a user application, the name of the send queue that is in error. For the Q Apply program, the name of the send queue that is in error and the ASN message number and space-separated tokens.
	
	CMD + REINIT_SUB
	The Q subscription or publication name.
	
	CMD + ADDCOL
	The Q subscription or publication name and the name of the source table column that you are adding, separated by a semicolon. Use the format *'subname;column_name;before_column_name;target_column_name'*. Follow these examples:
	
	Add column in source table to Q subscription and to target table
	`QSUB1;COL10`
	
	Add column and before image of the column (for CCD target tables)
	`QSUB1;COL10;XCOL10`
	
	Add column without before image but with different target column name `QSUB1;COL10;;TRGCOL10` (Use the double semicolon (;;) to indicate that you are omitting the before-image column.)
	
	CMD + STARTQ
	The queue name or ALL.
	
	CMD + STOPQ
	The queue name or ALL.

Table 74. Columns in the IBMQREP_SIGNAL table (continued)

Column name	Description
SIGNAL_INPUT_IN (Continued)	**CMD + START_SCHEMASUB** The schema-level subscription name. **CMD + STOP_SCHEMASUB** The schema-level subscription name. **CMD + REINIT_SCHEMASUB** The schema-level subscription name.
SIGNAL_STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates the status of the signal. **P (default)** The signal is pending; the Q Capture program did not receive it yet. **R** The Q Capture program received the signal. **C** The Q Capture program completed processing the signal. **F** The signal failed. For example, the Q Capture program cannot perform a CAPSTART because the Q subscription or publication is faulty.
SIGNAL_LSN	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** Yes The logical log sequence number of the log record for the insert into the SIGNAL table.

IBMQREP_SRC_COLS table

The IBMQREP_SRC_COLS table lists columns at the source table for which changes are to be captured.

Server: Q Capture server

Default schema: ASN

Primary key: SUBNAME, SRC_COLNAME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 75 provides a brief description of the columns in the IBMQREP_SRC_COLS table.

Table 75. Columns in the IBMQREP_SRC_COLS table

Column name	Description
SUBNAME	**Data type:** VARCHAR(30); **Nullable:** No The name of the Q subscription or publication for this source table.
SRC_COLNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the column in the source table for which changes will be captured.

Table 75. Columns in the IBMQREP_SRC_COLS table (continued)

Column name	Description
IS_KEY	**Data type:** SMALLINT; **Nullable:** No, with default A flag that indicates whether the column is part of the key to be used for replication or publishing. Any set of columns that are unique at the source can be used. **0 (default)** The column is not part of the unique key. Its order in the transaction message will be the same as its order in the source table. *n* The column is part of the unique key. In a multiple-column key, the column's order in the transaction message will be encoded based on the number *n* that you specify. At least one of the columns from the source table should have a value greater than 0 in the IBMQREP_SRC_COLS table or the Q subscription or publication will be invalid. **Restriction:** Large-object (LOB) columns and LONG columns cannot be used in the replication or publishing key.
COL_OPTIONS_FLAG	**Data type:** CHAR(10); **Nullable:** No, with default The first character determines whether the before-image value of the column is published or replicated. The first character can have the following values: **N** No before-image values are sent for this column. **D** Before-image values are sent for delete operations. **U** Before-image values are sent for update operations. **Y** Before-image values are sent for both delete and update operations. Default: NNNNNNNNNN

IBMQREP_SRCH_COND table

IBMQREP_SRCH_COND is an internal table that is used by the Q Capture program to evaluate the search condition for a Q subscription or publication.

Server: Q Capture server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 76 provides a brief description of the column in the IBMQREP_SRCH_COND table.

Table 76. Column in the IBMQREP_SRCH_COND table

Column name	Description
ASNQREQD	**Data type:** CHAR(8); **Nullable:** Yes This column contains no data.

IBMQREP_SUBS table

The IBMQREP_SUBS table contains information about Q subscriptions or publications, including the type of subscription or publication, search conditions, data-sending options, load options, and the subscription or publication state.

Server: Q Capture server

Default schema: ASN

Primary key: SUBNAME

Non-unique index: SUB_ID

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 77 provides a brief description of the columns in the IBMQREP_SUBS table.

Table 77. Columns in the IBMQREP_SUBS table

Column name	Description
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The Q subscription or publication name. For each instance of the Q Capture program, all Q subscription or publication names must be unique.
SOURCE_OWNER	**Data type**: VARCHAR(128); **Nullable**: No The schema name or high-level qualifier of the source table for this Q subscription or publication.
SOURCE_NAME	**Data type:** VARCHAR(128); VARCHAR(18) for DB2 for z/OS Version 7 and Version 8 compatibility mode; **Nullable:** No The name of the source table for this Q subscription or publication.
TARGET_SERVER	**Data type:** VARCHAR(18); **Nullable:** Yes The name of the database or subsystem where the Q Apply program runs and targets are defined. For z/OS, this is a location name.
TARGET_ALIAS	**Data type:** VARCHAR(8); **Nullable:** Yes The DB2 database alias that corresponds to the Q Apply server that is named in the TARGET_SERVER column.
TARGET_OWNER	**Data type:** VARCHAR(128); **Nullable:** Yes The schema name or high-level qualifier of the target table or stored procedure for a Q subscription.
TARGET_NAME	**Data type:** VARCHAR(128); **Nullable:** Yes The name of the target table for a Q subscription.

Table 77. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
TARGET_TYPE	**Data type:** INTEGER; **Nullable:** Yes A flag that indicates the type of replication target. **1** User table **2** Consistent-change-data (CCD) table **3** Reserved for future use. **4** Reserved for future use. **5** Stored procedure
APPLY_SCHEMA	**Data type:** VARCHAR(128); **Nullable:** Yes The schema of the Q Apply program that is applying transactions for this Q subscription.
SENDQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the WebSphere MQ queue that the Q Capture program uses to send transactional data for this Q subscription or publication. Each source table is paired with one send queue.
SEARCH_CONDITION	**Data type:** VARCHAR(2048); **Nullable:** Yes The search condition that is used to filter rows for the Q subscription or publication. This must be an annotated select WHERE clause, with a single colon directly in front of the names of the source columns.
SUB_ID	**Data type:** INTEGER; **Nullable:** Yes An integer that is generated by the Q Capture program and used to uniquely identify a Q subscription in the `subscription schema` message to the Q Apply program.
SUBTYPE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates the type of replication that a Q subscription is involved in, or whether this is a publication. A flag that indicates the type of replication that a Q subscription is involved. **U (default)** Unidirectional replication. This is the value used for publications. **B** Bidirectional replication. **P** Peer-to-peer replication.
ALL_CHANGED_ROWS	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program sends a message when a row in the source table changes, even if none of the columns that are part of a Q subscription changed: **N (default)** The Q Capture program sends a message only when columns that are part of a Q subscription change. **Y** When any row in the source table changes, the Q Capture program sends the columns from that row that are part of a Q subscription, even if none of them changed.
BEFORE_VALUES	**Data type:** CHAR(1); **Nullable:** No, with default This column is deprecated for Version 9 Fix Pack 1.

Table 77. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
CHANGED_COLS_ONLY	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program publishes columns that are part of a Q subscription or publication only if they have changed. This field applies to update operations only. **Y (default)** When the Q Capture program sends an updated row, it sends only the changed columns that are part of a Q subscription or publication. **N** The Q Capture program sends all columns in a row that are part of a Q subscription or publication whenever any of them has changed.
HAS_LOADPHASE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the target table for the Q subscription or publication will be loaded with data from the source: **N (default)** The target will not be loaded. **I** An automatic load. The Q Apply program calls one of several different utilities, depending on the LOAD_TYPE that is specified in the IBMQREP_TARGETS table, and on the platform of the Q Apply server and Q Capture server. **E** A manual load. An application other than the Q Apply program loads the target table. In this case, the user or Replication Center inserts the LOADDONE signal into the IBMQREP_SIGNAL table at the Q Capture server, or the Q Capture program inserts this signal after it receives the `load done` message.

Table 77. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that is inserted by the Q Capture program to indicate the current state of the Q subscription or publication. The initial state is new, and the STATE_INFO field is initially set to ASN7024I (new Q subscription or publication). **N (default)** The Q subscription or publication is new. The Q Capture program automatically activates this Q subscription or publication when the program is started or reinitialized. **I** The Q subscription or publication is inactive. The Q Capture program saw a CAPSTOP signal in the log, or an error occurred and the Q subscription or publication was deactivated. The Q Capture program stopped sending messages for this Q subscription or publication but continued with others. **L** The Q subscription is loading. The Q Capture program processed the CAPSTART signal and sent the `subscription schema` message to the Q Apply program or user application. The Q Capture program is sending transaction messages that include before values for all columns, and it is waiting for the LOADDONE signal. **A** The Q subscription or publication is active. If there is a load phase, the Q Capture program processed the LOADDONE signal and sent a `load done received` message to the Q Apply program or user application. The Q Capture program is sending data messages based on the options defined for the Q subscription or publication. **T** An internal state that indicates that the Q Capture program read a CAPSTART signal in the log for this peer-to-peer Q subscription, and the Q subscription is being initialized within the peer-to-peer group. **G** An internal state that indicates that the Q Capture program read a CAPSTOP signal in the log for this peer-to-peer Q subscription, and the Q subscription is being deactivated within the peer-to-peer group. **U** An internal state that indicates that the Q Capture program created this Q subscription, and the Q subscription is waiting to be activated.
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp of the last change in Q subscription or publication state. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:** Yes The number for the ASN message about the Q subscription state. For details, see the IBMQREP_CAPTRACE table, or the Q Capture diagnostic log.
STATE_TRANSITION	**Data type:** VARCHAR(256) FOR BIT DATA; **Nullable:** Yes An internal value used to store half state and related information.
SUBGROUP	**Data type:** VARCHAR(30); **Nullable:** Yes The name of the peer-to-peer group that includes this Q subscription. This column does not apply for a publication.
SOURCE_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the source server in a peer-to-peer Q subscription. This column does not apply for a publication. Default: 0

Table 77. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
TARGET_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the target server in a peer-to-peer Q subscription. This column does not apply for a publication. Default: 0
GROUP_MEMBERS	**Data type:** CHAR(254) FOR BIT DATA; **Nullable:** Yes This column is updated by the Q Capture program when members join or leave a peer-to-peer group.
OPTIONS_FLAG	**Data type:** CHAR(4) FOR BIT DATA; **Nullable:** No, with default Reserved for future.
SUPPRESS_DELETES	**Data type:** CHAR(1);**Nullable:** No, with default A flag that tells the Q Capture program whether to send rows that were deleted from the source table: **N (default)** Send deleted rows. **Y** Do not send deleted rows.
DESCRIPTION	**Data type:** VARCHAR(200); **Nullable:** Yes A user-supplied description of the Q subscription or publication.
TOPIC	**Data type:** VARCHAR(256); **Nullable:** Yes A user-supplied topic to be included in the JMS-compliant (MQRFH2) message header for each XML message that is sent for the publication.
CAPTURE_LOAD	**Data type:** CHAR(1); **Nullable:** No, with default The action that the Q Capture program takes when the recovery log shows a load operation that uses the DB2 LOAD utility occurred at the source table: **W (default)** Q Capture issues a warning message after the load completes. **R** Q Capture issues a warning message and then stops and starts the Q subscription for the source table, prompting a load of the target table if one is specified for the Q subscription.

Table 77. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
CHANGE_CONDITION	**Data type:** VARCHAR(2048); **Nullable:** Yes, with default An SQL predicate that uses log record variables to filter which rows are replicated or published. This predicate does not require a WHERE clause like that used in the SEARCH_CONDITION column, but it can include the following log record variables: **$OPERATION** The DML operation. Valid values are I (insert), U (update), and D (delete). **$AUTHID** The authorization ID of a transaction. z/OS **$AUTHTOKEN** The authorization token (job name) of a committed transaction. z/OS **$PLANNAME** The plan name of a committed transaction. For example, the following predicate tells Q Capture to only replicate or publish log records for insert operations committed by user HR for the affected Q subscription: `$OPERATION = 'I' AND $AUTHID = 'HR'` If a WHERE clause is specified in the SEARCH_CONDITION column, the value of CHANGE_CONDITION and SEARCH_CONDITION are combined into a single predicate by using the AND operator. However, you can leave the SEARCH_CONDITION column empty and specify the full predicate that includes a WHERE clause in this column. The value in this column is not used when the Q Apply program loads the target table. For more detail on specifying filters that use log records, see “Log record variables to filter rows (unidirectional replication)” on page 71.
REPL_ADDCOL	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Capture program whether to automatically add new source table columns to the Q subscription for the table. The columns are also added to the target table if they do not exist. **N (default)** Do not automatically add new columns to the Q subscription. **Y** When an ALTER TABLE ADD COLUMN statement is detected, automatically add any new columns to the Q subscription for the source table, and to the target table if the columns do not exist.

Table 77. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
IGNSETNULL	**Data type:** CHAR(1); **Nullable:** No, with default Linux UNIX Windows A flag that tells the Q Capture program whether to replicate UPDATE operations that result from the delete of parent rows in tables with referential integrity relationships (ON DELETE SET NULL rule). For Version 10.1 and later on Linux, UNIX, and Windows only. **N (default)** When a parent row is deleted at the source, Q Capture replicates the UPDATE operations from child rows in which one or more column values are set to NULL. **Y** Q Capture does not replicate UPDATE operations at child tables that result from the ON DELETE SET NULL rule. Only the deletion of the parent row is replicated. This option can be useful if you have the ON DELETE SET NULL rule in force at the target database and do not need these updates replicated.
SCHEMA_SUBNAME	**Data type:** VARCHAR(64); **Nullable:** Yes The name of the schema-level subscription that this table-level Q subscription belongs to. This column is populated by the ASNCLP command-line program when a schema-level subscription is created, and by the Q Capture program when it detects a CREATE TABLE operation within the schema.
SUB_CREATOR	**Data type:** VARCHAR(12); **Nullable:** Yes Identifies how this Q subscription was created. The values are "asnclp," "replcenter" (the Replication Center), and "asnqcap" (the Q Capture program). When the Q Capture control tables are migrated from a release before Version 10 on Linux, UNIX, and Windows, the column value is NULL.
CAPTURE_TRUNCATE	**Data type:** CHAR(1); **Nullable:** No, with default Reserved for future use.

IBMQREP_SUBS_PROF table

The IBMQREP_SUBS_PROF table stores the default values for automatically creating table-level Q subscriptions when tables are added to databases with a schema-level Q subscriptions. When you create a schema-level Q subscription, the replication administration tools populate this table with default options for unidirectional, bidirectional, or peer-to-peer configurations.

Server: Q Capture server

Default schema: ASN

Primary key: SUBPROFNAME

Important: After a profile is inserted in this table, the profile should not be modified because the default values for newly created table-level Q subscriptions should be consistent with existing Q subscriptions. If you want to change the profile for a schema-level Q subscription, use the ASNCLP command-line program to create a new profile with a different name and associate it with this schema-level Q subscription.

Table 78 provides a brief description of the columns in the IBMQREP_SUBS_PROF table.

Table 78. Columns in the IBMQREP_SUBS_PROF table

Column name	Description
SUBPROFNAME	**Data type:** VARCHAR(128); **Nullable:** No A unique identifier for the table-level Q subscription profile. Any name that is prefixed by "ASN" is reserved because the following profiles are automatically used for unidirectional or bidirectional replication: ASNUNI and ASNBIDI.
SUBNAME_PREFIX	**Data type:** VARCHAR(4); **Nullable:** Yes The identifier that is prefixed to table-level Q subscription names when Q Capture automatically creates Q subscriptions that are based on schema-level Q subscriptions. For example, if the source table name is T3 and the value of SUBNAME_PREFIX is PROD, the Q subscription name that would be generated by the Q Capture program would be PRODT30001. (If this column is null, Q Capture generates Q subscription names comprised of the source table name plus 000*n*, where *n* >= 1.)
SUBTYPE	For details about these columns, see the descriptions in “IBMQREP_SUBS table” on page 490.
CHANGED_COLS_ONLY	
HAS_LOADPHASE	
SUPPRESS_DELETES	
CAPTURE_LOAD	
CONFLICT_RULE	For details about these columns, see the descriptions in “IBMQREP_TARGETS table” on page 520.
CONFLICT_ACTION	
ERROR_ACTION	
OKSQLSTATES	
LOAD_TYPE	
MODELQ	

IBMQREP_TABVERSION table

The IBMQREP_TABVERSION table is used by the Q Capture and Capture programs to keep track of different versions of a source table. The Q Capture or Capture program inserts rows into this table when the Q subscription or registration for a source table is first activated, and then each time the source table is altered.

Server: Q Capture server

Default schema: ASN

Index: LSN, TABLEID1, TABLEID2, VERSION

Index: SOURCE_OWNER ASC, SOURCE_NAME ASC

Index: TABLEID1 ASC, TABLEID2 ASC

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 79 provides a brief description of the columns in the IBMQREP_TABVERSION table.

Table 79. Columns in the IBMQREP_TABVERSION table

Column name	Description
LSN	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** No The point in the DB2 recovery log where the Q Capture program or Capture program detected a new version of the source table.
TABLEID1	**Data type:** SMALLINT; **Nullable:** No The database identifier (DBID) in SYSIBM.SYSTABLES.
TABLEID2	**Data type:** SMALLINT; **Nullable:** No The object identifier (OBID) in SYSIBM.SYSTABLES.
VERSION	**Data type:** INTEGER; **Nullable:** No A number generated by the Q Capture or Capture program to keep track of the different versions of a source table.
SOURCE_OWNER	**Data type:** VARCHAR(128); **Nullable:** No The schema or high-level qualifier of the source table.
SOURCE_NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the source table.

Control tables at the Q Apply server

The control tables at the Q Apply server contain Q Apply operating parameters, Q subscription definitions, performance statistics, and other metadata. These tables are built according to options that you specify in the replication administration tools.

Table 80 describes the control tables at the Q Apply server.

Table 80. Control tables at the Q Apply server

Table name	Description
“IBMQREP_APPLYENQ table” on page 499	Ensures that only one Q Apply program with a given schema is running per Q Apply server.
“IBMQREP_APPENVINFO table” on page 500	Stores environment variables and other information that replication tools use to access remote programs.
“IBMQREP_APPLYMON table” on page 500	Contains statistics about the performance of a Q Apply program for each receive queue.
“IBMQREP_APPLYPARMS table” on page 504	Contains parameters that you can specify to control the operations of a Q Apply program.
“IBMQREP_APPLYTRACE table” on page 511	Contains informational, warning, and error messages from the Q Apply program.
“IBMQREP_DELTOMB table” on page 512	An internal table used by the Q Apply program to record conflicting deletes in peer-to-peer replication.
“IBMQREP_DONEMSG table” on page 513	An internal table used by the Q Apply program to record which messages were processed.
“IBMQREP_EXCEPTIONS table” on page 513	Contains row changes that could not be applied because of conflicts, errors, or rollbacks.

Table 80. Control tables at the Q Apply server (continued)

Table name	Description
"IBMQREP_RECVQUEUES table" on page 516	Identifies queues that a Q Apply program uses to receive transaction messages and send control message, and contains some operation parameters for the Q Apply program.
"IBMQREP_SAVERI table" on page 518	An internal table that the Q Apply program uses to store referential integrity constraints that are dropped while targets are being loaded.
"IBMQREP_SPILLEDROW table" on page 520	An internal table that the Q Apply program uses to keep track of rows sent to a temporary spill queue.
"IBMQREP_SPILLQS table" on page 519	Identifies temporary spill queues that will hold changes to source tables before they are applied to targets.
"IBMQREP_TRG_COLS table" on page 529	Contains information about the mapping between source and target columns.
"IBMQREP_TARGETS table" on page 520	Contains information about target tables or stored procedures, and options for Q subscriptions.

IBMQREP_APPLYENQ table

The IBMQREP_APPLYENQ table is used to ensure the uniqueness of the schema that is used to identify a Q Apply program and its control tables.

Server: Q Apply server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The IBMQREP_APPLYENQ table ensures that:

- Linux UNIX Windows For DB2 for Linux, UNIX, and Windows, only one Q Apply program with a given schema is running per database.
- z/OS For non-data-sharing DB2 for z/OS, only one Q Apply program with a given schema is running per subsystem.
- z/OS For data-sharing DB2 for z/OS, only one Q Apply program with a given schema is running per data-sharing group.

While running, a Q Apply program exclusively locks this table. Starting the Q Apply program twice will place the second instance on a lock wait over this table. The table is created empty.

Table 81 provides a brief description of the column in the IBMQREP_APPLYENQ table.

Table 81. Column in the IBMQREP_APPLYENQ table

Column name	Description
LOCKNAME	**Data type:** INTEGER; **Nullable:** Yes This column contains no data.

IBMQREP_APPENVINFO table

The IBMQREP_APPENVINFO table contains eight rows that are used to store the value of runtime environment variables and other information that the replication administration tools use to access remote programs.

Server: Q Apply server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 82 provides a brief description of the columns in the IBMQREP_APPENVINFO table.

Table 82. Columns in the IBMQREP_APPENVINFO table

Column name	Description
NAME	**Data type:** VARCHAR(30); **Nullable:** No **STARTTIME** The timestamp when the Q Apply program started. **HOSTNAME** The TCP/IP host name of the server where the Q Apply program is running. **LOGFILE** The path and file name of the Q Apply diagnostic log file. **TMPDIR** The path of the directory where the Inter-Process Communication (IPC) key of the Q Apply program is located. **ASNUSEMQCLIENT** The value of the Q Replication **ASNUSEMQCLIENT** environment variable. **MQSERVER** The value of the WebSphere MQ **MQSERVER** environment variable. **MQCHLLIB** The value of the WebSphere MQ **MQCHLLIB** environment variable. **MQCHLTAB** The value of the WebSphere MQ **MQCHLTAB** environment variable.
VALUE	**Data type:** VARCHAR(3800); **Nullable:** Yes For each row in the IBMQREP_APPENVINFO table, the VALUE column contains the value that is associated with the corresponding NAME column.

IBMQREP_APPLYMON table

The Q Apply program periodically inserts rows in the IBMQREP_APPLYMON table to record performance statistics, one row for each receive queue.

Server: Q Apply server

Default schema: ASN

Non-unique index: MONITOR_TIME DESC

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The value that you specify for MONITOR_INTERVAL in the IBMQREP_APPLYPARMS table determines how often the Q Apply program inserts rows into this control table. The MONITOR_LIMIT value determines how long rows remain in the table before they are eligible for pruning.

Table 83 provides a brief description of the columns in the IBMQREP_APPLYMON table.

Table 83. Columns in the IBMQREP_APPLYMON table

Column name	Description
MONITOR_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Apply server when the row was inserted into the IBMQREP_APPLYMON table.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that this row of Q Apply performance statistics pertains to.
QSTART_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Apply server when the receive queue was started.
CURRENT_MEMORY	**Data type:** INTEGER; **Nullable:** No The amount of memory in bytes that the Q Apply browser thread used for reading transactions from this queue.
QDEPTH	**Data type:** INTEGER; **Nullable:** No The queue depth (number of messages on the queue). The Q_PERCENT_FULL column expresses the fullness of the queue as a percentage.
END2END_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that transactions were committed to the source table and the time that they were committed to the target.
QLATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that the Q Capture program put messages on the send queue and the time that the Q Apply program got them from the receive queue.
APPLY_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that the Q Apply program read transactions from the receive queue and the time that they were committed to the target.
TRANS_APPLIED	**Data type:** INTEGER; **Nullable:** No The total number of transactions from this receive queue that the Q Apply committed to the target.
ROWS_APPLIED	**Data type:** INTEGER; **Nullable:** No The total number of insert, update, and delete operations from this receive queue that the Q Apply program applied to the target.

Table 83. Columns in the IBMQREP_APPLYMON table (continued)

Column name	Description
TRANS_SERIALIZED	**Data type:** INTEGER; **Nullable:** No The total number of transactions that conflicted with another transaction (either because of a row conflict or a referential integrity conflict). In these cases, the Q Apply program suspends parallel processing and applies the row changes within the transaction in the order they were committed at the source.
RI_DEPENDENCIES	**Data type:** INTEGER; **Nullable:** No The total number of referential integrity conflicts that were detected, forcing transactions to be serialized.
RI_RETRIES	**Data type:** INTEGER; **Nullable:** No The number of times that the Q Apply program had to re-apply row changes because of referential integrity conflicts when the transactions that they were part of were executed in parallel.
DEADLOCK_RETRIES	**Data type:** INTEGER; **Nullable:** No The number of times that the Q Apply program re-applied row changes because of lock timeouts and deadlocks.
ROWS_NOT_APPLIED	**Data type:** INTEGER; **Nullable:** No The number of rows that could not be applied, and were entered in the IBMQREP_EXCEPTIONS table.
MONSTER_TRANS	**Data type:** INTEGER; **Nullable:** No The number of transactions that exceeded the MEMORY_LIMIT for the receive queue set in the IBMQREP_RECVQUEUES table.
MEM_FULL_TIME	**Data type:** INTEGER; **Nullable:** No The number of seconds that the Q Apply program could not build transactions from this receive queue because its agents were using all available memory to apply transactions.
APPLY_SLEEP_TIME	**Data type:** INTEGER; **Nullable:** No The number of milliseconds that Q Apply agents for this receive queue were idle while waiting for work.
SPILLED_ROWS	**Data type:** INTEGER; **Nullable:** No The number of rows that the Q Apply program sent to temporary spill queues while targets were being loaded or while Q subscriptions were placed into a spill state by the **`spillsub`** parameter of the **`MODIFY`** or **`asnqacmd`** command.
SPILLEDROWSAPPLIED	**Data type:** INTEGER; **Nullable:** No The number of spilled rows that were applied to the target.

Table 83. Columns in the IBMQREP_APPLYMON table (continued)

Column name	Description
OLDEST_TRANS	**Data type:** TIMESTAMP; **Nullable:** No A timestamp that is based on the local time at the Q Apply server that helps determine how far Q Apply has caught up with respect to the source. At each monitor interval OLDEST_TRANS represents: • If Q Apply is processing transactions, the source commit time for which all transactions to that point have been applied to the target. (Other more recent transactions might also have been applied. Because the Q Apply program processes transactions in parallel, the commit times of these more recent transactions do not refer to a point at which all previous transactions have been applied.) • The latest heartbeat time, if no transactions are being processed and the heartbeat message arrived after the oldest applied transaction. • The value 1900-01-01-00.00.00.000000 if the Q Apply program has not seen any messages (transaction or heartbeat).
OLDEST_INFLT_TRANS	**Data type:** TIMESTAMP; **Nullable:** No At each monitor interval OLDEST_INFLT_TRANS represents: • If Q Apply is processing transactions, the source commit time of the oldest currently in-flight transaction. An in-flight transaction has not been fully applied and committed at the target. • The value NULL, if Q Apply is not processing transactions. • The value NULL, if Q Apply has not seen any transaction messages so far. OLDEST_INFLT_TRANS does not consider heartbeat messages. It only considers transaction messages. This value also does not reflect how far Q Apply has caught up with respect to the source because the source commit time belongs to a transaction that has not yet been fully processed or committed.
OKSQLSTATE_ERRORS	**Data type:** INTEGER; **Nullable:** No The number of row changes that caused an SQL error that is defined as acceptable in the OKSQLSTATES field of the IBMQREP_TARGETS table. The Q Apply program ignores these errors.
HEARTBEAT_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that `heartbeat` messages were sent by the Q Capture program and the time that they were received by the Q Apply program.
KEY_DEPENDENCIES	**Data type:** INTEGER; **Nullable:** No The total number of replication key constraints that were detected, forcing transactions to be serialized.
UNIQ_DEPENDENCIES	**Data type:** INTEGER; **Nullable:** No The total number of unique index constraints that were detected, forcing transactions to be serialized.
UNIQ_RETRIES	**Data type:** INTEGER; **Nullable:** No The number of times that the Q Apply program tried to re-apply rows that were not applied in parallel because of unique index constraints.
JOB_DEPENDENCIES	**Data type:** INTEGER; **Nullable:** No The number of transactions that are delayed because of correlation ID dependencies.

Table 83. Columns in the IBMQREP_APPLYMON table (continued)

Column name	Description
CAPTURE_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time transactions were committed to the source table and the time Q Capture puts the last message for the transactions on the send queue.
OLDEST_COMMIT_LSN	**Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** Yes The commit log sequence number (LSN) from the source recovery log that corresponds to the oldest transaction that was applied. All transactions with lower LSNs were applied. Some more recent transactions might also be applied. If no source transactions are committed yet, the value of this column is all zeroes, for example: x'00000000000000000000'. z/OS You can use the OLDEST_COMMIT_LSN value on z/OS as the value for the **maxcmtseq** parameter when you need to restart the Q Capture program from a known point in the recovery log.
ROWS_PROCESSED	**Data type:** INTEGER; **Nullable:** Yes The number of rows that were read from receive queues and applied but not yet committed to the target.
Q_PERCENT_FULL	**Data type:** SMALLINT; **Nullable:** Yes, with default The fullness of the receive queue expressed as a percentage, where the MAXDEPTH attribute of the queue is 100 percent. The QUEUE_DEPTH column expresses the fullness as a number of messages.
OLDEST_COMMIT_SEQ	Linux UNIX Windows **Data type:** VARCHAR(16) FOR BIT DATA; **Nullable:** Yes An internal log marker that represents the last transaction that was applied by the Q Apply program before which all previous transactions have also been applied. The value is a formatted timestamp with nanosecond precision that is encoded as two integers, seconds, and nanoseconds into a 10-byte sequence. You can use the OLDEST_COMMIT_SEQ value on Linux, UNIX, and Windows as the value for the **maxcmtseq** parameter when you need to restart the Q Capture program from a known point in the recovery log.
MQ_BYTES	**Data type:** INTEGER; **Nullable:** Yes The number of bytes of data read from all receive queues during the monitor interval, including message data and the message header.
MQGET_TIME	**Data type:** INTEGER; **Nullable:** Yes The number of milliseconds during the monitor interval that the Q Apply program spent interacting with the WebSphere MQ API for getting messages from all receive queues. The value includes unsuccessful MQGET calls.
NUM_MQGETS	**Data type:** INTEGER; **Nullable:** Yes The number of times that the Q APPLY program successfully used the MQGET call to retrieve a message from all receive queues during the monitor interval. MQGET calls that failed to retrieve a message from the receive queue are not counted.

IBMQREP_APPLYPARMS table

The IBMQREP_APPLYPARMS table contains parameters that you can modify to control the operation of the Q Apply program. For example, you can specify the name of the queue manager that the Q Apply program works with, or how long the Q Apply program retains data in the IBMQREP_APPLYMON table before pruning. The Q Apply program reads changes to this table only during startup.

Server: Q Apply server

Default schema: ASN

Unique index: QMGR

This table contains information that you can update by using SQL.

The IBMQREP_APPLYPARMS table contains a single row. If this table has no row, or more than one row, the Q Apply program will not run.

Table 84 provides a brief description of the columns in the IBMQREP_APPLYPARMS table.

Table 84. Columns in the IBMQREP_APPLYPARMS table

Column name	Description
QMGR	**Data type:** VARCHAR(48); **Nullable:** No The name of the WebSphere MQ queue manager that the Q Apply program works with.
MONITOR_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_APPLYMON table before they are eligible for pruning. At each pruning interval, rows in the IBMQREP_APPLYMON table are pruned if they are older than this limit based on the current timestamp. Default: 10080
TRACE_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_APPLYTRACE table before they are eligible for pruning. At each pruning interval, rows in the IBMQREP_APPLYTRACE table are pruned if they are older than this limit based on the current timestamp. Default: 10080
MONITOR_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in milliseconds, the Q Apply program adds a row to the IBMQREP_APPLYMON table. Default: 60000 milliseconds (1 minute) on z/OS; 30000 milliseconds (30 seconds) on Linux, UNIX, and Windows
PRUNE_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Apply program automatically prunes rows in the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables. Default: 300
AUTOSTOP	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program whether to stop when all receive queues have been emptied once. **N (default)** The Q Apply program continues running after all receive queues have been emptied once. **Y** The Q Apply program stops when all receive queues have been emptied once.

Table 84. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
LOGREUSE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Apply program reuses the Q Apply log file or appends to it. **N (default)** The Q Apply program appends new information to an existing Q Apply log file when it restarts. **Y** On restart, the Q Apply program reuses its log file by clearing the file then writing to the blank file.
LOGSTDOUT	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Apply program sends log messages to outputs other than its log file. **N (default)** The Q Apply program directs most log messages to the log file only. **Y** The Q Apply program sends log messages to both the log file and the console (stdout). Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file regardless of the setting for this parameter.
APPLY_PATH	**Data type:** VARCHAR(1040); **Nullable:** Yes, with default The path where files created by the Q Apply program are stored. By default, this is the directory where the Q Apply program is started.
ARCH_LEVEL	**Data type:** CHAR(4); **Nullable:** No, with default The version of the control tables. For Replication Server Version 10 on z/OS the value is 100Z. Other ARCH_LEVEL values are 0802, 0901, 0905, 0907, and 0973 for Replication Server Version 9.7 Fix Pack 3 on Linux, UNIX, and Windows. **Attention:** When updating the IBMQREP_APPLYPARMS table, do not change the value in this column.
TERM	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Apply program stops if the target DB2 or queue manager are unavailable. **Y (default)** The Q Apply program stops if DB2 or the queue manager are unavailable. **N** The Q Apply program continues running if DB2 or the queue manager are unavailable. When DB2 or the queue manager are available, Q Apply begins applying transactions where it left off without requiring you to restart the program.
PWDFILE	**Data type:** VARCHAR(48); **Nullable:** Yes, with default The name of the encrypted password file that the Q Apply program uses to connect to the Q Capture program if the Q subscription calls for an internal load of the target. The **asnpwd** command creates this file by default in the directory specified in the APPLY_PATH column.

Table 84. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
DEADLOCK_RETRIES	**Data type:** INTEGER; **Nullable:** No, with default The number of times the Q Apply program tries to reapply changes to target tables, or make inserts into its control tables, after SQL deadlocks. If the deadlock occurs at a target table, the Q Apply program keeps trying until it reaches the limit that you set. After the limit is reached, if deadlocks persist the browser thread stops. Default: 3 tries.
SQL_CAP_SCHEMA	**Data type:** VARCHAR(128); **Nullable:** Yes, with default The schema of the Capture control tables that the Q Apply program uses to manage CCD target tables that are registered as SQL replication sources. This column must contain a value in for the Q Apply program to manage data distribution (fan-out) configurations.
LOADCOPY_PATH	**Data type:** VARCHAR(1040); **Nullable:** Yes, with default Specifies the path where the DB2 LOAD utility creates a copy of loaded data on the primary server for a configuration that involves the DB2 High Availability Disaster Recovery (HADR) utility. Setting this parameter prompts Q Apply to start the LOAD utility with the option to create the copy when Q Apply loads the target table. The secondary server in the HADR configuration then looks for the copied data in this path.
NICKNAME_COMMIT_CT	**Data type:** INTEGER; **Nullable:** Yes, with default Specifies the number of rows after which the DB2 IMPORT utility commits changes to nicknames that reference the target table during the loading process. This parameter applies only to automatic loads for federated targets that use the IMPORT utility. The default is **`nickname_commit_ct`**=10. This parameter can be used to tune the performance of the DB2 IMPORT utility by reducing the number of commits for federated targets.
SPILL_COMMIT_COUNT	**Data type:** INTEGER; **Nullable:** Yes, with default Specifies how many rows are grouped together in a commit scope by the Q Apply spill agents that apply data that was replicated during a load operation. The default is **`spill_commit_count`**=10. Increasing the number of rows that are applied before a COMMIT is issued can improve performance by reducing the I/O resources that are associated with frequent commits. Balance the potential for improvement with the possibility that fewer commits might cause lock contention at the target table and the IBMQREP_SPILLEDROW control table.
LOAD_DATA_BUFF_SIZE	**Data type:** INTEGER; **Nullable:** Yes, with default Specifies the number of 4KB pages for the DB2 LOAD utility to use as buffered space for transferring data within the utility during the initial loading of the target table. This parameter applies only to automatic loads using the DB2 LOAD utility. By default, the Q Apply program starts the utility with the option to use a buffer of 8 pages. Load performance for multidimensional clustering (MDC) tables that are replication targets can be significantly improved by specifying a much higher number of pages.

Table 84. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
CLASSIC_LOAD_FILE_SIZE	**Data type:** INTEGER; **Nullable:** Yes with default z/OS Specifies the estimated number of rows in tables or views from a Classic replication data source that are to be loaded into target tables. The Q Apply program uses this estimate to calculate the DASD allocation of the data set that is used as input to the load utility. The default is 500,000 rows. Use this parameter when the default allocation is too small. This parameter applies only to automatic loads of z/OS target tables from Classic sources.
MAX_PARALLEL_LOADS	**Data type:** INTEGER; **Nullable:** Yes, with default Specifies the maximum number of automatic load operations of target tables that Q Apply can start at the same time for a given receive queue. The default differs depending on the platform of the target server: z/OS On z/OS the default is one load at a time because of potential issues with the DSNUTILS stored procedure that Q Apply uses to call the DB2 LOAD utility. Depending on your environment you can experiment with values higher than **max_parallel_loads**=1. If errors occur, reset the value to 1. Linux UNIX Windows On Linux, UNIX, and Windows the default is 15 parallel loads.
COMMIT_COUNT	**Data type:** INTEGER; **Nullable:** Yes, with default Specifies the number of transactions that each Q Apply agent thread applies to the target table within a commit scope. By default (**commit_count**=1), the agent threads commit after each transaction that they apply. By increasing **commit_count** and grouping more transactions within the commit scope, you might see improved performance. **Recommendation:** Use a higher value for **commit_count** only with row-level locking. This parameter requires careful tuning when used with a large number of agent threads because it could cause lock escalation resulting in lock timeouts and deadlock retries.
INSERT_BIDI_SIGNAL	**Data type:** CHAR(1); **Nullable:** Yes, with default Whether the Q Capture and Q Apply programs use P2PNORECAPTURE signal inserts to prevent recapture of transactions in bidirectional replication. **Y (default)** The Q Apply program inserts P2PNORECAPTURE signals into the IBMQREP_SIGNAL table to instruct the Q Capture program at its same server not to recapture applied transactions at this server. **N** The Q Apply program does not insert P2PNORECAPTURE signals. Instead, you insert Q Apply's AUTHTKN information into the IBMQREP_IGNTRAN table, which instructs the Q Capture program at the same server to not capture any transactions that originated from the Q Apply program, except for inserts into the IBMQREP_SIGNAL table.
APPLY_ALIAS	**Data type:** VARCHAR(8); **Nullable:** Yes The alias name for the database or subsystem that is used as the Q Apply server. This is the alias as cataloged on the system where the replication administration tools run and used to connect to the target database or subsystem to create Q Apply control tables. This column is populated by the Replication Center or ASNCLP command-line program when control tables are created.

Table 84. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
STARTALLQ	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program whether to activate all receive queues that are not already in active state when Q Apply starts. Receive queues that are already active are always processed when Q Apply starts. **Y** When the Q Apply program starts, it activates all receive queues that are not already in active (A) state. **N** When the Q Apply program starts, it does not activate receive queues that are in inactive (I) state. The default value for this column is N for z/OS and Y for Linux, UNIX, and Windows.
PRUNE_BATCH_SIZE	**Data type:** INTEGER; **Nullable:** No, with default The number of rows that are deleted from the IBMQREP_DONEMSG table in one commit scope when PRUNE_METHOD is 2. The default is 1000 rows. The minimum value is 2 rows.
PRUNE_METHOD	**Data type:** INTEGER; **Nullable:** No, with default z/OS The method that the Q Apply program uses to delete unneeded rows from the IBMQREP_DONEMSG table. **1** Q Apply deletes a message from the receive queue, queries the IBMQREP_DONESG table to see if data from the message was applied, and then prunes the corresponding row from IBMQREP_DONEMSG by issuing an individual SQL statement. **2 (default)** Q Apply prunes groups of rows based on the PRUNE_BATCH_SIZE value. A separate prune thread records which messages have been applied, then issues a single range-based DELETE against the IBMQREP_DONEMSG table.
IGNBADDATA	**Data type:** CHAR(1); **Nullable:** No, with default **Note:** This column applies only if the Q Apply program uses International Components for Unicode (ICU) for code page conversion (if the code page of the source database and the code page that Q Apply uses are different).Whether the Q Apply program checks for illegal characters in data from the source and continues processing even if it finds illegal characters. **Y** Q Apply checks for illegal characters. **N (default)** Q Apply does not check for illegal characters and does not report exceptions for illegal characters.

Table 84. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
RICHKLVL	**Data type:** INTEGER; **Nullable:** No, with default The level of referential integrity checking. By default, the Q Apply program checks for RI-based dependencies between transactions to ensure that dependent rows are applied in the correct order. **0** Q Apply does not check for RI-based dependencies. **2 (default)** Q Apply checks for RI-based dependencies when a key value is updated in the parent table or a row is deleted from the parent table. **5** Q Apply checks for RI-based dependencies when a key value is updated in the parent table, a row is updated in the parent table, or a row is deleted from the parent table.
TRACE_DDL	**Data type:** CHAR(1); **Nullable:** No, with default When DDL operations at the source database are replicated, this column indicates whether the SQL text of the operation that the Q Apply program performs at the target database is logged. **N (default)** Replicated DDL operations are not logged. **Y** The Q Apply program issues an ASN message to its log file, standard output, and IBMQREP_APPLYTRACE table with the text of the SQL statement that was used to replicate the source DDL. The SQL text is truncated to 1024 characters.
REPORT_EXCEPTIONS	**Data type:** CHAR(1); **Nullable:** No, with default A flag that controls whether the Q Apply program inserts data into the IBMQREP_EXCEPTIONS table when a conflict or SQL error occurs at the target table but the row is applied anyway because the conflict action that was specified for the Q subscription was F (force). **Y (default)** Q Apply inserts data into the IBMQREP_EXCEPTIONS table whether or not the row that caused the exception is applied. **N** Q Apply does not insert data into the IBMQREP_EXCEPTIONS table when the row that caused an exception is applied; data is inserted only when the row is not applied. With this setting, Q Apply also tolerates codepage conversion errors when writing SQL text into the IBMQREP_EXCEPTIONS table and continues normal processing.

Table 84. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
ORACLE_EMPTY_STR	**Data type:** CHAR(1); **Nullable:** No, with default A flag that specifies whether the Q Apply program replaces an empty string in VARCHAR columns with a space. DB2 allows empty strings in VARCHAR columns. When a source DB2 VARCHAR column is mapped to an Oracle target, or to a DB2 server that is running with Oracle compatibility mode, the empty string is converted to a NULL value. The operation fails when the target column is defined with NOT NULL semantics. With **oracle_empty_str**=y, Q Apply replaces the NULL value with a space just before applying the data to the target and after any codepage conversion. If you are using SQL expressions in any Q subscriptions, take the following considerations into account: **Y** Q Apply replaces the NULL value with a one-character space just before applying the data to the target and after any codepage conversion. **N (default)** Q Apply applies the NULL value even though this action can result in an error if the target column does not accept nulls.
LOGMARKERTZ	**Data type:** CHAR(8); **Nullable:** No, with default A value that determines the time zone that the Q Apply program uses when it inserts source commit data into the IBMSNAP_LOGMARKER column of consistent-change data (CCD) tables or point-in-time (PIT) tables. **GMT (default)** Q Apply records the source commit timestamp in Greenwich Median Time. **LOCAL** Q Apply inserts the source commit timestamp in the local time of the Q Apply server **Note:** Because the value in the IBMSNAP_LOGMARKER column records the time that a row was committed at the source database, specifying **logmarkertz**=local is useful only when the Q Capture and Q Apply servers are in the same time zone with the same Daylight Savings Time or other time change in effect.

IBMQREP_APPLYTRACE table

The IBMQREP_APPLYTRACE table contains informational, warning, and error messages from the Q Apply program. You can set up automatic pruning of this table by using the TRACE_LIMIT parameter in the IBMQREP_APPLYPARMS table.

Server: Q Apply server

Default schema: ASN

Non-unique index: TRACE_TIME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 85 on page 512 provides a brief description of the columns in the IBMQREP_APPLYTRACE table.

Table 85. Columns in the IBMQREP_APPLYTRACE table

Column name	Description
OPERATION	**Data type:** CHAR(8); **Nullable:** No The type of message from the Q Apply program: **INFO** Describe actions that the Q Apply program takes. **WARNING** Describe conditions that could cause errors for the Q Apply program. **ERROR** Describe errors encountered by the Q Apply program.
TRACE_TIME	**Data type:** TIMESTAMP; **Nullable:** No The time at the Q Apply server when the row was inserted into this table.
DESCRIPTION	**Data type:** VARCHAR(1024); **Nullable:** No The ASN message ID followed by the message text. This column contains English-only text.
REASON_CODE	**Data type:** INTEGER; **Nullable:** Yes The reason code for the replication error message.
MQ_CODE	**Data type:** INTEGER; **Nullable:** Yes The reason code for the WebSphere MQ error message.

IBMQREP_DELTOMB table

The IBMQREP_DELTOMB table is an internal table used by the Q Apply program to record conflicting deletes in peer-to-peer replication. This table is pruned by the Q Apply program.

Server: Q Apply server

Default schema: ASN

Non-unique index: TARGET_NAME, TARGET_OWNER, VERSION_TIME DESC, KEY_HASH

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 86 provides a brief description of the columns in the IBMQREP_DELTOMB table.

Table 86. Columns in the IBMQREP_DELTOMB table

Column name	Description
TARGET_OWNER	**Data type**: VARCHAR(128); **Nullable**: No The owner name of the target table for which the conflicting delete was recorded
TARGET_NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the table for which the conflicting delete was recorded
VERSION_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp of the conflicting delete at the originating server.

Table 86. Columns in the IBMQREP_DELTOMB table (continued)

Column name	Description
VERSION_NODE	**Data type:** SMALLINT; **Nullable:** No Identifies the server in a peer-to-peer group where the conflicting delete originated.
KEY_HASH	**Data type:** INTEGER; **Nullable:** No The hash value of the key for the conflicting delete.
PACKED_KEY	**Data type:** VARCHAR (4096); **Nullable:** No The packed key value of the conflicting delete.

IBMQREP_DONEMSG table

The IBMQREP_DONEMSG table is an internal table used by the Q Apply program to record all transaction or administrative messages that have been received.

Server: Q Apply server

Default schema: ASN

Primary key: RECVQ, MGMSGID

Important: Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The records in this table help ensure that messages are not processed more than once (for example in the case of a system failure) before being deleted. The Q Apply program removes entries in this table on startup and during regular execution.

Table 87 provides a brief description of the columns in the IBMQREP_DONEMSG table.

Table 87. Columns in the IBMQREP_DONEMSG table

Column name	Description
RECVQ	**Data type:** VARCHAR(97); **Nullable:** No The name of the receive queue where the message arrived.
MQMSGID	**Data type:** CHAR(24) FOR BIT DATA; **Nullable:** No The WebSphere MQ message identifier of the message.

IBMQREP_EXCEPTIONS table

The IBMQREP_EXCEPTIONS table contains the SQL code and other information for row changes that could not be applied because of a conflict or SQL error.

Server: Q Apply server

Default schema: ASN

Non-unique index: EXCEPTION_TIME

The SQLCODE, SQLSTATE, and SQLERRMC fields are set to NULL for rows that were rolled back by the Q Apply program.

The size of this table depends on the number of conflicts or errors that you expect. You can use SQL to delete unneeded rows from the table.

Table 88 provides a brief description of the columns in the IBMQREP_EXCEPTIONS table.

Table 88. Columns in the IBMQREP_EXCEPTIONS table

Column name	Description
EXCEPTION_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The local timestamp at the Q Apply server when the error or conflict occurred. Default: Current timestamp
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue where the transaction message arrived.
SRC_COMMIT_LSN	**Data type:** VARCHAR(48) FOR BIT DATA, VARCHAR(10) FOR BIT DATA for z/OS Version 7; **Nullable:** No The logical log sequence number at the Q Capture server for the transaction.
SRC_TRANS_TIME	**Data type:** TIMESTAMP; **Nullable:** No The GMT timestamp at the Q Capture server for when the transaction was committed at the source database.
SUBNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the Q subscription that the transaction belonged to.

Table 88. Columns in the IBMQREP_EXCEPTIONS table (continued)

Column name	Description
REASON	**Data type:** CHAR(12); **Nullable:** No A description of the error or conflict that caused the transaction to be logged into the IBMQREP_EXCEPTIONS table. **NOTFOUND** An attempt to delete or update a row that did not exist. **DUPLICATE** An attempt to insert a row that was already present. **CHECKFAILED** The conflict detection rule was to check all values or check changed values, and a nonkey value was not as expected. **LOBXMLTOOBIG** A large object (LOB) value or XML document was too large to fit into a transaction message. The TEXT column specifies which data type was too large. **SQLERROR** An SQL error occurred, and it was not on the list of acceptable errors in the OKSQLSTATES column of the IBMQREP_TARGETS table. **OKSQLSTATE** An SQL error occurred, and it was on the list of acceptable errors in the OKSQLSTATES column of the IBMQREP_TARGETS table. **P2PDUPKEY** In peer-to-peer replication, a key update failed because a target row with the same key already existed, but was newer. **P2PNOTFOUND** In peer-to-peer replication, a delete or update failed because the target row didn't exist. **P2PVERLOSER** In peer-to-peer replication, a delete or update failed because the target row was newer than the row in the change message. **P2PINSERTED** In peer-to-peer replication, a key update was successfully applied as an insert because the old key row and the new key row were not found. The new key row was inserted into the target table.
SQLCODE	**Data type:** INTEGER; **Nullable:** Yes The SQL code returned by DB2 for the transaction.
SQLSTATE	**Data type:** CHAR(5); **Nullable:** Yes The SQL state number returned by DB2 for the transaction.
SQLERRMC	**Data type:** CHAR(70) FOR BIT DATA; **Nullable:** Yes The error message tokens from the SQLCA structure used for executing the transaction.
OPERATION	**Data type:** VARCHAR(18); **Nullable:** No The type of SQL operation that failed. Possible values are INSERT, INSERT(LOAD), DELETE, DELETE(LOAD), UPDATE, UPDATE(LOAD), KEY UPDATE, KEY UPDATE(LOAD).

Table 88. Columns in the IBMQREP_EXCEPTIONS table (continued)

Column name	Description
TEXT	**Data type:** CLOB; **Nullable:** No A SQL statement that describes the row that caused an error.
IS_APPLIED	**Data type:** CHAR(1); **Nullable:** No A flag that indicates whether the row was applied to the target table even though it was entered into the IBMQREP_EXCEPTIONS table. **Y** The row was applied because the conflict action that was specified for the Q subscription was F (force). **N** The transaction was not applied.
CONFLICT_RULE	**Data type:** CHAR(1); **Nullable:** Yes The type of conflict detection that resulted in the row being entered in the IBMQREP_EXCEPTIONS table. **K** Only key values were checked. **C** Changed nonkey values as well as key values were checked. **A** All values were checked.
SRC_TRANS_ID	**Data type:** VARCHAR(48); **Nullable:** Yes The identifier for the transaction that the row that could not be applied belongs to.

IBMQREP_RECVQUEUES table

The IBMQREP_RECVQUEUES table contains information about the WebSphere MQ local queues that are used by a Q Apply program to receive transactions from the source. Each Q Apply program can work with multiple receive queues. Each receive queue is uniquely identified by a row in the Q Apply receive queues table.

Server: Q Apply server

Default schema: ASN

Primary key: RECVQ

Unique index: REPQMAPNAME

Important: Do not alter this table by using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 89 provides a brief description of the columns in the IBMQREP_RECVQUEUES table.

Table 89. Columns in the IBMQREP_RECVQUEUES table

Column name	Description
REPQMAPNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the replication queue map that includes this receive queue.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue for this Q subscription.

Table 89. Columns in the IBMQREP_RECVQUEUES table (continued)

Column name	Description
SENDQ	**Data type:** VARCHAR(48); **Nullable:** Yes The name of the send queue that is used by the Q Capture program for this Q subscription.
ADMINQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the administration queue that is used by the Q Apply program to send control and error messages to the Q Capture program.
NUM_APPLY_AGENTS	**Data type:** INTEGER; **Nullable:** No, with default The number of agent threads that the Q Apply program uses to concurrently apply transactions from this receive queue. A value of 1 requests that transactions be executed in the order that they were received from the source table. Default: 16. Maximum number of agent threads: 128.
MEMORY_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The maximum amount of memory in megabytes that the Q Apply program can use as a buffer for messages that it gets from this receive queue. The default value is 32 MB; the maximum value is 2 GB.
CAPTURE_SERVER	**Data type:** VARCHAR(18); **Nullable:** No The name of the database where the Q Capture program that uses this receive queue runs. z/OS This is a location name.
CAPTURE_ALIAS	**Data type:** VARCHAR(8); **Nullable:** No The DB2 database alias that corresponds to the Q Capture server that is named in the CAPTURE_SERVER column.
CAPTURE_SCHEMA	**Data type**: VARCHAR(128); **Nullable**: No, with default The schema of the Q Capture program that uses this receive queue. Default: ASN
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that shows the receive queue's current status. **A (default)** Active: The Q Apply program is processing and applying the transactions from this queue. **I** Inactive: A severe error was encountered on the queue.
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Apply server of the last state change for this receive queue. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:**Yes The number for the ASN message about the queue state. For details, see the IBMQREP_APPLYTRACE table, or the Q Apply diagnostic log.
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the replication queue map that contains this receive queue.

Table 89. Columns in the IBMQREP_RECVQUEUES table (continued)

Column name	Description
SOURCE_TYPE	**Data type:** CHAR(1); **Nullable:** Yes The value of this attribute indicates the type of data source for each record. **C** Classic data source **D** DB2 data source
MAXAGENTS_CORRELID	**Data type:** INTEGER; **Nullable:** Yes The maximum number of Q Apply agents that can concurrently apply transactions that have the same *correlation ID*. The correlation ID identifies transactions that were started from the same z/OS job on the Q Capture server. You can use the MAXAGENTS_CORRELID value to limit parallelism for batch jobs that might have many dependencies that could cause lock contention. You can set the value by using the **`CREATE REPLQMAP`** or **`ALTER REPLQMAP`** commands. The value for the MAXAGENTS_CORRELID column cannot be greater than the value for the NUM_APPLY_AGENTS. If MAXAGENTS_CORRELID value is 1, the transactions will be applied serially. If the value is greater than one, for example *4*, the first four transactions will be applied in parallel and the following transactions are marked as dependent to any one of the transactions. When a transaction finishes, a dependent transaction with the same correlation ID is applied.
BROWSER_THREAD_ID	**Data type:** VARCHAR(9); **Nullable:** Yes The correlation ID for the browser thread that is processing this receive queue.

IBMQREP_SAVERI table

The IBMQREP_SAVERI table is an internal table that the Q Apply program uses to save information about referential integrity constraints for target tables. The Q Apply program drops referential integrity constraints while target tables are being loaded. The constraints are saved in this control table, and then restored after tables are loaded.

Server: Q Apply server

Default schema: ASN

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 90 provides a brief description of the columns in the IBMQREP_SAVERI table.

Table 90. Columns in the IBMQREP_SAVERI table

Column name	Description
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription to which the target tables belong.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that is specified for the Q subscription.

Table 90. Columns in the IBMQREP_SAVERI table (continued)

Column name	Description
CONSTNAME	**Data type:** VARCHAR (128), VARCHAR(18) for z/OS Version 7; **Nullable:** No The unique name of the constraint.
TABSCHEMA	**Data type:** VARCHAR(128); **Nullable:** No The schema or high-level qualifier of the child table on which the constraint is defined.
TABNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the child table on which the constraint is defined.
REFTABSCHEMA	**Data type:** VARCHAR(128); **Nullable:** No The schema of the parent table on which the constraint is defined.
REFTABNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the parent table on which the constraint is defined.
ALTER_RI_DDL	**Data type:** VARCHAR(1680); **Nullable:** No The ALTER TABLE statement that is used to restore referential integrity constraints.
TYPE_OF_LOAD	**Data type:** CHAR(1); **Nullable:** No A flag that indicates the type of load phase. **I** An automatic load. **E** A manual load.
DELETERULE	**Data type:** CHAR(1); **Nullable:** Yes The delete rule that is defined for the constraint. **A** NO ACTION **C** CASCADE **N** SET NULL **R** RESTRICT
UPDATERULE	**Data type:** CHAR(1); **Nullable:** Yes The delete rule that is defined for the constraint. **A** NO ACTION **R** RESTRICT

IBMQREP_SPILLQS table

The IBMQREP_SPILLQS table is an internal table used by the Q Apply program to record the temporary spill queues that it creates to hold messages while target tables are being loaded. The Q Apply program removes spill queues when they are no longer needed.

Server: Q Apply server

Default schema: ASN

Primary key: SPILLQ

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 91 provides a brief description of the columns in the IBMQREP_SPILLQS table.

Table 91. Columns in the IBMQREP_SPILLQS table

Column name	Description
SPILLQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the temporary spill queue that is used for this Q subscription.
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that is used for this Q subscription.

IBMQREP_SPILLEDROW table

The IBMQREP_SPILLEDROW table is an internal table used by the Q Apply program to record messages that are sent to a temporary spill queue while targets are being loaded.

Server: Q Apply server

Default schema: ASN

Primary key: SPILLQ, MQMSGID

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The Q Apply program deletes rows in this table after the messages they represent are taken from the spill queue and applied to the target table.

Table 92 provides a brief description of the columns in the IBMQREP_SPILLEDROW table.

Table 92. Columns in the IBMQREP_SPILLEDROW table

Column name	Description
SPILLQ	**Data type:** VARCHAR2(48); **Nullable:** No The name of the spill queue where the message was temporarily stored.
MQMSGID	**Data type:** CHAR(24) FOR BIT DATA; **Nullable:** No The WebSphere MQ message identifier of the message.

IBMQREP_TARGETS table

The IBMQREP_TARGETS table stores Q subscriptions information for the Q Apply program, including type and state, default error actions, and rules for handling row conflicts.

Server: Q Apply server

Default schema: ASN

Unique index: SUBNAME, RECVQ

Non-unique index: TARGET_OWNER ASC, TARGET_NAME ASC, RECVQ ASC, SOURCE_OWNER ASC, SOURCE_NAME ASC

Non-unique index: RECVQ, SUB_ID

Non-unique index: SPILLQ, STATE

Important: Do not alter this table by using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 93 provides a brief description of the columns in the IBMQREP_TARGETS table.

Table 93. Columns in the IBMQREP_TARGETS table

Column name	Description
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription. It must be unique for each source-target pair, and cannot contain blanks.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue used for this Q subscription.
SUB_ID	**Data type:** INTEGER; **Nullable:** Yes An integer that is generated by the Q Capture program and used to uniquely identify a Q subscription in the `subscription schema` message to the Q Apply program.
SOURCE_SERVER	**Data type:** VARCHAR(18); **Nullable:** No The name of the database or subsystem that contains the source table for this Q subscription. For z/OS, this is a location name. The value of the *Data source* entry in the ASNCLP configuration file for Classic replication, which is the name of the query processor on the Classic data server.
SOURCE_ALIAS	**Data type:** VARCHAR(8); **Nullable:** No The DB2 database alias that corresponds to the Q Capture server that is named in the SOURCE_SERVER column.
SOURCE_OWNER	**Data type:** VARCHAR(128); **Nullable:** No The schema name or high-level qualifier of the source table for this Q subscription.
SOURCE_NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the source table for this Q subscription.
SRC_NICKNAME_OWNER	**Data type:** VARCHAR(128); **Nullable:** Yes The schema of the nickname that is assigned to the source table for automatic loads that uses the LOAD from CURSOR utility when the Q Apply program is running on a non-z/OS platform.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
SRC_NICKNAME	**Data type:** VARCHAR(128); **Nullable:** Yes The nickname that is assigned to the source table for automatic loads that uses the LOAD from CURSOR utility when the Q Apply program is running on a non-z/OS platform.
TARGET_OWNER	**Data type:** VARCHAR(128); **Nullable:** No The schema name or high-level qualifier of the target table or stored procedure for this Q subscription.
TARGET_NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the target table for this Q subscription.
TARGET_TYPE	**Data type:** INTEGER; **Nullable:** No, with default A flag that indicates the type of replication target. **1 (default)** User table **2** Consistent-change-data (CCD) table **3** Reserved for future use. **4** Reserved for future use. **5** Stored procedure
FEDERATED_TGT_SRVR	**Data type:** VARCHAR(18); **Nullable:** Yes The name of the non-DB2 relational database that contains the Q subscription target.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that is inserted by the Q Apply program to describe the current state of the Q subscription. **I (default)** The Q Apply is not applying changes to the target because the Q subscription is new or in error. The Q Apply program discards all transactions that it receives for the Q subscription and waits for a new `subscription schema` message. **L** The Q Capture program has begun activating the Q subscription by sending a `subscription schema` message, and is sending changes from the source table. **E** The target table is being loaded by an external application. The Q Apply program is putting change messages in a spill queue while it waits for the table to be loaded. **D** The target table is loaded and the Q Apply program is ready to send a `load done` message to the Q Capture program. This state is used for automatic loads only. **F** The Q Apply program is applying messages from the spill queue. **T** The Q Apply program is terminating because of an error. It deactivates the Q subscription, then empties and deletes the spill queue. **A** The Q Apply program is applying changes to the target. **R** The Q Apply program is resuming operations after a Q subscription was placed in the spilled state (S) by the **spillsub** parameter of the **MODIFY** or **asnqacmd** command, or during the initial Q subscription load phase when a `load done received` message was received from the Q Capture program. The **resumesub** parameter places the Q subscription in the resuming state (R) after target table maintenance is complete. This state means that the Q Apply program is processing rows that are in the spill queue. Until the Q Apply program empties the spill queue, incoming rows continue to be spilled. When the Q Apply program empties the spill queue, the Q subscription is placed in the active (A) state and normal operations resume. **S** The Q Apply program is placing rows for the Q subscription in a temporary spill queue. Specifying the **spillsub** parameter places the Q subscription in the spilling state so that you can perform maintenance on the target table. **W** **For peer-to-peer configurations with more than two servers and a load phase:** The Q Apply program has seen a subscription schema message, is actively spilling changes to the source table, and is waiting for a confirmation (another schema message) from the Q Capture program to start loading. **P** An SQL error was detected while Q Apply was applying messages from a spill queue and the Q subscription was in F state. The Q Apply agent thread that was processing the spill queue stops. Future changes for the Q subscription are placed in the same spill queue until the **resumesub** parameter of the MODIFY or asnqacmd commands is issued. **U** An internal state that indicates that the Q Apply program created this Q subscription, and the Q subscription has not been activated yet.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Apply server of the last change in state for this Q subscription. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:** Yes The number for the ASN message about the Q subscription state. For details, see the IBMQREP_APPLYTRACE table or the Q Apply diagnostic log.
SUBTYPE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates the type of replication that the Q subscription is involved in. **U (default)** Unidirectional replication. **B** Bidirectional replication. **P** Peer-to-peer replication.
CONFLICT_RULE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program how to look for conflicting changes to the target table. Inserts are always checked using the K (check only keys) rule because there are no before values and keys must be used to detect conflicts. **K (default)** Check only keys. The Q Apply program looks for conflicts by comparing the current value of the primary key in the target table with the old key value sent from the source table. **C** Check changed columns. Before updating target columns, the Q Apply program makes sure their current value matches the before values in the source columns. For deletes, the Q Apply program checks all columns. **A** Check all columns. Before updating or deleting a row, the Q Apply program makes sure that the current values in all columns match the old values in the source table. **V** Check version. In peer-to-peer replication, the Q Apply program checks the version column before applying a row.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
CONFLICT_ACTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program what to do when a row change conflicts: **I (default)** The Q Apply program does not apply the conflicting row but applies other rows in the transaction. **F** The Q Apply program tries to force the change. This requires that the Q Capture program send all columns, so the CHANGED_COLS_ONLY value must be set to N (no) in the IBMQREP_SUBS table. This is the default value while a target table is being loaded. **D** The Q Apply program does not apply the conflicting row but applies other rows in the transaction. Then it disables the Q subscription, stops applying transactions to the target, and sends an error report to the Q Capture program on the administration queue. **S** The Q Apply program rolls back the transaction, commits, and then stops. **Q** The Q Apply program stops reading from the queue. All conflicting rows are inserted into the IBMQREP_EXCEPTIONS table.
ERROR_ACTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program what to do in case of an error such as an SQL error (other than a conflict) that prevent it from applying a row change. This flag does not affect Q Apply behavior for errors that are not related to applying a row change, for example WebSphere MQ errors related to reading from a queue. **Q (default)** The Q Apply program stops reading from the queue. **D** The Q Apply program does not apply the conflicting row but applies other rows in the transaction. Then it disables the Q subscription, stops applying transactions to the target, and sends an error report to the Q Capture program on the administration queue. **S** The Q Apply program rolls back the transaction, commits, and then stops. **B** The Q Apply program starts putting change messages for the Q subscription in a temporary spill queue while the SQL error is being fixed. Use the **`resumesub`** parameter of the **`MODIFY`** command or **`asnqacmd`** command to prompt Q Apply to start applying messages from the spill queue to targets. To use this error action you must specify a model queue for the Q subscription when you create or change it with the replication administration tools. This error action is not supported for tables with referential integrity constraints. All conflicting rows are inserted into the IBMQREP_EXCEPTIONS table.
SPILLQ	**Data type:** VARCHAR(48); **Nullable:** Yes The name of the temporary spill queue that the Q Apply program creates when it loads targets.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
OKSQLSTATES	**Data type:** VARCHAR(128); **Nullable:** Yes A list of space-separated SQLSTATE values that the Q Apply program does not consider as errors. You specify these values when you define a Q subscription. Values that are entered for OKSQLSTATES prompt the Q Apply program to bypass the error action that is specified for the Q subscription. OKSQLSTATES values do not affect conflicts such as duplicates and row-not-found errors, which are handled by the conflict action that is specified for the Q subscription. **Restriction:** The OKSQLSTATES feature is not supported when all of these conditions are true: • The Q subscription uses expressions. • At least one key column is used in the expression. • The evaluation of the expression fails because of bad data or another reason.
SUBGROUP	**Data type:** VARCHAR(30); **Nullable:** Yes The name of the peer-to-peer replication group that includes this Q subscription.
SOURCE_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the source server in a peer-to-peer Q subscription. Default: 0
TARGET_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the target server in a peer-to-peer Q subscription. Default: 0
GROUP_INIT_ROLE	**Data type:** CHAR(1); **Nullable:** Yes The role of this target server in the process of initializing a peer-to-peer Q subscription. **I** The initiator of the peer-to-peer group, where the CAPSTART signal is entered into the IBMQREP_SIGNAL table to initialize the subscription. **M** A server in the peer-to-peer group that is not used to initialize the subscription. **N** A new server that is in the process of joining the peer-to-peer group.
HAS_LOADPHASE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the target table will be loaded with data from the source. **N (default)** The target will not be loaded. **I** An automatic load. The Q Apply program loads the target table. **E** A manual load. An application other than the Q Apply program loads the target table.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
LOAD_TYPE	**Data type:** SMALLINT; **Nullable:** No, with default A flag to indicate which utility is called to load the target table when HAS_LOADPHASE is I (automatic load). **0 (default)** The Q Apply program selects the load utility from among the options below. **1** Use the LOAD from CURSOR utility. The utility is invoked with an option to delete all data in the target table before replacing it with data from the source (this is called the replace option). **101** Use the LOAD from CURSOR utility. The utility is invoked with an option to append source data to the target table without deleting target table contents. This is called the resume option on z/OS targets and the insert option on Linux, UNIX, and Windows targets. **2** Use the EXPORT and IMPORT utilities. The utilities are invoked with an option to delete all data in the target table before replacing it with data from the source (this is called the replace option). **102** **Linux, UNIX, and Windows targets:** Use the EXPORT and IMPORT utilities. The LOAD utility is invoked with an option to append source data to the target table without deleting target table contents (this is called the insert option). **3** Use the EXPORT and LOAD utilities. The utilities are invoked with an option to delete all data in the target table before replacing it with data from the source (this is called the replace option). **103** **Linux, UNIX, and Windows targets:** Use the EXPORT and LOAD utilities. The LOAD utility is invoked with an option to append source data to the target table without deleting target table contents (this is called the insert option). **4** Select from a replication source and use the LOAD utility, or for Oracle targets use the SQL*Loader utility (unidirectional replication only). The utilities are invoked with an option to delete all data in the target table before replacing it with data from the source (replace option). **Oracle targets:** To use SQL*Loader, you must create a password file by using the **`asnpwd`** command in the directory that is specified by the **`apply_path`** parameter or the directory from which Q Apply is invoked with the following values for these keywords: • **`alias`**: The Oracle `tnsnames.ora` entry that refers to the Oracle server (the same name that is used for the NODE option of the CREATE SERVER command for setting up federation). • **`id`**: The remote user ID for connecting to Oracle. • **`password`**: The password for connecting to Oracle. The file must have the default name `asnpwd.aut`. Before starting the Q subscription, you should test connectivity with this command: `$> sqlplus` *`id/password@alias`*. **104** Select from a replication source and use the LOAD utility, or for Oracle targets use the SQL*Plus utility. The utilities are invoked with an option to append source data to the target table without deleting target table contents (resume or insert option). To use SQL*Plus, follow the instructions in the entry for LOAD_TYPE 4 above.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
LOAD_TYPE (Continued)	**5** Linux, UNIX, and Windows targets: Select from a replication source and use the IMPORT utility. The utility is invoked with an option to delete all data in the target table before replacing it with data from the source (this is called the replace option). **105** **Linux, UNIX, and Windows targets:** Select from a replication source and use the IMPORT utility. The utility is invoked with an option to append source data to the target table without deleting target table contents (this is called the insert option).
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the Q subscription.
SEARCH_CONDITION	**Data type:** VARCHAR(2048); **Nullable:** Yes The search condition that is used to filter rows for the Q subscription. This must be an annotated select WHERE clause, with a single colon directly in front of the names of the source columns.
MODELQ	**Data type:** VARCHAR(36); **Nullable:** Yes, with default The name of the model queue that the Q Apply program uses to create spill queues during the target loading process. Default: IBMQREP.SPILL.MODELQ
CCD_CONDENSED	**Data type:** CHAR(1); **Nullable:** Yes, with default A flag that indicates whether a CCD target table is condensed or noncondensed. **N (default)** The CCD table is noncondensed, which means that it contains multiple rows with the same key value, one row for every change that occurs to the source table. **Y** The CCD table is condensed, which means that it contains one row for every key value in the source table and contains only the latest value for the row.
CCD_COMPLETE	**Data type:** CHAR(1); **Nullable:** Yes, with default A flag that indicates whether a CCD target table is complete or noncomplete. **N (default)** The CCD table is noncomplete, which means that it contains only changes to the source table and starts with no data. **Y** The CCD table is complete, which means that it contains every row of interest from the source table and is initialized with a full set of source data.
SOURCE_TYPE	**Data type:** CHAR(1); **Nullable:** Yes The value of this attribute indicates the type of data source for each record. **C** Classic data source **D** DB2 data source
SCHEMA_SUBNAME	**Data type:** VARCHAR(64); **Nullable:** Yes The name of the source schema-level Q subscription if this table-level Q subscription was created by the Q Apply program or ASNCLP command-line program. The value is null otherwise.

Table 93. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
SUB_CREATOR	**Data type:** VARCHAR(12); **Nullable:** Yes Identifies how this Q subscription was created. The values are "asnclp," "replcenter" (the Replication Center), and "asnqapp" (the Q Apply program). When the Q Apply control tables are migrated from a release before Version 10 on Linux, UNIX, and Windows, the column value is NULL.

IBMQREP_TRG_COLS table

The IBMQREP_TRG_COLS table identifies the mapping between a column in the source table and a column in the target table, or between the source column and a parameter if the target is a stored procedure.

Server: Q Apply server

Default schema: ASN

Unique index: RECVQ, SUBNAME, TARGET_COLNAME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

The Q Apply program enters values in this table based on information that it receives in the schema message from the Q Capture program.

Table 94 provides a brief description of the columns in the IBMQREP_TRG_COLS table.

Table 94. Columns in the IBMQREP_TRG_COLS table

Column name	Description
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that is used for this Q subscription.
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription.
SOURCE_COLNAME	**Data type:** VARCHAR(1024); **Nullable:** No This column can contain one of the following values: • The name of the source column that is being replicated or published • An SQL or XML expression that is used to create the target column contents • An auditing column in a consistent-change-data (CCD) table
TARGET_COLNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the target column. If the target is a stored procedure, this column contains the name of the parameter to which the Q Apply program passes the source column value.
TARGET_COLNO	**Data type:** INTEGER; **Nullable:** Yes A number assigned to a target column. If the target is a stored procedure, this column contains a number assigned to the parameter to which the Q Apply program passes the source column value.

Table 94. Columns in the IBMQREP_TRG_COLS table (continued)

Column name	Description
MSG_COL_CODEPAGE	**Data type:** INTEGER; **Nullable:** Yes An identifier for the code page that is used to encode the value of the source column.
MSG_COL_NUMBER	**Data type:** SMALLINT; **Nullable:** Yes The source column's order of appearance in a change message, starting from 0.
MSG_COL_TYPE	**Data type:** SMALLINT; **Nullable:** Yes The DB2 data type of the target column.
MSG_COL_LENGTH	**Data type:** SMALLINT; **Nullable:** Yes The maximum data length defined on the target column.
IS_KEY	**Data type:** CHAR(1); **Nullable:** No A flag that indicates whether the source column is part of the key for the source table. If the value of this flag does not match the target table key definition, the Q Apply program rejects the schema message and invalidates the Q subscription: **Y** The column is part of the source table key. **N** The column is not part of the source table key. **Restriction:** Large-object (LOB) columns and LONG columns cannot be used in the replication or publishing key.
MAPPING_TYPE	**Data type:** CHAR(1); **Nullable:** Yes, with default A flag that indicates the type of mapping between the source column and the target column. **R (default)** A regular mapping where the value in the source column corresponds to the value in the target column. **E** An SQL expression is used to generate the target column contents. **A** An auditing column in a CCD table. The column does not exist at the source table.
SRC_COL_MAP	**Data type:** VARCHAR(2000); **Nullable:** Yes, with default The column position, data type, length, and code page for all of the columns that are used in an SQL expression. Default value: NULL
BEF_TARG_COLNAME	**Data type:** VARCHAR(128); **Nullable:** Yes, with default Specifies the before-image column name, if one exists (for CCD targets only). Default value: NULL

Detailed structure of versioning columns for peer-to-peer replication

Tables that participate in peer-to-peer replication require two versioning columns, a timestamp column and a small integer column, which are maintained by triggers. These two columns allow the Q Capture and Q Apply programs to perform version-based conflict checking that is required for peer-to-peer replication.

The columns also allow the two programs to resolve conflicts so that the tables within a Q subscription group maintain convergence. The values in these columns reflect which version of a row is most current.

These extra columns and triggers are created when you use the ASNCLP command-line program or Replication Center to create Q subscriptions for peer-to-peer replication. When you create a Q subscription for peer-to-peer replication, you must subscribe to both of these columns.

Table 95 provides a brief description of the extra columns in user tables that are required for peer-to-peer replication.

Table 95. Extra columns required for peer-to-peer replication

Column name	Description
IBMQREPVERNODE	**Data type:** SMALLINT; **Nullable:** No, with default A number that identifies the database or subsystem that contains the table within a peer-to-peer group. Default: 0
IBMQREPVERTIME	**Data type:** TIMESTAMP; **Nullable:** No, with default A timestamp that records when a change occurs in the table. Default: 0001-01-01-00.00.00

Chapter 26. Structure of XML messages for event publishing

In event publishing, a Q Capture program and a user application exchange Extensible Markup Language (XML) messages.

The following topics explain more about the XML messages that are exchanged and their structure:

XML message types and requirements

A Q Capture program uses XML messages to send transactions or row-level changes to a user application. The Q Capture program and user application also use XML messages to communicate.

The following topics describe the XML message types and their technical requirements:

Message types

A Q Capture program sends data messages and informational messages to a user application, and the user application sends control messages to the Q Capture program.

Table 96 describes the three types of messages.

Table 96. Messages sent by a Q Capture program and user application

Type of message	Direction	Description
Data	Q Capture to user application	Contains one of the following things from the source table: • All or part of a transaction • A single row operation • All or part of a large object (LOB) value from a row operation within a transaction
Informational	Q Capture to user application	Provides information about the status of the Q Capture program or a publication.
Control	User application to Q Capture	Asks the Q Capture program to activate or deactivate a publication, invalidate a send queue, or confirm that a target table is loaded.

Technical requirements for XML messages

The Q Capture program generates messages in the form of XML document instances according to the following guidelines.

- Messages are encoded in Unicode by using UTF-8 (code page 1208) as specified in the XML 1.0 (2nd edition), W3C Recommendation, 6 October 2000.
- Message structure follows the XML Schema Language (Part 1: Structure and Part 2: Datatypes), W3C Recommendation, 2 May 2001.

Changes from the source database are converted into messages by using the version of IBM's International Components for Unicode (ICU4C) that is shipped with DB2.

To interpret control messages from the subscribing application, the Q Capture program uses the IBM XML parser XML4C version 5.3.

How XML delimiters are handled in character data

In data messages from a Q Capture program, the values from subscribed columns appear between XML tags that describe the column data type.

For example, the values 222 and Hello from a source table are encoded as `<integer>222</integer>` and `<varchar>Hello</varchar>`.

Because the angle bracket (< or >) and ampersand (&) characters are predefined XML delimiters, the Q Capture program translates these characters when they occur in column values as follows:

- < to &lt;
- > to &gt;
- & to &amp;

Also, when the apostrophe (') or double quotation mark (") appear in attribute values, the Q Capture program translates these characters as follows:

- ' to &apos;
- " to &quot;

The resulting messages are valid XML document instances.

Structure of messages from Q Capture to a user application

A Q Capture program sends both data messages and informational messages to a user application. The data messages convey changes to a source table that is part of a publication. The informational messages either confirm a user application's request with a control message, report on the status of the Q Capture program, or report publication errors.

List of messages from Q Capture to a user application

The Q Capture program sends two types of messages to a user application: data messages and control messages.

Data messages
Contain changes to a source table. Table 97 provides a quick reference to the types of data messages.

Informational messages
Report on the status of a Q Capture program or publication. Table 98 on page 535 provides a quick reference to the types of informational messages.

Table 97. Data messages from the Q Capture program to a user application

Message type	Description
Transaction	Contains one or more insert, delete, or update operations to a source table. These operations belong to the same database transaction. Also contains commit information for the transaction.

Table 97. Data messages from the Q Capture program to a user application (continued)

Message type	Description
Row operation	Contains a single insert, delete, or update operation to a source table. Also contains commit information about the database transaction that this row is part of.
Large object (LOB)	Contains some or all of the data from a LOB value in the source table. LOB messages are sent separately from the transaction messages and row operation messages that the LOB values belong to.

Table 98. Informational messages from the Q Capture program to a user application

Message type	Description
Subscription deactivated	Tells the user application that the Q Capture program deactivated a publication.
Load done received	Acknowledges that the Q Capture program received the message that the target table is loaded.
Error report	Tells the user application that the Q Capture program encountered a publication error.
Heartbeat	Tells the user application that the Q Capture program is still running when it has no data messages to send.
Subscription schema	Contains information about the source table and its columns. Also contains data-sending options, send queue name, and information about the Q Capture program and source database.
Add column	Contains information about a column that was added to an existing publication.

msg: Root element for XML messages from Q Capture to a user application

The msg element is the root element for all data messages and informational messages from the Q Capture program to a user application.

Table 99 describes the msg element.

Table 99. Element description for the msg element (Q Capture program to a user application)

Name	Properties
msg	Not empty, complex type, complex content

Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="XML_schema_instance"
      xsi:noNamespaceSchemaLocation="schema_document"
      version="version" dbName="database_name">

      elements

</msg>
```

Details

XML_schema_instance
The URL of the XML schema instance. For event publishing, the URL is www.w3.org/2001/XMLSchema-instance. XML data type: string.

schema_document
The file name of the XML schema document. XML namespace is not supported in event publishing because messages refer to one XML schema only. Messages from a Q Capture program to a user application refer to the mqcap.xsd schema document. XML data type: string.

version
The version of the XML message schema. XML data type: string.

database_name
The name of the source database or subsystem. XML data type: string.

elements
One of the elements that the msg element contains. Only one of these elements appears in each message:

- trans
- rowOp
- lob
- subDeactivated
- loadDoneRcvd
- heartbeat
- errorRpt
- subSchema

Example:

The following example shows a message element.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1">

       elements

 </msg>
```

Where *elements* represents one of the following elements: trans, rowOp, lob, subDeactivated, loadDoneRcvd, heartbeat, errorRpt, or subSchema.

Transaction message

A `transaction` message contains one or more insert, update, or delete row operations on the source table. The `transaction` message also contains information about the time that the transaction was committed at the source database, and a time-based log sequence number.

If a `transaction` message exceeds the maximum message size defined for the send queue, the Q Capture program can divide it into multiple `transaction` messages. Each message in a divided transaction is numbered by using the segment number attribute of the transaction element (trans). All of the messages in a divided transaction share the same value for commit time and commit logical sequence number.

Within a `transaction` message, the trans element contains a hierarchy of other elements that describe the type of row operation, the attributes of each column, the data type of the column value, and the value itself. The following sections describe the elements contained by the trans element.

- "Transaction element (trans)"
- "Row operation elements (insertRow, updateRow, and deleteRow)" on page 538
- "Column element (col)" on page 540
- "Elements for a single-column value" on page 541
- "Elements for a double-column value" on page 543
- "Before-value and after-value elements (beforeVal and afterVal)" on page 544

Transaction element (trans)

The transaction element (trans) is contained by the msg element, and it contains one of the three row operation elements (insertRow, updateRow, or deleteRow).

Table 100 describes the trans element.

Table 100. Element description for trans

Name	Properties
trans	Not empty, complex type, complex content

Structure

```
<trans isLast="is_last_indicator" segmentNum="segment_number"
     cmitLSN="commit_logical_sequence_number" cmitTime="commit_time"
     authID="authorization_ID" correlationID="correlation_ID"
     planName="plan_name">

              elements

</trans>
```

Details

is_last_indicator
: A boolean value that indicates whether the `transaction` message is the last message in a database transaction. If it is the last message, the value is 1 (true). If it is not the last message, the value is 0 (false). XML data type: boolean.

  If a database transaction contains row operations with LOB columns, and there are LOB values to be published, then these LOB values are sent in separate LOB messages after the last `transaction` message. In this case, the last message in a database transaction is not the last `transaction` message, but a LOB message.

segment_number
: A positive integer that indicate the message's segment number in a divided `transaction` message. XML data type: positiveInteger.

commit_logical_sequence_number
: The commit logical sequence number (a time-based log sequence number) of the COMMIT statement for the transaction. XML data type: string.

commit_time
: The timestamp of the COMMIT statement for the transaction in Greenwich mean time (GMT), formatted in microseconds. XML data type: dateTime.

authorization_ID
The user ID of the user who updated the source table. XML data type: string.

correlation_ID
z/OS The correlation ID (normally a job name) that ran the source update. XML data type: string.

plan_name
z/OS The plan name that is associated with the transaction. XML data type: string.

elements
Each trans element contains one or more of these elements:

- insertRow
- updateRow
- deleteRow

Example

The following example shows a transaction element that contains one or more of the insert row, update row, or delete row elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1">
       <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
             cmitTime="2003-10-31T12:12:12.000122">

             insertRow, updateRow, or deleteRow

       </trans></msg>
```

Where *insertRow, updateRow, or deleteRow* represents the elements that are explained in "Row operation elements (insertRow, updateRow, and deleteRow)."

Row operation elements (insertRow, updateRow, and deleteRow)

Within a transaction element, the row operation elements (insertRow, updateRow, and deleteRow) describe the type of SQL operation that is performed on a row of the source table. Each of these elements contains one or more column elements (col) that describe changes to subscribed columns.

Table 101 describes the insertRow, deleteRow, and updateRow elements.

Table 101. Element description for insertRow, deleteRow, and updateRow

Name	Properties
insertRow	Not empty, complex type, complex content
deleteRow	Not empty, complex type, complex content
updateRow	Not empty, complex type, complex content

Structure

```
<insertRow subName="publication_name" srcOwner="source_owner"
       srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">

             elements
```

```
</insertRow>


<deleteRow subName="publication_name" srcOwner="source_owner"
       srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">

              elements

</deleteRow>


<updateRow subName="publication_name" srcOwner="source_owner"
       srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">

               elements

</updateRow>
```

Details

publication_name
: The name of the publication to which this row operation belongs. XML data type: string.

source_owner
: The schema of the source table where the row operation originated. XML data type: string.

source_name
: The name of the source table. XML data type: string.

row_number
: If a row operation includes large object (LOB) columns, this attribute will be generated to identify the position number of the row operation in the database transaction. This attribute does not have a default value. XML data type: positiveInteger.

LOB_indicator
: A boolean value that indicates whether the row operation contains LOB columns. If it contains LOB columns, the value is 1 (true). The default value is 0 (false). XML data type: boolean.

elements
: One or more column elements (col) contained by the insertRow, updateRow, or deleteRow element.

Example

The following example shows insertRow, updateRow, and deleteRow elements within a `transaction` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
        <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <insertRow subName="S1" srcOwner="USER1" srcName="T1">

                     column_element

              </insertRow>

              <deleteRow subName="S1" srcOwner="USER1" srcName="T1">
```

```
          column_element

     </deleteRow>

     <updateRow subName="S1" srcOwner="USER1" srcName="T1">

          column_element

     </updateRow>       </trans>
</msg>
```

Where *column_element* represents the column element that is explained in "Column element (col)."

Column element (col)

The column element (col) describes the name of a subscribed column in the source table, and it also tells whether the column is part of the key to be used for publishing. A col element within an insert or delete operation contains a single value only. Within an update operation, the col element can contain a before value and an after value, depending on the options for sending data that you specified for the publication.

If you specified for the publication queue map to send bad data if code page conversion fails, the invalidData and rawData attributes are set to 1 for col elements for insert operations. For update operations, the invalidData and rawData attributes are set to 1 on either the beforeVal or afterVal elements.

Table 102 describes the col element.

Table 102. Element description for col

Name	Properties
col	Not empty, complex type, complex content

Structure

```
<col name="column_name" isKey="key_indicator" invalid_data_options>

    single_or_double_column_value

</col>
```

Details

column_name
: The name of a subscribed column in the source table. XML data type: string.

key_indicator
: Optional: A boolean value that indicates whether the column is part of the key to be used for publishing. The default is 0 (false). If it is a key column, the value is 1 (true). XML data type: boolean.

invalid_data_options
: Optional: If you specify for the publication queue map to publish data when code page conversion errors are encountered, the data is published in a hexadecimal format. For insert operations, the invalidData and rawData attributes are set to 1 (true) on the col element. For update operations, the invalidData and rawData attributes are set to 1 (true) on either the beforeVal or afterVal elements. The default is 0 (false). XML data type: boolean.

single_or_double_column_value
If the column element is part of an insert or delete operation at the source table, it will contain one of the single-column- value elements. For update operations, the column element can contain a double-column value, which includes both a before value and an after value.

Example

The following example shows an insert operation that contains single column values, and an update operation that contains double column values.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
        <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <insertRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">

                            single_column_value

                     </col>
                     <col name="COL2">

                            single_column_value

                     </col>
              </insertRow>
              <updateRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">

                            double_column_value

                     </col>
                     <col name="COL2">

                            double_column_value

                     </col>
              </updateRow>       </trans>
</msg>
```

Where *single_column_value* represents the elements that are explained in "Elements for a single-column value," and *double_column_value* represents the elements that are explained in "Elements for a double-column value" on page 543.

Elements for a single-column value

A single-column-value element contains an actual value from the source table. The Q Capture program uses single-column-value elements for insert and delete operations. These elements are named for the data type of the source column, and do not contain other elements. If the value from the source table is NULL, the element is empty and the xsi:nil attribute is set to 1 (true).

Table 103 on page 542 describes the single-column-value elements. All of the elements are of complex type, and simple content. The blob, clob, and dbclob elements, which convey LOB data, are always empty because the data from a large object is sent in a separate LOB message. The blob, clob, and dbclob elements do not have the xsi:nil attribute.

Table 103. Element descriptions for single column value

Name	XML data type	Value's data format
smallint	short	
integer	integer	
bigint	long	
float	float (32 bits) double (64 bits)	[-]d.dddddE[-|+]dd [-]d.dddddddddddddddE[-|+]dd
real	float	
double	double	
decimal	decimal	
date	date	YYYY-MM-DD
time	time	HH:MM:SS.SSS
timestamp	dateTime	YYYY-MM-DDTHH:MM:SS.SSS
char	string	
varchar	string	
long varchar	string	
bitchar	hexBinary	
bitvarchar	hexBinary	
bitlongvarchar	hexBinary	
graphic	string	
vargraphic	string	
longvargraphic	string	
rowid	hexBinary	
blob	hexBinary	
clob	string	
dbclob	string	

Structure

```
<data_type xsi:nil="null_indicator">value</data_type>
```

Details

data_type
: The data type of the column in the source table. This data type is used to name the element.

null_indicator
: Optional: An integer that indicates whether the source column contains a NULL value. The default is 0 (false). If the source column contains a NULL value, the value of this attribute is 1 (true). The blob, clob, and dbclob elements do not have this attribute. XML data type: boolean.

value
: The actual value in the source column. If the source value is NULL or a LOB value, the element is empty.

Example

The following example shows an insert operation with single column values of 222 in a key column with an integer data type and Hello in a nonkey column with a varchar data type. The example also shows a delete operation of the row with a single-column value of 222 in a key column with an integer data type.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
       <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <insertRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>222</integer>
                     </col>
                     <col name="COL2">
                            <varchar>Hello</varchar>
                     </col>
              </insertRow>
              <deleteRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>222</integer>
                     </col>
              </deleteRow>
       </trans>
</msg>
```

Elements for a double-column value

Double-column-value elements are used in update operations when the Q Capture program needs to send both before and after values from source columns. In messages, the Q Capture program sends before values of key columns that have changed. It sends before values of nonkey columns that have changed if the BEFORE_VALUES data-sending option is set to "Yes" for the publication. If the before and after values are the same, only the after-value element (afterValue) is used.

All double-column-value elements except blob, clob, and dbclob are not empty, have a complex type, and have complex content. The elements blob, clob, and dbclob are always empty and not nullable. Double-column-value elements have no attributes. For a description of the double-column-value elements, see “Elements for a single-column value” on page 541.

Structure

```
<data_type>

  elements

</data_type>
```

Details

data_type
: The data type of the column in the source table. This data type is used to name the element. If the data type is blob, clob, or dbclob, the elements are empty and not nullable.

elements
: One or both of the beforeValue or afterValue elements, or empty for blob, clob, and dbclob.

Example

The following example shows double-column-value elements for:

- A key column (integer data type) that has changed.
- A nonkey column (varchar data type) that has changed, but the BEFORE_VALUES data-sending option for the publication is set to "No."

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
        <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <updateRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>

                                   beforeValue
                                   afterValue

                            </integer>
                     </col>
                     <col name="COL2">
                            <varchar>

                                  afterValue

                            </varchar>
                     </col>
              </updateRow>
       </trans>
</msg>
```

Where *beforeValue* and *afterValue* represent the elements that are explained in "Before-value and after-value elements (beforeVal and afterVal)."

Before-value and after-value elements (beforeVal and afterVal)

Before-value and after-value elements (beforeVal and afterVal) contain actual values from the source table. These elements are used in update operations for key columns that have changed, and for changed nonkey columns when the BEFORE_VALUES data-sending-option for the publication is set to "Yes." If the publication calls for before values to be sent and the value in the source column has not changed, only the afterVal element is used. If the value from the source table is NULL, the elements are empty and the xsi:null attribute is set to 1 (true).

Table 104 describes the beforeVal and afterVal elements.

Table 104. Element descriptions for beforeVal and afterVal

Name	Properties
beforeVal	Nullable, complex type, simple content
afterVal	Nullable, complex type, simple content, optional

Structure

```
<beforeVal xsi:nil="null_indicator" invalidData="invalid_indicator"
    rawData="rawdata_indicator">value</beforeVal>
<afterVal xsi:nil="null_indicator" invalidData="invalid_indicator"
    rawData="rawdata_indicator">value</afterVal>
```

Details

null_indicator
: Optional: An integer that indicates whether the value in the source column is NULL. The default is 0 (false). If the source column contains a NULL value, the value of this attribute is 1 (true). XML data type: boolean.

value
: The actual value in the source column. If the source value is NULL, the element will be empty.

invalid_indicator
: Optional: If you specify for the publication queue map to publish data when code page conversion errors are encountered, the data is published in a hexadecimal format. If the before value cannot be converted, the invalidData and rawData attributes are set to 1 (true) on the beforeVal element. The default is 0 (false). XML data type: boolean.

rawdata_indicator
: Optional: If you specify for the publication queue map to publish data when code page conversion errors are encountered, the data is published in a hexadecimal format. If the after value cannot be converted, the invalidData and rawData attributes are set to 1 (true) on the afterVal element. The default is 0 (false). XML data type: boolean.

Example

The following example shows an update operation where the key column's value of 222 did not change (only the afterVal element is used), and where a varchar column in the same row changed from "Hello" to NULL. In this case, the BEFORE_VALUES option for the publication is set to "Yes."

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
       <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
            cmitTime="2003-10-31T12:12:12.000122">
              <updateRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>
                                   <afterVal>222</afterVal>
                            </integer>
                     </col>
                     <col name="COL2">
                            <varchar>
                                   <beforeVal>Hello</beforeVal>
                                   <afterVal xsi:nil="1"/>
                            </varchar>
                     </col>
              </updateRow>
       </trans>
</msg>
```

Row operation message

A `row operation` message contains one insert, update, or delete operation from the source table. In a `row operation` message, the message element (msg) contains a row operation element (rowOp).

A `row operation` message must not exceed the maximum message size that is defined for the send queue. `Row operation` messages that exceed this size cannot be divided into multiple messages. In `row operation` messages, any inserts, updates, or deletes that belong to a transaction have the same commit time and commit logical sequence number. If LOB messages follow the row operation message, the row operation message contains an attribute to indicate that it is not the last message in the row operation from the source database.

Table 105 describes the rowOp element.

Table 105. Element description for rowOp

Name	Properties
rowOp	Not empty, complex type, complex content

Structure

```
<rowOp cmitLSN="commit_logical_sequence_number"
    cmitTime="commit_time" isLast="is_last_indicator"
    authID="authorization_ID" correlationID="correlation_ID"
    planName="plan_name">


        elements

</rowOp>
```

Details

commit_logical_sequence_number
: The commit logical sequence number (a time-based log sequence number) of the COMMIT statement for the transaction. XML data type: string.

commit_time
: The timestamp of the COMMIT statement for the transaction in Greenwich mean time (GMT), formatted in microseconds. XML data type: dateTime.

is_last_indicator
: Optional: A boolean value that indicates whether the row operation message is the last message in a row operation from the source database. If LOB messages follow the row operation message, the value is set to 0 (false). This attribute has no default value. XML data type: boolean.

authorization_ID
: The user ID of the user who updated the source table. XML data type: string.

correlation_ID
: **z/OS only:** The correlation ID (normally a job name) that ran the source update. XML data type: string.

plan_name
: **z/OS only:** The plan name that is associated with the transaction that the row belongs to. XML data type: string.

elements
: Each rowOp element contains one of these elements:
- insertRow
- updateRow
- deleteRow

Example

The following example shows a row operation element that contains an insertRow, updateRow, or deleteRow element.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0" dbName="DB1">
       <rowOp cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">

              insertRow, deleteRow, or updateRow

       </rowOp></msg>
```

Where *insertRow, updateRow, or deleteRow* represents the elements that are explained in "Transaction message" on page 536.

Large object (LOB) message

A large object (LOB) message transmits some or all of the data from a column in the source table that contains a large object value: BLOB (binary large object), CLOB (character large object), or DBCLOB (double-byte character large object).

Each LOB message contains data from at most one LOB value in the source table. The Q Capture program can divide a LOB value into multiple LOB messages if the value exceeds the LOB message buffer size determined by the Q Capture program. The buffer size can be up to the maximum message size defined for the send queue. All messages that contain part of the same LOB value have the same publication name, source table owner, source table name, row number, and column name.

Messages that contain LOB values are sent after the messages that contain the transaction or row operation that the LOB values belong to. The isLast attribute denotes the last message of a divided LOB value, which is also the last message in a transaction or row operation.

Within a LOB message, the large object element (lob) is contained by the message element (msg), and contains a single LOB column value element.

Table 106 describes the lob element.

Table 106. Element description for lob

Name	Properties
lob	Not empty, complex type, complex content

Structure

```
<lob isLast="is_last_indicator" subName="publication_name"
    srcOwner="source_owner" srcName="source_name" rowNum="row_number"
    colName="column_name" totalDataLen="LOB_data_length"
    dataLen="segment_data_length">

        LOB_column_value

</lob>
```

Details

is_last_indicator
A boolean value that indicates whether this is the last message in a transaction or row operation. If this is the last message, the value is 1 (true). If it is not the last message, the value is 0 (false). XML data type: boolean.

publication_name
: The name of the XML publication that includes the LOB value. XML data type: string.

source_owner
: The schema of the source table where the LOB originated. XML data type: string.

source_name
: The name of the source table. XML data type: string.

row_number
: Within the database transaction, the position number of the row operation that contains the LOB value. XML data type: positiveInteger.

column_name
: The name of the column in the source table that contains the LOB value. XML data type: string.

LOB_data_length
: The total length of the LOB value contained in the source table, in bytes. XML data type: nonNegativeInteger.

segment_data_length
: The length of the LOB data contained in a single message segment, in bytes. XML data type: nonNegativeInteger.

LOB_column_value
: One of the three LOB column value elements that describe the data type of the LOB value. The three elements are blob, clob, and dbclob.

Example

The following example shows a LOB message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1">
       <lob isLast="0" subName="S1" srcOwner="USER1" srcName="T1" rowNum="3"
          colName="LOBCOL" totalDataLen="92675" dataLen="100">

            LOB_column_value

       </lob>
</msg>
```

Where *LOB_column_value* describes one of the three elements that are explained in "LOB column value."

LOB column value

The three LOB column value elements each contain actual LOB data from the source table. The elements are named for their data type, either blob, clob, or dbclob. If the value from the source table is NULL, the elements are empty and the xsi:nil attribute is set to 1 (true).

Table 107 on page 549 describes the LOB column value elements.

Table 107. Element description for LOB column values

Name	Properties
blob	Nullable, complex type, simple content
clob	Nullable, complex type, simple content
dbclob	Nullable, complex type, simple content

Structure

```
<data_type xsi:nil="null_indicator">

  LOB_value

</data_type>
```

Details

data_type
: The data type of the column in the source table. This data type is used to name the element.

null_indicator
: Optional: A boolean value that indicates whether the value in the source column is NULL. The default is 0 (false). If the source column contains a NULL value, the value of this attribute is 1 (true). XML data type: boolean.

LOB_value
: Actual data from the large object in the source table.

Example

The following example shows a LOB message that includes 100 bytes of the 92,675 total bytes of data from a CLOB (character large object) value.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1>
       <lob isLast="0" subName="S1" srcOwner="USER1" srcName="T1" rowNum="3"
          colName="LOBCOL" totalDataLen="92675" dataLen="100">

            <clob>LOB data</clob>

       </lob>
</msg>
```

Subscription deactivated message

A subscription deactivated message confirms that the Q Capture program received the deactivate subscription message from the user application.

In a subscription deactivated message, the message element (msg) contains a subscription deactivated element (subDeactivated).

Table 108 describes the subDeactivated element.

Table 108. Element description for subDeactivated

Name	Properties
subDeactivated	Empty, complex type

Structure

```
<subDeactivated subName="publication_name" srcOwner="source_owner"
       srcName="source_name" stateInfo="state_information"/>
```

Details

publication_name
: The name of the publication that was deactivated. XML data type: string.

source_owner
: The schema of the source table for the publication. XML data type: string.

source_name
: The name of the source table. XML data type: string.

state_information
: Additional information regarding the state of the publication. This attribute contains an ASN message number. XML data type: string.

Example

The following example shows a `subscription deactivated` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">

       <subDeactivated subName="S1" srcOwner="USER1" srcName="T1"
              stateInfo="ASN7019I"/>

</msg>
```

Load done received message

The `load done received` message acknowledges that the Q Capture program received the `load done` message from the user application. The `load done` message signifies that a target table is loaded.

In a `load done received` message, the message element (msg) contains a load done received element (loadDoneRcvd).

Table 109 describes the loadDoneRcvd element.

Table 109. Element description for loadDoneRcvd

Name	Properties
loadDoneRcvd	Empty, complex type

Structure

```
<loadDoneRcvd subName="publication_name" srcOwner="source_owner"
       srcName="source_name" stateInfo="state_information"/>
```

Details

publication_name
: The name of the publication for which the target table was loaded. XML data type: string.

source_owner
: The schema of the source table for the publication. XML data type: string.

source_name
The name of the source table. XML data type: string.

state_information
Additional information regarding the state of the publication. This attribute contains an ASN message number. XML data type: string.

Example

The following example shows a `load done received` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">

       <loadDoneRcvd subName="S1" srcOwner="USER1" srcName="T1"
             stateInfo="ASN7019I"/>

</msg>
```

Error report message

The Q Capture program sends an `error report` message when it cannot perform the request of a user application that was made through a control message.

For example, the Q Capture program sends an error report message if it cannot activate or deactivate a publication or acknowledge a `load done` message. The Q Capture program also writes these errors to its log. If the Q Capture program cannot send an error report message because the send queue is not available, it will still write the error to its log. Error report messages are not generated by errors related to WebSphere MQ.

In an `error report` message, the message element (msg) contains an error report element (errorRpt).

Table 110 describes the errorRpt element.

Table 110. Element description for errorRpt

Name	Properties
errorRpt	Empty, complex type

Structure

```
<errorRpt subName="publication_name" srcOwner="source_owner"
       srcName="source_name" errorMsg="message_text"/>
```

Details

publication_name
The name of the publication that generated an error. XML data type: string.

source_owner
The schema of the source table for the publication. XML data type: string.

source_name
The name of the source table. XML data type: string.

message_text
The text of the error message. XML data type: string.

Example

The following example shows an `error report` message generated after the Q Capture program was unable to activate a publication

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">

        <errorRpt subName="S1" srcOwner="USER1" srcName="T1"
           errorMsg="message_text"/>

</msg>
```

Where *message_text* is the text of the error message.

Heartbeat message

A `heartbeat` message tells the user application that a Q Capture program is still running. The Q Capture program puts these messages on active send queues each time the heartbeat interval for the publishing queue map that contains the send queue is reached if there are no messages to put on the queue. If the Q Capture program reaches the end of the log before this interval occurs, it sends a `heartbeat` message with no information about the last commit time.

In a `heartbeat` message, the message element (msg) contains a heartbeat element (heartbeat).

Table 111 describes the heartbeat element.

Table 111. Element description for heartbeat

Name	Properties
heartbeat	Empty, complex type

Structure

```
<heartbeat sendQName="send_queue_name" lastCmitTime="last_commit_time"/>
```

Details

`send_queue_name`
: The name of the send queue where the Q Capture program put the `heartbeat` message. XML data type: string.

`last_commit_time`
: Optional: The timestamp of the last committed transaction in Greenwich mean time (GMT). This attribute is optional and has no default value. XML data type: dateTime.

Example

The following example shows a `heartbeat` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
```

```
    <heartbeat sendQName="Q1" lastcmitTime="2003-10-31T12:12:12.000122"/>

</msg>
```

Subscription schema message (subSchema)

The Q Capture program sends a `subscription schema` message to acknowledge that it activated or reinitialized a publication. The message conveys details about the publication, including the names of the source table and send queue, data-sending options, and information about the load phase. The `subscription schema` message is sent in response to an `activate subscription` message, a **`reinit`** command, or a REINIT_SUB signal.

Within a `subscription schema` message, the message element (msg) contains a subscription schema element (subSchema), which contains one or more column elements (col). The following sections describe the two elements.

- “Subscription schema element (subSchema)”
- “Column element (col) in a subscription schema message” on page 555

Subscription schema element (subSchema)

Through its attributes, the subSchema element provides details about a publication. The subSchema element contains one or more column elements (col).

Table 112 describes the subSchema element.

Table 112. Element description for subSchema

Name	Properties
subSchema	Not empty, complex type, simple content

Structure

```
<subSchema subname="publication_name"
           srcOwner="source_owner"
           srcName="source_name"
           sendQName="send_queue_name"
           allChangedRows="ALL_CHANGED_ROWS_option"
           beforeValues="BEFORE_VALUES_option"
           changedColsOnly="CHANGED_COLS_ONLY_option"
           hasLoadPhase="load_phase_option"
           dbServerType="operating_system"
           dbRelease="DB2_release_level"
           dbInstance="DB2_instance_name"
           capRelease="Q_capture_release_level">

    column_elements

</subSchema>
```

Details

`publication_name`
: The name of the publication that was activated or reinitialized. XML data type: string.

`source_owner`
: The schema of the source table for the publication. XML data type: string.

source_name
The name of the source table. XML data type: string.

send_queue_name
The name of the send queue that is specified for the publication. XML data type: string.

ALL_CHANGED_ROWS_option
Optional: A boolean value that indicates whether the ALL_CHANGED_ROWS data-sending option is specified for the publication. The default is 0 (false). If the option is specified, the value is 1 (true). XML data type: boolean.

BEFORE_VALUES_option
Optional: A boolean value that indicates whether the BEFORE_VALUES data-sending option is specified for the publication. The default is 0 (false). If the option is specified, the value is 1 (true). XML data type: boolean.

CHANGED_COLS_ONLY_option
Optional: A boolean value that indicates whether the CHANGED_COLS_ONLY data-sending option is specified for the publication. The default is 0 (false). If the option is specified, the value is 1 (true). XML data type: boolean.

load_phase_option
Optional: An indicator of whether the publication has a load phase. The default is "none" for no load phase. If a load phase is specified, the value is "external." XML data type: loadPhaseEnumType.

operating_system
Optional: The operating system of the source database or subsystem. The default is QDB2/6000 (DB2 for AIX). XML data type: dbServerTypeEnumType.

DB2_release_level
The DB2 release level of the source database or subsystem. XML data type: string.

DB2_instance_name
The name of the DB2 instance that contains the source database. XML data type: string.

Q_capture_release_level
The release level of the Q Capture program. XML data type: string.

column_elements
One or more column elements (col) that convey information about each column in the source table.

Table 113 provides additional details about two XML data types used in attributes for the subSchema element.

Table 113. Additional data type descriptions for subSchema attributes

Type name	Base Type	Values
loadPhaseEnumType	string	none, external
dbServerTypeEnumType	string	QDB2, QDB2/6000, QDB2/HPUX, QDB2/NT, QDB2/SUN, QDB2/LINUX, QDB2/Windows **Note:** QDB2 by itself implies DB2 for z/OS.

Example

The following `subscription schema` message would be sent for a publication that specifies the BEFORE_VALUES data-sending option and a load phase.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
       <subSchema subname="S1"
                  srcOwner="USER1"
                  srcName="T1"
                  sendQName="Q1"
                  beforeValues="yes"
                  hasLoadPhase="external"
                  dbServerType="QDB2/6000"
                  dbRelease="8.2.0"
                  dbInstance="DB2INST"
                  capRelease="8.2.0">

              column_element

       </subSchema>
</msg>
```

Where *column_element* represents one or more column elements, which are explained in "Column element (col) in a subscription schema message."

Column element (col) in a subscription schema message

Within the `subscription schema` message, the column element (col) conveys information about each column in the source table.

Table 114 describes the col element within a schema message.

Table 114. Element description for col

Name	Properties
col	Empty, complex type

Structure

```
<col name="column_name"
     type="data_type"
     len="data_length"
     precision="data_precision"
     scale="decimal_scale"
     codepage="codepage_number"
     isKey="key_indicator"/>
```

Details

`column_name`
: The name of the column in the source table. XML data type: string.

`data_type`
: The data type of the source column. This must be one of the data types defined for the dataTypeEnumType XML data type. For a list, see Table 115 on page 556. XML data type: dataTypeEnumType.

Table 115. Additional data type description for dataTypeEnumType

Type name	Base type	Values
dataTypeEnumType	string	smallint, integer, bigint, float, real, double, decimal, char, varchar, longvarchar, bitchar, bitvarchar, bitlongvarchar, graphic, vargraphic, longvargraphic, time, timestamp, date, rowid, blob, clob, dbclob

data_length
: Optional: The maximum length of the data in the source column. XML data type: unsignedInt.

data_precision
: Optional: For decimal data types, the precision of the number. XML data type: unsignedShort.

decimal_scale
: Optional: For decimal data types, the scale of the number. XML data type: unsignedShort.

codepage_number
: Optional: The code page for character data types. The default is 0. XML data type: unsignedShort.

key_indicator
: Optional: A boolean value that indicates whether this is a key column. The default is 0 (false). If it is a key column, the value is 1 (true). XML data type: boolean.

Example

The following example shows two column elements within a subscription schema message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
     dbName="DB1">
       <subSchema subname="S1"
                  srcOwner="USER1"
                  srcName="T1"
                  sendQName="Q1"
                  beforeValues="yes"
                  hasLoadPhase="external"
                  dbServerType="QDB2/6000"
                  dbRelease="8.2.0"
                  dbInstance="DB2INST"
                  capRelease="8.2.0">
             <col name="COL1" type="integer" len="4"/>
             <col name="COL2" type="varchar" len="50" codepage="1208"/>
       </subSchema>
</msg>
```

Add column message

An add column message tells the user application that a Q Capture program added a column to an existing publication. This message is sent in response to a user or user application inserting an ADDCOL signal into the IBMQREP_SIGNAL table.

In an add column message, the message element (msg) contains an add column element (addColumn). The add column element contains a column schema (col) element that conveys information about the source table column that was added.

Table 116 describes the addColumn element.

Table 116. Element description for addColumn

Name	Properties
addColumn	Not empty, complex type, simple content

Structure

```
<addColumn subName="publication_name" srcOwner="source_owner"
    srcName="source_name">

      column_element

</addColumn>
```

Details

publication_name
: The name of the publication that the column was added to. XML data type: string.

srcOwner
: The schema of the source table for the publication. XML data type: string

srcName
: The name of the source table. XML data type: string.

column_element
: A column schema (col) element that contains details about the column that was added, such as name, data type, data length, and whether the column is a key column.

Example

The following example shows an add column message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
    <addColumn subName="S1" srcOwner="USER1" srcName="T1">

      column_element

    </addColumn>
</msg>
```

Where *column_element* represents the column element that is explained in "Column element (col) in a subscription schema message" on page 555.

Structure of messages from a user application to Q Capture

A user application communicates with a Q Capture program by sending messages to its administration queue. These messages are known as control messages. The user application uses these messages to report that a target table is loaded, or to request that the Q Capture program activate or deactivate a publication, or invalidate a send queue.

The following topics describe the structure of control messages from a user application to a Q Capture program.

List of messages from a user application to Q Capture

Table 117 describes the four types of control messages from a user application to a Q Capture program.

Table 117. Control messages from a user application to a Q Capture program

Message type	Description
Invalidate send queue	Requests that a Q Capture program invalidate a send queue by performing the queue error action that you specified.
Load done	Tells a Q Capture program that the target table for a publication is loaded.
Activate subscription	Requests that a Q Capture program activate a publication.
Deactivate subscription	Requests that a Q Capture program deactivate a publication.

msg: Root element for XML messages from a user application to Q Capture

The msg element is the root element for all control messages from a user application to a Q Capture program.

Table 118 describes the msg element.

Table 118. Element description for msg (user application to a Q Capture program)

Name	Properties
msg	Not empty, complex type, complex content

Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="XML_schema_instance"
      xsi:noNamespaceSchemaLocation="schema_document"
      version="version">

      elements

<msg>
```

Details

XML_schema_instance
: The URL of the XML schema instance. For event publishing, the URL is www.w3.org/2001/XMLSchema-instance. XML data type: string.

schema_document
: The file name of the XML schema document. XML namespace is not supported in event publishing because messages refer to one XML schema only. Messages from a user application to a Q Capture program refer to the mqsub.xsd schema document. XML data type: string.

version
: The version of the XML message schema. For DB2 UDB Version 8.2, the version is 1.0.0. XML data type: string.

elements
One of the elements that the msg element contains. Only one of these elements appears in each message:
- invalidateSendQ
- loadDone
- activateSub
- deactivateSub

Example

The following example shows a message from a user application to the Q Capture program.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       elements

</msg>
```

Where *elements* represents one of the following elements: invalidateSendQ, loadDone, activateSub, deactivateSub.

Invalidate send queue message

A subscribing application sends the Q Capture program an `invalidate send queue` message when it detects an error on a send queue and wants the Q Capture program to perform the error action specified for the XML publication.

Table 119 describes the invalidateSendQ element.

Table 119. Element description for invalidateSendQ

Name	Properties
invalidateSendQ	Empty, complex type

Structure

```
<invalidateSendQ sendQName="send_queue_name"/>
```

Details

send_queue_name
The name of the send queue that the Q Capture program is being asked to invalidate. XML data type: string.

Example

The following example shows an `invalidate send queue` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <invalidateSendQ sendQName="S1"/>

</msg>
```

Load done message

A `load done` message notifies the Q Capture program that a target table is loaded. The Q Capture program responds to a `load done` message by sending a `load done received` message.

Table 120 describes the loadDone element.

Table 120. Element description for loadDone

Name	Properties
loadDone	Empty, complex type

Structure

```
<loadDone subName="publication_name"/>
```

Details

`publication_name`
The name of the publication that completed its load phase. XML data type: string.

Example

The following example shows a `load done` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <loadDone subName="S1"/>

</msg>
```

Activate subscription message

An `activate subscription` message tells a Q Capture program to begin capturing changes for a publication.

Table 121 describes the activateSub element.

Table 121. Element description for activateSub

Name	Properties
activateSub	Empty, complex type

Structure

```
<activateSub subName="publication_name"/>
```

Details

`publication_name`
The name of the publication that the Q Capture program is being asked to activate. XML data type: string.

Example

The following example shows an `activate subscription` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <activateSub subName="S1"/>

</msg>
```

Deactivate subscription message

A `deactivate subscription` message tells the Q Capture program to stop capturing changes for a publication.

Table 122 describes the deactivateSub element.

Table 122. Element description for deactivateSub

Name	Properties
deactivateSub	Empty, complex type

Structure

```
<deactivateSub subName="publication_name"/>
```

Details

publication_name
: The name of the publication that the Q Capture program is being asked to deactivate. XML data type: string.

Example

The following example shows a `deactivate subscription` message.

```
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <deactivateSub subName="S1"/>

</msg>
```

Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You also can provide feedback to IBM about products and documentation.

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 123. IBM resources

Resource	Description and location
IBM Support Portal	You can customize support information by choosing the products and the topics that interest you at www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server
Software services	You can find information about software, IT, and business consulting services, on the solutions site at www.ibm.com/businesssolutions/
My IBM	You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at www.ibm.com/account/
Training and certification	You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at http://www.ibm.com/software/sw-training/
IBM representatives	You can contact an IBM representative to learn about solutions at www.ibm.com/connect/ibm/us/en/

Federation, replication, and event publishing products support

For support, go to:

- IBM InfoSphere Federation Server
  www.ibm.com/software/data/integration/support/federation_server/
- IBM InfoSphere Replication Server
  www.ibm.com/software/data/integration/support/replication_server/
- IBM InfoSphere Data Event Publisher
  www.ibm.com/software/data/integration/support/data_event_publisher/

Classic products support

For support, go to:

- IBM InfoSphere Classic Federation Server for z/OS
  www.ibm.com/software/data/integration/support/classic_federation_server_z/

- IBM InfoSphere Classic Replication Server for z/OS www.ibm.com/software/data/infosphere/support/replication-server-z/
- IBM InfoSphere Classic Data Event Publisher for z/OS www.ibm.com/software/data/integration/support/data_event_publisher_z/
- IBM InfoSphere Data Integration Classic Connector for z/OS www.ibm.com/software/data/integration/support/data_integration_classic_connector_z/

Providing feedback

The following table describes how to provide feedback to IBM about products and product documentation.

Table 124. Providing feedback to IBM

Type of feedback	**Action**
Product feedback	You can provide general product feedback through the Consumability Survey at www.ibm.com/software/data/info/consumability-survey
Documentation feedback	To comment on the information center, click the Feedback link on the top right side of any topic in the information center. You can also send comments about PDF file books, the information center, or any other documentation in the following ways: • Online reader comment form: www.ibm.com/software/data/rcf/ • E-mail: comments@us.ibm.com

How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

```
►►──required_item──────────────────────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►──required_item──┬───────────────┬───────────────────────────────►◄
                   └─optional_item─┘
```

  If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

```
                   ┌─optional_item─┐
►►──required_item──┴───────────────┴───────────────────────────────►◄
```

- If you can choose from two or more items, they appear vertically, in a stack.

  If you must choose one of the items, one item of the stack appears on the main path.

```
►►──required_item──┬─required_choice1─┬────────────────────────────►◄
                   └─required_choice2─┘
```

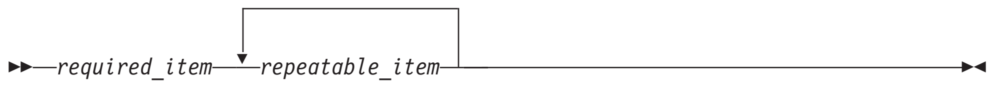
  If choosing one of the items is optional, the entire stack appears below the main path.

```
►►──required_item──┬──────────────────┬────────────────────────────►◄
                   ├─optional_choice1─┤
                   └─optional_choice2─┘
```

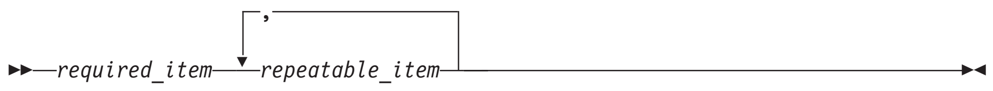
  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

```
                   ┌─default_choice───┐
►►──required_item──┼──────────────────┼────────────────────────────►◄
                   ├─optional_choice1─┤
                   └─optional_choice2─┘
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.

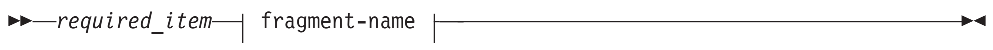


If the repeat arrow contains a comma, you must separate repeated items with a comma.

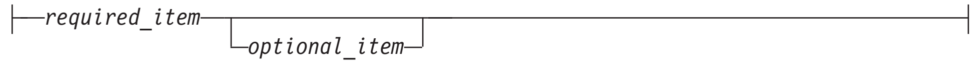


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



Fragment-name:



- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown.
- Variables appear in all lowercase italic letters (for example, **column-name**). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

Notices and trademarks

This information was developed for products and services offered in the U.S.A.

Notices

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 grant 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 character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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

Trademarks

Index

Printed in USA

SC19-3637-00

Spine information:

IBM InfoSphere Data Replication **Version 10.1.3** **Replication and Event Publishing Guide and Reference**