Before using this information and the product that it supports, read the information in "Notices" on page 515.
Contents

Chapter 1. Planning memory and storage requirements for Q replication and event publishing ................ 1
Planning memory requirements for Q replication and event publishing ........................................... 1
Memory used by the Q Capture program ........................................ 1
Memory used by the Q Apply program ........................................ 4
Memory for LOB data types for Q replication and event publishing .............................................. 6
Planning storage requirements for Q replication and event publishing ........................................... 6
Storage requirements for database logs for Q replication and event publishing .............................. 6

Chapter 2. Data conversion for Q replication and event publishing ................................................. 9
Code pages for Q replication ........................................ 9
Code pages for event publishing ........................................ 10

Chapter 3. Setting up user IDs and passwords .................................................................................. 11
Authorization requirements for Q replication and event publishing ................................................. 11
Authorization requirements for the Q Capture program ........................................ 11
Authorization requirements for the Q Apply program ........................................ 11
Authorization requirements to administer Q replication and event publishing ................................ 12
Connectivity requirements for Q replication and event publishing ................................................ 13
Managing user IDs and passwords for remote servers (Linux, UNIX, Windows) ............................... 13

Chapter 4. Setting up WebSphere MQ for Q replication and event publishing—Overview ............. 17
WebSphere MQ objects required for Q replication and event publishing—Overview ......................... 17
WebSphere MQ objects required for unidirectional replication (remote) ........................................ 17
WebSphere MQ objects required for unidirectional replication (same system) ................................ 18
WebSphere MQ objects required for event publishing ................................................................. 19
WebSphere MQ objects required for bidirectional or peer-to-peer replication (two remote servers) .... 21
WebSphere MQ objects required for peer-to-peer replication (three or more remote servers) .......... 23
Required settings for WebSphere MQ objects ................................................................................. 23
Sample commands for creating WebSphere MQ objects for Q replication and event publishing .... 25
Running the replication programs on a WebSphere MQ client ...................................................... 30
Validating WebSphere MQ objects for Q replication and event publishing .................................... 34
Sending test messages between queues in a replication queue map ................................................. 35
Connectivity and authorization requirements for WebSphere MQ objects .................................... 37
Storage requirements for WebSphere MQ for Q replication and event publishing ............................. 38
WebSphere MQ message size ........................................................................................................... 39
Recovering when WebSphere MQ messages are lost ....................................................................... 40
Recovering after messages are sent to dead-letter queues .............................................................. 42
Queue depth considerations for large object (LOB) values ............................................................. 43
Queue manager clustering in Q replication and event publishing .................................................... 44

Chapter 5. Configuring databases for Q replication and event publishing (Linux, UNIX, Windows) .... 45
Setting environment variables (Linux, UNIX, Windows) ................................................................. 45
Setting the TMPDIR environment variable (Linux, UNIX) ............................................................. 46
Addressing memory issues for Q Capture and Q Apply (AIX) ........................................................ 46
Configuring the source database to work with the Q Capture program (Linux, UNIX, Windows) .... 48
Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows) .......... 49
Optional: Binding the program packages (Linux, UNIX, Windows) ............................................ 50
Optional: Binding the Q Capture program packages (Linux, UNIX, Windows) ............................... 50
Optional: Binding the Q Apply program packages (Linux, UNIX, Windows) .................................. 51
Optional: Binding the Replication Alert Monitor packages (Linux, UNIX, Windows) .................... 52
Configuring databases for Q replication and event publishing (z/OS) ............................................. 52
Software prerequisites for the Replication Center ............................................................................ 53
Q replication and event publishing for multiple database partitions .............................................. 53
Replication of partitioned tables ....................................................................................................... 54
Creating control tables for the Q Capture and Q Apply programs .................................................. 55
Creating control tables at a different version (Linux, UNIX, Windows) .......................................... 56
Configuring for older or remote DB2 databases .............................................................................. 58

Chapter 6. Setting up unidirectional Q replication ............................................................................ 63
Unidirectional replication ................................................................................................................... 63
Grouping replication queue maps and Q subscriptions ...................................................................... 64
Changing the Q Capture parameters .................................. 191
Methods of changing the Q Capture operating parameters .............. 191
Changing parameters while a Q Capture program is running ............. 192
Changing saved Q Capture parameters in the IBMQREP_CAPPARMS table ....................... 194
Stopping a Q Capture program ........................................ 195
Starting Q subscriptions ............................................. 196
Stopping Q subscriptions ............................................. 197
Starting publications .................................................. 198
Stopping publications ............................................... 199
Managing Q Capture message activity at the send queue level .......... 200
Starting message activity on one or more send queues .................. 203
Stopping message activity on one or more send queues .................. 204

Chapter 14. Operating a Q Apply program ................................ 207
Starting a Q Apply program ............................................ 207
Parameters of a Q Apply program ..................................... 209
  Default values for Q Apply operating parameters ....................... 209
  Descriptions of Q Apply parameters ................................ 210
Changing the Q Apply parameters .................................... 223
Changing parameters while a Q Apply program is running ............... 224
Changing saved Q Apply parameters in the IBMQREP_APPLYPARMS table ........... 225
Stopping a Q Apply program ........................................... 226
Stopping message processing on a receive queue ........................ 227
Starting message processing on a receive queue ......................... 228
Prompting a Q Apply program to ignore transactions .................... 229

Chapter 15. Changing a Q replication environment ...................... 233
Changing the properties of unidirectional Q subscriptions ............... 233
Adding columns to replicate (unidirectional replication) ................ 235
Adding columns to replicate (bidirectional or peer-to-peer replication) ........ 236
How Q Capture handles alterations of source tables ..................... 238
Changing properties of replication queue maps ........................ 239
Restarting failed Q subscriptions without dropping and recreating them ...... 241
Deleting Q subscriptions ............................................ 241
Deleting replication queue maps ..................................... 242
Dropping Q Capture or Q Apply control tables ........................ 243

Chapter 16. Changing an event publishing environment ................. 245
Changing properties of publications .................................. 245
Adding columns to existing publications ................................ 246
Deleting publications .................................................. 248
Changing properties of publishing queue maps ........................ 249
Deleting publishing queue maps ....................................... 250
Dropping Q Capture control tables .................................... 251

Chapter 17. Viewing reports about the Q replication and event publishing programs ........................................... 253
Checking the status of the Q replication and event publishing programs ....................... 253
Threads of the Q Capture, Q Apply, and Replication Alert Monitor programs .................... 255
Historical and performance data for Q replication and event publishing programs .......... 257
Latency ............................................................... 258
Exceptions ............................................................ 260

Chapter 18. Replication Alert Monitor .................................. 263
Monitoring replication with the Replication Alert Monitor .................. 263
Alert conditions and notifications for the Replication Alert Monitor .............. 265
E-mail notifications for replication alert conditions ........................ 268
Specifying where to send monitor alerts ................................ 270
The ASNMAIL exit routine for sending alerts in replication (Linux, UNIX, Windows) ........ 270
Setting up the Replication Alert Monitor ................................ 271
Memory used by the Replication Alert Monitor ................................ 271
Authorization requirements for the Replication Alert Monitor ............... 272
Optional: Binding the Replication Alert Monitor program packages (Linux, UNIX, Windows) ...... 272
Creating control tables for the Replication Alert Monitor .................. 273
Defining contact information for the Replication Alert Monitor ............... 278
Creating monitors for replication or publishing ................................ 279
Changing alert conditions for the Replication Alert Monitor ............... 279
Defining suspension periods for the Alert Monitor ........................ 280
Operating the Replication Alert Monitor ................................ 280
Starting monitors ..................................................... 280
Reinitializing monitors ................................................ 280
Suspending and resuming a monitor .................................... 280
Ending a monitor suspension ....................................... 280
Reviewing Monitor program messages .................................. 281
Parameters of the Replication Alert Monitor ................................ 281
Default values of Replication Alert Monitor parameters .................... 281
Descriptions of the Replication Alert Monitor parameters .................. 281
Changing runtime parameters for the Replication Alert Monitor .............. 284
Specifying how often the Replication Alert Monitor runs .................... 285
<table>
<thead>
<tr>
<th>Chapter 19. Maintaining a Q replication and event publishing environment</th>
<th>287</th>
</tr>
</thead>
<tbody>
<tr>
<td>Considerations for maintaining Q replication and event publishing source systems</td>
<td>287</td>
</tr>
<tr>
<td>Maintaining source tables in a Q replication and event publishing environment</td>
<td>287</td>
</tr>
<tr>
<td>Retaining log files for Q replication and event publishing</td>
<td>287</td>
</tr>
<tr>
<td>Considerations for managing compression dictionaries in Q replication and event publishing (z/OS)</td>
<td>290</td>
</tr>
<tr>
<td>Maintaining control tables in Q replication and event publishing</td>
<td>291</td>
</tr>
<tr>
<td>Pruning control tables in Q replication and event publishing</td>
<td>291</td>
</tr>
<tr>
<td>Considerations for using the RUNSTATS utility on control tables for Q replication and event publishing</td>
<td>292</td>
</tr>
<tr>
<td>Reorganizing control tables</td>
<td>292</td>
</tr>
<tr>
<td>When replication programs cannot connect to their DB2 server</td>
<td>293</td>
</tr>
<tr>
<td>Maintaining target tables</td>
<td>294</td>
</tr>
<tr>
<td>Considerations for rebinding packages and plans for Q replication and event publishing</td>
<td>295</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 20. Table differencing and repair</th>
<th>297</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table difference utility (asntdiff)</td>
<td>297</td>
</tr>
<tr>
<td>Table repair utility (asntrep)</td>
<td>302</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 21. Using system services to operate the Q replication and event publishing programs</th>
<th>303</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using z/OS system services to run the Q replication and event publishing programs</td>
<td>303</td>
</tr>
<tr>
<td>Running the Q replication and event publishing programs by using JCL</td>
<td>303</td>
</tr>
<tr>
<td>Running the Q replication and event publishing programs with system-started tasks</td>
<td>309</td>
</tr>
<tr>
<td>The Automatic Restart Manager (ARM) recovery system</td>
<td>310</td>
</tr>
<tr>
<td>Automatically restart Q replication and event publishing programs</td>
<td>310</td>
</tr>
<tr>
<td>Replication services (Windows)</td>
<td>311</td>
</tr>
<tr>
<td>Description of Windows services for replication</td>
<td>311</td>
</tr>
<tr>
<td>Creating a replication service</td>
<td>312</td>
</tr>
<tr>
<td>Starting a replication service</td>
<td>313</td>
</tr>
<tr>
<td>Stopping a replication service</td>
<td>313</td>
</tr>
<tr>
<td>Viewing a list of replication services</td>
<td>313</td>
</tr>
<tr>
<td>Dropping a replication service</td>
<td>314</td>
</tr>
<tr>
<td>Scheduling the replication programs</td>
<td>314</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 22. Naming rules and guidelines for Q replication and event publishing—Overview</th>
<th>317</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naming rules for Q replication and event publishing objects</td>
<td>317</td>
</tr>
<tr>
<td>How lowercase object names are handled for Q replication and event publishing</td>
<td>320</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 23. System commands for Q replication and event publishing</th>
<th>323</th>
</tr>
</thead>
<tbody>
<tr>
<td>asncap: Starting a Q Capture program</td>
<td>324</td>
</tr>
<tr>
<td>asncmd: Working with a running Q Capture program</td>
<td>332</td>
</tr>
<tr>
<td>asnpapp: Starting a Q Apply program</td>
<td>336</td>
</tr>
<tr>
<td>asncacmd: Working with a running Q Apply program</td>
<td>348</td>
</tr>
<tr>
<td>asnmmon: Starting a Replication Alert Monitor</td>
<td>352</td>
</tr>
<tr>
<td>asnnmcmd: Working with a running Replication Alert Monitor</td>
<td>357</td>
</tr>
<tr>
<td>asnpwd: Creating and maintaining password files</td>
<td>360</td>
</tr>
<tr>
<td>asnsrt: Creating a replication service</td>
<td>363</td>
</tr>
<tr>
<td>asnsdrop: Dropping a replication service</td>
<td>366</td>
</tr>
<tr>
<td>asnslist: Listing replication services</td>
<td>367</td>
</tr>
<tr>
<td>asntdiff: Comparing data in source and target tables</td>
<td>368</td>
</tr>
<tr>
<td>asntdiff –f (input file) command option</td>
<td>374</td>
</tr>
<tr>
<td>asntrc: Operating the replication trace facility</td>
<td>377</td>
</tr>
<tr>
<td>asntrep: Repairing differences between source and target tables</td>
<td>384</td>
</tr>
<tr>
<td>asncanalyze: Operating the Q Replication Analyzer</td>
<td>387</td>
</tr>
<tr>
<td>asnxfmt: Formatting and viewing Q replication and event publishing messages</td>
<td>395</td>
</tr>
<tr>
<td>asnxfmt: Formatting and viewing event publishing messages (z/OS)</td>
<td>397</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 24. Control tables for Q replication and event publishing</th>
<th>399</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control tables at the Q Capture server</td>
<td>399</td>
</tr>
<tr>
<td>IBMQREP_ADMINMSG table</td>
<td>400</td>
</tr>
<tr>
<td>IBMQREP_CAPPARM table</td>
<td>400</td>
</tr>
<tr>
<td>IBMQREP_CAPPARMINFO table</td>
<td>401</td>
</tr>
<tr>
<td>IBMQREP_CAPMON table</td>
<td>402</td>
</tr>
<tr>
<td>IBMQREP_CAPQMON table</td>
<td>404</td>
</tr>
<tr>
<td>IBMQREP_CAPQMON table</td>
<td>408</td>
</tr>
<tr>
<td>IBMQREP_CAPTRACE table</td>
<td>410</td>
</tr>
<tr>
<td>IBMQREP_COLVERSION table (z/OS)</td>
<td>410</td>
</tr>
<tr>
<td>IBMQREP_EOLFLUSH table</td>
<td>412</td>
</tr>
<tr>
<td>IBMQREP_IGNTRAN table</td>
<td>412</td>
</tr>
<tr>
<td>IBMQREP_IGNTRANTRC table</td>
<td>413</td>
</tr>
<tr>
<td>IBMQREP_SENDQUEUES table</td>
<td>414</td>
</tr>
<tr>
<td>IBMQREP_SIGNAL table</td>
<td>418</td>
</tr>
<tr>
<td>IBMQREP_SRC_COLS table</td>
<td>421</td>
</tr>
<tr>
<td>IBMQREP_SRCH_COND table</td>
<td>422</td>
</tr>
</tbody>
</table>
Chapter 1. Planning memory and storage requirements for Q replication and event publishing

Before you begin replicating or publishing source changes, ensure that you have sufficient memory so that the programs avoid writing transactions to spill files. Also, in the event that the programs do need to spill to files, ensure that you have adequate storage to accommodate the spilled data.

Q replication and publishing environments require a certain amount of memory to capture, apply, and monitor transactions. The amount of memory that is required depends on a number of factors, some of which you can tune.

Planning memory requirements for Q replication and event publishing

You should plan the memory requirements for Q replication and event publishing to ensure that there is enough memory for the replication and event publishing programs to run on your system.

Memory is a system-managed resource. Processes (jobs in z/OS®) generally have a memory quota that determines the maximum amount of memory that the process can use. It is recommended that you do not set such a quota for Q replication or event publishing, or that if you set a quota, you set a high quota. The programs use only as much memory as they need. The programs have a memory limit parameter that you can use to control how much memory the programs consume.

**Recommendation:** Plan for the maximum memory that could be required for your anticipated workload even though the programs will probably not use that amount of memory most of the time.

By understanding how the Q Capture program and the Q Apply program use memory, you can determine how much memory they will use in your environment to replicate or publish data. Basically they use more memory if they are processing larger transactions, if there are many concurrent transactions, and if you configure the Q Capture program to commit transactions less frequently. The programs also manage the amount of memory that is required to process LOB and BLOB values. After you start replicating or publishing data, you should tune the amount of memory that the programs use to avoid spilling transactions.

A small amount of memory is also required by the Replication Alert Monitor.

**Memory used by the Q Capture program**

The Q Capture program uses memory to store information about sources and to reconstruct transactions from the DB2® log.

**Memory for storing information about active sources**

The Q Capture program uses memory to store information about sources that are active, such as what source data you want to replicate or publish. The Q Capture program retrieves data from the control tables and stores this data in memory when you start the program or when you activate Q subscriptions or publications while the Q Capture program is running. Each Q subscription or publication consumes a maximum of 1000 bytes of memory.
Memory for reconstructing transactions

When the Q Capture program reads the DB2 log, it stores individual transaction records in memory until it reads the associated commit or abort record. Data that is associated with an aborted transaction is cleared from memory, and data that is associated with a commit record is sent to the WebSphere MQ queue that you specified in the replication or publishing queue map. The committed transactions stay in memory until the Q Capture program commits its work; the Q Capture program commits its work when it reaches its commit interval.

Memory for other uses

When the Q Capture program reads log records, the Q Capture program uses a memory buffer. The default size on z/OS operating systems is 66 1-KB pages, and the memory is ECSA (extended common service area) storage. The default size of the buffer on Linux®, UNIX®, and Windows® operating systems is 200 1-KB pages.

Most of the memory that is used by the Q Capture program is used to reconstruct transactions, and the amount of memory that is needed for that activity depends on the following factors:

Commit interval

The Q Capture program stores transactions that were committed by DB2 in memory until the Q Capture program can commit them to the send queue. You use the `commit_interval` parameter to specify how frequently the Q Capture program commits the transactions to the send queues. Typically, a smaller value for the `commit_interval` means that the Q Capture program uses less memory and is less likely to write transactions to spill files because the Q Capture program commits the transactions frequently. A larger value for the `commit_interval` means that the Q Capture program commits transactions less frequently.

Size of transactions that are replicated or published

The amount of memory that is required by the Q Capture program is proportional to the scope of the transactions and the size of the records that are updated. If transactions are very large and the Q Capture program runs out of memory, the Q Capture program will write the transactions to spill files. Long running transactions with infrequent commits can impact the memory and storage requirements of Q replication. Consider adding more frequent commits to such transactions.

Number of concurrent transactions

If the Q Capture program processes many transactions at the same time or processes interleaved transactions, then the Q Capture program needs to store more information in memory or on disk.

You can control how much memory the Q Capture program uses to buffer transactions by setting the `memory_limit` parameter when you start the Q Capture program or while it is running. By default, this parameter is set to 500 MB on Linux, UNIX, and Windows, and to 0 on z/OS. The value of 0 means that Q Capture calculates a memory allocation that is based on the region size of the Q Capture job or started task. The larger the `memory_limit`, the less likely the Q Capture program writes transactions to spill files.

Use Table 1 on page 3 as a guide to help you determine the memory limit for each Q Capture program. By using these estimates as a guide, you can ensure that the Q Capture program has enough memory for your workload and you can prevent...
the program from writing transactions to spill files. The table uses this formula to calculate the number of transactions per `commit_interval`:

\[ \text{txr}/1000 \times \text{ci} = \text{txci} \]

Where:
- **txr**: Transaction rate (transactions per second)
- **ci**: Commit_interval
- **txci**: Transactions per `commit_interval`

The transactions per `commit_interval` are used in the following formula to come up with the memory limit:

\[ \text{txci} \times \text{mtxs} = \text{ml} \]

Where:
- **txci**: Transactions per `commit_interval`
- **mtxs**: Maximum transaction size
- **ml**: Memory limit in bytes

### Table 1. Approximate memory requirements of a Q Capture program

<table>
<thead>
<tr>
<th>Transaction rate (txr)</th>
<th>Maximum transaction size (mtxs)</th>
<th>Commit_interval in milliseconds (ci)</th>
<th>Transactions per <code>commit_interval</code> (txci)</th>
<th>Memory limit in bytes (ml)</th>
</tr>
</thead>
<tbody>
<tr>
<td>730</td>
<td>1 340</td>
<td>1 000</td>
<td>730</td>
<td>978 200</td>
</tr>
<tr>
<td>1 033</td>
<td>12 000</td>
<td>3 000</td>
<td>3 099</td>
<td>37 188 000</td>
</tr>
<tr>
<td>5 000</td>
<td>2 340</td>
<td>500</td>
<td>2 500</td>
<td>5 850 000</td>
</tr>
<tr>
<td>12 000</td>
<td>7 000</td>
<td>5 000</td>
<td>60 000</td>
<td>420 000 000</td>
</tr>
</tbody>
</table>

**Recommendation:** Do not set a memory quota (region size for z/OS) for the Q Capture program; however, if you need to set one, use Table 2 as a guide. The table shows an example of how to derive a memory quota using this general formula:

\[ \text{ml} + (\text{mas} \times 1000) + 2000000 = \text{q} \]

Where:
- **ml**: Memory limit in bytes
- **mas**: Maximum active subscriptions
- **q**: Memory quota in bytes

### Table 2. Approximate memory quota for the Q Capture program

<table>
<thead>
<tr>
<th>Memory_limit in bytes (ml)</th>
<th>Maximum active subscriptions (mas)</th>
<th>Memory quota in bytes (q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>37 188 000</td>
<td>500</td>
<td>39 688 000</td>
</tr>
<tr>
<td>420 000 000</td>
<td>300</td>
<td>422 300 000</td>
</tr>
</tbody>
</table>

After you start replicating or publishing data, use one of the following methods to check how much memory the Q Capture program is using for transactions, and use this information to tune the `memory_limit` value.
In the Replication Center, select the **Memory usage** option in the Q Capture Throughput window. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to check and select **Reports → Q Capture Throughput**.

Check the value in the CURRENT_MEMORY column of the IBMQREP_CAPMON table to see how much memory a Q Capture program is using to reconstruct transactions from the log.

Also, you can check the value in the TRANS_SPILLED column of the IBMQREP_CAPMON table to see if the Q Capture program is spilling transactions to files.

If the Q Capture program does not have enough memory, it tries to spill to files. The maximum spill file size on z/OS is 2 GB. If there is not enough storage, the Q Capture program terminates.

### Memory used by the Q Apply program

The Q Apply program uses memory to store information about targets and to store committed transactions from the source that it retrieved from the receive queues.

**Memory for storing information about active targets**

The Q Apply program uses memory to store information about active targets, such as the targets to which you want to replicate source data. The Q Apply program retrieves and stores this data when you start the program or when you activate Q subscriptions while the Q Apply program is running. Each Q subscription consumes a maximum of 1000 bytes worth of memory.

**Memory for applying source transactions to targets**

The Q Apply program uses memory to store source transactions that it retrieved from the receive queue, and the Q Apply program keeps the source data in memory while it applies the transactions to the targets. The amount of memory that the Q Apply program uses is proportional to the size of the transactions that it receives, the number of transactions that it receives, and the transaction dependency among the transactions. The amount of memory that the Q Apply program uses is inversely proportional to the rate at which the Q Apply program applies those transactions to the target.

Most of the memory that is used by the Q Apply program is used to apply transactions to targets, and the amount of memory that is needed for that activity depends on the size of transaction. The amount of memory that is required by the Q Apply program is proportional to the scope of the transactions and the size of the records that are updated. If transactions are very large and the Q Apply program runs out of memory, the Q Apply program will stop caching the transactions in memory. Parallel processing will be limited and, depending on the size of the transaction that is being processed, the Q Apply program might need to process transactions serially, one after the other. Long-running transactions with infrequent commits can impact the memory and storage requirements of the Q Apply program. Consider adding more frequent commits to such transactions.

You set the **memory_limit** parameter when you create or modify a replication queue map, which specifies a receive queue. This value controls how much memory the Q Apply program can use to buffer transactions from the receive queue. While the Q Apply program is buffering transactions it is also applying transactions to the target.
As a general requirement, the memory limit for a receive queue must be at least three times the size of the `max_message_size` parameter for the corresponding send queue, which is also set in the replication queue map.

Use Table 3 as a guide to help you determine the `memory_limit` value that is needed for each receive queue for the Q Apply program. Optimally, set the memory limit so that all agents are working on transactions. The table uses this general formula:

\[(\text{mtxs} + 44000) \times (a + 1) = \text{ml}\]

Where:
- \(\text{mtxs}\) Maximum transaction size, in bytes
- \(a\) Number of agents
- \(\text{ml}\) Memory limit for a receive queue, in bytes

Table 3. Approximate memory requirements of a Q Apply program per receive queue

<table>
<thead>
<tr>
<th>Maximum transaction size (mtxs)</th>
<th>Number of agents (a)</th>
<th>memory_limit (ml)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 340</td>
<td>128</td>
<td>5 848 860</td>
</tr>
<tr>
<td>12 000</td>
<td>16</td>
<td>952 000</td>
</tr>
</tbody>
</table>

Set the memory limit so that all agents are kept busy. If you increase the number of agents, check that the memory limit can accommodate the additional transactions to keep them all busy.

**Recommendation:** Do not set a memory quota for the Q Apply program; however, if you need to set one, use Table 4 as a guide after you determine the appropriate memory limit for each receive queue. The table uses this general formula:

\[\text{sml} + (\text{mas} \times 1000) + 500000 = q\]

Where:
- \(\text{sml}\) Sum of memory limit values
- \(\text{mas}\) Maximum active subscriptions
- \(q\) Memory quota in bytes

Table 4. Approximate memory quota for a Q Apply program

<table>
<thead>
<tr>
<th>Sum of memory_limit values for all receive queues (sml)</th>
<th>Maximum active subscriptions (mas)</th>
<th>Memory quota (q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 848 860 + 952 000 = 6 800 860</td>
<td>100</td>
<td>7 400(^{k}) 860</td>
</tr>
</tbody>
</table>

After you start replicating or publishing data, check the value in the CURRENT_MEMORY column of the IBMQREP_APPLYMON table to see how much memory each Q Apply browser is using. Use this information to tune the `memory_limit` value for each Q Apply program queue.

If the Q Apply program runs out of memory and there is not enough disk storage, the diagnostic log files cannot be saved. The Q Apply program keeps running.
Memory for LOB data types for Q replication and event publishing

The Q Capture program manages the amount of memory that it consumes when it replicates or publishes large object (LOB) data types.

If you set your LOB_SEND_OPTION to S to have LOB values sent in a separate LOB message, your 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. The value of the MAX_MESSAGE_SIZE attribute must not be larger than the WebSphere MQ maximum message length (MAXMSGL) attribute that is defined for the queue manager.

For example, assume that the WebSphere MQ maximum message length (MAXMSGL) is 100 MB or higher. If you set the MAX_MESSAGE_SIZE to 20 MB, each 100 MB message will be divided into 5 messages, each of which is 20 MB long.

If you expect to replicate or publish many LOB values or BLOB values, allocate sufficient memory and storage, and set the queue depth accordingly.

Planning storage requirements for Q replication and event publishing

Before you begin replicating or publishing source changes, you first must plan how much storage is required for Q replication and event publishing.

Storage requirements for database logs for Q replication and event publishing

Q replication requires storage in the database logs for the changes that occur in the source servers and the target servers.

Database log requirements for a DB2 source

At the source server, you need to allocate enough log space for the changes that occur in the source tables. DB2 keeps a record of full-row images for each UPDATE statement because each source table has the DATA CAPTURE CHANGES keyword turned on.

You also need a small amount of additional space for the control tables. Q replication log requirements are limited to the space that is required for the changes that occur in the control tables that are needed to operate the Q Capture program. The amount of storage that is needed to operate the Q Capture program is very small.

Database log requirements for a DB2 target

At a DB2 target server, you need to allocate enough log space for the changes that occur in the target tables. You also need additional space for the changes that occur in the control tables for the Q Apply program for tracking transactions and statistics. The amount of storage needed for this information is very small.
**Database log requirements for a non-DB2 target**

At a non-DB2 target server, you need to allocate enough log space for the changes that occur in the target tables. At both the DB2 federated server and the non-DB2 target server, you also need additional space for the changes that occur in the control tables for the Q Apply program for tracking transactions and statistics. The amount of storage needed for this information is very small.

**Storage for diagnostic files for Q replication and event publishing**

The Q Capture program, Q Apply program, and the Replication Alert Monitor use diagnostic log files to store information about the program’s activities, such as when the program started and stopped, and other informational or error messages. By default, each program appends messages to its log file, even after the program is restarted.

Ensure that the directories that contain these log files have enough space to store the files. The location of these files depends on the value that you set for the `capture_path`, `apply_path`, and `monitor_path` parameters when you started the Q Capture program, Q Apply program, and Replication Alert Monitor, respectively.

If you are concerned about storage, you have the option of reusing the program log instead of appending to it. You can use the `logreuse` parameter to specify that you want a program to reuse its log, so that each time that the program starts, it deletes its log and recreates it.

**Storage requirements for when the Q Capture program exceeds its memory limit**

The Q Capture program uses spill files to temporarily store data when the Q Capture program exceeds its memory limit. You must plan for storage requirements of these spill files.

Each transaction spills to its own file. The Q Capture program writes the largest transaction to the file; however, the largest transaction is not necessarily the one that exceeded the memory limit.

The maximum supported size for a spill file is 2 GB.

The size of the spill files for the Q Capture program depends on the following factors:

- **Memory limit**
  The more memory that the Q Capture program is allowed to use, the less likely that it will write to spill files. Use the `memory_limit` parameter to specify how much memory that the Q Capture program can use to buffer transactions.

- **Size of transactions that are replicated or published**
  The amount of memory that is required by the Q Capture program is proportional to the scope of the transactions and the size of the records that are updated. If transactions are very large and the Q Capture program runs out of memory, it writes the transactions to spill files. Long running transactions with infrequent commits can increase the memory and storage requirements of Q replication. Consider adding more frequent commits to such transactions.
Number of concurrent transactions
If the Q Capture program processes many transactions at the same time or processes interleaved transactions, then the Q Capture program might need to spill multiple transactions to files.

Commit interval
You use the commit_interval parameter to specify how frequently the Q Capture program commits the transactions to the send queues. The Q Capture program stores transactions that were committed by DB2 in memory until the Q Capture program can commit the transactions to the send queue. You can use the commit_interval parameter to control how frequently the Q Capture program commits those transactions. Typically, a smaller value for the commit_interval means that the Q Capture program uses less memory and that the Q Capture program is less likely to write transactions to spill files because it commits them frequently. A larger value for the commit_interval means that the Q Capture program commits transactions less frequently.

z/OS
Spill files go to VIO by default, if no CAPSPILL DD card is specified. You can direct the spill file to go to DASD or VIO by specifying UNIT=SYSDA or UNIT=VIO, respectively.

Linux UNIX Windows
Spill files are always on disk. One file per transaction is created in the capture_path directory.

Storage requirements for traces for Q replication and event publishing
You must plan for the storage requirements of the log files that the trace facility uses. The trace facility logs the flow of the Q Capture program, Q Apply program, and Replication Alert Monitor.

You can provide this logged information to IBM® Software Support for troubleshooting assistance.

There are several factors to consider when you estimate the size of a trace file:
• How long the trace will run
• The level of detail that you specified for the trace
• The volume of transactions that will occur during the trace
Chapter 2. 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.

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.

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(3x).
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 3. 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.

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.

For a list of authorization requirements on z/OS, see Replication installation and customization for z/OS.

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.
- 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.

For a list of authorization requirements on z/OS, see Replication installation and customization for z/OS.

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.
• 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 to administer Q replication and event publishing
To administer Q replication and event publishing, you can choose from several administrative interfaces including the Replication Center, system commands, or SQL, depending upon the administrative task.

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. These tasks include:
• Creating and accessing control tables
• Accessing the DB2 system catalog
• Binding plans and packages
• Running generated SQL to create target tables and table spaces, Q subscriptions and publications, and other objects

To simplify administration using 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.

User IDs that administer Q replication and event publishing must have the following authorities and privileges:
• 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
• SELECT, UPDATE, INSERT, and DELETE privileges for all control tables on the Q Capture server, Q Apply server, and Monitor control server
• SELECT privilege for the DB2 system catalog
• All required table, table space, and index privileges to create targets at the Q Apply server
• 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
• Privileges to create stored procedures using a shared library, and to call stored procedures (Linux, UNIX, Windows only).

**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 5. Password file requirements**

<table>
<thead>
<tr>
<th>Program</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>apply_path</td>
</tr>
<tr>
<td>Q Apply</td>
<td>apply_path</td>
</tr>
<tr>
<td>Q Capture</td>
<td>capture_path</td>
</tr>
<tr>
<td>Replication Alert Monitor</td>
<td>monitor_path</td>
</tr>
<tr>
<td>asntdiff or asntrep command</td>
<td>DIFF_PATH</td>
</tr>
</tbody>
</table>

- 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 4. Setting up WebSphere MQ for Q replication and event publishing—Overview

Q replication and event publishing use WebSphere MQ, formerly known as MQ Series, to transmit transactional data and exchange other messages.

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—Overview

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

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 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

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 21 shows the WebSphere MQ objects that are required for event publishing between remote servers.
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:

---

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.
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 23 shows the WebSphere MQ objects that are required for bidirectional or peer-to-peer Q replication between two remote servers.
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.

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.
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 25 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.
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 26
- “WebSphere MQ objects at the target” on page 27
- “WebSphere MQ channel objects” on page 28

Note: Starting with Version 9.7 or the equivalent PTF on DB2 for z/OS Version 9.1, Q replication and event publishing no longer require persistent WebSphere MQ transmission queue.
messages, although persistent messages are still the default. 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), theses 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](http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp).

### WebSphere MQ objects at the source

Table 6 provides required values for selected parameters for WebSphere MQ objects at the source.

<table>
<thead>
<tr>
<th>Object name</th>
<th>Required settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue manager</strong></td>
<td><strong>MAXMSGL</strong></td>
</tr>
<tr>
<td></td>
<td>(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 MAXMSGGL that is defined for each send queue, transmission queue, and the administration queue.</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong> This parameter is not valid on z/OS.</td>
</tr>
<tr>
<td><strong>Send queue</strong></td>
<td><strong>PUT (ENABLED)</strong></td>
</tr>
<tr>
<td></td>
<td>Allows the Q Capture program to put data messages and informational messages on the queue.</td>
</tr>
<tr>
<td></td>
<td><strong>INDXTYPE(MSGID)</strong></td>
</tr>
<tr>
<td></td>
<td>Allows the queue manager to maintain an index of message identifiers.</td>
</tr>
<tr>
<td></td>
<td><strong>SHARE</strong></td>
</tr>
<tr>
<td></td>
<td>More than one application instance can get messages from the queue.</td>
</tr>
<tr>
<td><strong>Administration queue</strong></td>
<td><strong>GET (ENABLED)</strong></td>
</tr>
<tr>
<td></td>
<td>Allows the Q Capture program to get messages from the queue.</td>
</tr>
<tr>
<td></td>
<td><strong>INDXTYPE(MSGID)</strong></td>
</tr>
<tr>
<td></td>
<td>Allows the queue manager to maintain an index of message identifiers.</td>
</tr>
<tr>
<td></td>
<td><strong>SHARE</strong></td>
</tr>
<tr>
<td></td>
<td>More than one application instance can get messages from the queue.</td>
</tr>
</tbody>
</table>
Table 6. Required parameter values for WebSphere MQ objects at the source (continued)

<table>
<thead>
<tr>
<th>Object name</th>
<th>Required settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart queue</td>
<td>PUT (ENABLED)</td>
</tr>
<tr>
<td></td>
<td>Allows the Q Capture program to put a restart message on the queue.</td>
</tr>
<tr>
<td></td>
<td>GET (ENABLED)</td>
</tr>
<tr>
<td></td>
<td>Allows the Q Capture program to get the restart messages from the queue.</td>
</tr>
<tr>
<td></td>
<td>INDXTYPE(MSGID)</td>
</tr>
<tr>
<td></td>
<td>Allows the queue manager to maintain an index of message identifiers.</td>
</tr>
</tbody>
</table>

**WebSphere MQ objects at the target**

Table 7 provides required values for selected parameters for WebSphere MQ objects at the target.

Table 7. Required parameter values for WebSphere MQ objects at the target

<table>
<thead>
<tr>
<th>Object name</th>
<th>Required settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue manager</td>
<td>MAXMSGL (The maximum size of messages allowed on queues for this queue manager.)</td>
</tr>
<tr>
<td></td>
<td>This value must be at least as large as the MAX_MESSAGE_SIZE and MEMORY_LIMIT that</td>
</tr>
<tr>
<td></td>
<td>you define when you create a replication queue map. The MAX_MESSAGE_SIZE defines</td>
</tr>
<tr>
<td></td>
<td>the message buffer that is allocated for each send queue. The MEMORY_LIMIT defines</td>
</tr>
<tr>
<td></td>
<td>the message buffer that a Q Apply program uses for buffering transactions from each</td>
</tr>
<tr>
<td></td>
<td>receive queue.</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is not valid on z/OS.</td>
</tr>
</tbody>
</table>

Receive queue

| GET(ENABLED) | Allows the Q Apply program to get messages from 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.                                                                        |
| SHARE        | Allows multiple Q Apply threads to work with this queue.                          |
| MSGDLVSQ(PRIORITY) | Specifies that messages on the queue are delivered in first-in, first-out order |
|               | within priority.                                                                  |
| INDXTYPE(MSGID) | Specify that the queue manager maintain an index of messages based on the message |
|               | identifier to expedite MQGET operations on the queue. This parameter is supported  |
|               | only on local and model queues.                                                   |
Table 7. Required parameter values for WebSphere MQ objects at the target (continued)

<table>
<thead>
<tr>
<th>Object name</th>
<th>Required settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration queue</td>
<td><strong>PUT(ENABLED)</strong></td>
</tr>
<tr>
<td></td>
<td>Allows the Q Apply program or user application to put control messages on the queue.</td>
</tr>
<tr>
<td></td>
<td><strong>SHARE</strong></td>
</tr>
<tr>
<td></td>
<td>Allows multiple Q Apply threads to work with this queue.</td>
</tr>
<tr>
<td>Model (spill) queue</td>
<td><strong>Queue name</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>DEFTYPE(PERMDYN)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>DEFSOPT(SHARED)</strong></td>
</tr>
<tr>
<td></td>
<td>Allows more than one thread (different agent threads and the spill agent thread) to access messages on the spill queue at the same time.</td>
</tr>
<tr>
<td></td>
<td><strong>MAXDEPTH(500000)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>MAXMSGL(100000)</strong></td>
</tr>
<tr>
<td></td>
<td>This is a recommended limit for the maximum message size. Make sure this value is at least as large as the MAXMSGL for the source queue manager and transmission queue.</td>
</tr>
<tr>
<td></td>
<td><strong>MSGDLVSQ(FIFO)</strong></td>
</tr>
<tr>
<td></td>
<td>Specifies that messages on the spill queue are delivered in first-in, first-out order.</td>
</tr>
<tr>
<td></td>
<td><strong>INDXTYPE(MSGID)</strong></td>
</tr>
<tr>
<td></td>
<td>Specify that the queue manager maintain an index of messages based on the message identifier to expedite MQGET operations on the queue. This parameter is supported only on local and model queues.</td>
</tr>
</tbody>
</table>

**WebSphere MQ channel objects**

Table 8 on page 29 describes the WebSphere MQ objects that are used to create channels between the source and target systems, and provides required values for selected parameters.
Table 8. Required parameter values for WebSphere MQ objects used in channels

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
<th>Required settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission queue (Q Capture to Q Apply or user application)</td>
<td>Holds transaction messages and informational messages from the Q Capture program that are bound for a remote Q Apply program or user application.</td>
<td>USAGE(XMITQ) Transmission queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAXDEPTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you plan to use a single transmission queue for multiple send queues, ensure that the maximum number of messages allowed on the transmission queue is at least as large as the combined maximums for all send queues.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAXMSGL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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 MAX_MESSAGE_SIZE and MEMORY_LIMIT that you set when you create a replication queue map.</td>
</tr>
<tr>
<td>Transmission queue (Q Apply or user application to Q Capture program)</td>
<td>Holds control messages from the Q Apply program or user application that are destined for the administration queue of the Q Capture program.</td>
<td>USAGE(XMITQ) Transmission queue.</td>
</tr>
<tr>
<td>Channel (Q Capture program to Q Apply or user application)</td>
<td>Defined within the source queue manager for outbound data and informational messages.</td>
<td>CHLTYPE(SDR) A sender channel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISCINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensure that the disconnect interval is large enough to keep this channel from timing out during periods when there are no transactions to replicate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONVERT(NO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify that the sending message channel agent should not attempt conversion of messages if the receiving message channel agent cannot perform this conversion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HBINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Channel (Q Apply or user application from Q Capture program)</td>
<td>Defined within the target queue manager for inbound transaction or informational messages.</td>
<td>CHLTYPE(RCVR) A receiver channel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HBINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 8. Required parameter values for WebSphere MQ objects used in channels (continued)

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
<th>Required settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel (Q Apply or user application to Q Capture program)</td>
<td>Defined within the target queue manager for outbound control messages.</td>
<td>CHLTYPE(SDR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A sender channel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISCINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HBINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Q Apply program does not send heartbeat messages, so there is no need to coordinate this value with any Q replication parameters.</td>
</tr>
<tr>
<td>Channel (Q Capture from Q Apply or user application)</td>
<td>Defined within the source queue manager for inbound control messages.</td>
<td>CHLTYPE(RCVR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A receiver channel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HBINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Q Apply program does not send heartbeat messages, so there is no need to coordinate this value with any Q replication parameters.</td>
</tr>
</tbody>
</table>

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
```bash
crtmqm queue_manager_name
```

Start queue manager
```bash
strmqm queue_manager_name
```

After you create and start a queue manager, you can use the MQSC commands in Table 9 on page 31 and Table 10 on page 31 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 event 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. To download sample scripts, see WebSphere MQ setup scripts for Q replication.
### Table 9. Sample MQSC commands for WebSphere MQ objects that are used by the Q Capture program

<table>
<thead>
<tr>
<th>Object</th>
<th>Sample command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue manager</strong></td>
<td>A program that manages queues for the Q Capture program.</td>
</tr>
<tr>
<td></td>
<td>crtmqm QM1</td>
</tr>
<tr>
<td><strong>Send queue</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>DEFINE QREMOTE('ASN.QM1_TO_QM2.DATAQ') RNAME('ASN.QM1_TO_QM2.DATAQ') RQNAME('QM2') XMITQ('QM2') DEFPSIST(YES) INDXTYPE(GROUPID)</td>
</tr>
<tr>
<td><strong>Administration queue</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>DEFINE QLOCAL('ASN.QM1.ADMINQ') DEFPSIST(YES) INDXTYPE(GROUPID)</td>
</tr>
<tr>
<td><strong>Restart queue</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>DEFINE QLOCAL('ASN.QM1.RESTARTQ') DEFPSIST(YES) INDXTYPE(GROUPID)</td>
</tr>
<tr>
<td><strong>Transmission queue</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>DEFINE QLOCAL('QM2') USAGE(XMITQ) DEFPSIST(YES)</td>
</tr>
<tr>
<td><strong>Sender channel</strong></td>
<td>The sending end of the channel from the source system to the target system.</td>
</tr>
<tr>
<td></td>
<td>DEFINE CHL ('QM1_TO_QM2') + CHLTYPE(SDR) TRPTYPE(TCP) + CONNAME('IP_address (port)') + XMITQ('QM2') + DISCINT (0) 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.</td>
</tr>
<tr>
<td><strong>Receiver channel</strong></td>
<td>The receiving end of the channel from the target system to the source system.</td>
</tr>
<tr>
<td></td>
<td>DEFINE CHL ('QM2_TO_QM1') + CHLTYPE(RCVR) TRPTYPE(TCP)</td>
</tr>
</tbody>
</table>

### Table 10. Sample MQSC commands for WebSphere MQ objects that are used by the Q Apply program

<table>
<thead>
<tr>
<th>Object</th>
<th>Sample command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue manager</strong></td>
<td>A program that manages queues for Q Apply.</td>
</tr>
<tr>
<td></td>
<td>crtmqm QM2</td>
</tr>
</tbody>
</table>
Table 10. Sample MQSC commands for WebSphere MQ objects that are used by the Q Apply program (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Sample command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive queue</td>
<td>DEFINE QLOCAL('ASN.QM1_TO_QM2.DATAQ') DEFPSIST(YES)</td>
</tr>
<tr>
<td>Administration queue</td>
<td>DEFINE QREMOTE('ASN.QM1.ADMINQ') RNAME('ASN.QM1.ADMINQ') RQNAME('QM1') XMITQ('QM1') DEFPSIST(YES)</td>
</tr>
<tr>
<td>Spill queue</td>
<td>DEFINE QMODEL('IBMQREP.SPILL.MODELQ') DEFSOPT(SHARED) MAXDEPTH(500000) MSGDLSQQ(QIFO) DEFTYPE(PERMDYN)</td>
</tr>
<tr>
<td>Transmission queue</td>
<td>DEFINE QLOCAL('QM1') USAGE(XMITQ) DEFPSIST(YES)</td>
</tr>
<tr>
<td>Sender channel</td>
<td>DEFINE CHL ('QM2_TO_QM1') + CHLTYPE(SDR) TRPTYPE(TCP) + CONNAME('IP_address (port)') + XMITQ('QM1') + DISCINT (0)</td>
</tr>
<tr>
<td>Receiver channel</td>
<td>DEFINE CHL ('QM1_TO_QM2') + CHLTYPE(RECVR) TRPTYPE(TCP)</td>
</tr>
</tbody>
</table>

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 (LIBPATH on AIX®, LD_LIBRARY_PATH on SunOS, SHLIB_PATH on HP-UX, and PATH on Windows).

Restrictions

z/OS A WebSphere MQ for z/OS subsystem cannot be a client.

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 file on the client system that contains connection 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 WebSphere MQ Clients 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 environment variable ASNUSEMQCLIENT=TRUE.
2. Create a client-connection channel by using one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQSERVER environment variable</td>
<td>Set the MQSERVER system environment variable to point to the queue manager that the replication program works with. For example, if the Q Apply program runs on a WebSphere MQ client system and works with the remote queue manager RQMGR2, set MQSERVER=RQMGR2 on the client system.</td>
</tr>
<tr>
<td>MQCHLLIB or MQCHLTAB environment variables</td>
<td>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.</td>
</tr>
</tbody>
</table>

Note: Because MQCHLLIB and MQCHLTAB have platform-specific default values, you can omit setting these environment variables if the configuration file for the MQ client channel definitions 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).
3. Optional: To use the replication administration tools to list and validate
WebSphere MQ objects, or to use the Replication Alert Monitor to monitor the
QUEUE_DEPTH alert condition, follow these steps:
   a. Define an MQSERVER system environment variable for each remote queue
      manager that is used by a replication program that runs on a WebSphere
      MQ client.
      For example, if the Q Capture program uses remote queue manager
      RQMGR1 and the Q Apply program uses RQMGR2, define two system
      variables, MQSERVER_RQMGR1 and MQSERVER_RQMGR2.
   b. For every remote queue manager, define another environment variable:
      ASNUSEMQCLIENT_queuemanager=TRUE.
      For example, if the Q Capture program uses remote queue manager
      RQMGR1 and the Q Apply program uses RQMGR2, define these two
      system variables:
         ASNUSEMQCLIENT_RQMGR1=TRUE
         ASNUSEMQCLIENT_RQMGR2=TRUE
   c. Add the new environment variables to the DB2ENVLIST
      variable.
      For example:
      DB2ENVLIST="ASNUSEMQCLIENT_RQMGR1 MQSERVER_RQMGR1
                  ASNUSEMQCLIENT_RQMGR2 MQSERVER_RQMGR2"
   d. Stop and restart DB2 after setting DB2ENVLIST.

Validating WebSphere MQ objects for Q replication and event
publishing

You can use the ASNCLP command-line program or Replication Center to check
whether the WebSphere MQ queue managers and queues in your configuration
have the correct properties.

Before you begin
• The queue manager where the objects are defined must be started.
• The command server for the queue manager must be
  running to display queues in the Replication Center. If you start the queue
  manager with the WebSphere MQ Explorer tool, the command server starts
  automatically. You can also start the server by using the strmqcsv command.
• 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.
• 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
• 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 32.

- **Linux UNIX** On Linux and UNIX, to use this function you must first add the path to the WebSphere MQ libraries to the DB2LIBPATH environment variable and stop and start DB2 before you start the Replication Center.

**Procedure**

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASNCLP command-line program</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Replication Center</strong></td>
<td>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. You can modify the queue manager or queue and click the validation link again to verify your changes.</td>
</tr>
</tbody>
</table>

For more information about WebSphere MQ reason codes, see the WebSphere MQ information center.

**Sending test messages between queues in a replication queue map**

You can use the ASNCLP command-line program or Replication Center to check the message flow between the queues in a replication queue map by sending test messages between the queues.

**Before you begin**

- Ensure that the queue manager where the objects are defined is started.
- **Windows** The command server for the queue manager must be running to display queues in the Replication Center. If you start the queue manager with the WebSphere MQ Explorer tool, the command server starts automatically. You can also start the server by using the strmqcsv command.
- **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.
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.

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 32.

Restrictions
- You can send test messages only when the Q Apply program is stopped.
- If any of the validation tests on WebSphere MQ objects fails, the test message is not sent. You can fix any problems with object definitions and then retry the test.

About this task

The Replication administration tool 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.

Procedure

To send test messages between queues in a replication queue map, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASNCLP command-line program</strong></td>
<td>Use the <code>VALIDATE WSMQ MESSAGE FLOW FOR REPLQMAP</code> 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: <code>VALIDATE WSMQ MESSAGE FLOW FOR REPLQMAP SAMPLE ASN TO TARGET ASN</code></td>
</tr>
</tbody>
</table>

| Replication Center      | 1. Open one of the following windows:                                        |
|                        |   • The Create Replication Queue Map window by right-clicking a Replication Queue Maps folder in the object tree and clicking Create. |
|                        |   • The Replication Queue Map Properties window by right clicking a replication queue map in the contents pane and clicking Properties. |
|                        | 2. Make sure that the send queue, receive queue, and Q Apply administration queue that you want to test are specified, then click Validate queues. |
|                        | 3. 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. |

For more information about WebSphere MQ reason codes, see the WebSphere MQ information center.
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), and get messages (MQGET).

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.

---

### Recovering when WebSphere MQ messages are lost

You can keep source and target tables synchronized even in situations where the WebSphere MQ messages that are used to convey replicated data are lost.

**About this task**

To do so, you determine the point from which the Q Capture program needs to reread the recovery log to recapture any missing transactions, and then restart the program. By starting from a known point in the log, you avoid the need to reload target tables with all source data (full refresh). Figure 6 shows the information that you need to provide to Q Capture for its restart.

![Recovery log](image)

**Figure 6. Recovery point for Q Capture program.** The restart point is the first log record that is part of the oldest transaction for which a commit or rollback was not seen when Q Capture stopped. This point is known as the minimum log sequence number (LSN), and is used as the value for the lsn parameter. Because Q Capture goes back in the log far enough to read transactions that were incomplete at shutdown, when it restarts it might also recapture already-published transactions. Q Capture only sends transactions that are newer than the last published transaction at shutdown. This point is called the maximum commit sequence and is used as the value for the maxcmtseq parameter.

The following situations might require you to recover:
Nonpersistent messages are lost and the queue manager stops because of error or outage

If you start Q Capture with the `msg_persistence=n` parameter, it uses nonpersistent messages. When the queue manager stops, any message that are still in queues are lost.

**Note:** If you use `msg_persistence=n`, ensure that queues are empty before any planned shutdown of the queue manager by first stopping Q Capture and then verifying that Q Apply has emptied all queues. Restart messages are always persistent and survive the shutdown or failure of the queue manager. Messages in the Q Apply spill queue are also persistent, but if they are lost you must stop and restart the Q subscription, which prompts a new load of the target table.

Q Capture uses persistent messages, but a nonrecoverable WebSphere MQ disk failure occurs

In the case of a disk failure, messages that contain captured changes that are not yet applied at the target might be lost.

**Procedure**

1. Determine how current the target is by getting the commit timestamp of the last transaction that was applied at the target before the failure. Use one of the following counters from the IBMQREP_APPLYMON table at the target:

   **OLDEST_COMMIT_LSN (use this value for maxcmtseq with z/OS sources)**
   The earliest LSN of the commit for the oldest transaction that has been applied at the target.

   **OLDEST_COMMIT_SEQ (use this value for maxcmtseq with Linux, UNIX, and Windows sources)**
   A formatted timestamp that denotes the oldest transaction that has been applied at the target.

   **Note:** For a planned outage, you can republish only those transactions that were not applied at the target. For unplanned outages, Q Capture might have to republish some transactions that were already applied, which could cause conflicts at the target. The Q Apply program resolves and reports any conflicts if the conflict action for the Q subscription was set to F (force) or I (ignore) before the outage. Q Apply might not be able to resolve some types of conflicts, depending on the nature of changes that were applied and resent by Q Capture.

2. Determine how far back Q Capture needs to read in the log to capture and send any transaction that was not successfully replicated before the failure. This enables Q Capture to resend any transaction that could not be applied at the target because it was not completely captured at the source, or was captured but the transaction message was lost because of the WebSphere MQ failure. Use the following counter from the IBMQREP_CAPMON table at the source:

   **RESTART_SEQ (use this value for lsn)**
   The logical LSN at which the Q Capture program starts during a warm restart. This value represents the earliest LSN that the Q Capture program found for which a commit or abort record was not found at the time that the monitor table was updated. Choose an entry in the monitor table that is older than the time of the crash to ensure that any lost messages are recaptured. Going back further than necessary in the log poses no harm except for the extra time that Q Capture spends...
reading the log. Any transactions that were committed before the value of the `maxcmtseq` parameter are not republished.

3. Restart the Q Capture program from this point in the log by specifying the value from Step 1 in the `maxcmtseq` parameter, and the value from Step 2 in the `lsn` parameter. Use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCL</td>
<td>Use JCL similar to the following example:</td>
</tr>
<tr>
<td></td>
<td>//QCAP EXEC PGM=ASNQCAP,</td>
</tr>
<tr>
<td></td>
<td>// PARM=&quot;/CAPTURE_SERVER=DSN1 CAPTURE_SCHEMA=CAPCAT&quot;</td>
</tr>
<tr>
<td></td>
<td>// LSN=000000000000115b7704</td>
</tr>
<tr>
<td></td>
<td>// MAXCMTSEQ=41c22264000000040000</td>
</tr>
<tr>
<td>asncap command</td>
<td>Use a command similar to the following example:</td>
</tr>
<tr>
<td></td>
<td>asncap capture_server=source</td>
</tr>
<tr>
<td></td>
<td>lsn=000000000000115b7704</td>
</tr>
<tr>
<td></td>
<td>maxcmtseq=41c22264000000040000</td>
</tr>
</tbody>
</table>

You can specify the `maxcmtseq` value with or without colons. For example, both of the following values are acceptable:

- `maxcmtseq=41c22264000000040000`
- `maxcmtseq=41c2:2264:0000:0004:0000`

### Recovering after messages are sent to dead-letter queues

If your WebSphere MQ environment uses dead-letter queues, you might need to recover from a situation where messages are sent to dead-letter queues and as a result the Q Apply program stops processing messages.

**About this task**

By default, WebSphere MQ messages that cannot be delivered are placed on dead-letter queues. These queues are not necessary for Q replication, and if a queue manager is being used strictly for replication IBM recommends that you do not define a dead-letter queue.

You might not always be able to follow this recommendation. For example, applications outside of replication that share the same queue manager might require a dead-letter queue. In some environments, it might be a standard practice to create a dead-letter queue for all queue managers.

An environment with dead-letter queues can cause the Q Apply program to stop replication if a message is not delivered to the receive queue:

- The Q Apply program processes messages in a strict order. The Q Capture program generates a message sequence number to manage this strict processing order.
- When a message has not arrived in the receive queue in its expected order, the Q Apply program continues to look for the message.
- If the message is sent to a dead-letter queue, Q Apply cannot find it. At some point while looking for the message, the program stops processing messages until it finds the message with the missing sequence number.

The Q Apply program issues the ASN7551E message when it detects a gap in the message sequence. If you receive this message, the Q Apply program has stopped processing messages from the receive queue, although the Q Apply program continues to run.
**Note:** The following procedure describes steps to take when one or more messages are sent to a dead-letter queue. This scenario can occur when a receive queue fills up because it is too small or because of a problem that prevents the Q Apply program from applying messages to the target. For more detail on recovering from the full-queue scenario, see the article "Q Replication recovery" on the IBM developerWorks Web site at the following URL: [http://www.ibm.com/developerworks/db2/library/techarticle/dm-0709hamel/](http://www.ibm.com/developerworks/db2/library/techarticle/dm-0709hamel/)

**Procedure**

To recover after messages are sent to dead-letter queues:

1. Do one of the following:
   - For a single missing message, use the message ID that is shown in the ASN7551E message to look for the message with the expected ID on the dead-letter queue. In the following example from the Q Apply diagnostic log, the expected message ID is 5152455046CB582B00000000000000000000000001:
     
     2007-08-21-14.35.52.172000 <browser::readNextDenseMsg ASN7551E> "Q Apply" :
     "ASN" : "BR00001" : The Q Apply program detected a gap in message numbers on receive queue "asn.qm1.dataq3", replication queue map LONDON ASN_TO_TOKYO ASN". It read message ID "5152455046CB582B00000000000000000000000003", but expected to find message ID "5152455046CB582B00000000000000000000000001". The Q Apply program cannot process any messages until it finds the expected message.
     
     If you recover the message, put it on the receive queue, preserving the WebSphere MQ message header information (especially the message ID).
   - For multiple missing messages, run the DLQ handler using the runmqd1q command to forward missing messages to the receive queue.

2. If the message cannot be recovered, follow these steps:

   **Recommendation:** These steps prompt a new load (full refresh) of any affected target tables and should be avoided if possible.
   - a. Ensure that the Q Apply program has stopped reading from the receive queue by issuing the asnqacmd stopq command.
   - b. Stop all Q subscriptions that use the replication queue map that contains the receive queue.
   - c. Remove all messages from the receive queue and its corresponding send queue.
   - d. Use the `startq` parameter with the MODIFY command on z/OS or asnpacmd command on Linux, UNIX, Windows, and UNIX System Services on z/OS so that the Q Apply program resumes reading from the receive queue.
   - e. Start all of the Q subscriptions that you stopped in Step b.

**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 5. Configuring databases for Q replication and event publishing (Linux, UNIX, Windows)

Setting environment variables (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.

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.
   - For Linux and UNIX, use the following command:
     ```
     export DB2INSTANCE=db2_instance_name
     ```
   - 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. 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
   ```
   - Ensure that the EXTSHM environment variable is set each time you start DB2. Do this by editing the `~/db2inst/sql1ib/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'
   - Add the following line to the `~/db2inst/sql1ib/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.
Chapter 5. Configuring databases (Linux, UNIX, Windows)

47

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:

   ```bash
   lspath -E -1 sys0
   ```

   Set maxuproc to a higher value using the following command as user root:

   ```bash
   chdev -l syst -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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command line</td>
<td>1. Issue the following command:</td>
</tr>
<tr>
<td></td>
<td>db2 update database configuration for database using LOGARCHMETH1=LOGRETAIN</td>
</tr>
<tr>
<td></td>
<td>2. If you want to use an exit routine to manage archived logs, set the USEREXIT database configuration parameter to ON.</td>
</tr>
<tr>
<td></td>
<td>3. Optional: Use the update database configuration command to increase values for the source database depending on your replication or publishing needs.</td>
</tr>
<tr>
<td></td>
<td>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, 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.</td>
</tr>
<tr>
<td></td>
<td>4. Issue the backup database command using appropriate parameters for the source database.</td>
</tr>
</tbody>
</table>

### 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**

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:
\[ n \geq (\text{number of applications other than Q Apply that can use the database at the same time}) + (3 \times \text{the number of Q Apply programs on the database}) + (\text{number of receive queues, each with a browser thread} + \text{total number of Q Apply agents for all receive queues}) \]

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).

\[ n \geq 2 + (3 \times 2) + (5 + 15) \]

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, LOGFILSZ 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:
   ```sql
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 command:
   
   `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_install_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 command:
      
      `db2 bind @qapply.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 qapply.lst.

   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`. 

 configuring databases for Q replication and event publishing (z/OS)
You must set up and customize the replication programs when you install Q replication and event publishing on z/OS.

For information, see [Replication installation and customization for z/OS](#).

---

### Software prerequisites for the Replication Center

The Replication Center requires the correct Java™ Runtime Environment (JRE) and requires the DB2 Administration Server (DAS).

When you install DB2, you have the option to install the JRE. It is strongly recommended that you do so. If you choose not to install the JRE, you must ensure that your system has at least Version 1.3.1 of an IBM-approved Java 2 Runtime Environment or Java 2 Software Development Kit.

If you will use the Replication Center to operate the Q Capture, Q Apply, or Replication Alert Monitor programs on remote systems, ensure that the DB2 Administration Server is running on each of the remote DB2 systems that will run the Q Capture or Q Apply programs. You can use the Replication Center for configuration tasks without the DAS. The DAS for z/OS is available only with DB2 for OS/390® and z/OS Version 7 or later.

**z/OS:** For more information on FMIDs that are required to support the DAS and Replication Center, see the [IBM WebSphere Information Integration z/OS Install and Customization Guide for Replication and Event Publishing](#).

---

### 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 partitioned tables

Starting with Version 9.7, Q replication and event publishing support source tables that are partitioned by range (using the PARTITION BY clause of the CREATE TABLE statement).

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 partition at which they occur, are replicated or published.

You can perform several alterations on a partitioned table, including adding a partition, attaching a partition, or detaching a 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:

**ADD**
Adds a new, empty partition to the source table. If you require the new 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 partition at the target before adding it at the source. Q Capture automatically begins replicating changes to the partition.

- **Version 9.5 or 9.1**
  Q Capture does not recognize the addition of the partition until the program is reinitialized or stopped and restarted. Add the partition at both the source and target before restarting Q Capture. Do not change data in the source partition until Q Capture is restarted.

**ATTACH**
Creates a new partition at the source by using an existing table. The ATTACH operation is not replicated and the data in the new partition is not replicated to the target. If you require the new 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 partition at the target.

**Note:** If the Q Capture program is stopped when a partition is attached, rows that are inserted, updated, or deleted at the partition before it is attached are replicated. If Q Capture is running when the partition is attached, these rows are not replicated. To ensure consistent behavior, make sure the DATA CAPTURE CHANGES clause for the table is set to OFF if you need to make any changes to the table before attaching the table as a partition. For example, the following statements create a table, insert values into the table, and then attach the table as a 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 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 partition into a separate table, you need to do so manually.

**Note:** DB2 logs updates that cause rows to move across 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.

---

**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 128.

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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:  
  ASNCPL 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 event 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 event 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.

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.

Procedure

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line program</td>
<td>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 &quot;ASN1.QM1.RESTARTQ&quot; ADMINQ &quot;ASN1.QM1.ADMINQ&quot; version 9.5</td>
</tr>
<tr>
<td>Replication Center</td>
<td>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.</td>
</tr>
</tbody>
</table>

Configuring for older or remote DB2 databases

You can configure WebSphere 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 Replication Server or Data Event Publisher to reside 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 WebSphere Replication Server with an older version of DB2, in this example, Replication Server V9.5 with DB2 V8.2.
The following figure shows the basic configuration for using WebSphere Replication Server with remote DB2 databases.

**Figure 8. A configuration of Replication Server with older 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, read 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, review release coexistence information here: 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
WebSphere Replication Server or Data Event Publisher is installed and licensed, its replication programs can work with any DB2 database that is at the same release or lower (down to V8.2), regardless whether that database resides in another DB2 instance or on another system. A new DB2 instance is created as part of this V9.5 installation, however this new V9.5 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

- For running Q Capture remotely, the Q Capture program and the source database must be on Linux, UNIX, or Windows.
- For running Q Apply remotely, the Q Apply program and the target database must be on Linux, UNIX, or Windows.
- You cannot use the replication administration tools to list or validate queues for any Q Capture or Q Apply programs that run on a system that is remote from the source or target database.
- 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 V8.2) 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 event 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 6. 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 10 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.

![Diagram showing multiple Q subscriptions for unidirectional replication]

**Figure 10. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| ASNCLP command-line      | 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. |
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.

**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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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  
TGEMPLOYEE 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 11 on page 73 shows how a subset of columns are replicated.
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 74

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
  #-----------------------------------------------
  # Incorrect: Don't do this
  #-----------------------------------------------
  
  WHERE :LOCATION = 'EAST' AND SUM(:SALES) > 100000
  ```

  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
  #-----------------------------------------------
  # 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
  #-----------------------------------------------
  # 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.

How source columns map to target columns (unidirectional replication)

For existing target tables or for stored procedures, you can specify how you want the data from source columns to map to target columns or to parameters in a stored procedure. If a target table does not exist, then the Replication Center creates the target table (or nickname if you have non-DB2 target tables) with the same columns that the source table has.

Attention: The source table cannot replicate more columns than are in the target table or more columns than parameters in the stored procedure. However, the target table can contain more columns than the number of source columns that you selected for replication. (Stored procedures cannot have extra parameters that do not map to source columns.) 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.

If you are creating a single Q subscription, then the Replication Center tries to map the columns in the source table to columns in the target table or to parameters in the stored procedure by pairing identical column names and data types. If any columns do not map to identical column names and data types, then you must manually map those extra columns. The Replication Center also allows you to change the mapping of compatible columns if you choose.

Non-DB2 target tables:

- 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.
- 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.

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.
When you create Q subscriptions, 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.

**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(COL1,COL2) 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 12 shows how expressions can be used in Q replication to create computed columns.

**Expression Combinations**

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column name</strong></td>
<td><strong>Column type</strong></td>
</tr>
<tr>
<td>KEY 1</td>
<td>INT</td>
</tr>
<tr>
<td>C1</td>
<td>CHAR(10)</td>
</tr>
<tr>
<td>C2</td>
<td>CHAR(10)</td>
</tr>
<tr>
<td>C3</td>
<td>INT</td>
</tr>
<tr>
<td>C4</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 12. 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 (:), 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.
- Constants and derived constants are not supported as key columns for replication (IS_KEY=Y in the IBMQREP_TRG_COLS table).

Figure 13 on page 81 shows how expressions can be used in Q replication to create computed columns.
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;
```

---

![Expression Combinations (keys)](image)

**Figure 13. 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).
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 83
- “[Restrictions” on page 83
- “[Examples of XML expressions” on page 84
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 11 provides examples of supported XML expressions and lists unsupported expressions.

Table 11. Supported and unsupported XML expressions

<table>
<thead>
<tr>
<th>Examples of supported XML expressions</th>
<th>Unsupported XML expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• XMLATTRIBUTES scalar function</td>
<td>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.</td>
</tr>
<tr>
<td>• XMLCOMMENT scalar function</td>
<td>• PARAMETER scalar function: You can use this function in the db2-fn:sqlquery function; a query is not part of an expression</td>
</tr>
<tr>
<td>• XMLCAST specification</td>
<td>• XMLAGG: aggregate function</td>
</tr>
<tr>
<td>• XMLCONCAT scalar function</td>
<td>• XMLEXIST predicate: Can be used in the WHERE clause of a query</td>
</tr>
<tr>
<td>• XMLDOCUMENT scalar function</td>
<td>• XMLGROUP: aggregate function</td>
</tr>
<tr>
<td>• XMLELEMENT scalar function</td>
<td>• XMLTABLE: aggregate function</td>
</tr>
<tr>
<td>• XMLFOREST scalar function</td>
<td>• XMLXMLSROBJECTID: scalar function with no XML value output</td>
</tr>
<tr>
<td>• XMLNAMESPACE scalar function</td>
<td>• XMLTRANSFORM: scalar function; not supported because other output than an XML value is possible.</td>
</tr>
<tr>
<td>• XMLPARSE scalar function</td>
<td></td>
</tr>
<tr>
<td>• XMLPI scalar function</td>
<td></td>
</tr>
<tr>
<td>• XMLQUERY scalar function</td>
<td></td>
</tr>
<tr>
<td>• XMLROW scalar function</td>
<td></td>
</tr>
<tr>
<td>• XMLELEMENT scalar function</td>
<td></td>
</tr>
<tr>
<td>• XMLEXISTS predicate</td>
<td></td>
</tr>
<tr>
<td>• XMLTEXT scalar function</td>
<td></td>
</tr>
<tr>
<td>• XMLVALIDATE scalar function</td>
<td></td>
</tr>
</tbody>
</table>

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>  
| 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

Output

Resulting data at target

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

Expression

Resulting data at target

CCD tables as targets for Q replication
Consistent-change-data (CCD) target tables provide committed transactional data that can be read and used by other applications, for example WebSphere DataStage® or the SQL Apply program. 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 targets in unidirectional Q replication. 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:
• Force the source change into the target table (CONFLICT_ACTION=F).
• Ignore the condition and continue (CONFLICT_ACTION=I).

Any combination other than condensed and complete
The only valid option is to force the change into the target table.

For all CCD table types, the only valid conflict option is to check only key columns (CONFLICT_RULE=K).

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:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMSNAP_INTENTSEQ</td>
<td>Data type: CHAR(10) FOR BIT DATA; Nullable: No</td>
</tr>
<tr>
<td></td>
<td>A sequence number that uniquely identifies a change. This value is ascending in a transaction.</td>
</tr>
<tr>
<td></td>
<td>z/OS The log sequence number (LRSN or RBA) of each update, delete, and insert.</td>
</tr>
<tr>
<td>IBMSNAP_OPERATION</td>
<td>Data type: CHAR(1); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates the type of operation: I (INSERT), U (UPDATE), or D (DELETE).</td>
</tr>
</tbody>
</table>
### Column Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
</table>
| IBMSNAP_COMMITSEQ   | **Data type:** CHAR(10) FOR BIT DATA; Nullable: No  
A sequence number for each row within a transaction.  

<table>
<thead>
<tr>
<th>z/OS</th>
<th>The log sequence number (LRSN or RBA) of the source commit record.</th>
</tr>
</thead>
</table>
| IBMSNAP_LOGMARKER   | **Data type:** TIMESTAMP; Nullable: No  
The time at the source server that the data was committed, measured in Greenwich Mean Time (GMT). |

### Optional auditing columns

When you create a noncomplete (COMPLETE=N) CCD table with the ASNCNP command-line program or Replication Center, you can specify additional auditing columns. The following table describes these columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
</table>
| IBMSNAP_AUTHID       | **Data type:** VARCHAR(30); Nullable: Yes  
The user ID that updated the source table.  

<table>
<thead>
<tr>
<th>z/OS</th>
<th>This column is the primary authorization ID.</th>
</tr>
</thead>
</table>
| IBMSNAP_AUTHTKN      | **Data type:** VARCHAR(30); Nullable: Yes  
The authorization token that is associated with the transaction.  

<table>
<thead>
<tr>
<th>z/OS</th>
<th>The correlation ID (normally a job name) that ran the source update.</th>
</tr>
</thead>
</table>
| IBMSNAP_PLANID       | **Data type:** VARCHAR(8); Nullable: Yes  
The plan name that is associated with the transaction. This column will be null for DB2 for Linux, UNIX, and Windows.  

<table>
<thead>
<tr>
<th>z/OS</th>
<th>The plan name that is associated with the transaction. This column will be null for DB2 for Linux, UNIX, and Windows.</th>
</tr>
</thead>
</table>
| 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. |
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 primary key, unique constraint, or unique index for the target.

If the Replication Center cannot find uniqueness at the source, it will automatically specify all valid, subscribed source columns as key columns for replication, which allows you to replicate between source and target tables with no primary keys or unique constraints.

**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 89

**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.

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 existing target table that is based on all valid, subscribed source columns.

**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” on page 90
- “Ignore the unexpected condition” on page 90
- “Deactivate the corresponding Q subscription” on page 90
- “Have the Q Apply program stop reading from the corresponding receive queue” on page 91
- “Stop the Q Apply program” on page 92
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 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 94
- “Stop the Q Apply program” on page 94

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 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.
Chapter 7. 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 14 shows bidirectional replication of one logical table between two servers. For one logical table, you need two Q subscriptions and two replication queue maps.

![Bidirectional Replication Diagram](image)

**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
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” on page 101
- “Replication objects for peer-to-peer replication with three or more servers” on page 103
- “Conflict resolution and referential integrity” on page 103

**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 15 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.

**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 16 on page 103](#) 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.
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 concomitantly 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.

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 ASINCLP
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:
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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 SERVER MULTIDIR TO DB "SAMPLE";
  SET SERVER MULTIDIR TO DB "SAMPLE2";
  SET MULTIDIR SCHEMA "SAMPLE".RED;
  SET MULTIDIR SCHEMA "SAMPLE2".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.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 A 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.  
  The commands are saved to a script file, which is then invoked by using the LOAD MULTIDIR REPL SCRIPT command. |
| **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>
<thead>
<tr>
<th>Configuration</th>
<th>Recapture avoidance method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple bidirectional configurations between two servers, or a combination of bidirectional, unidirectional, and peer-to-peer configurations</td>
<td>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.</td>
</tr>
<tr>
<td>Pure, two-server bidirectional configuration</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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**

   ```sql
   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.
Note: 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.

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.

3. Optional: 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.

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” on page 109
- “How conflicts are resolved at each server” on page 109
- “Outcomes of different choices for checking and resolving conflicts” on page 110
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 13 on page 111 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 13. Outcomes of different combinations of options for conflict detection

<table>
<thead>
<tr>
<th>How to check for conflicts</th>
<th>How to resolve conflicts</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check all columns for conflicts.</td>
<td>The first server takes precedence.</td>
<td>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.</td>
</tr>
<tr>
<td>The second server takes precedence.</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Neither server takes precedence.</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>How to check for conflicts</td>
<td>How to resolve conflicts</td>
<td>Outcome</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Check only changed non-key columns for conflicts.</td>
<td>The first server takes precedence.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>The second server takes precedence.</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Neither server takes precedence.</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 13. Outcomes of different combinations of options for conflict detection (continued)

<table>
<thead>
<tr>
<th>How to check for conflicts</th>
<th>How to resolve conflicts</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check only key columns for conflicts.</td>
<td>The first server takes precedence.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>The second server takes precedence.</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Neither server takes precedence.</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

### 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.
- 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:

```sql
SET SUBGROUP "p2p2group";
SET SERVER MULTIDIR TO DB "SAMPLE";
SET SERVER MULTIDIR TO DB "SAMPLPEER";

SET MULTIDIR SCHEMA "SAMPLE".GREEN;
SET MULTIDIR SCHEMA "SAMPLPEER".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.

The commands are saved to a script file, which is then invoked by using the LOAD MULTIDIR REPL SCRIPT command.

#### 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.
- 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:
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 SERVER MULTIDIR TO DB "SAMPLE";
  SET SERVER MULTIDIR TO DB "SAMPLE2";
  SET SERVER MULTIDIR TO DB "SAMPLE3";
  SET MULTIDIR SCHEMA "SAMPLE".GRAY;
  SET MULTIDIR SCHEMA "SAMPLE2".BROWN;
  SET MULTIDIR SCHEMA "SAMPLE3".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.  
  The commands are saved to a script file, which is then invoked by using the LOAD MULTIDIR REPL SCRIPT command. |

Chapter 7. Setting up multidirectional Q replication 117
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>Use the Create Q Subscriptions wizard. To open the wizard, expand the appropriate Q Capture or Q Apply schema, right-click the <strong>Q Subscriptions</strong> folder, and select <strong>Create</strong>.</td>
</tr>
<tr>
<td><strong>Servers page</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Target Tables page</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Loading Target Table page</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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 peer-to-peer 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 peer-to-peer 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 peer-to-peer 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 peer-to-peer 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 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 17 shows the relationships between the replication programs, DB2 servers, and non-DB2 server in federated Q replication.

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.
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 18 depicts these relationships.

![Figure 18. Sources and targets in federated Q replication](image)

**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).

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 19 on page 126 shows the steps involved in setting 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.

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:
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Control Center</strong></td>
<td>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 <strong>Configure Parameters</strong>. Under the <strong>Environment</strong> heading, click <strong>Federated</strong> and click the ellipsis button to turn on federated support.</td>
</tr>
<tr>
<td><strong>UPDATE DBM CFG command</strong></td>
<td>Ensure that you are attached to the instance that contains the Q Apply server, and issue the following command: <strong>UPDATE DBM CFG USING FEDERATED YES</strong></td>
</tr>
</tbody>
</table>

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 14 on page 129 shows the location of the control tables.
**Table 14. Location of control tables for federated Q replication**

<table>
<thead>
<tr>
<th>Tables in the federated server</th>
<th>Tables in non-DB2 target server</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMQREP_APPLYENQ</td>
<td>IBMQREP_DONEMSG</td>
</tr>
<tr>
<td>IBMQREP_APPLYTRACE</td>
<td>IBMQREP_EXCEPTIONS</td>
</tr>
<tr>
<td>IBMQREP_APPLYMON</td>
<td>IBMQREP_RECVQUEUES</td>
</tr>
<tr>
<td>IBMQREP_APPLYPARMS</td>
<td>IBMQREP_SAVERI</td>
</tr>
<tr>
<td></td>
<td>IBMQREP_SPILLEDROW</td>
</tr>
<tr>
<td></td>
<td>IBMQREP_SPILLQS</td>
</tr>
<tr>
<td></td>
<td>IBMQREP_TRG_COLS</td>
</tr>
<tr>
<td></td>
<td>IBMQREP_TARGETS</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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; |

Chapter 8. Q replication to federated targets  129
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>Use the Create Q Apply Control Tables wizard. To open the wizard, right-click the <strong>Q Apply Servers</strong> folder and click <strong>Create Q Apply Control Tables</strong>.</td>
</tr>
</tbody>
</table>

#### 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 perform a load by using the EXPORT and IMPORT utilities, the target table that is referred to by the nickname must be empty.
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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 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); |
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

**Note:** The ASNCLP or Replication Center always create a new nickname for a non-DB2 target table even if one or more nicknames already exist for the table. Creating a new nickname gives the tools control over nickname alterations. This control allows the tools to resolve data type differences between DB2 and the non-DB2 target without changing an existing nickname.

**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.

**Loading the Target Table page**

Choose a load option. For automatic loading of the target table by the Q Apply program, federated Q replication supports only the EXPORT and IMPORT utilities.
Chapter 9. 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.

For automatic loading of the target table by the Q Apply program, federated Q replication supports only the EXPORT and IMPORT utilities.

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. If you specify this utility on Linux, UNIX, or Windows operating systems, you must specify a nickname for the source table, unless the source and target tables are in the same database. 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 MVS OPTIONS (REMOVE AUTHID 'AZUMA', REMOVE_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.

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

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 to create a nickname, you must create a server definition, wrapper, and user mapping.

About this task

Nicknames are required if the target tables are on Linux, UNIX, or Windows operating systems, and if the Q Capture server is remote to the Q Apply server.

When you create Q subscriptions, you can have the Replication Center create nicknames, or you can specify existing nicknames.

Procedure

To specify nicknames for unidirectional Q subscriptions:
- If you are creating a single Q subscription, use the Loading Target Tables page of the Create Q Subscriptions wizard in the Replication Center to specify an existing nickname or create a new one:
  - To specify an existing nickname, provide the name and owner of the nickname.
  - To create a new nickname, select a registered server definition on the Q Apply server and specify an owner and name for the new nickname.
- If you are creating multiple Q subscriptions:
1. In the Replication Center, use the Loading Target Tables page of the Create Q Subscriptions wizard to specify whether you want to create new nicknames or use existing nicknames. If you choose to create new nicknames, select a registered server definition on the Q Apply server.

2. Use the Q Subscriptions Properties notebook that is launched from the Review Q Subscriptions page of the Create Q Subscription wizard to select each Q subscription that requires a nickname. On the properties notebook for each Q subscription, specify the name and owner of an existing nickname, or the name and owner to be used for creating a new nickname.

To open the Create Q Subscriptions wizard, expand a Q Capture or Q Apply schema in the object tree, right-click the Q Subscriptions folder and select Create.

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.

• 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:

<table>
<thead>
<tr>
<th>Version of Q Apply program</th>
<th>Procedure</th>
</tr>
</thead>
</table>
| **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 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 20 illustrates the 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>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 &quot;Requires manual load.&quot;</td>
</tr>
<tr>
<td>SQL</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line program</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>ASNCLP SESSION SET TO Q REPLICATION; SET SERVER CAPTURE TO DB SAMPLE; SET CAPTURE SCHEMA SOURCE ASN1; LOAD DONE QSUB SUBNAME EMPLOYEE0001;</td>
</tr>
<tr>
<td>Replication Center</td>
<td>In the Manage Q Subscriptions window, select the Q subscription for the target table that you loaded and click <strong>Load done</strong>.</td>
</tr>
<tr>
<td>SQL</td>
<td>Insert a LOADDONE signal into the IBMQREP_SIGNAL table at the Q Capture server, as follows:</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>insert into schema.IBMQREP_SIGNAL(</td>
</tr>
<tr>
<td></td>
<td>SIGNAL_TIME,</td>
</tr>
<tr>
<td></td>
<td>SIGNAL_TYPE,</td>
</tr>
<tr>
<td></td>
<td>SIGNAL_SUBTYPE,</td>
</tr>
<tr>
<td></td>
<td>SIGNAL_INPUT_IN,</td>
</tr>
<tr>
<td></td>
<td>SIGNAL_STATE</td>
</tr>
<tr>
<td></td>
<td>) values (</td>
</tr>
<tr>
<td></td>
<td>CURRENT_TIMESTAMP,</td>
</tr>
<tr>
<td></td>
<td>'CMD',</td>
</tr>
<tr>
<td></td>
<td>'LOADDONE',</td>
</tr>
<tr>
<td></td>
<td>'subname',</td>
</tr>
<tr>
<td></td>
<td>'P');</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>Where <code>schema</code> identifies the Q Capture program that you want to signal, and <code>subname</code> is the name of the Q subscription for which you are performing a manual load.</td>
</tr>
</tbody>
</table>

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 142
- "Load options for peer-to-peer replication with three or more servers" on page 143

---

**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. To use these utilities, the target table must be empty.

**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.


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**

Replication of load operations that use the DB2 LOAD utility enables you to keep the source and target tables synchronized even though the individual insert, update, and deletes are not logged as they are with the IMPORT utility.

By default, when the Q Capture program reads a log record that indicates the source table was successfully loaded by DB2 LOAD it issues a warning message. When you enable replication of source table loads, Q Capture issues a warning message and then stops and starts the Q subscription after detecting the load.
operation, which prompts a new load of the target table. The replication programs follow the load options that were specified for the Q subscription.

Q Capture detects the following operations and treats them as loads:

- **z/OS**
  - LOAD RESUME YES SHRLEVEL NONE
  - LOAD REPLACE
  - REORG TABLESPACE DISCARD

- **Linux UNIX Windows**
  - LOAD REPLACE
  - LOAD INSERT

**Restrictions**

- **z/OS** The source database in DB2 for z/OS and the Q Capture program must be at APAR PK78558 or newer.
- **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.

**Procedure**

To replicate load operations at the source table, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 REPLMAP SAMPLE ASN TO TARGET ASN  
(SUBNAME EMPLOYEE0001 EMPLOYEE OPTIONS HAS LOAD PHASE I  
TARGET NAME TEMPLLOYEE LOAD TYPE 2 CAPTURE_LOAD R); |
| 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. |

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 CAPTURE_LOAD 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 full refresh of the target.
Chapter 10. Setting up event publishing

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 asnqxml 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 messages (for event publishing), but not both.

Procedure

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| ASNCLP command-line     | 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.

**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.
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 21 on page 151 shows how a single publication connects a source table to a WebSphere MQ send queue.
Figure 22 shows how multiple publications can use the same publishing queue map and Q Capture program.

Figure 21. Single publication. Changes from a source table are published over WebSphere MQ queues.

Figure 22. Multiple publications. Changes from source tables are published over WebSphere MQ queues.

Procedure
To create publications, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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**

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 a 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 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

```sql
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:

```sql
#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:

```sql
#Incorrect: Don't do this
WHERE :COL1 > 3 GROUP BY COL1, COL2
```

Valid WHERE clause with GROUP BY:

```sql
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” on page 157
• “Both changed and unchanged columns are sent” on page 157
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" on page 158

**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 11. 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 types

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:

**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.

**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.

**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
- Any column on which a VALIDPROC is defined.

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 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.

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:
\[ \text{rtrim}(\text{column}) \]

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.

Considerations for large object (LOB) data types for Q replication and event publishing

DB2 supports binary large object (BLOB), character large object (CLOB), and double-byte character large object (DBCLOB) data types.

You must have a unique index in your source table to replicate LOB data.

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 current LOB value from the source table.

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.

You can choose that LOB values are sent within the transaction message (inline or I) or in separate LOB messages (S) by specifying the lob_send_option parameter when you start the Q Capture program. Sending LOB values within the transaction message can improve performance. You might need to adjust your maximum message size to ensure that the messages are large enough to hold LOB values. The LOB_TOO_BIG_ACTION value in the IBMQREP_SENDQUEUES determines what action Q Capture takes when a LOB value does not fit into a message:

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.

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.

Recommendation: If you are replicating the DBCLOB data type, you should set lob_send_option=I to direct Q Capture to send LOB values within the transaction message, 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.

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 162
• "NUMBER" on page 163

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 15 shows a value at the source and resulting truncated value at the target.

Table 15. Truncation of TIMESTAMP(12) during replication

<table>
<thead>
<tr>
<th>Source value in TIMESTAMP(12)</th>
<th>Target value in TIMESTAMP(6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-07-10-10.33.42.458499823012</td>
<td>2009-07-10-10.33.42.458499</td>
</tr>
</tbody>
</table>

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.

**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 16 shows an example of padding.

Table 16. Padding of older TIMESTAMP value during replication

<table>
<thead>
<tr>
<th>Source value in TIMESTAMP</th>
<th>Target value in TIMESTAMP(12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-07-10-10.33.42.458499</td>
<td>2009-07-10-10.33.42.458499000000</td>
</tr>
</tbody>
</table>

**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):
If the source database does not have date compatibility enabled, the target value is padded to YYYY-MM-DD-00:00:00.

If the target database does not have date compatibility enabled, the TIMESTAMP(0) value is truncated to YYYY-MM-DD.

The NUMBER data type supports applications that use the Oracle NUMBER data type. DB2 treats NUMBER data internally as DECIMAL if no precision or scale are specified, and as DECIMAL with precision or scale if these attributes are specified.

Because Q replication already supports DECIMAL and DECIMAL, for unidirectional replication you can map columns defined with any of these three numeric types to each other: NUMBER to DECIMAL or DECIMAL, DECIMAL to NUMBER or DECIMAL, and DECIMAL to NUMBER or DECIMAL.

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 other.

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 12. 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 13. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>z/OS</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| z/OS console or TSO | 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:

```c
//* 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
```

| asncap command   | Use the asncap command to start a Q Capture program and specify startup parameters. For example:

```c
asncap 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 services | You can create a replication service on Windows operating systems to start the Q Capture program automatically when the system is started. |

| Replication Center | Use the Start Q Capture window. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to start and select **Start Q Capture Program**. You can start the program using saved parameter values from the IBMQREP_CAPPARMS control table, or specify new run-time values before starting. |
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.
- 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**.
- Use the Check Status window in the Replication Center to view the status of all Q Capture threads. To open the window, right-click the Q Capture server where the Q Capture program that you want to check is located and select **Check Status**.

---

**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 the asnqcap command and specify both the **lsn** and **maxcmtseq** parameters.
   - **Attention:** You must use both parameters in the same 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 0000:0000:0000:115b:7704 and a maxcmtseq value of
41c2:2264:0000:0004:0000, use this 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 Q Capture from the end of the log without triggering a load of the target,
specify the following values in the asnqcap command:

```
```

**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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:
```
asnmfmt 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.

**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 maxmtseq 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".

**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.
Parameters of a Q Capture program

The Q Capture operating parameters govern the way that the program starts, the amount of memory that it uses, which queue manager it connects to, and how frequently it commits messages to queues, among other things.

Default values for Q Capture operating parameters

You can change the default parameter values to suit your replication environment by updating the IBMQREP_CAPPARMS table, or by temporarily overriding the saved values when you start the Q Capture program or while the program is running.

When you create control tables using the ASNCLP or Replication Center, the IBMQREP_CAPPARMS table is created with a single row that contains default values for the Q Capture operating parameters. Table 17 shows these values.

You supply values for the adminq (administration queue), qmgr (queue manager), and restartq (restart queue) parameters when you create the control tables.

Table 17. Default values for Q Capture operating parameters

<table>
<thead>
<tr>
<th>Operating parameter</th>
<th>Default value</th>
<th>Column name in IBMQREP_CAPPARMS table</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_partition</td>
<td>N ¹</td>
<td>not applicable</td>
</tr>
<tr>
<td>adminq</td>
<td>None ²</td>
<td>ADMINQ</td>
</tr>
<tr>
<td>autostop</td>
<td>N ¹</td>
<td>AUTOSTOP</td>
</tr>
<tr>
<td>arch_level</td>
<td>0907</td>
<td>ARCH_LEVEL</td>
</tr>
<tr>
<td>arm</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>caf</td>
<td>N ¹</td>
<td>not applicable</td>
</tr>
<tr>
<td>capture_path</td>
<td>Directory where Q Capture was started ³</td>
<td>CAPTURE_PATH</td>
</tr>
<tr>
<td>capture_schema</td>
<td>ASN ⁴</td>
<td>not applicable</td>
</tr>
<tr>
<td>capture_server</td>
<td>DB2DBDFT ⁵</td>
<td>not applicable</td>
</tr>
<tr>
<td>commit_interval</td>
<td>500 ⁶</td>
<td>COMMIT_INTERVAL</td>
</tr>
<tr>
<td>compatibility</td>
<td>0907</td>
<td>COMPATIBILITY</td>
</tr>
<tr>
<td>ignore_transid</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>lob_send_option</td>
<td>I</td>
<td>LOB_SEND_OPTION</td>
</tr>
<tr>
<td>lsn</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>logrdbufsz</td>
<td>66 ¹³ for z/OS; 256 ¹³ for Linux, UNIX, and Windows</td>
<td>LOGRDBUF_SZ</td>
</tr>
<tr>
<td>logreuse</td>
<td>N ¹</td>
<td>LOGREUSE</td>
</tr>
<tr>
<td>logstdout</td>
<td>N ¹</td>
<td>LOGSTDOUT</td>
</tr>
<tr>
<td>maxcmtseq</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>memory_limit</td>
<td>0 ¹⁴ for z/OS; 500 ¹⁵ for Linux, UNIX, and Windows</td>
<td>MEMORY_LIMIT</td>
</tr>
<tr>
<td>msg_persistence</td>
<td>Y</td>
<td>MSG_PERSISTENCE</td>
</tr>
<tr>
<td>monitor_interval</td>
<td>300000 ⁶</td>
<td>MONITOR_INTERVAL</td>
</tr>
</tbody>
</table>
Table 17. Default values for Q Capture operating parameters (continued)

<table>
<thead>
<tr>
<th>Operating parameter</th>
<th>Default value</th>
<th>Column name in IBMQREP_CAPPARMS table</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor_limit</td>
<td>10080 9</td>
<td>MONITOR_LIMIT</td>
</tr>
<tr>
<td>prune_interval</td>
<td>300 8</td>
<td>PRUNE_INTERVAL</td>
</tr>
<tr>
<td>pwdfile</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>qfull_num_retries</td>
<td>30</td>
<td>QFULL_NUM_RETRIES</td>
</tr>
<tr>
<td>qfull_retry_delay</td>
<td>250 13</td>
<td>QFULL_RETRY_DELAY</td>
</tr>
<tr>
<td>qmgr</td>
<td>None 2</td>
<td>QMGR</td>
</tr>
<tr>
<td>restartq</td>
<td>None 2</td>
<td>RESTARTQ</td>
</tr>
<tr>
<td>signal_limit</td>
<td>10080 9</td>
<td>SIGNAL_LIMIT</td>
</tr>
<tr>
<td>sleep_interval</td>
<td>5000 6</td>
<td>SLEEP_INTERVAL</td>
</tr>
<tr>
<td>startmode</td>
<td>warmsi 10</td>
<td>STARTMODE</td>
</tr>
<tr>
<td>term</td>
<td>Y 11</td>
<td>TERM</td>
</tr>
<tr>
<td>trace_limit</td>
<td>10080 9</td>
<td>TRACE_LIMIT</td>
</tr>
</tbody>
</table>

Note:
1. No.
2. You must specify this value when you create the Q Capture control tables. This is not a command parameter. The Q Capture program reads this value from the IBMQREP_CAPPARMS table.
3. If Q Capture is started using JCL on z/OS, the default is the user ID associated with the started task or job. If Q Capture starts as a Windows service, the capture_path is \SQLLIB\bin.
4. You cannot change the default schema. To use a Q Capture program with a different schema, you specify the capture_schema start parameter.
5. For z/OS, there is no default Q Capture server. For Linux, UNIX, and Windows, the Q Capture program server is the value of the DB2DBDFT environment variable, if specified.
7. Megabytes.
8. Seconds.
9. Minutes.
10. The Q Capture program warm starts. It switches to cold start only if this is the first time that the program is starting.
11. Yes.
12. The default delay is 250 milliseconds or the value of the commit_interval parameter, whichever is less.
13. Kilobytes
14. When memory_limit is set to 0 on z/OS, Q Capture calculates a memory allocation that is based on the Q Capture region size in the JCL or started task.

Descriptions of Q Capture parameters

The Q Capture program reads parameters from the IBMQREP_CAPPARMS table when you start the program. You can specify temporary run-time values for some parameters when you start the program, and also while the program is running.
The following sections describe the Q Capture program’s operating parameters and discuss reasons that you might want to change the default values based on your needs.

- “add_partition (DB2 for Linux, UNIX, Windows)”
- “adminq” on page 177
- “arm” on page 177
- “autostop” on page 177
- “caf” on page 178
- “capture_path” on page 178
- “capture_schema” on page 178
- “capture_server” on page 179
- “commit_interval” on page 179
- “ignore_transid” on page 181
- “lob_send_option” on page 181
- “logrdbufsz” on page 182
- “logreuse” on page 182
- “logstdout” on page 183
- “lsn” on page 183
- “maxcommitseq” on page 183
- “memory_limit” on page 184
- “monitor_interval” on page 185
- “msg_persistence” on page 185
- “prune_interval” on page 186
- “qfull_num_retries” on page 187
- “qfull_retry_delay” on page 187
- “qmgr” on page 188
- “restartq” on page 188
- “signal_limit” on page 188
- “sleep_interval” on page 189
- “startmode” on page 190
- “term” on page 190
- “trace_limit” on page 191

**add_partition (DB2 for Linux, UNIX, Windows)**

Default: add_partition=N

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.
The *adminq* parameter defines the Q Capture administration queue. A Q Capture program uses this queue to receive control messages from the Q Apply program, a user application, or a WebSphere MQ message channel agent. It is a local queue that is defined on the same system where a Q Capture program runs. You must supply the name of the administration queue when you create the Q Capture control tables, and a Q Capture program must be able to connect to this queue or it does not run.

You specify the name of an administration queue when you create the Q Capture control tables. The value is saved in the IBMQREP_CAPPARMS control table. You can change the value by updating this table. You cannot alter the value when you start a Q Capture program, or while the program is running.

The Q Apply program always sends persistent messages to the administration queue, regardless of whether the queue is created as persistent or not and regardless of the setting for the Q Capture *message_persistence* parameter.

**arm**

**Default:** None

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: ASNQCxxxxyyyy (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`

The *autostop* parameter controls whether a Q Capture program terminates when it reaches the end of the active DB2 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.

You can set the *autostop* parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.
The Q Capture program runs with the default of Recoverable Resource Manager Services (RRS) connect (café=N). You can override this default and prompt the Q Capture program to use the Call Attach Facility (CAF) by specifying the café=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

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.

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 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).

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.

You can set the capture_path parameter when you start a Q Capture program, or you can change the saved value of the parameter in the IBMQREP_CAPPARMS table. You cannot alter this parameter while a Q Capture program is running.

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**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>Default: None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux UNIX Windows</td>
<td>Default: <code>capture_server=value of DB2DBDFT environment variable, if it is set</code></td>
</tr>
</tbody>
</table>

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.

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)` |

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 (check the TRANS_PUBLISHED value in the IBMQREP_CAPQMON table), you might want to have the Q Capture program commit fewer transactions at a time to WebSphere MQ (see “Procedure for determining transactions published per commit interval”).

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.

You can set the commit_interval parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**Procedure for determining transactions published per commit interval**

To calculate the number of transactions published per commit interval, use the following formula.

\[
\text{TRANS\_PUBLISHED} / (\text{MONITOR\_INTERVAL} / \text{COMMIT\_INTERVAL})
\]

Follow these steps:
1. Determine the monitor interval (the number of seconds between row inserts into the IBMQREP_CAPQMON table).
2. Divide this number by the commit interval.
   
   You can use the following windows in the Replication Center to view the commit interval and monitor interval:
   
   • Change Parameters Running Q Capture Program (if the Q Capture program is running).
   • Change Parameters Saved (if the Q Capture program is stopped).

   To open the windows, right-click the Q Capture server that contains the Q Capture program that you want to check, and select the appropriate menu item. You might need to convert one or both values to seconds for this step.

   Save the result because you will use this result in Step 3.

3. Decide if you want to see transactions published per commit interval for one send queue, or for all send queues that are used by this Q Capture program:
   
   • If you want to calculate the number for one send queue, find the latest value for TRANS_PUBLISHED for that queue in the IBMQREP_CAPQMON table.
   • If you want to calculate the number for this Q Capture program, sum the TRANS_PUBLISHED numbers for all send queues that are used by this Q Capture program.

   You can use the Q Capture Throughput window in the Replication Center to check this value for one or all send queues. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to check, and select Reports → Q Capture Throughput.
4. Divide TRANS_PUBLISHED by the result from Step 2. The result shows you how many transactions a Q Capture program published for each commit interval.

**ignore_transid**

Default: None

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:

```
<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:xxxx:xxxx:xxxx:mmmm</td>
<td>z/OS</td>
</tr>
</tbody>
</table>
```

Where `xxxx:xxxx:xxxx:mmmm` 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.

```
<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnnn:0000:xxxx:xxxx</td>
<td>Linux/UNIX/Windows</td>
</tr>
</tbody>
</table>
```

Where `xxxx:xxxx:xxxx:nnnn` 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.

**lob_send_option**

Default: `lob_send_option=I`

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.

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.

**logdbufsz**

Default: `logdbufsz=66K` for z/OS; 256K for Linux, UNIX, and Windows.

The `logdbufsz` parameter specifies the size of the buffer 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 `logdbufsz` 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.

Decreasing the buffer size might increase performance when a data partition is used with a source table on DB2 for Linux, UNIX, or Windows.

**logreuse**

Default: `logreuse=n`

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.

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.
The log file name is
\texttt{db2instance.capture_server.capture_schema.QCAP.log}. For example,
\texttt{DB2.SAMPLE.ASN.QCAP.log}.

You can set the \texttt{logreuse} parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**logstdout**

Default: \texttt{logstdout=n}

By default, a Q Capture program writes its work history only to the log. You can change the \texttt{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.

You can set the \texttt{logstdout} parameter when you start a Q Capture program with the \texttt{asnqcap} command, or while the program is running by using the \texttt{asnqccmd} command. You can also change the saved value of the parameter by updating the IBMQREP_CAPPARMS table. If you use the Replication Center to start the Q Capture program, this parameter is not applicable.

**lsn**

Default: None

The \texttt{lsn} parameter specifies the log sequence number at which the Q Capture program starts during a warm restart. You specify both the \texttt{lsn} and \texttt{maxcmtseq} parameters to start Q Capture from a known point in the DB2 log. When specifying the \texttt{lsn} parameter, you also must specify the \texttt{maxcmtseq} parameter in the same command invocation, and you cannot use these parameters if the value of \texttt{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 \texttt{lsn} from the restart message by using the \texttt{asnqmfmt} command. You can also use the value in the \texttt{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 the following values: \texttt{lsn=FFFF:FFFF:FFFF:FFFF:FFFF} and \texttt{maxcmtseq=0000:0000:0000:0000:0000}.

You can also specify \texttt{lsn} and \texttt{maxcmtseq} without colons to save space.

**maxcmtseq**

Default: None
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 `asqfmt` 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 the following values: `lsn=FFFF:FFFF:FFFF:FFFF` and `maxcmtseq=0000:0000:0000:0000:0000`.

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

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.

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.

You can set the memory_limit parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**msg_persistence**

Default: msg_persistence=y

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.

**monitor_interval**

Default: monitor_interval=60000 milliseconds (1 minute)

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 60000 milliseconds (1 minute). 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.

You can set the `monitor_interval` parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**monitor_limit**

Default: `monitor_limit`=10080 minutes (7 days)

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.

You can set the `monitor_limit` parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**prune_interval**

Default: `prune_interval`=300 seconds (5 minutes)

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.

You can set the prune_interval parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

pwdfile

Default: None

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.

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. 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

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

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.) 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):

```
asncap capture_server=SAMPLE capture_schema=ASN1 qfull_num_retries=50
qfull_retry_delay=10000
```

### qmgr

**Default:** None

The *qmgr* parameter specifies the name of a WebSphere MQ queue manager that a
Q Capture program works with. The queue manager manages queues and
messages for the Q Capture program.

The queue manager owns the queues that a Q Capture program uses to send data
messages and informational messages, and to receive control messages. All
communication between Q replication and event publishing programs and
WebSphere MQ goes through queue managers.

You specify the name of a queue manager when you create the Q Capture control
tables. The value is saved in the IBMQREP_CAPPARMS control table. You can
change the value by updating this table. You cannot alter the value when you start
a Q Capture program, or while the program is running.

### restartq

**Default:** None

The *restartq* parameter specifies the restart queue that you want the Q Capture
program to use. The restart queue contains a single message that tells a Q Capture
program where to start reading in the DB2 recovery log for each send queue after
the Q Capture program restarts. It is a local queue that is defined on the same
system where the Q Capture program runs. You must supply the name of this
queue when you create the Q Capture control tables, and a Q Capture program
must be able to connect to this queue to run.

You specify the name of a restart queue when you create the Q Capture control
tables. The value is saved in the IBMQREP_CAPPARMS control table. You can
change the value by updating this table. You cannot alter the value when you start
a Q Capture program, or while the program is running.

The Q Capture program sends persistent messages to the restart queue, regardless
of whether the queue is created as persistent or not and regardless of the setting
for the *message_persistence* parameter.

### signal_limit

**Default:** *signal_limit*=10080 minutes (7 days)

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 that 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.**
You can set the signal_limit parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**sleep_interval**

Default: sleep_interval=5000 milliseconds (5 seconds)

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.

You can set the sleep_interval parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.
### startmode

**Default**: startmode=warmsi

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 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. You cannot use a cold start to force a full refresh (a new load) of targets. The only way to force a full refresh is to deactivate and then activate a Q subscription or publication that has a load phase.

  **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.

  You can set the `startmode` parameter when you start a Q Capture program, or you can change the saved value of the parameter in the IBMQREP_CAPPARMS table. You cannot alter this parameter while a Q Capture program is running.

### term

**Default**: term=Y

The `term` parameter controls whether a Q Capture program keeps running when the source DB2 is unavailable (quiesced, stopped, etc.).

By default, a Q Capture program quits running when DB2 is unavailable. You can change the default if you want a Q Capture program to keep running while DB2 is unavailable.
unavailable. In this case, when DB2 is available, the Q Capture program goes back to capturing changes from the last restart point in the log without requiring you to restart the program.

You can set the term parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

**trace_limit**

Default: \texttt{trace\_limit}=10080 minutes (7 days)

The \texttt{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.

You can set the \texttt{trace\_limit} parameter when you start a Q Capture program or while the program is running. You can also change the saved value of the parameter in the IBMQREP_CAPPARMS table.

---

**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 program’s operating parameters 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 will use the parameter values that are saved in the IBMQREP_CAPPARMS table. You can update this table using the Change Parameters – Saved window in the Replication Center, or by using 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 the Start Q Capture window in the Replication Center or the asnqcap system command to set values for the operating parameters. Your changes will take effect when the program starts, but will last only while the program is running.

**Dynamically changing parameters while a Q Capture program is running**

You can dynamically change a Q Capture program’s parameter values without needing to stop capturing changes from the source. Use the
Three ways to change Q Capture operating parameters

Assume that you want to shorten the default setting for the commit interval of 5000 milliseconds (five seconds) for a Q Capture program identified by schema ASN1:

1. Update the IBMQREP_CAPPARMS table for Q Capture schema ASN1. Set the commit interval to 1000 milliseconds (one second). You can use the Change Parameters – Saved window in the Replication Center, or the following SQL:

   ```sql
   update asn1.ibmqrep_capparms set commit_interval=1000
   ```

   When you start this Q Capture program in the future, the commit interval will default to 1000 milliseconds.

2. You want to see the effect of an even 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 the commit interval set to 2000 milliseconds (2 seconds). You can use the Start Q Capture window in the Replication Center, or the asnqcap command:

   ```bash
   asnqcap capture_server=srcdb1 capture_schema="ASN1" commit_interval=2000
   ```

   While the program runs using a 2-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 1500 milliseconds (1.5 seconds), and monitor the change. You can 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, or use the Change Parameters – Running Q Capture Program window in the Replication Center:

   ```bash
   f myqcap,chgparms commit_interval=1500
   asnqccmd capture_server=srcdb1 capture_schema="ASN1" chgparms
       commit_interval=1500
   ```

   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:
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 while the Q Capture program is running. To change the value you must first stop the Q Capture program.

When you change the values, the effects might not be immediate. A delay of 1 to 2 seconds can occur between the time that you or the Replication Center issues the `asnqccmd chgparms` command and the time that a Q Capture program changes its operation.

Procedure

To dynamically change parameters while a Q Capture program is running, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chgparms parameter</td>
<td>Use this parameter with the MODIFY command on z/OS or asnqccmd command on Linux, UNIX, Windows, and UNIX System Services for z/OS to change parameters for a running Q Capture program:</td>
</tr>
<tr>
<td></td>
<td><strong>MODIFY</strong></td>
</tr>
<tr>
<td></td>
<td>myqcap,chgparms parameter=value</td>
</tr>
<tr>
<td></td>
<td>Where <code>myqcap</code> is the Q Capture job name.</td>
</tr>
<tr>
<td></td>
<td><strong>asnqccmd</strong></td>
</tr>
<tr>
<td></td>
<td>asnqccmd capture_server=server</td>
</tr>
<tr>
<td></td>
<td>capture_schema=schema</td>
</tr>
<tr>
<td></td>
<td>chgparms parameters</td>
</tr>
<tr>
<td></td>
<td>Where <code>server</code> is the name of the Q Capture server, <code>schema</code> identifies a running Q Capture program, and <code>parameters</code> is one or more parameters that you want to change.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>Use the Change Parameters – Running Q Capture Program window. To open the window, right-click the Q Capture server that contains the Q Capture program whose parameters you want to change and select Change Parameters → Running Q Capture Program.</td>
</tr>
<tr>
<td>Replication Dashboard</td>
<td>On the Programs tab, click Properties and then select a Q Capture server from the Program field to view or change saved values.</td>
</tr>
</tbody>
</table>
| 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.ibmrep_capparms  
 set monitor_interval=600, logstdout=Y  

Where schema identifies the Q Capture program whose saved parameter values you want to change. |
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop parameter</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>MODIFY</strong></td>
</tr>
<tr>
<td></td>
<td>f myqcap,stop</td>
</tr>
<tr>
<td></td>
<td>Where myqcap is the Q Capture job name.</td>
</tr>
<tr>
<td></td>
<td><strong>asnqccmd</strong></td>
</tr>
<tr>
<td></td>
<td>asnqccmd capture_server=server_name  capture_schema=schema stop</td>
</tr>
<tr>
<td></td>
<td>Where <em>server_name</em> is the name of the database or subsystem where the Q Capture program is running, and <em>schema</em> identifies the Q Capture program that you want to stop.</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Stop Q Capture window. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to stop and select <strong>Stop Q Capture Program</strong>.</td>
</tr>
</tbody>
</table>
Starting Q subscriptions

By default, newly created Q subscriptions 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 Q subscriptions after you create them.

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 Q subscriptions, the program will process the signal or message 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, a password file must be created on the Q Apply server to allow the utility to connect to the Q Capture server.
- 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.

About this task

When you start Q subscriptions, the Q Capture program starts capturing source changes and putting the change messages on WebSphere MQ queues.

Procedure

### Method | Description
--- | ---
SQL | From 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:

```sql
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 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.
To start Q subscriptions, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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; |
| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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; |
| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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” on page 201
- “Setting the error action for send queues to Q” on page 201
- “Keeping disabled queues inactive when Q Capture starts” on page 202
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 23* 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.

**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 `Isn` and `maxcmtnseq` 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 171.

You can format the newer restart message by using the `asnmfmt` or `asqnxmlfmt`
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)  | 414d5120535a50515445535420202048c1cd6620004e02 |
| CorrelId| 51505552214d41494e4494e4500000000000000000000000000 |
| 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.qRestartMsgSize : 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: 0000000000000000000000000000000003 |
| sendq lastCommitSEQ: 0000:0000:0000:0000:0000 |
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startq command</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>f myqcap,startq=asn.qm1.dataq</td>
</tr>
<tr>
<td></td>
<td>f myqcap,startq=all</td>
</tr>
<tr>
<td></td>
<td>The following commands use asnqccmd for the same actions:</td>
</tr>
<tr>
<td></td>
<td>asnqccmd capture_server=sourcedb capture_schema=&quot;asn&quot; startq=&quot;asn.qm1.dataq&quot;</td>
</tr>
<tr>
<td></td>
<td>asnqccmd capture_server=sourcedb capture_schema=&quot;asn&quot; startq=all</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Reinitializing the Q Capture program</td>
<td>The Q Capture reinit command starts inactive send queues if the Q Capture program was started with the <code>startallq=y</code> option (the default).</td>
</tr>
</tbody>
</table>
| 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:  
```sql
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopq command</td>
<td>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.</td>
</tr>
<tr>
<td>SQL</td>
<td>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:</td>
</tr>
</tbody>
</table>

```sql
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 14. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS console or TSO</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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 (/).</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>/// asterisk indicates a comment line</td>
</tr>
<tr>
<td></td>
<td>// QAPP EXEC PGM=ASNQAPP,PARMS='LE/Q Apply parameters'</td>
</tr>
<tr>
<td></td>
<td>/// Parameters can be any or no LE parameters and any or no Q Apply parameters</td>
</tr>
<tr>
<td></td>
<td>/// no Q Apply parameters</td>
</tr>
<tr>
<td></td>
<td>//SYSIN DD *</td>
</tr>
<tr>
<td></td>
<td>/// additional Q Apply parameters, one or more</td>
</tr>
<tr>
<td></td>
<td>/// parameters on each line</td>
</tr>
<tr>
<td></td>
<td>APPLY_SERVER=DSN!! APPLY_SCHEMA=APPCAT</td>
</tr>
<tr>
<td></td>
<td>DEBUG=Y LOGSTDOUT=N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>asnqapp command</th>
<th>From a command line, use the asnqapp command to start a Q Apply program and optionally specify startup parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>asnqapp apply_server=server_name</td>
</tr>
<tr>
<td></td>
<td>apply_schema=schema</td>
</tr>
<tr>
<td></td>
<td>parameters</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Windows</th>
<th>You can create a replication service on the Windows operating system to start the Q Apply program automatically when the system starts.</th>
</tr>
</thead>
</table>

| Replication Center | Use the Start Q Apply window. To open the window, right-click the Q Apply server that contains the Q Apply program that you want to start and select Start Q Apply Program. You can start the program using saved parameter values from the IBMQREP_APPLYPARMS control table, or specify new run time values before starting. |

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.
- 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.

- Use the Check Status window in the Replication Center to view the status of all Q Apply threads. To open the window, right-click the Q Apply server where the Q Apply program that you want to check is located and select Check Status.

### Parameters of a Q Apply program

A Q Apply program’s operating parameters specify which queue manager it connects to, how frequently it prunes its control tables, and whether it stops when it has no work to do, among other things.

### Default values for Q Apply operating parameters

When you create control tables, the IBMQREP_APPLYPARMS table is created with a single row that contains default values for the Q Apply program’s operating parameters.

Table 18 shows the default values for the Q Apply program operating parameters.

You can change the default parameter values to suit your replication environment by updating the IBMQREP_APPLYPARMS table, or by temporarily overriding the saved values when you start the Q Apply program or while the program is running.

**Note:** You supply a value for the `qmgr` (queue manager) parameter when you create the control tables.

**Table 18. Default values for Q Apply operating parameters**

<table>
<thead>
<tr>
<th>Operating parameter</th>
<th>Default value</th>
<th>Column name in IBMQREP_APPLYPARMS table</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_path</td>
<td>Directory where Q Apply was started¹</td>
<td>APPLY_PATH</td>
</tr>
<tr>
<td>apply_schema</td>
<td>ASN²</td>
<td>not applicable</td>
</tr>
<tr>
<td>apply_server</td>
<td>DB2DBDFT³</td>
<td>not applicable</td>
</tr>
<tr>
<td>arch_level</td>
<td>0907</td>
<td>ARCH_LEVEL</td>
</tr>
<tr>
<td>arm</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>applyupto</td>
<td>None⁴</td>
<td>not applicable</td>
</tr>
<tr>
<td>autostop</td>
<td>N ⁵</td>
<td>AUTOSTOP</td>
</tr>
<tr>
<td>caf</td>
<td>N⁵</td>
<td>not applicable</td>
</tr>
<tr>
<td>classic_load_file_sz</td>
<td>500,000</td>
<td>CLASSIC_LOAD_FILE_SZ</td>
</tr>
<tr>
<td>commit_count</td>
<td>1</td>
<td>COMMIT_COUNT</td>
</tr>
<tr>
<td>deadlock_retries</td>
<td>3</td>
<td>DEADLOCK_RETRIES</td>
</tr>
<tr>
<td>dftmodelq</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>ignbaddata</td>
<td>N⁵</td>
<td>not applicable</td>
</tr>
<tr>
<td>insert_bidi_signal</td>
<td>Y ¹⁰</td>
<td>INSERT_BIDI_SIGNAL</td>
</tr>
</tbody>
</table>
Table 18. Default values for Q Apply operating parameters (continued)

<table>
<thead>
<tr>
<th>Operating parameter</th>
<th>Default value</th>
<th>Column name in IBMQREP_APPLYPARMS table</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadcopy_path</td>
<td>Null</td>
<td>LOADCOPY_PATH</td>
</tr>
<tr>
<td>load_data_buff_sz</td>
<td>8</td>
<td>LOAD_DATA_BUFF_SZ</td>
</tr>
<tr>
<td>logreuse</td>
<td>N(^5)</td>
<td>LOGREUSE</td>
</tr>
<tr>
<td>logstdout</td>
<td>N(^5)</td>
<td>LOGSTDOUT</td>
</tr>
<tr>
<td>max_parallel_loads</td>
<td>1 (z/OS); 15 (Linux, UNIX, Windows)</td>
<td>MAX_PARALLEL_LOADS</td>
</tr>
<tr>
<td>monitor_interval</td>
<td>30000(^6)</td>
<td>MONITOR_INTERVAL</td>
</tr>
<tr>
<td>monitor_limit</td>
<td>10080(^8)</td>
<td>MONITOR_LIMIT</td>
</tr>
<tr>
<td>nickname_commit_ct</td>
<td>10</td>
<td>NICKNAME_COMMIT_CT</td>
</tr>
<tr>
<td>p2p_2nodes</td>
<td>N(^5)</td>
<td>not applicable</td>
</tr>
<tr>
<td>prune_interval</td>
<td>300(^7)</td>
<td>PRUNE_INTERVAL</td>
</tr>
<tr>
<td>pwdfile</td>
<td>asnpwd.aut</td>
<td>PWDFILE</td>
</tr>
<tr>
<td>qmgr</td>
<td>None(^9)</td>
<td>QMGR</td>
</tr>
<tr>
<td>richklvl</td>
<td>2</td>
<td>not applicable</td>
</tr>
<tr>
<td>skiptrans</td>
<td>None</td>
<td>not applicable</td>
</tr>
<tr>
<td>spill_commit_count</td>
<td>10</td>
<td>SPILL_COMMIT_COUNT</td>
</tr>
<tr>
<td>sql_cap_schema</td>
<td>None</td>
<td>SQL_CAP_SCHEMA</td>
</tr>
<tr>
<td>term</td>
<td>Y(^{10})</td>
<td>TERM</td>
</tr>
<tr>
<td>trace_limit</td>
<td>10080(^8)</td>
<td>TRACE_LIMIT</td>
</tr>
</tbody>
</table>

**Note:**

1. If Q Apply is started using JCL on z/OS, the default is the user ID associated with the started task or job. If Q Apply starts as a Windows service, the `apply_path` is `\SQLLIB\bin`.

2. You cannot change the default schema. To use a Q Apply program with a different schema, you specify the `apply_schema` parameter when you start a Q Apply program.

3. For z/OS, there is no default Q Apply server. For Linux, UNIX, and Windows, the default Q Apply server is the value of the DB2DBDFT environment variable, if specified.

4. If you specify a timestamp for the `applyupto` parameter, the Q Apply program stops when it receives a transaction with a source timestamp that is later than the specified timestamp. This timestamp must be specified as a full or partial timestamp in Greenwich mean time (GMT).

5. No.


7. Seconds.

8. Minutes.

9. You must specify this value when you create the Q Apply control tables. This is not a command parameter. The Q Apply program reads this value from the IBMQREP_APPLYPARMS table.

10. Yes.

---

**Descriptions of Q Apply parameters**

The Q Apply program reads parameters from the IBMQREP_APPLYPARMS table when you start the program. You can specify temporary run time values for some
parameters when you start the program, and also while the program is running. Restart the Q Apply program to activate any changes that you make to the IBMQREP_APPLYPARMS table.

- **autostop**
- **apply_path** on page 212
- **apply_schema** on page 212
- **apply_server** on page 213
- **applyupto** on page 213
- **arm** on page 214
- **buffered_inserts** on page 214
- **caf** on page 215
- **commit_count** on page 215
- **deadlock_retries** on page 215
- **dftmodelq** on page 216
- **ignbaddata** on page 216
- **insert_bidi_signal** on page 217
- **loadcopy_path** on page 217
- **load_data_buff_sz** on page 217
- **logstdout** on page 218
- **max_parallel_loads** on page 218
- **monitor_interval** on page 219
- **monitor_limit** on page 219
- **nickname_commit_ct** on page 220
- **p2p_2nodes** on page 220
- **pwdfile** on page 220
- **prune_interval** on page 221
- **qmgr** on page 221
- **richklvl** on page 222
- **skiptrans** on page 222
- **spill_commit_count** on page 222
- **term** on page 223
- **trace_limit** on page 223

### autostop

**Default**: autostop=N

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.

You can set the `autostop` 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.

**Restriction:** You cannot specify both the `autostop` parameter and the `applyupto` parameter.

### apply_path

**Default:** None

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 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).

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.

### 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.

On z/OS, no special characters are allowed in the Q Apply schema except for the underscore (_).

**apply_server**

By default: None

**applyupto**

By default: None

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 the specified 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

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: `ASNQAxxxxyyyy` (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.

---

**buffered_inserts**

**Default:** buffered_inserts=N

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

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

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

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.

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`. However, you can raise the value.

You can set the `deadlock_retries` parameter when you start a Q Apply program or while the program is running. You can also change its saved value in the `IBMQREP_APPLYPARMS` table.

**dfmodelq**

Default: None

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 `dfmodelq` parameter. The following list summarizes the behavior of the parameter:

**If you specify dfmodelq 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 `dfmodelq` 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 `dfmodelq` parameter.

**If you do not specify dfmodelq 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

**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`

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 IBMQREPIGNTRAN 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 IBMQREPIGNTRAN 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 IBMQREPIGNTRANTRC table for each transaction that it does not recapture.

**loadcopy_path**

Default: `loadcopy_path=Value of apply_path parameter`

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`

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.

**logreuse**

Default: logreuse=N

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.

You can set the logreuse parameter when you start a Q Apply program or while the program is running. You can also change its saved value in the IBMQREP_APPLYPARMS table.

**logstdout**

Default: logstdout=N

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 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.

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. You can also change the value of the logstdout parameter while the Q Apply program is running by using the asnqacmd command. You can change the saved value in the IBMQREP_APPLYPARMS table by using SQL.

**max_parallel_loads**

Default: max_parallel_loads=1 (z/OS); 15 (Linux, UNIX, Windows)
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)

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.

By default, rows are inserted into this table every 60000 milliseconds (1 minute). 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.

You can set the `monitor_interval` 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.

**monitor_limit**

Default: `monitor_limit`=10080 minutes (7 days)

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

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, which must use the EXPORT and IMPORT utilities.

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

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 (Default=N) 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.

pwdfile

Default: None
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.

**prune_interval**

Default: `prune_interval`=300 seconds (5 minutes)

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.

You can set the **prune_interval** 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.

**qmgr**

Default: None

The **qmgr** parameter specifies the name of a WebSphere MQ queue manager that a Q Apply program works with. The queue manager’s job is to manage queues and
messages for the Q Apply program. It must run on the same system as the Q Apply program. Although client connections to queue managers are supported by WebSphere MQ, this configuration is not supported for Q Apply.

The queue manager owns the queues that a Q Apply program uses to receive data and informational messages and send control messages. All communication between Q replication programs and WebSphere MQ goes through queue managers.

You specify the name of a queue manager when you create the Q Apply control tables. The value is saved in the IBMQREP_APPLYPARMS control table. You can change the value by updating this table. You cannot alter the value when you start the Q Apply program, or while the program is running.

richklvl

Default: richklvl=2

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

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

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.

**term**

Default: term=Y

The term parameter controls whether a Q Apply program keeps running when DB2 is unavailable (quiesced, HADR failover, stopped, etc.).

By default (term=Y), the Q Apply program terminates when DB2 is unavailable. You can change the default if you want a Q Apply program to keep running while DB2 is unavailable. In this case, when DB2 is available, the Q Apply program begins applying transactions where it left off without requiring you to restart the program.

You can set the term 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.

**trace_limit**

Default: trace_limit=10080 minutes (7 days)

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.

You can set the trace_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.

---

**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 191. 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`

A delay of 1 to 2 seconds can occur between the time that you or the Replication Center issues the `asnqacmd chgparms` command and the time that a Q Apply program changes its operation.

Procedure

To dynamically change parameters while a Q Apply program is running, use one of the following methods:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| chgparms parameter | Use this parameter with the MODIFY command on z/OS or `asnqacmd` command on Linux, UNIX, Windows, and UNIX System Services for z/OS to change parameters for a running Q Apply program:  
  **MODIFY**  
  f myqapp,chgparms parameter=value  
  Where myqapp is the Q Apply job name.  
  **asnqacmd**  
  `asnqacmd` 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. |
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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>Use the Change Parameters – Running Q Apply Program window. To open the window, right-click the Q Apply server that contains the Q Apply program whose parameters you want to change and select Change Parameters → Running Q Apply Program.</td>
</tr>
<tr>
<td>Replication Dashboard</td>
<td>On the Programs tab, click Properties and then select a Q Apply server from the Program field to view or change saved values.</td>
</tr>
<tr>
<td>SQL</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
| Replication Center | Use the Stop Q Apply window. To open the window, right-click the Q Apply server that contains the Q Apply program that you want to stop and select Stop Q Apply Program. |
| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopq parameter</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>MODIFY f myqapp,stopq</td>
</tr>
<tr>
<td></td>
<td>Where myqapp is the Q Apply job name.</td>
</tr>
<tr>
<td>asnqacmd</td>
<td>asnqacmd apply_server=server_name apply_schema=schema stopq</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Replication Center</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

  ```sql
  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 shut down or to stop reading from the receive queue.
• You instructed the Q Apply program to stop processing messages from the receive queue.

Procedure

To start message processing on a receive queue:

Use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startq parameter</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>MODIFY f myqapp,startq</td>
</tr>
<tr>
<td></td>
<td>Where myqapp is the Q Apply job name.</td>
</tr>
<tr>
<td></td>
<td>asnqacmd apply_server=server_name apply_schema=schema startq</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Replication Center</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Situation</th>
<th>Method</th>
</tr>
</thead>
</table>
| 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=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:  
  `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**  
  `mmmm:0000:xxxx:xxxx:xxxx`  
  Where `xxxx:xxxx:xxxx:xxxx` 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;0000:0000:0000:51b0-0000:0000:0000:51g0"  ```  

Chapter 14. Operating a Q Apply program 231
**Situation** | **Method**
--- | ---
While starting a receive queue | When you start a receive queue with the `startq` parameter of the `asnqcacmd` 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 `asnqcacmd` is as follows:

```
asnqcacmd 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:

```
asnqcacmd 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 15. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>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 <strong>Manage + Q Subscriptions</strong>.</td>
</tr>
</tbody>
</table>
| SQL                   | Insert a REINIT_SUB signal into the IBMQREP_SIGNAL table at the Q Capture server:  
```sql
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:  
```sql
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 columns to replicate (unidirectional replication)

You can specify additional columns to replicate from the source table while the replication programs are running. The programs update the Q subscription that specifies the source table and begin replicating the new columns automatically.

**Before you begin**

- The Q subscription 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.
- If you add columns with default values and the conflict rule for the Q subscription is A (check all columns), you must run the REORG utility on the source table before you begin replicating the new column. For more detail, see the Technote, How do I get around CHECKFAILED errors reported in IBMOREP_EXCEPTIONS table, after I add a column to a target DB2 for z/OS table?
- You cannot add more than 20 columns during a single WebSphere MQ commit interval, which is set by the Q Capture COMMIT_INTERVAL parameter.
- **Federated targets:** If the nickname for the target table has columns that are not part of the existing Q subscription, you can use the ADDCOL signal to add them to the Q subscription. You must drop the Q subscription and re-create it after you alter the target table (you cannot add columns to a nickname).

**About this task**

You use an SQL signal to add a column to a Q subscription, either by using the ASNCLP command-line program or by performing an SQL insert.

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. For unidirectional replication, the Q Capture program can be stopped when you insert the signals and it will read them from the log when it restarts.

In the ASNCLP CREATE QSUB command, you can add columns to all Q subscriptions for a source table by specifying the SOURCE option instead of the QSUB option.

**Recommendation:** Let the replication programs automatically add new columns to the target table to ensure that they match the columns at the source. Columns will be added to the target table with the same name, data type, null characteristic, and default value as the matching columns in the source table.

**Tip:** If you plan to quiesce the source database or instance before adding columns, you can set the term parameter for the Q Capture program to N (no) by using the chgparms parameter with the MODIFY or asnmqcmd commands or the Replication Center. 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 begins capturing changes from the last restart point in the log without requiring you to restart the program.

Procedure

To add columns to a unidirectional Q subscription, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line program.</td>
<td>Use the ALTER ADD COLUMN command. For example, the following command adds the column ADDRESS to the \ EMPLOYEE0001 Q subscription: ALTER ADD COLUMN USING SIGNAL (BONUS) QSUB DEPARTMENT0001 USING REQPMAP SAMPLE_ASN_TO_TARGET_ASN;</td>
</tr>
<tr>
<td>SQL</td>
<td>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 that is processing the Q subscription that you are adding a column to. subname;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.</td>
</tr>
</tbody>
</table>

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 columns to replicate (bidirectional or peer-to-peer replication)

After you add a column to a table that is involved in bidirectional or peer-to-peer replication, you can update the Q subscriptions that specify the table so that the column is replicated at all servers.

Before you begin

- The Q subscriptions that specify the table must be in the A (active) state at all servers.
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 and the conflict rule for the Q subscription is A (check all columns), you must run the REORG utility on the source table before you begin replicating the new column. For more detail, see the Technote How do I get around CHECKFAILED errors reported in IBMQREP_EXCEPTIONS table, after I add a column to a target DB2 for z/OS table?
- 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 during a single WebSphere MQ commit interval, which is set by the Q Capture COMMIT_INTERVAL parameter.

About this task

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 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:

   ```sql
   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.
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.


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.

**Important:** If you manually add columns to the logical table at both servers (bidirectional replication) or all servers (peer-to-peer replication), you need to insert an ADDCOL signal into the IBMQREP_SIGNAL tables at each server.

### How Q Capture handles alterations of source tables

When you alter a 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 Data Definition Language (DDL) operations and what you need to do for any affected Q subscriptions.

<table>
<thead>
<tr>
<th>DDL operation</th>
<th>How it is handled</th>
<th>What you need to do</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER TABLE ADD (COLUMN)</td>
<td>Q Capture leaves the Q subscription active, but does not replicate the added column until it receives an ADDCOL signal.</td>
<td>Use the ASNCLP ALTER ADD COLUMN command, or manually insert an ADDCOL signal, to indicate that you want to replicate the new column.</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>Q Capture leaves the Q subscription active, but there are no log records to read for the source table.</td>
<td>Stop the Q subscription, and then delete the Q subscription.</td>
</tr>
<tr>
<td>DDL that alters the structure of a table</td>
<td>Q Capture leaves the Q subscription unchanged.</td>
<td>1. Stop the Q subscription. 2. Alter the source and target tables. 3. Start the Q subscription.</td>
</tr>
</tbody>
</table>

On z/OS, the ASN0197W warning message is issued.
Table 19. How Q Capture handles DDL changes to source tables and what you need to do (continued)

<table>
<thead>
<tr>
<th>DDL operation</th>
<th>How it is handled</th>
<th>What you need to do</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDL that does not alter the table structure</td>
<td>Q Capture leaves the Q subscription active.</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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.

#### 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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.             |
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q Replication Dashboard</td>
<td>Use the Queues page. Select the Q Capture-Q Apply program pair, and then on the Properties tab make your changes and click <strong>Save</strong>.</td>
</tr>
<tr>
<td><strong>reinitq parameter</strong></td>
<td>Use this parameter with the MODIFY command on z/OS or asnqcmd 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:</td>
</tr>
<tr>
<td><strong>send queue</strong></td>
<td>MODIFY</td>
</tr>
<tr>
<td></td>
<td>f myqcap,reinitq=queue_name</td>
</tr>
<tr>
<td></td>
<td>Where <strong>myqcap</strong> is the Q Capture job name.</td>
</tr>
<tr>
<td></td>
<td>asnqcmd</td>
</tr>
<tr>
<td></td>
<td>asnqcmd capture_server=server_name</td>
</tr>
<tr>
<td></td>
<td>capture_schema=schema reinitq=queue_name</td>
</tr>
<tr>
<td></td>
<td>Where <strong>server_name</strong> is the name of the database or subsystem where the Q Capture program is running,</td>
</tr>
<tr>
<td></td>
<td><strong>capture_schema</strong> identifies the Q Capture program that uses the send queue, and <strong>queue_name</strong> is the name of the send queue that you want to reinitialize.</td>
</tr>
<tr>
<td><strong>receive queue</strong></td>
<td>Use this parameter with the MODIFY command on z/OS or asnqcmd 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:</td>
</tr>
<tr>
<td></td>
<td>MODIFY</td>
</tr>
<tr>
<td></td>
<td>f myqapp,reinitq=queue_name</td>
</tr>
<tr>
<td></td>
<td>Where <strong>myqapp</strong> is the Q Apply job name.</td>
</tr>
<tr>
<td></td>
<td>asnqacmd</td>
</tr>
<tr>
<td></td>
<td>asnqacmd apply_server=server_name</td>
</tr>
<tr>
<td></td>
<td>apply_schema=schema_name reinitq=queue_name</td>
</tr>
<tr>
<td></td>
<td>Where <strong>server_name</strong> is the name of the database or subsystem where the Q Apply program is running,</td>
</tr>
<tr>
<td></td>
<td><strong>schema_name</strong> identifies the Q Apply program that uses the receive queue, and <strong>queue_name</strong> is the name of the receive queue that you want to reinitialize.</td>
</tr>
</tbody>
</table>
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**
     ```sql
     UPDATE schema.IBMQREP_SUBS
     SET STATE='I', STATE_TRANSITION=NULL,
     GROUP_MEMBERS=NULL WHERE SUBNAME='Q_subscription_name'
     ```
   - **Q Apply server**
     ```sql
     UPDATE schema.IBMQREP_TARGETS
     SET STATE='I' WHERE SUBNAME='Q_subscription_name'
     ```
3. Use the replication administration tools to restart the Q subscriptions.

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, the nickname for the target table will be dropped 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line program</td>
<td>Use one of the following commands, depending on your configuration:</td>
</tr>
<tr>
<td></td>
<td>DROP QSUB</td>
</tr>
<tr>
<td></td>
<td>DROP SUBTYPE (bidirectional replication)</td>
</tr>
<tr>
<td></td>
<td>DROP SUBTYPE (peer-to-peer replication)</td>
</tr>
<tr>
<td></td>
<td>For example, the following commands set the environment and drop the Q</td>
</tr>
<tr>
<td></td>
<td>subscription for unidirectional replication EMPLOYEE0001:</td>
</tr>
<tr>
<td></td>
<td>ASNCLP SESSION SET TO Q REPLICAION;</td>
</tr>
<tr>
<td></td>
<td>SET SERVER CAPTURE TO DB SAMPLE;</td>
</tr>
<tr>
<td></td>
<td>SET CAPTURE SCHEMA SOURCE ASN1;</td>
</tr>
<tr>
<td></td>
<td>SET SERVER TARGET TO DB TARGET;</td>
</tr>
<tr>
<td></td>
<td>SET APPLY SCHEMA ASN1;</td>
</tr>
<tr>
<td></td>
<td>SET RUN SCRIPT LATER;</td>
</tr>
<tr>
<td></td>
<td>DROP QSUB (SUBNAME EMPLOYEE0001 USING REPLQMAP</td>
</tr>
<tr>
<td></td>
<td>SAMPLE_ASN1_TO_TARGETDB_ASN1));</td>
</tr>
<tr>
<td></td>
<td>Use the SET DROP command to specify whether to drop the target table</td>
</tr>
<tr>
<td></td>
<td>and its table space.</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Delete Q Subscriptions window. To open the window, right-click a</td>
</tr>
<tr>
<td></td>
<td>Q subscription and select Delete.</td>
</tr>
<tr>
<td></td>
<td>You can optionally use this window to drop the target tables for Q</td>
</tr>
<tr>
<td></td>
<td>subscriptions that you delete. If you drop a target table, you can</td>
</tr>
<tr>
<td></td>
<td>optionally use the window to drop the associated table space if no other</td>
</tr>
<tr>
<td></td>
<td>tables are using the table space.</td>
</tr>
</tbody>
</table>

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.
2. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASNCLP command-line program</strong></td>
<td>Use the DROP REPLQMAP command. For example, the following commands set the environment and drop the queue map</td>
</tr>
<tr>
<td></td>
<td>SAMPLE ASN1 TO TARGET ASN1:</td>
</tr>
<tr>
<td></td>
<td>ASNCLP SESSION SET TO Q REPLICATION;</td>
</tr>
<tr>
<td></td>
<td>SET SERVER CAPTURE TO DB SAMPLE;</td>
</tr>
<tr>
<td></td>
<td>SET CAPTURE SCHEMA SOURCE ASN1;</td>
</tr>
<tr>
<td></td>
<td>SET SERVER TARGET TO DB TARGET;</td>
</tr>
<tr>
<td></td>
<td>SET APPLY SCHEMA ASN1;</td>
</tr>
<tr>
<td></td>
<td>SET RUN SCRIPT LATER;</td>
</tr>
<tr>
<td></td>
<td>DROP REPLQMAP</td>
</tr>
<tr>
<td></td>
<td>SAMPLE ASN1 TO TARGETDB ASN1;</td>
</tr>
<tr>
<td><strong>Replication Center</strong></td>
<td>Use the Delete Replication Queue Maps window. To open the window, right-click the replication queue map and select <strong>Delete</strong>.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 ASNI:
  ASNCLP SESSION SET TO Q REPLICATION;
  SET SERVER APPLY TO DB TARGET;
  SET APPLY SCHEMA ASNI;
  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 16. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASNCLP command-line program</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>ASNCLP SESSION SET TO Q REPLICATION;</td>
</tr>
<tr>
<td></td>
<td>SET SERVER CAPTURE TO DB SAMPLE;</td>
</tr>
<tr>
<td></td>
<td>SET CAPTURE SCHEMA SOURCE EP1;</td>
</tr>
<tr>
<td></td>
<td>SET RUN SCRIPT LATER;</td>
</tr>
<tr>
<td></td>
<td>ALTER PUB EMPLOYEE0001</td>
</tr>
<tr>
<td></td>
<td>FOR JK.EMPLOYEE OPTIONS</td>
</tr>
<tr>
<td></td>
<td>CHANGED COLS ONLY N SUPPRESS DELETES Y;</td>
</tr>
<tr>
<td><strong>Replication Center</strong></td>
<td>Use the Publication Properties notebook. To open the notebook, right-click a publication and select Properties.</td>
</tr>
</tbody>
</table>

2. If you are changing a single publication, reinitialize the publication by using one of the following methods:
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:

```sql
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit parameter</td>
<td>Use this parameter with the MODIFY command on z/OS or asnqcmd command on Linux, UNIX, Windows, and UNIX System Services on z/OS to reinitialize all of the publications for one Q Capture schema:</td>
</tr>
<tr>
<td></td>
<td><code>MODIFY</code></td>
</tr>
<tr>
<td></td>
<td><code>f myqcap,reinit</code></td>
</tr>
<tr>
<td></td>
<td>Where <code>myqcap</code> is the Q Capture job name.</td>
</tr>
<tr>
<td></td>
<td><code>asnqcmd</code></td>
</tr>
<tr>
<td></td>
<td><code>asnqcmd capture_server=server_name</code></td>
</tr>
<tr>
<td></td>
<td><code>capture_schema=schema reinit</code></td>
</tr>
<tr>
<td></td>
<td>Where <code>server_name</code> is the name of the database or subsystem where the Q Capture program is running, and <code>schema</code> identifies the Q Capture program for which you want to reinitialize all publications.</td>
</tr>
</tbody>
</table>

| 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line</td>
<td>Use the ALTER ADD COLUMN command. For example, the following command adds</td>
</tr>
<tr>
<td>program.</td>
<td>the column BONUS to the DEPARTMENT0001 publication:</td>
</tr>
<tr>
<td></td>
<td>ALTER ADD COLUMN USING SIGNAL (BONUS)</td>
</tr>
<tr>
<td></td>
<td>PUB DEPARTMENT0001;</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Center</td>
<td>Use the Delete Publications window. To open the window, right-click a publication and select <strong>Delete</strong>.</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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**  

```plaintext
f myqcap,reinitq=queue_name
```

Where `myqcap` is the Q Capture job name.

**asnqccmd**  

```plaintext
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.
   - 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*.
   - If any publications are using the queue map, delete the publications.

2. Delete the publishing queue map. Use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 17. Viewing reports about the Q replication and event publishing programs

You can use the Replication Center and system commands to check the status of the Q Capture program, Q Apply program, and Replication Alert Monitor. You can also use the Replication Center to view historical data that the programs save about their performance.

Checking the status of the Q replication and event publishing programs

You can use the Replication Center 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

Both the Replication Center and the commands allow you 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:

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**

```plaintext
f myqapp,status show details
```

Where `myqapp` is the Q Apply job name.

**asnqacmd**

```plaintext
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 asnmcmd 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**

```plaintext
f mymon,status
```

Where `mymon` is the monitor job name.

**asnmcmd**

```plaintext
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.

Replication Center

**Q Capture**

Use the Check Status window. Right-click the Q Capture server that contains the Q Capture program that you want to check, and select Check Status.

**Q Apply**

Use the Check Status window. Right-click the Q Apply server that contains the Q Apply program that you want to check, and select Check Status.
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:

**Q Capture program**

A Q Capture program has the following threads:

- **Transaction**
  Reads the DB2 recovery log, captures changes for subscribed tables, and rebuilds log records into transactions in memory before passing them to the worker thread.

- **Worker**
  Receives completed transactions from the transaction thread, turns transactions into WebSphere MQ messages, and puts the messages on send queues.

- **Administration**
  Handles control messages that are put by the Q Apply program or a user application on the administration queue, and is also used for error logging and monitoring.

- **Prune**
  Deletes old data from some of the Q Capture control tables.

- **Holdl**
  Prevents two Q Capture programs with the same schema from running on a server, and handles signals sent to the Q Capture program.

Worker, administration, and prune threads are typically in running or resting states. Holdl threads are typically in a waiting state. If the worker thread is in a running state but data is not being captured, you can use the Q Capture Messages window in the Replication Center or the IBMQREP_CAPTRACE table to look for messages that might explain why data is not being captured.

**Q Apply program**

A Q Apply program has the following threads:

- **Browser**
  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 BRxxxxx, where xxxxx specifies the number of browser threads for a Q Apply program.

- **Agent**
  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**
  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.

**Housekeeping**
Maintains the Q Apply control tables by saving and deleting data.

**Monitor**
Logs information about the Q Apply program’s performance in the IBMQREP_APPLYMON control table.

Browser, agent, and housekeeping 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 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 20.

**Table 20. Descriptions of thread states and suggested actions**

<table>
<thead>
<tr>
<th>Thread state</th>
<th>Description</th>
<th>Suggested action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>The thread is actively processing.</td>
<td>No action.</td>
</tr>
<tr>
<td>Exists</td>
<td>The thread exists but cannot start.</td>
<td>If the thread does not leave this state, stop the program and restart it.</td>
</tr>
<tr>
<td>Started</td>
<td>The thread started but cannot initialize.</td>
<td>Investigate potential system resource problems, such as too many threads or not enough processing power.</td>
</tr>
<tr>
<td>Initializing</td>
<td>The thread is initialized but cannot work.</td>
<td>If the thread does not leave this state, stop the program and restart it.</td>
</tr>
<tr>
<td>Resting</td>
<td>The thread is sleeping and will wake up when there is work to do.</td>
<td>No action.</td>
</tr>
</tbody>
</table>
### Table 20. Descriptions of thread states and suggested actions (continued)

<table>
<thead>
<tr>
<th>Thread state</th>
<th>Description</th>
<th>Suggested action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped</td>
<td>The thread is not running.</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Historical and performance data for Q replication and event publishing programs

You can use the Replication Center to check a variety of historical and performance data about a Q Capture program, Q Apply program, or Replication Alert Monitor.

The programs save this data in the following control tables: IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_CAPTRACE, IBMQREP_APPLYMOM, IBMQREP_EXCEPTIONS, IBMQREP_APPLYTRACE, IBMSNAP_ALERTS, and IBMSNAP_MONTRACE. The amount of data stored in each table depends on how frequently the tables are pruned.

**Recommendation:** Keep at least one week of data in these tables so that you can examine the data when you are troubleshooting or evaluating performance.

The Replication Center provides windows that let you view the data and tailor it to your needs. You can also print or export the data for use in a spreadsheet program.

Table 21 shows typical questions that you might ask about the performance or history of the Q replication and event publishing programs, and the Replication Center window that provides the data to help answer the questions.

### Table 21. Where to find historical and performance information in the Replication Center

<table>
<thead>
<tr>
<th>To answer this question</th>
<th>Use this window</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are the recent messages from the Q Capture, Q Apply, and Monitor programs?</td>
<td>Q Capture Messages, Q Apply Messages, Monitor Messages</td>
</tr>
<tr>
<td>• How many rows and transactions were put on send queues?</td>
<td>Q Capture Throughput</td>
</tr>
<tr>
<td>• How many rows and transactions were captured but not put on send queues because of Q subscription or publication definitions?</td>
<td></td>
</tr>
<tr>
<td>• How many transactions were spilled to a file?</td>
<td></td>
</tr>
<tr>
<td>• What was the largest transaction processed?</td>
<td></td>
</tr>
<tr>
<td>• How much memory is the Q Capture program using?</td>
<td></td>
</tr>
</tbody>
</table>
Table 21. Where to find historical and performance information in the Replication Center (continued)

<table>
<thead>
<tr>
<th>To answer this question</th>
<th>Use this window</th>
</tr>
</thead>
<tbody>
<tr>
<td>• How many rows and transactions were applied?</td>
<td>Q Apply Throughput</td>
</tr>
<tr>
<td>• How many rows were not applied because of conflicts or errors?</td>
<td></td>
</tr>
<tr>
<td>• How many times were rows retried because of lock timeouts and deadlocks?</td>
<td></td>
</tr>
<tr>
<td>• How many transactions were not applied in parallel because of conflicts?</td>
<td></td>
</tr>
<tr>
<td>• How many transactions violated referential integrity constraints?</td>
<td></td>
</tr>
<tr>
<td>• How many times were rows retried because of referential integrity constraints?</td>
<td></td>
</tr>
<tr>
<td>• How many rows were put in temporary spill queues?</td>
<td></td>
</tr>
<tr>
<td>• How many spilled rows were applied?</td>
<td></td>
</tr>
<tr>
<td>Approximately how current is the Q Capture program in reading the DB2 recovery log?</td>
<td>Q Capture Latency</td>
</tr>
<tr>
<td>How much time elapsed between:</td>
<td>Latency</td>
</tr>
<tr>
<td>• Transactions being committed at the source database and being put on send queues?</td>
<td></td>
</tr>
<tr>
<td>• Transactions being put on send queues and being taken from receive queues?</td>
<td></td>
</tr>
<tr>
<td>• Transactions being taken from receive queues and being committed at the target?</td>
<td></td>
</tr>
<tr>
<td>• Transactions being committed at the source and being committed at the target?</td>
<td></td>
</tr>
<tr>
<td>What kinds of conflicts and SQL errors caused some rows to not be applied? What were the details?</td>
<td>Exceptions</td>
</tr>
<tr>
<td>What alerts were issued by the Replication Alert Monitor?</td>
<td>Show Alerts</td>
</tr>
</tbody>
</table>

To open the report windows, right-click a Q Capture server, Q Apply server, or Monitor qualifier and select one of these menu items:

<table>
<thead>
<tr>
<th>Q Capture server</th>
<th>Q Apply server</th>
<th>Monitor qualifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reports &gt;</td>
<td>Reports &gt;</td>
<td>Show Monitor Messages</td>
</tr>
<tr>
<td>Messages</td>
<td>Messages</td>
<td>Show Alerts</td>
</tr>
<tr>
<td>Q Capture Throughput</td>
<td>Q Apply Throughput</td>
<td></td>
</tr>
<tr>
<td>Q Capture Latency</td>
<td>Latency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exceptions</td>
<td></td>
</tr>
</tbody>
</table>

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 18. Replication Alert Monitor

You can use the Replication Alert Monitor to monitor an SQL replication, Q replication, or event publishing environment.

The Replication Alert Monitor cannot check the status of Classic replication sources but it can monitor the DB2 or federated target servers in a Classic replication configuration.

The following topics explain how the Replication Alert Monitor works and how to configure and operate monitors for your replication or publishing environment.

Monitoring replication with the Replication Alert Monitor

The Replication Alert Monitor is a program that can alert you to changes in the status of your replication environment.

When the Replication Alert Monitor is running, it automatically checks the status of replication and notifies you about certain conditions that occur in your replication environment. For example, in SQL replication, the Replication Alert Monitor can notify you when any Apply program terminates. Similarly, in Q replication, the Replication Alert Monitor can notify you when any Q Capture program deactivates a Q subscription.

Restriction: The Replication Alert Monitor cannot check the status of Classic replication sources but it can monitor the DB2 or federated target servers in a Classic replication configuration.

You can configure the Replication Alert Monitor in one of two ways:

One monitor

Typically you use one monitor when you have few replication programs to monitor. If you set up a single monitor, all the control information is stored on one server. Each monitor can monitor multiple replication programs, but the monitor checks for alerts on each server one at a time. It must check all of the other servers that it monitors before it returns to any one server.

Multiple monitors

Use additional monitors to monitor many replication programs, prioritize the monitoring of certain programs, or split up the monitoring workload. You create independent monitors to check the servers in your system. These monitors do not communicate with each other, but they each send alerts about the servers. When you set up multiple monitors, the control information for each monitor is stored on the server that it is assigned to monitor. Use multiple monitors to:

- Monitor some replication programs more frequently than others. Set up a monitor with a smaller monitor_interval to check replication programs for alert conditions more frequently. For example, you can assign one monitor to monitor one Capture server for the CAPTURE_WARNINGS alert condition every 15 minutes. You can assign another monitor to monitor another Capture server for the CAPTURE_WARNINGS alert condition every 50 minutes.
- **Monitor different applications separately.** Set up monitors for each replication application. For example, separate monitors can send alerts to different groups or help an administrator separate the alerts for two different applications. Similarly, separate monitors can be assigned to check for different alert conditions.

- **Prioritize alert conditions.** For example, you might want to monitor the status of a Q Apply program every 10 minutes by using the QAPPLY_STATUS alert condition. But, you might also want to monitor the memory of the same Q Apply program every 300 minutes by using the QAPPLY_MEMORY alert condition.

The following terms describe components of the Replication Alert Monitor:

**Monitor**
A monitor is one instance or occurrence of the Replication Alert Monitor. You can set up a monitor to check the status of the replication programs that are running on a server or servers. Each monitor checks the replication activity on the server, or servers, that it is assigned to.

**Monitor qualifier**
A monitor qualifier is a name that you specify for a monitor. Every monitor has a unique monitor qualifier.

**Monitor control server**
A monitor control server is any server containing control information for the Replication Alert Monitor.

**Alerts**
Alerts are notices that tell you about events and conditions in your replication environment. The Replication Alert Monitor sends alerts by means of e-mail or pager. You can also send alerts to the z/OS console.

**Alert conditions**
Alert conditions are conditions of the replication environment that cause the Replication Alert Monitor to send alerts. There are three kinds of alert conditions: alert conditions that are triggered by status, alert conditions that are triggered by events, and alert conditions that are triggered by thresholds.

**Alert conditions that are triggered by status**
Status alert conditions inform you about the state of the replication programs. For example, if you specify the APPLY_STATUS alert condition, the Replication Alert Monitor will send an alert if an Apply program is not running.

**Alert conditions that are triggered by events**
Event alert conditions tell you when specific events in replication happen. For example, if you specify the QAPPLY_ERRORS alert condition, the Replication Alert Monitor will send an alert anytime the Q Apply program records an error in the IBMQREP_APPLYTRACE table.

**Alert conditions that are triggered by thresholds**
Threshold alert conditions tell you when a threshold has been exceed in your replication environment. For example, if you specify the QCAPTURE_MEMORY alert condition, the Replication Alert Monitor will notify you anytime the Q Capture program uses more memory than its threshold allows.

**Contacts**
A contact is an e-mail address or a pager address where alerts from the
Replication Alert Monitor are sent. Alerts can also be directed to the z/OS console. The ASNMAIL exit routine sends e-mail notifications for the monitor. You can modify this exit routine to put the alerts elsewhere such as in a problem management system.

Contact groups
A contact group is a collection of contacts that receive the same alerts.

- z/OS
  You can also specify that alerts are sent to the z/OS console.

The Replication Alert Monitor monitors servers on DB2 for Linux, UNIX, Windows, or z/OS operating systems.

A monitor that runs from a z/OS server can monitor the status of replication programs that are running locally or remotely on other z/OS data-sharing systems as well as on other Linux, UNIX, or Windows servers. The monitor will check the status by querying the appropriate Q Capture, Q Apply, Capture, or Apply monitor tables.

You can monitor replication programs whose control tables are at Version 8 architecture or later.

Restrictions
- For non-DB2 relational databases, the Replication Alert Monitor does not monitor triggers that are associated with such databases used as sources in a federated database system.
- z/OS
  The Replication Alert Monitor can send e-mail notifications by using an SMTP server, but cannot use the ASNMAIL exit routine to handle notification.
- System i
  To monitor System i® servers, the Replication Alert Monitor must run on a Linux, UNIX, or Windows server and monitor the System i server remotely. You cannot set up Monitor control servers on DB2 for i5/OS® servers.

Alert conditions and notifications for the Replication Alert Monitor
The Replication Alert Monitor can send notifications when certain alert conditions occur.

Alert conditions for the Replication Alert Monitor
Alert conditions are conditions of the replication environment that cause a monitor to send alerts. Alerts are messages that describe the status, event or threshold that triggers an alert condition.

Some alerts also report relevant parameter values. For example, the message for the QCAPTURE_MEMORY alert condition reports the amount of memory that the Q Capture program is using and the memory threshold value that was exceeded.

The following sections describe alert conditions that you can use to monitor your replication environment.
- "Alert conditions for the Q Capture program” on page 266
- "Alert conditions for the Q Apply program” on page 266
- "Alert conditions for the Capture program” on page 267
- "Alert conditions for the Apply program” on page 267
## Alert conditions for the Q Capture program

Table 22 describes the alert conditions for the Q Capture program.

<table>
<thead>
<tr>
<th>Alert condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCAPTURE_STATUS</td>
<td>The Replication Alert Monitor sends an alert when a Q Capture program is not running.</td>
</tr>
<tr>
<td>QCAPTURE_ERRORS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of ERROR in the OPERATION column of the IBMQREP_CAPTRACE table.</td>
</tr>
<tr>
<td>QCAPTURE_WARNINGS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of WARNING in the OPERATION column in the IBMQREP_CAPTRACE table.</td>
</tr>
<tr>
<td>QCAPTURE_LATENCY</td>
<td>Q Capture latency measures the difference between the time that data was written to the database and the time that the Q Capture program passes it on. The Replication Alert Monitor sends an alert when the Q Capture latency exceeds the threshold that you specify. Q Capture latency is measured in seconds.</td>
</tr>
<tr>
<td>QCAPTURE_MEMORY</td>
<td>The Replication Alert Monitor sends an alert when a Q Capture program uses more memory than the threshold that you specify. Memory is measured in megabytes.</td>
</tr>
<tr>
<td>QCAPTURE_TRANSIZE</td>
<td>The Replication Alert Monitor sends an alert when a transaction that the Q Capture program is processing uses more memory than the threshold that you specify. Memory is measured in megabytes.</td>
</tr>
<tr>
<td>QCAPTURE_SUBSINACT</td>
<td>The Replication Alert Monitor sends an alert when a Q Capture program deactivates a Q subscription.</td>
</tr>
</tbody>
</table>

## Alert conditions for the Q Apply program

Table 23 describes the alert conditions for the Q Apply program.

<table>
<thead>
<tr>
<th>Alert condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QAPPLY_STATUS</td>
<td>The Replication Alert Monitor sends an alert when a Q Apply program is not running.</td>
</tr>
<tr>
<td>QAPPLY_ERRORS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of ERROR in the OPERATION column in the IBMQREP_APPLYTRACE table.</td>
</tr>
<tr>
<td>QAPPLY_WARNINGS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of WARNING in the OPERATION column in the IBMQREP_APPLYTRACE table.</td>
</tr>
<tr>
<td>QAPPLY_LATENCY</td>
<td>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 Replication Alert Monitor sends an alert when the Q Apply latency exceeds the threshold that you specify. Q Apply latency is measured in milliseconds.</td>
</tr>
<tr>
<td>QAPPLY_EELATENCY</td>
<td>Q Apply end-to-end latency measures the total time that replication requires to capture changes and apply those changes to a target database. The Replication Alert Monitor sends an alert when the Q Apply end-to-end latency exceeds the threshold that you specify. Q Apply end-to-end latency is measured in seconds.</td>
</tr>
</tbody>
</table>
### Table 23. Alert conditions for the Q Apply program (continued)

<table>
<thead>
<tr>
<th>Alert condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QAPPLY_MEMORY</td>
<td>The Replication Alert Monitor sends an alert when a Q Apply program uses more memory than the threshold that you specify. Memory is measured in megabytes.</td>
</tr>
<tr>
<td>QAPPLY.Exceptions</td>
<td>The Replication Alert Monitor sends an alert when a row is inserted in the IBMQREP_EXCEPTIONS table because of a conflict or SQL error at a target.</td>
</tr>
<tr>
<td>QAPPLY_RECVQINACT</td>
<td>The Replication Alert Monitor sends an alert when a receive queue is deactivated.</td>
</tr>
<tr>
<td>QAPPLY_SPILLQDEPTH</td>
<td>The Replication Alert Monitor sends an alert when the fullness of the spill queue exceeds the threshold that you specify. Fullness is expressed as a percentage. This alert condition is not supported when the Q Apply program is remote from the target database or subsystem.</td>
</tr>
<tr>
<td>QAPPLY_QDEPTH</td>
<td>The Replication Alert Monitor sends an alert when the fullness of any queue exceeds the threshold that you specify. Fullness is expressed as a percentage. This alert condition is not supported when the Q Apply program is remote from the target database or subsystem.</td>
</tr>
</tbody>
</table>

### Alert conditions for the Capture program

Table 24 describes the alert conditions for the Capture program.

### Table 24. Alert conditions for the Capture program

<table>
<thead>
<tr>
<th>Alert condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPTURE_STATUS</td>
<td>The Replication Alert Monitor sends an alert when a Capture program is not running.</td>
</tr>
<tr>
<td>CAPTURE_ERRORS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of ERROR in the OPERATION column of the IBMSNAP_CAPTRACE table.</td>
</tr>
<tr>
<td>CAPTURE_WARNINGS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of WARNING in the OPERATION column of the IBMSNAP_CAPTRACE table.</td>
</tr>
<tr>
<td>CAPTURE_LASTCOMMIT</td>
<td>The Replication Alert Monitor sends an alert when the time that elapsed from the last commit of a Capture program exceeds the threshold that you specify. Time elapsed is measured in seconds.</td>
</tr>
<tr>
<td>CAPTURE_CLATENCY</td>
<td>The current capture latency measures the difference between the time that data was written to the database and the time that the Q Capture program passes it on. The Replication Alert Monitor sends an alert when the current Capture latency exceeds the threshold that you specify.</td>
</tr>
<tr>
<td>CAPTURE_HLATENCY</td>
<td>Historic Capture latency is a composite of every Capture latency measurement since the monitor last checked a server for alert conditions. The Replication Alert Monitor sends an alert when the historic Capture latency exceeds the threshold that you specify.</td>
</tr>
<tr>
<td>CAPTURE_MEMORY</td>
<td>The Replication Alert Monitor sends an alert when a Capture program uses more memory than the threshold that you specify. Memory is measured in megabytes.</td>
</tr>
</tbody>
</table>

### Alert conditions for the Apply program

Table 25 on page 268 describes the alert conditions for the Apply program.
<table>
<thead>
<tr>
<th>Alert condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY_STATUS</td>
<td>The Replication Alert Monitor sends an alert when an Apply program is not running.</td>
</tr>
<tr>
<td>APPLY_SUBSFALING</td>
<td>The Replication Alert Monitor sends an alert when a subscription fails.</td>
</tr>
<tr>
<td>APPLY_SUBSINACT</td>
<td>The Replication Alert Monitor sends an alert when a subscription is deactivated.</td>
</tr>
<tr>
<td>APPLY_ERRORS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of ERROR in the OPERATION column in the IBMSNAP_APPLYTRACE table</td>
</tr>
<tr>
<td>APPLY_WARNINGS</td>
<td>The Replication Alert Monitor sends an alert when it finds a row with the value of WARNING in the OPERATION column in the IBMSNAP_APPLYTRACE table</td>
</tr>
<tr>
<td>APPLY_FULLREFRESH</td>
<td>The Replication Alert Monitor sends an alert when there is a full refresh.</td>
</tr>
<tr>
<td>APPLY_REJTRANS</td>
<td>The Replication Alert Monitor sends an alert when a transaction is rejected in a subscription set.</td>
</tr>
<tr>
<td>APPLY_SUBSDELAY</td>
<td>The Replication Alert Monitor sends an alert when the delay in processing a subscription is longer than the threshold that you specify.</td>
</tr>
<tr>
<td>APPLY_REWORKED</td>
<td>The Replication Alert Monitor sends an alert when the Apply program reworks more rows in a subscription set than the threshold that you specify.</td>
</tr>
<tr>
<td>APPLY_LATENCY</td>
<td>Apply end-to-end latency measures the total time that replication requires to capture changes and apply those changes to a target database. The Replication Alert Monitor sends an alert when the Apply end-to-end latency exceeds the threshold that you specify. Apply end-to-end latency is measured in seconds.</td>
</tr>
</tbody>
</table>

**E-mail notifications for replication alert conditions**

The Replication Alert Monitor can send an e-mail when an alert condition occurs.

The content of the e-mail notification depends on whether the e-mail address that you provided is for a pager. The following examples show the type of information that you can expect in each case, for one set of alerts. The e-mail that is sent to non-pager devices shows the time when each alert condition occurred at the specific server. It also shows the number of times that each alert condition occurred and the associated message. The e-mail that the Replication Alert Monitor sends to pagers contains a summary of the parameters that triggered the alert instead of a complete message. If an alert condition occurred many times, the timestamp reflects the last time that the alert condition occurred.

**Setting ASNSENDER variable to prevent e-mail filtering**

Some providers such as pager services require a full valid return address for filtering unsolicited messages. If a valid return address is not provided, the e-mail might be blocked.

If you are not receiving alerts to your e-mail address or pager, take the following steps:
1. Stop the Replication Alert Monitor.
2. Set the ASNSENDER environment variable to a valid e-mail address. For example:
3. Start the monitor.

**Example e-mail notification to nonpager devices (SQL replication)**

To: repladmin@company.com
From: replmon@server.com
Subject: Monitor: "MONQUAL" Alerts issued

ASN5129I MONITOR "MONQUAL". The Replication Alert Monitor on server "WSDB" reports an e-mail alert

2002-01-20-10.00.00 1 ASN0552E Capture : "ASN" The program encountered an SQL error. The server name is "CORP". The SQL request is "PREPARE". The table name "PROD1.INVOICESCD". The SQLCODE is "+204". The SQLSTATE is "42704". The SQLERRMC is "PROD1.INVOICESCD". The SQLERRP is "readCD"

2002-01-20-10.05.00 2 ASN5152W Monitor "MONQUAL". The current Capture latency exceeds the threshold value. The Capture control server is "CORP". The schema is "ASN". The Capture latency is "90" seconds. The threshold is "60" seconds

2002-01-20-10.05.00 4 ASN5154W Monitor "MONQUAL". The memory used by the Capture program exceeds the threshold value. The Capture control server is "CORP". The schema is "ASN". The amount of memory used is "34" bytes. The threshold is "30" megabytes.

**Example e-mail notification to pagers (SQL replication)**

To: repladmin@company.com
From: replmon@server.com
Subject: Monitor: "MONQUAL" Alerts issued

MONQUAL - MONDB

2002-01-20-10.00.00 ASN0552E 1 CAPTURE-ERRORS - CORP - ASN
2002-01-20-10.05.00 ASN5152W 2 CAPTURE_CLATENCY - CORP - ASN - 90 - 60
2002-01-20-10.05.00 ASN5154W 4 CAPTURE_MEMORY - CORP - ASN - 34 - 30

In SQL replication, the monitor groups alerts by Capture control servers and Apply control servers when it sends notifications. If a server is both a Capture control server and an Apply control server, then the monitor groups all alerts for that server together.

In Q replication, the monitor groups alerts by Q Capture servers and Q Apply servers when it sends notifications. If a server is both a Q Capture server and a Q Apply server, then the monitor groups all alerts for that server together.

If the size of the e-mail notification exceeds the limit for the type of e-mail, the monitor sends notification in multiple e-mails. The maximum size of a regular e-mail notification is 1024 characters. For a pager e-mail address the limit is 250 characters.

The ASNMAIL exit routine sends e-mail notifications for the monitor. You can modify this exit routine to handle alerts differently. For example, you could have the ASNMAIL user exit routine store the alerts in a problem management system.
Specifying where to send monitor alerts

The Replication Alert Monitor can send alerts to the z/OS console, to an e-mail address or pager, or to both the console and an e-mail address or pager.

Procedure

To specify where monitor alerts are sent, choose one of the following options:

<table>
<thead>
<tr>
<th>To send to ...</th>
<th>Take these steps ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail only</td>
<td>In the ASNCLP CREATE ALERT CONDITIONS FOR command, use the NOTIFY CONTACT or NOTIFY GROUP keywords to specify a person or group to notify by email. The following example shows the syntax for specifying that alerts are sent to the e-mail contact REPLADMIN: CREATE CONDITIONS FOR QAPPLY MONITOR QUALIFIER MONQUAL NOTIFY CONTACT REPLADMIN (STATUS DOWN, ERRORS, WARNINGS, LATENCY 360, EXCEPTIONS)</td>
</tr>
<tr>
<td>z/OS console only</td>
<td>In the ASNCLP CREATE ALERT CONDITIONS FOR command, use the NOTIFY OPERATOR CONSOLE keywords to define the z/OS console as the destination for alerts. For example: CREATE ALERT CONDITIONS FOR QCAPTURE SCHEMA ASN1 MONITOR QUALIFIER MONQUAL NOTIFY OPERATOR CONSOLE (STATUS DOWN, ERRORS, WARNINGS)</td>
</tr>
<tr>
<td>E-mail and z/OS console</td>
<td>1. Follow the steps above to specify an e-mail contact. 2. Start the monitor by using the console=y option. Use one of the following methods: JCL Specify CONSOLE=Y in the job that will start the monitor. asnmon command (USS) From a USS command prompt, issue the asnmon command with the console parameter set to Y. For example: asnmon monitor_server=SAMPLE monitor_qualifier=monqual console=y</td>
</tr>
</tbody>
</table>

You can also use the Replication Center to specify that alerts be sent to both e-mail and the z/OS console. In the Alert Conditions window, check the Send notification to the operator console check box. Then use the Run Now or Save Command window to edit the operational command that will start the monitor so that the console parameter is set to Y, and start the monitor.

The ASNMAIL exit routine for sending alerts in replication (Linux, UNIX, Windows)

The ASNMAIL exit routine distributes alerts that notify you about specific conditions in your replication environment.

**z/OS** The Replication Alert Monitor cannot use the ASNMAIL exit routine to handle notification on z/OS. An SMTP server can be used instead.

This exit routine takes the following input:

```
asnmail email_server to_address subject alert_message alert_message
```
Table 26 describes the inputs for the ASNMAIL exit routine.

Table 26. Inputs for the ASNMAIL exit routine

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email_server</td>
<td>This is the address of an e-mail server that uses the SMTP protocol. This server address is passed from the email_server parameter specified in at the start of the asnmon command.</td>
</tr>
<tr>
<td>to_address</td>
<td>This is the e-mail address of the contact to be notified.</td>
</tr>
<tr>
<td>subject</td>
<td>This is the subject in the notification.</td>
</tr>
<tr>
<td>alert_message</td>
<td>This is the string that contains the alert message.</td>
</tr>
</tbody>
</table>

Instead of sending alerts by e-mail, you can modify the ASNMAIL exit routine to put the alerts elsewhere such as in a problem management system. The \sqllib\samples\rep1\ directory contains a sample of the ASNMAIL exit routine. The asnameail.smp sample contains the input parameters and directions for using the sample program.

**Setting up the Replication Alert Monitor**

Your replication environment consists of the replication programs that run on servers and the control tables that support them. The Replication Alert Monitor monitors this environment for you.

**About this task**

The following topics describe things to consider before setting up the monitor.

- [“Authorization requirements for the Replication Alert Monitor” on page 272](#)
- [“Memory used by the Replication Alert Monitor”](#)
- [“Optional: Binding the Replication Alert Monitor program packages (Linux, UNIX, Windows)” on page 272](#)

**Procedure**

To set up the monitor:

1. Create control tables for each Monitor control server
2. Define contact information for the monitor
3. Create one or more monitors
4. Select alert conditions
5. Operate the monitor
6. Optional: Define suspension periods for the monitor

**Memory used by the Replication Alert Monitor**

The Replication Alert Monitor uses memory to store definitions and to keep alerts in memory before they are sent as notifications.

The amount of memory needed for the definitions is directly proportional to the number of definitions. The Replication Alert Monitor reserves 32 KB of memory for storing alert notifications. More memory is requested, as needed, and released when no longer required.

**Recommendation:** Do not set a memory quota for the Replication Alert Monitor. If you need to set one, set it to 3 MB.
Authorization requirements for the Replication Alert Monitor

All user IDs that run a Replication Alert Monitor must have authorization to access the Q Capture server or Q Apply server that you want to monitor. A user ID must also have access to the Monitor control tables on the Monitor control server.

User IDs that run a monitor must have the following authorities and privileges:

- SELECT, UPDATE, INSERT, and DELETE privileges for the Monitor control tables
- SELECT privileges on the Q Capture and Q Apply control tables on the servers that you want to monitor
- BINDADD authority (required only if you want to use the autobind feature for the monitor packages)
- EXECUTE privilege for the Monitor program packages
- WRITE privilege on the monitor_path directory where the Replication Alert Monitor stores diagnostic files
- Read access to the password file that is used by the Replication Alert Monitor
- Authority to create global objects

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

The Replication Alert Monitor program is bound automatically on Linux, UNIX, and Windows during execution. You can bind packages manually if you want to specify bind options, schedule binding, or check that all bind processes completed successfully.

Procedure

To bind the Monitor program packages:

1. Change to the directory where the Monitor program bind files are located.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Location of bind files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>drive:\..\sqllib\bnd</td>
</tr>
<tr>
<td></td>
<td>Where drive is the drive where DB2 is installed.</td>
</tr>
<tr>
<td>Linux UNIX</td>
<td>db2homedir\sqllib\bnd</td>
</tr>
<tr>
<td></td>
<td>Where db2homedir is the DB2 instance home directory.</td>
</tr>
</tbody>
</table>

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 Monitor control 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. For example:
      
      db2 connect to database user userid using password
b. Create and bind the Replication Alert Monitor program 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. For each server that you are monitoring and to which the Replication Alert Monitor program connects, do the following steps:
   a. Connect to the database by entering the following command:
      
      ```
      db2 connect to database
      ```

      Where `database` is the server that you want to monitor. 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. For example:
      
      ```
      db2 connect to database user userid using password
      ```

   b. Create and bind the Replication Alert Monitor program 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.

      These commands create packages, the names of which are in the file `asnmonit.lst`.

### Creating control tables for the Replication Alert Monitor

Before you can use the Replication Alert Monitor, you must create monitor control tables. These tables store alert conditions, contact information, run-time parameters, and other metadata for the monitor.

#### About this task

The server where you create the monitor control tables is called a Monitor control server.

The Monitor control server can be a DB2 for Linux, UNIX, Windows database or a DB2 for z/OS subsystem. In most cases you will need only one Monitor control server, but you can use multiple servers depending on your replication environment. For example, if you want monitors to run on the same system as the replication programs that they monitor, create one set of control tables for each local monitor on the server where the monitor runs.

#### Procedure

To create monitor control tables, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP</td>
<td>Use the CREATE CONTROL TABLES FOR command. For example: CREATE CONTROL</td>
</tr>
<tr>
<td>command-line</td>
<td>TABLES FOR MONITOR CONTROL SERVER;</td>
</tr>
<tr>
<td>program</td>
<td></td>
</tr>
</tbody>
</table>
Defining contact information for the Replication Alert Monitor

You must define contact information for the individuals or groups that you want to notify of alert conditions before you use the Replication Alert Monitor for the first time.

About this task

Contact information is stored on Monitor control servers. Monitors running on the same Monitor control server can share contacts. If you have multiple Monitor control servers, you must define contacts for each server. You can change contact information after monitors are running.

After you define contacts by specifying the e-mail address for and the name of each contact, you can put contacts into groups. For example, you could set up a contact group called replication administrators that contains the contact information for all your replication administrators. You can also copy contact and group information from one server to another.

Contacts that you create for the Replication Alert Monitor in the Replication Center cannot be used in other DB2 centers such as the Task Center or the Health Center. Contacts created in other DB2 centers cannot be used by the Replication Alert Monitor.

Procedure

To define contact information for the Replication Alert Monitor:
1. Create contacts and contact groups for the monitors on a Monitor control server by using one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line</td>
<td>Use the CREATE CONTACT command. For example:</td>
</tr>
<tr>
<td></td>
<td>CREATE CONTACT REPLADMIN</td>
</tr>
<tr>
<td></td>
<td>EMAIL &quot;<a href="mailto:repladmin@us.ibm.com">repladmin@us.ibm.com</a>&quot;</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION &quot;replication administration&quot;;</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Create Contact window or Create Contact Group window. To open the</td>
</tr>
<tr>
<td></td>
<td>windows, expand the Monitor control server for which you want to add a</td>
</tr>
<tr>
<td></td>
<td>contact or contact group, right-click the Contacts folder and select Create</td>
</tr>
<tr>
<td></td>
<td>Contact + Person or Create Contact + Group.</td>
</tr>
</tbody>
</table>

2. Optional: Copy contact information from one Monitor control server to another by using the Copy Contacts and Contact Groups window in the Replication Center. To open the window, expand the Monitor control server on which the contacts or contact groups are located. Select the Contacts folder. In the contents pane, right-click the contacts or contact groups that you want to copy, and select Copy.

3. Optional: Use the DELEGATE CONTACTS command in the ASNCLP command-line program to delegate an existing contact to a new contact for a specific period of time. For example:
Creating monitors for replication or publishing

After you create monitor control tables, you can use the Create Monitor wizard in the Replication Center to create monitors and select the alert conditions that will be used to monitor your replication or publishing environment.

Before you begin

Before you create monitors, you must set up the Replication Alert Monitor.

Procedure

To create a monitor:

1. In the Replication Center, open the Create Monitor wizard and specify the name of the monitor and the replication or publishing programs that the monitor will check for alert conditions:
   a. To open the wizard, expand the Monitor control server on which you want to create a monitor, right-click the Monitors folder, and select Create.
   b. On the Start page, specify a monitor qualifier. Then, specify the programs that you want this monitor to check for alert conditions. You can also monitor subscription sets that are used in SQL replication.

   The wizard directs you to one or more of the following pages where you can select alert conditions, depending on which replication programs you want this monitor to check for alert conditions:
   • Select alert conditions for Q Capture programs
   • Select alert conditions for Q Apply programs
   • Select alert conditions for Capture programs
   • Select alert conditions for Apply programs
   • Select alert conditions for subscription sets

   See the online help for details. For example, if you specified that you want to monitor Q Capture programs and Q Apply programs, then the Create Monitor wizard directs you to the Select alert conditions for Q Capture programs page and the Select alert conditions for Q Apply programs page.

2. From one of the pages that are listed above, open secondary dialogs where you can:
   • Specify the programs or subscription sets that you want to monitor.
   • Specify the alert conditions that you want to check for, and the parameters for the appropriate alert conditions. For example, you can set the monitor_interval parameter value to 60 to make the monitor check for alert conditions once every minute.


Selecting alert conditions for the Replication Alert Monitor

When you create a monitor, you select the alert conditions that will prompt that monitor to send alerts. You can select alert conditions for each Q Capture program, Q Apply program, Capture program, Apply program, or subscription set that a monitor is monitoring.

About this task
The Replication Alert Monitor monitors the activity of the replication and publishing programs at the following times:

- Each monitor checks for alert conditions immediately when you start it.
- Each monitor checks for alert conditions periodically, at timed intervals that you specify.

**Procedure**

To select alert conditions for the Replication Alert Monitor, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line program</td>
<td>Use one of the following commands:</td>
</tr>
<tr>
<td></td>
<td>• CREATE ALERT CONDITIONS FOR APPLY</td>
</tr>
<tr>
<td></td>
<td>• CREATE ALERT CONDITIONS FOR CAPTURE</td>
</tr>
<tr>
<td></td>
<td>• CREATE ALERT CONDITIONS FOR Q CAPTURE</td>
</tr>
<tr>
<td></td>
<td>• CREATE ALERT CONDITIONS FOR Q APPLY</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use one or more of the following pages in the Create Monitor wizard in the Replication Center, depending on which program you chose to monitor:</td>
</tr>
<tr>
<td></td>
<td>• Select alert conditions for Q Capture programs</td>
</tr>
<tr>
<td></td>
<td>• Select alert conditions for Q Apply programs</td>
</tr>
<tr>
<td></td>
<td>• Select alert conditions for Capture programs</td>
</tr>
<tr>
<td></td>
<td>• Select alert conditions for Apply programs</td>
</tr>
<tr>
<td></td>
<td>• Select alert conditions for subscription sets</td>
</tr>
</tbody>
</table>

Specify thresholds that are compatible with your environment. For example, if a Capture program is running with a commit interval of 30 seconds, specify a threshold for Capture latency that is greater than 30 seconds. Or, if you schedule an Apply program to process subscription sets every 10 minutes, set the threshold of the APPLY_SUBSDELAY alert condition to a value that is greater than 10 minutes.

**Changing alert conditions for the Replication Alert Monitor**

You can change alert conditions while the monitor is running. You do this by changing the alert conditions and then reinitializing the monitor.

**Procedure**

To change alert conditions, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNCLP command-line program</td>
<td>Use one of the following commands:</td>
</tr>
<tr>
<td></td>
<td>• ALTER ALERT CONDITIONS FOR APPLY</td>
</tr>
<tr>
<td></td>
<td>• ALTER ALERT CONDITIONS FOR CAPTURE</td>
</tr>
<tr>
<td></td>
<td>• ALTER ALERT CONDITIONS FOR Q CAPTURE</td>
</tr>
<tr>
<td></td>
<td>• ALTER ALERT CONDITIONS FOR Q APPLY</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Alert Conditions window for a Q Capture program, Q Apply program, Capture program, Apply program, or subscription set. To open the windows, select a monitor in the object tree, right-click a schema or subscription set in the contents pane, and click Change.</td>
</tr>
</tbody>
</table>
After you change the alert conditions, reinitialize the monitor.

**Defining suspension periods for the Alert Monitor**

You can define suspension periods for a Replication Alert Monitor program. You can create a repeating suspension (for example, every Sunday morning for two hours) or suspend the monitor for a single time period.

**About this task**

While the monitor is suspended, it will stop checking Q Capture, Q Apply, Capture control, or Apply control servers for all defined alert conditions. When the suspension period ends, the monitor will resume checking.

To define a suspension that repeats, you create a *suspension template*. If you create a template, you can reuse the template on multiple monitored servers.

If you do not create a template, you can specify a start date and time and end date and time for which monitoring on a server is suspended one time.

All dates and times for monitor suspensions are based on the clock at the system where the monitor is running.

**Restrictions**

Suspensions and suspension templates can only be defined through the ASNCLP command-line program and cannot be defined or viewed through the Replication Center.

**Procedure**

To suspend the monitor for a defined period:

1. Optional: Use the CREATE MONITOR SUSPENSION TEMPLATE command in the ASNCLP command-line program to create a template to define a repeating suspension.
   For example, the following command creates a template that suspends the monitor program from 00:00:00 to 04:00:00 every Sunday:
   ```
   CREATE MONITOR SUSPENSION TEMPLATE SUNDAY START TIME 00:00:00 REPEATS WEEKLY DAY OF WEEK SUNDAY FOR DURATION 4 HOURS
   ```

2. Use the CREATE MONITOR SUSPENSION command in the ASNCLP command-line program to define a start and end point for a one-time suspension or use a suspension template.
   For example, the following command creates a suspension called S1, which uses the template SUNDAY to suspend the monitor control server QSRVR1:
   ```
   CREATE MONITOR SUSPENSION NAME S1 FOR SERVER QSRVR1 STARTING DATE 2006-12-10 USING TEMPLATE SUNDAY ENDING DATE 2007-12-31
   ```

3. Reinitialize the monitor that you want to suspend by using the asnmcmd reinit command.
   You can also use the Reinitialize Monitor window in the Replication Center. To open the window, right-click a monitor qualifier in the contents pane and click **Reinitialize Monitor**.

4. Optional: Use one of the following commands in the ASNCLP command-line program to list, change, or drop monitor suspensions or suspension templates:
### Operating the Replication Alert Monitor

You can start, stop, suspend, reinitialize, and perform other operations on the Replication Alert Monitor.

#### Starting monitors

You can use several methods to start a monitor. You can also decide whether to run the monitor continuously or for only one monitor cycle. You can also set values for parameters, and enter the e-mail address of the person to contact if the monitor itself encounters an error while running.

**Before you begin**

- Create monitor control tables and a monitor, which includes selecting contacts and alert conditions.
- Create a password file.
- Make sure that you have the correct authorization to access the Monitor control tables and servers where the programs that you want to monitor are running.

**Procedure**

To start a monitor, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Replication Center</strong></td>
<td>Use the Start Monitor window. To open the window, right-click the Monitor qualifier that identifies the monitor that you want to start, and select <strong>Start Monitor</strong>.</td>
</tr>
<tr>
<td><em>z/OS</em></td>
<td>Use this command to start a monitor and optionally specify startup parameters.</td>
</tr>
<tr>
<td><em>Linux</em> <em>UNIX</em> <em>Windows</em></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST MONITOR SUSPENSION</td>
<td>Generates a list of suspensions on a monitor control server.</td>
</tr>
<tr>
<td>ALTER MONITOR SUSPENSION</td>
<td>Allows you to change the following properties of a suspension:</td>
</tr>
<tr>
<td></td>
<td>• The template that is used</td>
</tr>
<tr>
<td></td>
<td>• The start or end date for using a template</td>
</tr>
<tr>
<td></td>
<td>• The start or end date for suspending the monitor program one time</td>
</tr>
<tr>
<td>DROP MONITOR SUSPENSION</td>
<td>Deletes a suspension from the monitor control tables.</td>
</tr>
<tr>
<td>LIST MONITOR SUSPENSION TEMPLATE</td>
<td>Generates a list of suspension templates on a monitor control server.</td>
</tr>
<tr>
<td>ALTER MONITOR SUSPENSION TEMPLATE</td>
<td>Allows you to change the frequency and length of monitor suspensions as defined in a suspension template.</td>
</tr>
<tr>
<td>DROP MONITOR SUSPENSION TEMPLATE</td>
<td>Deletes a suspension template from the monitor control tables.</td>
</tr>
</tbody>
</table>
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>z/OS</strong></td>
<td>You can set up the ARM recovery system to start a monitor from the z/OS console or TSO.</td>
</tr>
<tr>
<td>Automatic Restart Manager</td>
<td></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td>You can set up the monitor to run as a Windows service.</td>
</tr>
<tr>
<td>Windows service</td>
<td></td>
</tr>
</tbody>
</table>

### Reinitializing monitors

You can reinitialize a monitor while it is running. Reinitializing a monitor causes it to recognize any updates that you have made to contacts, alert conditions, and parameter values. For example, reinitialize a monitor if you added a new e-mail address for a contact while the monitor is running.

**Procedure**

To reinitialize a monitor, use one of the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnmcmd system command</td>
<td>Use the asnmcmd reinit command to reinitialize a running monitor.</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Reinitialize Monitor window to reinitialize a monitor. To open the window, right-click the Monitor qualifier that identifies the monitor that you want to reinitialize, and select Reinitialize Monitor.</td>
</tr>
</tbody>
</table>

### Suspending and resuming a monitor

You can suspend and resume a monitor when you want to temporarily stop monitoring your replication or publishing environment.

**About this task**

You might consider suspending and resuming a monitor instead of stopping and restarting it in the following situations:

- You do not have the authority to stop and start a monitor.
- A server in your replication environment is being serviced. For example, if a monitor named MONITOR1 is monitoring SERVER_GREEN, which is a Q capture server, and SERVER_GREEN will be shut down for maintenance between 4 and 7 p.m., you could suspend MONITOR1 at 4 p.m. and resume it at 7 p.m. This prevents MONITOR1 from issuing a QCAPTURE_STATUS alert condition.

If you suspend the monitor while the Capture, Apply, Q Capture, or Q Apply programs are running, the monitor continues where it left off when you resume it. When a monitor is suspended and then resumed, it will not check for alert conditions or issue alerts for conditions that were met while the monitor was suspended.

**Procedure**

To suspend and resume a monitor:
1. Suspend the monitor by issuing the `asnmcmd suspend command`. The monitor stops checking for alert conditions.

2. Resume the monitor by issuing the `asnmcmd resume command`. The monitor resumes checking for alert conditions.

**Ending a monitor suspension**

You can end a monitor suspension before its regularly scheduled expiration time by removing the suspension from the monitor control tables and then reinitializing the monitor.

**Procedure**

To end a monitor suspension:

1. Use the DROP MONITOR SUSPENSION command in the ASNCLP command-line program to remove the suspension from the control tables at the monitor server.
   
   For example, the following command removes a suspension named SUSP1:
   
   ```
   DROP MONITOR SUSPENSION NAME SUSP1
   ```
   
2. Reinitialize the monitor by using one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnmcmd command</td>
<td>Use the asnmcmd reinit command to prompt the monitor to read its control tables for the most recent changes. The following command reinitializes the monitor identified by the monitor qualifier myqual at the Monitor control server wsdb:</td>
</tr>
<tr>
<td></td>
<td><code>asnmcmd monitor_server=wsdb monitor_qual=myqual reinit</code></td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Reinitialize Monitor window. In the contents pane, right-click the monitor qualifier that identifies the monitor that you want to reinitialize and click <strong>Reinitialize Monitor</strong>.</td>
</tr>
</tbody>
</table>

*Note:* You can also stop and then start the monitor to prompt it to read its control tables.

**Stopping monitors**

When you stop a monitor, it stops checking the replication or publishing programs for alert conditions. You can use the Replication Center, a system command, or a DB2 replication service to stop a monitor.

**About this task**

If the monitor stops while the Capture, Apply, Q Capture, or Q Apply programs are running, then the next time the monitor starts it performs the following actions:

- Checks for alert conditions that were met while the monitor was stopped.
- Issues alerts for any conditions that were met.

**Procedure**

To stop a monitor, use one of the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnmcmd stop command</td>
<td>Use this command to stop a monitor.</td>
</tr>
</tbody>
</table>
Method | Description
--- | ---
Replication Center | Use the Stop Monitor window to stop a monitor. To open the window, right-click the Monitor qualifier that identifies the monitor that you want to stop, and select **Stop Monitor**.
Windows services | Stop the replication service. The monitor stops automatically.

### Reviewing Monitor program messages

Use the Monitor Messages window to review the messages that were inserted in the IBM_SNAP_MONTRACE table over a specified period of time. The IBM_SNAP_MONTRACE table contains rows for significant events such as actions, warnings, and errors that are issued by the Monitor program.

For example, from the Monitor Messages window, you can review all the error and warning messages that are recorded by the Monitor program during one week. You can also print or save data to a file from the Monitor Messages window.

### Parameters of the Replication Alert Monitor

You can determine the behavior of the Replication Alert Monitor by setting values for various parameters.

#### Default values of Replication Alert Monitor parameters

When you use the replication administration tools to create Monitor control tables, default values are set for the monitor operating parameters.

<table>
<thead>
<tr>
<th>Operational parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert_prune_limit</td>
<td>10080 minutes</td>
</tr>
<tr>
<td>autopruner</td>
<td>Y</td>
</tr>
<tr>
<td>email_server</td>
<td>no default value</td>
</tr>
<tr>
<td>max_notification_minutes</td>
<td>60 minutes</td>
</tr>
<tr>
<td>max_notifications_per_alert</td>
<td>3</td>
</tr>
<tr>
<td>monitor_errors</td>
<td>no default value</td>
</tr>
<tr>
<td>monitor_interval</td>
<td>300 seconds</td>
</tr>
<tr>
<td>monitor_limit</td>
<td>10080 minutes</td>
</tr>
<tr>
<td>monitor_path</td>
<td>the directory where the asnmon command was invoked.</td>
</tr>
<tr>
<td>runonce</td>
<td>N</td>
</tr>
<tr>
<td>trace_limit</td>
<td>10080 minutes</td>
</tr>
</tbody>
</table>

### Descriptions of the Replication Alert Monitor parameters

This topic describes the following parameters you can use to operate the Replication Alert Monitor:
alert_prune_limit

Default: alert_prune_limit=10080 minutes (seven days.)

When the Replication Alert Monitor starts a new monitor cycle, it prunes the rows from the IBMSNAP_ALERTS table that are eligible for pruning. By default, the Replication Alert Monitor prunes the rows that are older than 10080 minutes (seven days). The alert_prune_limit parameter controls how much old data the Replication Alert Monitor stores in the table. The parameter specifies how old the data must be before the Replication Alert Monitor prunes it.

You can reduce the value of alert_prune_limit parameter if the storage space on your system is small for the IBMSNAP_ALERTS table. A lower prune limit saves space, but increases processing costs. Alternatively, you might want to increase the value for the alert_prune_limit parameter to maintain a history of all the alert activity. In SQL replication only, a higher prune limit requires more space for change-data (CD) tables and UOW tables, but decreases processing costs.

autoprune

Default: autoprune=y

The autoprune parameter controls automatic pruning. The Replication Alert Monitor automatically prunes rows from the IBMSNAP_ALERTS table that it has already copied into the Monitor control tables.

email_server

The email_server parameter enables the ASNMAIL exit routine. The default ASNMAIL routine enables the Replication Alert Monitor to send alerts by using e-mail. Set the value of this parameter to the address of an e-mail server that is set to use the Simple Mail Transfer Protocol (SMTP).

max_notification_minutes

Default: max_notifications_minutes=60

The max_notification_minutes parameter specifies how long that a monitor will track an alert condition to see if it occurs more than once. By default, if an alert condition occurs more than once within 60 minutes, the Replication Alert Monitor will send a maximum of three alerts during the 60 minute period. The
The **max_notifications_per_alert** parameter tells the Monitor how many notifications to send during the period of time specified by the **maxNotificationsMinutes** parameter for any alert condition.

**max_notifications_per_alert**

Default: **max_notifications_per_alert**=3

The **max_notifications_per_alert** parameter tells the Replication Alert Monitor the maximum number of notifications to send for any one alert. By default, if the Replication Alert Monitor receives an alert condition more than once, it sends a maximum of three notifications for that alert condition in a period of 60 minutes.

**monitor_errors**

The Replication Alert monitor stores any errors that occur in the monitoring process. One example of an operational error is when the Replication Alert Monitor cannot connect to the Monitor control server. You must specify an e-mail address for the **monitor_errors** parameter if you want to receive notification of operational errors. If you do not specify an e-mail address, the Replication Alert Monitor logs operational errors, but it does not send notification of the errors.

The Replication Alert Monitor ignores the **monitor_errors** parameter if the **email_server** parameter does not describe a valid e-mail server.

**monitor_interval**

Default: **monitor_interval**=300 seconds (5 minutes)

The **monitor_interval** parameter tells the Replication Alert Monitor how often to check for alert conditions. By default, the Replication Alert Monitor checks for all alert conditions for the specific monitor on the server every 300 seconds.

**monitor_limit**

Default: **monitor_limit**=10080 minutes (7 days)

For Q replication, the **monitor_limit** parameter specifies how long to keep rows in the IBMQREP_CAPMON and the IBMQREP_CAPQMON tables before the Q Capture program can prune them. For SQL replication, the **monitor_limit** parameter specifies how long to keeps rows in the IBMSNAP_CAPMON table before the Q Capture program can prune them. At each pruning interval, the Capture and Q Capture programs prune rows in these tables if the rows are older than this limit based on the current timestamp.

**monitor_path**

Default: **monitor_path**=the directory where the asnmon command was invoked

The **monitor_path** parameter specifies the location of the log files that the Replication Alert Monitor uses.

**runonce**

Default: **runonce**=n
When you start the Replication Alert Monitor, by default, it runs at intervals to monitor any alert conditions that you selected. You can schedule the Replication Alert Monitor to run hourly, at some other time interval, or even just one time.

When you specify runonce=y, the Replication Alert Monitor checks one time for all the alert conditions that you selected and ignores the monitor_interval parameter. You can use runonce when you run the Replication Alert Monitor in a batch process. For example, after the Apply program completes, you can use runonce=y to determine if any subscription sets failed. Then, if a subscription set did fail, the Replication Alert Monitor sends notification to your contact person or group.

By default, the monitor_interval is 300 seconds (five minutes). The Replication Alert Monitor checks for all the alert conditions for each monitor on the server every 300 seconds. If the Replication Alert Monitor finds an alert condition, it sends notification.

**trace_limit**

*Default: trace_limit=10080 minutes (7 days)*

The trace_limit parameter tells the Replication Alert Monitor how often to prune the IBMSNAP_MONTRACE and IBMSNAP_MONTRAIL tables. The Replication Alert Monitor stores the rows in these tables for 10080 minutes (seven days). The Replication Alert Monitor prunes any rows older than the value that you specify for the trace_limit parameter.

### Changing runtime parameters for the Replication Alert Monitor

You can change runtime parameters for the Replication Alert Monitor when you start the monitor or while the monitor is running.

**About this task**

You set initial parameter values when you create a monitor. These values are stored in the IBMSNAP_MONPARMS control table. When you start the monitor, it reads this control table and uses the parameter values.

You can override the saved values at runtime when you start the monitor or while the monitor is running. Any runtime values that you set will only last for the current run. If the monitor is stopped and restarted, it uses the values saved in the control table.

**Procedure**

1. Change parameters when you start the monitor. Use one of the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnmon system command</td>
<td>Specify one or more parameters and values when you use this command to start the monitor.</td>
</tr>
<tr>
<td>Replication Center</td>
<td>Use the Specify Monitor Startup Parameters window. To open the window, right-click a Monitor qualifier in the contents pane that identifies the monitor that you want to start, and click Start Monitor. Then click Parameters on the Start Monitor window.</td>
</tr>
</tbody>
</table>

2. Use the asnmcmd chgparms system command to change parameters while the monitor is running. You can change the following parameters:
Specifying how often the Replication Alert Monitor runs

You must decide how often the Replication Alert Monitor will check for alert conditions for your replication environment.

Procedure

To specify how often the Replication Alert Monitor runs, use the following methods:

- Use the `runonce` parameter of the `asnmon` command to specify if the Replication Alert Monitor will run repeatedly or only once.
- Use the `monitor_interval` parameter of the `asnmon` command to specify how often the Replication Alert Monitor will run when `runonce`=n.
- Use the Replication Center to specify run times when you start the Replication Alert Monitor.

Specifying notification criteria for selected alert conditions

The Replication Alert Monitor stores any alert conditions that you select. You can set up notification parameters to notify a contact of the alert conditions automatically with electronic mail (e-mail).

Procedure

To specify notification criteria for alert conditions, use the following methods:

- Set the `max_notifications_per_alert` parameter to control the maximum notification for a particular time period. Specify the maximum number of notifications you want to receive about a particular alert condition within the time period specified by the `max_notifications_minutes` parameter.
- Set the `email_server` parameter to enable DB2 to notify you by e-mail when an alert condition occurs. Set the value of this parameter to the address of an e-mail server by using the SMTP protocol.
- Optional: Write your own extensions to the `ASNMAIL` exit routine to customize how alert conditions are handled. This option is useful for integrating with problem management and other systems.

Specifying notification criteria for operational errors

The Replication Alert Monitor sends notification if it causes an error during its operation.

Procedure

To specify notification criteria for operational errors, set the value of the `monitor_errors` parameter to an e-mail address. The monitor will send notification of operational errors that it causes to this address. Enter the e-mail address by using the Simple Mail Transfer Protocol (SMTP) protocol.
Specifying prune intervals for data from the Replication Alert Monitor

The Replication Alert Monitor can automatically prune your Monitor control tables. You must decide whether the Monitor will prune the tables automatically, and if so, how the Monitor will prune the tables.

Procedure

To specify how often to prune your monitor tables, use the following methods:

• Specify whether you want the Replication Alert Monitor to automatically prune its control tables by using the **autoprune** parameter.

• Change the value for the **alert_prune_limit** parameter to control how much historic data you want the Replication Alert Monitor to store in the table. Specify how old the data must be before the Replication Alert Monitor prunes it from the IBMSNAP_ALERTS table.

• Change the value for the **trace_limit** parameter to control how long the Replication Alert Monitor stores rows in your monitor tables.
Chapter 19. 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:

   ```sql
   SELECT max(RESTART_SEQ)
   FROM schema.IBMQREP_CAPMON
   WITH UR;
   ```

   This is an example of a log sequence number:
   000555551F031230000

   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.LOLOGCOPY1.DS02
   ```
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/SQLO0002/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 28 lists control tables that the Q Capture program prunes that can grow regularly.

Table 28. Control tables that the Q Capture program prunes

<table>
<thead>
<tr>
<th>Control table</th>
<th>Parameter that specifies which rows are eligible for pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMQREP_CAPMON</td>
<td>monitor_limit</td>
</tr>
<tr>
<td>IBMQREP_CAPQMON</td>
<td>monitor_limit</td>
</tr>
<tr>
<td>IBMQREP_CAPTRACE</td>
<td>trace_limit</td>
</tr>
<tr>
<td>IBMQREP_SIGNAL</td>
<td>signal_limit</td>
</tr>
</tbody>
</table>

The prune_interval parameter specifies how often the Q Capture program checks for rows that are eligible for pruning.

Table 29 lists control tables that the Q Apply program prunes that can grow regularly.

Table 29. Control tables that the Q Apply program prunes

<table>
<thead>
<tr>
<th>Control table</th>
<th>Parameter that specifies which rows are eligible for pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMQREP_APPLYMON</td>
<td>monitor_limit</td>
</tr>
<tr>
<td>IBMQREP_APPLYTRACE</td>
<td>trace_limit</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System command</strong></td>
<td>Use the <code>prune</code> 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:</td>
</tr>
<tr>
<td></td>
<td>Q Capture control tables</td>
</tr>
<tr>
<td></td>
<td><code>asnqc</code>cmd</td>
</tr>
<tr>
<td></td>
<td>Q Apply control tables</td>
</tr>
<tr>
<td></td>
<td><code>asnqc</code>cmd</td>
</tr>
<tr>
<td></td>
<td>Monitor control tables</td>
</tr>
<tr>
<td></td>
<td><code>asnmc</code>cmd</td>
</tr>
<tr>
<td><strong>Replication Center</strong></td>
<td>Use one of the following windows:</td>
</tr>
<tr>
<td>Q Capture control tables</td>
<td>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.</td>
</tr>
<tr>
<td>Q Apply control tables</td>
<td>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.</td>
</tr>
<tr>
<td>Monitor control tables</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

- Use the REORG utility with the PREFORMAT option. The PREFORMAT option speeds up the insert processing of the Q Capture program.
- 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place a Q subscription into</td>
<td>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 <code>spillsub</code> and <code>resumesub</code> 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 <code>asnqacmd</code> command on Linux, UNIX, Windows, and UNIX System Services on z/OS. For example:</td>
</tr>
<tr>
<td>spill mode</td>
<td><strong>MODIFY</strong></td>
</tr>
<tr>
<td></td>
<td><code>f myqapp,spillsub=receive_queue_name:q_subscription_name</code></td>
</tr>
<tr>
<td></td>
<td><code>asnqacmd</code></td>
</tr>
<tr>
<td></td>
<td><code>asnqacmd apply_server=server_name apply_schema=schema</code></td>
</tr>
<tr>
<td></td>
<td><code>spillsub=receive_queue_name:q_subscription_name</code></td>
</tr>
<tr>
<td></td>
<td><strong>Restriction:</strong> If your tables have referential integrity constraints, you must use another method such as stopping message processing on the receive queue.</td>
</tr>
</tbody>
</table>

Restriction: If your tables have referential integrity constraints, you must use another method such as stopping message processing on the receive queue.
Option | Description
--- | ---
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, ASNCAPTURE, 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 20. Table differencing and repair

The asndiff and asntrep utilities allow you to detect and repair differences between source and target tables in Q replication and SQL replication without manually comparing the tables or performing a load (full refresh) of the target.

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 asndiff and asntrep utilities 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 utilities do not depend on logs, triggers, or isolation level.

Procedure

To detect and repair differences between source and target tables, run the asndiff utility, and then run the asntrep utility.

Table difference utility (asndiff)

The asndiff utility compares all columns in a source table to their corresponding columns in a target table and generates a list of differences between the two tables in the form of a DB2 table.

To use the asndiff utility, you run the asndiff 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.

The following sections explain how to use the asndiff command:

- "Overview of the asndiff command"
- "Difference table" on page 298
- "Suppressed delete operations" on page 299
- "Different data types in sources and targets" on page 300
- "Comparing the GRAPHIC data type" on page 300
- "Predicates" on page 300
- "When to use the asndiff utility" on page 300
- "Using an input file to specify tables to compare" on page 301
- "Specifying location and size of temporary files or data sets" on page 301

Overview of the asndiff command

You can run the asndiff 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 asndiff command can be used with federated sources and targets if the corresponding columns in the two tables have the same data types.
Note: 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 table and not a stored procedure. For SQL replication, the target must be a user table, point-in-time table, replica table, or user-copy table.

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:

```sql
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:

```sql
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.

**Difference table**
The asntdiff command creates a difference table in the source database or subsystem for Q replication and SQL replication.

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.

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.

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.
Different data types in sources and targets

The asntdiff utility builds 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.

Comparing the GRAPHIC data type

Columns with the GRAPHIC data type at the source and target might not match when you use the asntdiff 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 asntdiff utility compares the GRAPHIC data in a consistent manner.

Predicates

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.

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.

When to use the asntdiff utility

The best time to use the asntdiff 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.

Using an input file to specify tables to compare

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 374.

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"
```

Specifying location and size of temporary files or data sets

The asntdiff command creates temporary files or data sets for spilling data and for writing differences before inserting them into the difference table. You specify the location of the temporary files or data sets differently, depending on the platform:

**z/OS**

On z/OS, the temporary files are written by default to the UNIX System Services (USS) 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.

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:
You can use the TMPDIR environment variable to specify the location of the temporary spill files.

**Table repair utility (asntrep)**

The asntrep utility repairs differences between source and target tables on all DB2 servers by deleting, inserting, and updating rows in the target table. The utility runs on Linux, UNIX, or Windows operating systems.

The asntrep utility uses the difference table that is generated by the asntdiff utility to do the following:

- 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.

To use the asntrep utility, 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.

The asntrep command does not drop the difference table from the target database or subsystem. You must drop the table manually.

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.

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.
Chapter 21. 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

Running the Q replication and event publishing programs by using JCL

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.D7DPJAY.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, 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', 300, 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, 300, 5000,
'N', 'N', 'Y', ''/SYSADM.XYZ'', '901');
```

**Starting the Q Capture program with JCL**

---

304  Replication and Event Publishing Guide and Reference
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'
```
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 30. Sample jobs to start the replication programs in JCL

<table>
<thead>
<tr>
<th>Sample</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASNQSTRA</td>
<td>Q Apply</td>
</tr>
<tr>
<td>ASNQSTRC</td>
<td>Q Capture</td>
</tr>
<tr>
<td>ASNSTRM</td>
<td>Replication Alert Monitor</td>
</tr>
<tr>
<td>ASNQTON</td>
<td>Trace (for the Q Capture program or the QApply program)</td>
</tr>
</tbody>
</table>
```

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 asncmd: Working with a running Q Capture program, asnacmd: 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: asnqcmd, 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 31 shows the Q Capture commands that you can run with the `f` keyword. In all examples, the job name is myqcap.

**Table 31. Sample MODIFY commands for the Q Capture program**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sample command that uses f keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>prune</td>
<td>f myqcap,prune</td>
</tr>
<tr>
<td>qryparms</td>
<td>f myqcap,qryparms</td>
</tr>
<tr>
<td>reinit</td>
<td>f myqcap,reinit</td>
</tr>
<tr>
<td>reinitq</td>
<td>f myqcap,reinitq=send.queue.name</td>
</tr>
<tr>
<td>startq</td>
<td>f myqcap,startq=send.queue.name</td>
</tr>
<tr>
<td>startq all</td>
<td>f myqcap,startq all</td>
</tr>
<tr>
<td>stopq</td>
<td>f myqcap,stopq=send.queue.name</td>
</tr>
<tr>
<td>status</td>
<td>f myqcap,status</td>
</tr>
<tr>
<td>status show details</td>
<td>f myqcap,status show details</td>
</tr>
<tr>
<td>stop</td>
<td>f myqcap,stop</td>
</tr>
<tr>
<td>chgparms</td>
<td>f myqcap,chgparms autostop=y</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms commit_interval=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms logreuse=y</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms logstdout=y</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms monitor_interval=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms monitor_limit=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms prune_interval=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms qfull_num_retries=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms qfull_retry_delay=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms sleep_interval=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms signal_limit=n</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms term=y</td>
</tr>
<tr>
<td></td>
<td>f myqcap,chgparms trace_limit=n</td>
</tr>
</tbody>
</table>

Table 32 shows the Q Apply commands that you can run with the `f` keyword. In all examples, the job name is myqapp.

**Table 32. Sample MODIFY commands for the Q Apply program**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sample command that uses f keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>prune</td>
<td>f myqapp,prune</td>
</tr>
<tr>
<td>qryparms</td>
<td>f myqapp,qryparms</td>
</tr>
<tr>
<td>stopq</td>
<td>f myqapp,stopq=receive.queue.name</td>
</tr>
<tr>
<td>startq</td>
<td>f myqapp,startq=receive.queue.name</td>
</tr>
</tbody>
</table>
Table 32. Sample MODIFY commands for the Q Apply program (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sample command that uses f keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>startq;skiptrans</td>
<td>f myqapp,startq=&quot;receive.queue.name;skiptrans=transaction.ID&quot;</td>
</tr>
<tr>
<td>reinitq</td>
<td>f myqapp,reinit=&quot;receive.queue.name&quot;</td>
</tr>
<tr>
<td>stop</td>
<td>f myqapp,stop</td>
</tr>
<tr>
<td>status</td>
<td>f myqapp,status</td>
</tr>
<tr>
<td>status show details</td>
<td>f myqapp,status show details</td>
</tr>
<tr>
<td>spillsub</td>
<td>f myqapp,spillsub=&quot;receive.queue.name:q.sub.name&quot;</td>
</tr>
<tr>
<td>resumesub</td>
<td>f myqapp,resumesub=&quot;receive.queue.name:q.sub.name&quot;</td>
</tr>
<tr>
<td>chgparms</td>
<td>f myqapp,chgparms autostop=y</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms logreuse=y</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms logstdout=y</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms monitor_interval=n</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms monitor_limit=n</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms prune_interval=n</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms term=y</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms trace_limit=n</td>
</tr>
<tr>
<td></td>
<td>f myqapp,chgparms deadlock_retries=n</td>
</tr>
</tbody>
</table>

Table 33 shows asntrc program commands that you can run with the f keyword. In all examples, the job name is myqcap.

Table 33. Sample MODIFY commands for the asntrc program

<table>
<thead>
<tr>
<th>Task</th>
<th>Sample command that uses f keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start a program trace with the asntrc command</td>
<td>f myqcap,asntrc on</td>
</tr>
<tr>
<td></td>
<td>f myqcap,asntrc statlong</td>
</tr>
<tr>
<td>Format an asntrc fmt report and direct the output to a z/OS data set</td>
<td>F myqcap, asntrc fmt -ofn '&lt;USRT001.TRCFMT&gt;'</td>
</tr>
<tr>
<td>Format an asntrc flw report and direct the output to a z/OS data set</td>
<td>F myqcap, asntrc flw -ofn '&lt;USRT001.TRCFLW&gt;'</td>
</tr>
<tr>
<td>Stop a program trace</td>
<td>F myqcap, asntrc off</td>
</tr>
</tbody>
</table>

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 34 on page 309 shows the Replication Alert Monitor commands that you can run with the f keyword. In all examples, the job name is mymon.
Table 34. Sample MODIFY commands for the Replication Alert Monitor program

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sample command that uses f keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit</td>
<td>f mymon, reinit</td>
</tr>
<tr>
<td>status</td>
<td>f mymon, status</td>
</tr>
<tr>
<td>qryparms</td>
<td>f mymon, qryparms</td>
</tr>
<tr>
<td>suspend</td>
<td>f mymon, suspend</td>
</tr>
<tr>
<td>resume</td>
<td>f mymon, resume</td>
</tr>
<tr>
<td>stop</td>
<td>f mymon, stop</td>
</tr>
<tr>
<td>chgparms</td>
<td>f mymon, chgparms</td>
</tr>
<tr>
<td></td>
<td>monitor_interval=n</td>
</tr>
<tr>
<td></td>
<td>chgparms autoprune=y</td>
</tr>
<tr>
<td></td>
<td>chgparms trace_limit=n</td>
</tr>
<tr>
<td></td>
<td>chgparms alert_prune_limit=n</td>
</tr>
<tr>
<td></td>
<td>chgparms max_notifications_per_alert=n</td>
</tr>
<tr>
<td></td>
<td>chgparms max_notifications_minutes=n</td>
</tr>
</tbody>
</table>

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=xmhlqual.SIXMMOD1,DISP=SHR
   //CAPSPILL DD DSN=&&CAPSPILL,DISP=(NEW,DELETE,DELETE),
   // UNIT=VIO, SPACE=(CYL,(50,70)),
   // DCB=(RECFM=VB,BLKSIZ=6404)
   //MSGS DD PATH='/usr/lpp/db2repl_09_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
   ```
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.

The Automatic Restart Manager (ARM) recovery system

You can use the Automatic Restart Manager (ARM) recovery system on z/OS to restart the Q Capture program, Q Apply program, and Replication Alert Monitor.

ARM can restart a job that failed or task that failed without operator intervention. ARM can also restart a job or a task that is running on a system that has failed.

ARM uses element names to identify applications. Each application that is set up to use ARM generates a unique element name for itself that it uses in all communication with ARM. ARM tracks the element name and defines its restart policy in terms of element names. For details about setting up ARM, see z/OS MVS Sysplex Services Guide

Table 35 shows the element names to use for each of the replication programs when you configure ARM.

<table>
<thead>
<tr>
<th>Replication program</th>
<th>Element name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q Capture</td>
<td>ASNQCxxxxyyyyyy</td>
</tr>
<tr>
<td>Q Apply</td>
<td>ASNQAxxxxyyyyyy</td>
</tr>
<tr>
<td>Replication Alert Monitor</td>
<td>ASNAMxxxxyyyyyy</td>
</tr>
</tbody>
</table>

In Table 35, xxxx represents the DB2 subsystem name, and yyyy represents the data-sharing member name. For configurations that do not use data-sharing, replace yyyy with blanks. The element name must be 16 characters long. Element names must be unique in the entire sysplex; therefore, to use ARM, you can run only one instance of a particular program per subsystem.

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. ARM does not need the event exit because the replication programs do not run as an MVS subsystem.

ARM restarts registered programs if they terminate abnormally. For example, ARM would restart the Q Capture program if a segment violation occurs during operation. A registered replication program de-registers if it terminates normally or if it encounters an invalid registration. For example, the Q Apply program would de-register if it terminates because of a STOP command.

Automatically restart Q replication and event publishing programs

310  Replication and Event Publishing Guide and Reference
You can set up the Automatic Restart Manager (ARM) to restart the Q Apply program, Q Capture program, and the Replication Alert Monitor when replicating data on the z/OS operating system.

About this task

If you start the Q Capture or Q Apply program using the parameter TERM=N, the program does not stop when DB2 is quiesced. In this case, the program does not de-register from ARM. It continues to run but does not capture data until DB2 is restarted.

Procedure

To set up the Automatic Restart Manager for Q replication and event publishing programs:

1. Install the ARM. See z/OS MVS Sysplex Services Guide for details.
2. Set up the replication programs.
3. Copy the appropriate load module into an APF-authorized library. The Q Capture program must be APF authorized even if you are not using the ARM.

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 36 describes the inputs for the replication service name.

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>The name of the DB2 instance.</td>
</tr>
<tr>
<td>alias</td>
<td>The database alias of the Q Capture server, Q Apply server, Capture control server, Apply control server, or Monitor control server.</td>
</tr>
</tbody>
</table>
Table 36. Inputs for the replication service name (continued)

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
<td>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).</td>
</tr>
<tr>
<td>qualifier_or_schema</td>
<td>One of the following identifiers: Q Capture schema, Q Apply schema, Capture schema, Apply qualifier, or Monitor qualifier.</td>
</tr>
</tbody>
</table>

**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**

**Windows**

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.

**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 `asncrt` 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

**Windows**

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

**Windows**

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, asncmd, asncmcmd, 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

**Windows**

You can view a list of all your replication services and their properties by using the `asnlst` 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 37 shows commands that are used to start the replication programs at 3:00 p.m. on Friday.

<table>
<thead>
<tr>
<th>Replication program</th>
<th>Linux or UNIX command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q Capture</td>
<td>at 3pm Friday <code>asnqcap autoprune=n</code></td>
</tr>
<tr>
<td>Q Apply</td>
<td>at 3pm Friday <code>asnqapply applyqual=myqual</code></td>
</tr>
<tr>
<td>Replication Alert Monitor</td>
<td>at 3pm Friday <code>asnmom</code></td>
</tr>
<tr>
<td></td>
<td><code>monitor_server=db2srv1</code></td>
</tr>
<tr>
<td></td>
<td><code>monitor_qualifier=mymon</code></td>
</tr>
</tbody>
</table>

**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:
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 38 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>
<thead>
<tr>
<th>Replication program</th>
<th>Windows command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q Capture</td>
<td>c:&gt;at 15:00 db2cmd asnqcap</td>
</tr>
<tr>
<td></td>
<td>capture_server=qcapdb</td>
</tr>
<tr>
<td></td>
<td>capture_schema=schema</td>
</tr>
<tr>
<td></td>
<td>capture_path=c:\capture ^&gt;</td>
</tr>
<tr>
<td></td>
<td>c:\capture\asnqcap.out</td>
</tr>
<tr>
<td>Q Apply</td>
<td>c:&gt;at 15:00 db2cmd asnqapp</td>
</tr>
<tr>
<td></td>
<td>apply_server=qappdb</td>
</tr>
<tr>
<td></td>
<td>apply_schema=applyqual</td>
</tr>
<tr>
<td></td>
<td>apply_path=c:\apply ^&gt;</td>
</tr>
<tr>
<td></td>
<td>c:\apply\asnqapp.out</td>
</tr>
<tr>
<td>Replication Alert Monitor</td>
<td>c:&gt;at 15:00 db2cmd asnmon</td>
</tr>
<tr>
<td></td>
<td>monitor_server=mondb</td>
</tr>
<tr>
<td></td>
<td>monitor_qual=monqual monitor_path=c:\</td>
</tr>
<tr>
<td></td>
<td>monitor ^&gt; c:\monitor\asnmon.out</td>
</tr>
</tbody>
</table>

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 22. 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 39 lists the limits for names of objects in Q replication and event publishing.

Table 39. Name limits for objects in Q replication and event publishing

<table>
<thead>
<tr>
<th>Object</th>
<th>Name limits</th>
<th>Length limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source and target</td>
<td><strong>DB2</strong>: The names of DB2 source tables and target tables must follow the</td>
<td><strong>z/OS</strong></td>
</tr>
<tr>
<td>tables</td>
<td>naming rules for DB2 table names.</td>
<td>Both short and long</td>
</tr>
<tr>
<td></td>
<td><strong>non-DB2</strong>: The names of non-DB2</td>
<td>schema names are</td>
</tr>
<tr>
<td></td>
<td>target tables must follow the table</td>
<td>supported for tables</td>
</tr>
<tr>
<td></td>
<td>naming rules that are required by the</td>
<td>on DB2 for z/OS.</td>
</tr>
<tr>
<td></td>
<td>DB2 federated server to set up</td>
<td>Table names can</td>
</tr>
<tr>
<td></td>
<td>nicknames.</td>
<td>include up to:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 18 bytes for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subsystems that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>are running DB2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for z/OS Version 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compatibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode or earlier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 128 bytes for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subsystems that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>are running DB2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for z/OS Version 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new-function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Linux UNIX Windows</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>30 or fewer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>characters</td>
</tr>
<tr>
<td>Object</td>
<td>Name limits</td>
<td>Length limit</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Source and target columns</td>
<td><strong>DB2</strong>: 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.</td>
<td>Replication and publishing do not support source table column names that are longer than 30 characters (15 characters for double-byte languages). 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 29 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.</td>
</tr>
</tbody>
</table>
| Table owner                 | Both short and long schema names are supported for table owner for DB2 for z/OS. | z/OS: The names of the table owner can include up to:  
  - 30 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 |
| 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 |
Table 39. Name limits for objects in Q replication and event publishing (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Name limits</th>
<th>Length limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart queue</td>
<td>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.</td>
<td>48 or fewer characters</td>
</tr>
<tr>
<td>Q subscription</td>
<td>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 (;).</td>
<td>30 or fewer characters</td>
</tr>
<tr>
<td>SUBGROUP</td>
<td>The name of the SUBGROUP for bidirectional and peer-to-peer replication can include any characters that DB2 allows for VARCHAR data type columns. <strong>Recommendation:</strong> Use a unique group name for the set of Q subscriptions for a logical table.</td>
<td>30 or fewer characters</td>
</tr>
<tr>
<td>Publication</td>
<td>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 (;).</td>
<td>30 or fewer characters</td>
</tr>
</tbody>
</table>
### Table 39. Name limits for objects in Q replication and event publishing  (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Name limits</th>
<th>Length limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q Capture schema</td>
<td>The name of the Q Capture schema can include only the following valid characters:</td>
<td>The name of the Q Capture schema can be a string of:</td>
</tr>
<tr>
<td></td>
<td>• A through Z (uppercase letters)</td>
<td>• Linux UNIX Windows: 30 or fewer characters</td>
</tr>
<tr>
<td></td>
<td>• a through z (lowercase letters)</td>
<td>• z/OS Subsystems that are running Version 8 compatibility mode or earlier: 18 or fewer characters</td>
</tr>
<tr>
<td></td>
<td>• Numerals (0 through 9)</td>
<td>• z/OS Subsystems that are running Version 8 new-function mode: 128 or fewer characters</td>
</tr>
<tr>
<td></td>
<td>• The underscore character ( _ )</td>
<td></td>
</tr>
<tr>
<td>Q Apply schema</td>
<td>The name of the Q Apply schema can include only the following valid characters:</td>
<td>The name of the Q Apply schema can be a string of:</td>
</tr>
<tr>
<td></td>
<td>• A through Z (uppercase letters)</td>
<td>• Linux UNIX Windows: 30 or fewer characters</td>
</tr>
<tr>
<td></td>
<td>• a through z (lowercase letters)</td>
<td>• z/OS Subsystems that are running Version 8 compatibility mode or earlier: 18 or fewer characters</td>
</tr>
<tr>
<td></td>
<td>• Numerals (0 through 9)</td>
<td>• z/OS Subsystems that are running Version 8 new-function mode: 128 or fewer characters</td>
</tr>
<tr>
<td></td>
<td>• The underscore character ( _ )</td>
<td></td>
</tr>
<tr>
<td>Monitor qualifier</td>
<td>The name of the monitor qualifier can include only the following valid characters:</td>
<td>The name of the monitor qualifier can be a string of 18 or fewer characters.</td>
</tr>
<tr>
<td></td>
<td>• A through Z (uppercase letters)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• a through z (lowercase letters)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Numerals (0 through 9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The underscore character ( _ )</td>
<td></td>
</tr>
</tbody>
</table>

**How lowercase object names are handled for Q replication and event 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 (\).

**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 23. 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 \texttt{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 not recommended. For example, specifying \texttt{logreuse} is the same as \texttt{logreuse=y}. But to specify no logreuse, you must use \texttt{logreuse=n}.

Invoking commands with a question mark (for example, \texttt{asnqapp ?}), displays a help message that shows the command syntax.

Table \ref{tab:systemcommands} helps you match common tasks with the system commands.

\begin{table}[ht]
\centering
\begin{tabular}{|l|l|}
\hline
If you want to ... & Use this system command \\
\hline
Start a Q Capture program and specify startup parameters (Linux, UNIX, Windows, z/OS) & \texttt{asnqcap: Starting a Q Capture program} on page 324 \\
\hline
Work with a running Q Capture program (Linux, UNIX, Windows, z/OS) & \texttt{asnqcmd: Working with a running Q Capture program} on page 332 \\
\quad • Check parameter values & \\
\quad • Change parameters & \\
\quad • Prune the control tables & \\
\quad • Check Q Capture status & \\
\quad • Stop Q Capture & \\
\quad • Reinitialize all Q subscriptions or publications & \\
\quad • Reinitialize one send queue & \\
\hline
Start a Q Apply program and specify startup parameters (Linux, UNIX, Windows, z/OS) & \texttt{asnqapp: Starting a Q Apply program} on page 336 \\
\hline
Work with a running Q Apply program (Linux, UNIX, Windows, z/OS) & \texttt{asnqacmd: Working with a running Q Apply program} on page 348 \\
\quad • Check parameter values & \\
\quad • Change parameters & \\
\quad • Check Q Apply status & \\
\quad • Prune the control tables & \\
\quad • Stop Q Apply & \\
\quad • Stop Q Apply reading from a queue & \\
\quad • Start Q Apply reading from a queue & \\
\quad • Reinitialize one receive queue & \\
\hline
Operate the Q Replication Analyzer (Linux, UNIX, Windows) & \texttt{asnqanalyze: Operating the Q Replication Analyzer} on page 387 \\
\hline
\end{tabular}
\caption{Q replication and event publishing tasks and their corresponding system commands}
\end{table}
Table 40. Q replication and event publishing tasks and their corresponding system commands (continued)

<table>
<thead>
<tr>
<th>If you want to ...</th>
<th>Use this system command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format and view WebSphere MQ messages that are used in Q replication and event publishing (Linux, UNIX, Windows, z/OS)</td>
<td><code>asnqmfmt: Formatting and viewing Q replication and event publishing messages</code> on page 395</td>
</tr>
<tr>
<td>Format and view delimited or XML messages that are used in event publishing on z/OS.</td>
<td><code>asnqxmfmt: Formatting and viewing event publishing messages (z/OS)</code> on page 397</td>
</tr>
</tbody>
</table>

**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=dbname
               [capture_schema=schema]
               [capture_path=path]
               [add_partition=y]
               [arm=identifier]
               [autostop=y]
               [caf=y]
               [commit_interval=n]
               [commit_exe=commit_exe]
               [logrdbufsz=n]
               [logreuse=y]
               [logstdout=y]
               [lsn]=formatted_lsn
               [maxcmtseq]=formatted_lsn
               [memory_limit=n]
               [msg_persistence=n]
               [migrate=n]
               [monitor_interval=n]
               [monitor_limit=n]
               [monitor_monitored=n]
               [prune_interval=n]
               [prune_lock=y]
               [pwdfile=filename]
```

---

Page 324  Replication and Event Publishing Guide and Reference
Parameters

The following table defines the invocation parameters for the asnqcap command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_server</td>
<td>Specifies the name of the database or subsystem that contains the Q Capture control tables.</td>
</tr>
<tr>
<td>capture_schema</td>
<td>Specifies a name that identifies the Q Capture program that you want to start.</td>
</tr>
<tr>
<td>capture_path</td>
<td>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 asnqcap command. This location is an absolute path name, and you must enclose it in double quotation marks to preserve case.</td>
</tr>
<tr>
<td>add_partition</td>
<td>Specifies whether the Q Capture program starts reading the log file for partitions that were added since the last time the Q Capture program was restarted.</td>
</tr>
</tbody>
</table>

- **n (default)**
  
  No new partitions have been added since the last time the Q Capture program was restarted.

- **y**
  
  The Q Capture program starts reading the log file on one or more of the new partitions. On each partition, the Q Capture program starts reading the log from the log sequence number (LSN) that was initially used the last time the database was started.
### Table 41. Definitions for asnqcap invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm=identifier</td>
<td>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: ASNQCxxxxyyyy (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.</td>
</tr>
<tr>
<td>autostop=y/n</td>
<td>Specifies whether a Q Capture program stops after reaching the end of the active DB2 log.</td>
</tr>
<tr>
<td></td>
<td><strong>n (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Capture program does not stop after reaching the end of the active DB2 log.</td>
</tr>
<tr>
<td></td>
<td><strong>y</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Capture program stops when it reaches the end of the active DB2 log.</td>
</tr>
<tr>
<td>caf=y/n</td>
<td>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. The caf =y option specifies that the replication program overrides the default RRS connect and runs with CAF connect.</td>
</tr>
<tr>
<td></td>
<td><strong>n (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Capture program uses Recoverable Resource Manager Services (RRS) to connect to the database.</td>
</tr>
<tr>
<td></td>
<td><strong>y</strong></td>
</tr>
<tr>
<td></td>
<td>Specifies that the Q Capture program overrides the default RRS connect and runs with CAF connect.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>commit_interval=n</td>
<td>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 500 milliseconds (0.5 seconds).</td>
</tr>
</tbody>
</table>
### Table 41. Definitions for asncap invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ignore_transid=</strong></td>
<td>Specifies that the Q Capture program ignores the transaction that is identified by <code>transaction_ID</code>. The transactions are not replicated or published. The value for <code>transaction_ID</code> is a 10-byte hexadecimal identifier in the following format:</td>
</tr>
<tr>
<td><strong>Tip:</strong></td>
<td>The shortened version <code>transid</code> is also acceptable for this parameter.</td>
</tr>
<tr>
<td><strong>lob_send_option=I/S</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>logrdbufsz=n</strong></td>
<td>Specifies the size of the buffer 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. The default value for DB2 for z/OS is 66KB; for DB2 for Linux, UNIX, and Windows the default is 256KB. For partitioned databases, Q Capture allocates a buffer of the size that is specified by <code>logrdbufsz</code> for each partition.</td>
</tr>
<tr>
<td><strong>logreuse=y/n</strong></td>
<td>Specifies whether a Q Capture program reuses or appends messages to its diagnostic log file. On z/OS the log file name is <code>capture_server.capture_schema.OCAP.1og</code>. On Linux, UNIX, and Windows the log file name is <code>db2instance.capture_server.capture_schema.OCAP.1og</code>.</td>
</tr>
<tr>
<td><strong>n (default)</strong></td>
<td>The Q Capture program appends messages to the log file, even after the Q Capture program is restarted.</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>On restart, the Q Capture program reuses its log file by clearing the file and then writing to the blank file.</td>
</tr>
<tr>
<td><strong>logstdout=y/n</strong></td>
<td>Specifies whether a Q Capture program sends log messages to both its diagnostic log file and the console.</td>
</tr>
<tr>
<td><strong>n (default)</strong></td>
<td>The Q Capture program directs most log messages to the log file only.</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>The Q Capture program sends log messages to both its log file and the console (stdout).</td>
</tr>
</tbody>
</table>

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 41. Definitions for asnqcap invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsn=formatted_lsn</td>
<td>Specifies 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. When specifying the lsn parameter, you also must specify the maxcmtseq parameter. 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 this 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. Specify lsn=FFFF:FFFF:FFFF:FFFF:FFFF and maxcmtseq=0000:0000:0000:0000:0000 to start from the end of the log without triggering a load (full refresh) of the target table. You can also specify the lsn value without colons to save space, for example LSN=FFFFFFFFFFFFFFFFFFFF.</td>
</tr>
<tr>
<td>maxcmtseq=formatted_lsn</td>
<td>The commit log record position of the last transaction that was successfully sent by the Q Capture program before shutdown. This log marker is internal to replication, subject to change, and 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) You can specify the maxcmtseq value with or without colons. For example, both of the following values are acceptable: maxcmtseq=41c2:2264:0000:0004:0000 maxcmtseq=41c22264000000040000</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>memory_limit=n</td>
<td>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 differs depending on the source platform:</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong></td>
</tr>
<tr>
<td></td>
<td>By default, the <code>memory_limit</code> 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.</td>
</tr>
<tr>
<td></td>
<td><strong>Linux UNIX Windows</strong></td>
</tr>
<tr>
<td></td>
<td>On Linux, UNIX, and Windows, a Q Capture program uses a maximum of 500 MB by default.</td>
</tr>
<tr>
<td>msg_persistence=y/n</td>
<td>Specifies whether a Q Capture program writes persistent (logged) messages to WebSphere MQ queues.</td>
</tr>
<tr>
<td></td>
<td><strong>y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>Q Capture write persistent messages to all queues. The messages are logged by the queue manager and can be recovered.</td>
</tr>
<tr>
<td></td>
<td><strong>n</strong></td>
</tr>
<tr>
<td></td>
<td>Q Capture write nonpersistent messages to all queues. The messages are not logged and cannot be recovered.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you specify <code>msg_persistence=n</code>, 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.</td>
</tr>
<tr>
<td>migrate=y/n</td>
<td>Specifies that the Q Capture program starts from the beginning of the log after DB2 is migrated.</td>
</tr>
<tr>
<td></td>
<td><strong>Attention:</strong> Use this option only the first time that Q Capture is started and specify <code>startmode=warmns</code>.</td>
</tr>
<tr>
<td>monitor_interval=n</td>
<td>Specifies how often, in milliseconds, a Q Capture program adds a row to the IBMQREP_CAPMON and IBMQREP_CAPQMON tables. The default is 60000 milliseconds (one minute).</td>
</tr>
<tr>
<td>monitor_limit=n</td>
<td>Specifies the number of minutes that rows remain in the IBMQREP_CAPMON and 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).</td>
</tr>
<tr>
<td>pwdfile=filename</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>This command searches for the password file in the directory specified by the <code>capture_path</code> parameter. If no <code>capture_path</code> parameter is specified, this command searches for the password file in the directory where the command was invoked.</td>
</tr>
</tbody>
</table>
### Table 41. Definitions for asnqcap invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>prune_interval=$n$</td>
<td>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).</td>
</tr>
<tr>
<td>qfull_num_retries=$n$</td>
<td>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.</td>
</tr>
<tr>
<td>qfull_retry_delay=$n$</td>
<td>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.</td>
</tr>
<tr>
<td>signal_limit=$n$</td>
<td>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).</td>
</tr>
<tr>
<td>sleep_interval=$n$</td>
<td>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 5000 milliseconds (5 seconds).</td>
</tr>
<tr>
<td>startallq=y/n</td>
<td>Specifies whether the Q Capture program activates all send queues during startup. You can use this parameter to keep a disabled send queue inactive.</td>
</tr>
<tr>
<td>y (default)</td>
<td>When the Q Capture program starts, it activates all send queues that are not already in active (A) state.</td>
</tr>
<tr>
<td>n</td>
<td>When the Q Capture program starts, it does not activate send queues that are in inactive (I) state.</td>
</tr>
</tbody>
</table>
Table 41. Definitions for asnqcap invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>startmode=mode</td>
<td>Specifies the actions that a Q Capture program takes when it starts.</td>
</tr>
<tr>
<td></td>
<td><strong>warnsi (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Capture program starts reading the log at the point where it left off, except 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 warnsi start mode ensures that the Q Capture program cold starts only when it initially starts.</td>
</tr>
<tr>
<td></td>
<td><strong>warmns</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>cold</strong></td>
</tr>
<tr>
<td></td>
<td>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 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.</td>
</tr>
<tr>
<td>term=y/n</td>
<td>Specifies whether the Q Capture program terminates if the source DB2 is unavailable (quiesced, stopped, etc.).</td>
</tr>
<tr>
<td></td>
<td><strong>y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Capture program terminates if DB2 is unavailable.</td>
</tr>
<tr>
<td></td>
<td><strong>n</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Capture program continues running if DB2 is unavailable. When DB2 is available, the Q Capture program starts in warmns mode and begins capturing at the point where it left off.</td>
</tr>
<tr>
<td>trace_limit=n</td>
<td>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).</td>
</tr>
</tbody>
</table>

Examples for asnqcap

The following examples illustrate how to use the asnqcap 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, 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
```

**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:

```
asnqcap 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:

```
asnqcap 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:

```
asnqcap 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 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:
```

**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:

```
asnqcap capture_server=testdb lsn=FFFF:FFFF:FFFF:FFFF:FFFF
maxcmtseq=0000:0000:0000:0000:0000:
```

**Example - 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
```

**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

```
asnqccmd capture_server=db_name capture_schema=schema
```

```

...capture_server, ...capture_schema
```

```
chgparms parameters
```

```
prune
qryparms
reinit
reinitq=send_queue
startq send_queue
stopq send_queue
status show details
stop
```

Parameters:

```
autostop=y
commit_interval=n
logreuse=y
logstdout=y
memory_limit=n
monitor_interval=n
monitor_limit=n
prune_interval=n
signal_limit=n
sleep_interval=n
term=y
trace_limit=n
```

Parameters

Table 42 on page 334 defines the invocation parameters for the asnqccmd command.
Table 42. Definitions for asncqcmd invocation parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture_server</td>
<td>Specifies the name of the database or subsystem that contains the Q Capture control tables.</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong> Specifies the name of the DB2 subsystem where the Q Capture program is running. For data sharing, the value of <code>capture_server</code> might be the group attach name that was used to run Q Capture in any LPAR without changing the JCL for the started task.</td>
</tr>
<tr>
<td></td>
<td><strong>Linux UNIX Windows</strong> If you do not specify a Q Capture server, this parameter defaults to the value from the <code>DB2DBDFT</code> environment variable.</td>
</tr>
<tr>
<td>capture_schema</td>
<td>Specifies a name that identifies the Q Capture program that you want to work with.</td>
</tr>
</tbody>
</table>
| chgparsms       | 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`  
|                 | **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. |
| 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. |
Table 42. Definitions for asncqcmd invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>qryparms</strong></td>
<td>Specify if you want the current operational parameter values for a Q Capture program written to the standard output (stdout).</td>
</tr>
<tr>
<td><strong>reinit</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>reinitq=send_queue</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>startq=send_queue/all</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>status</strong></td>
<td>Specify to receive messages that indicate the state of each Q Capture thread (main, administration, prune, holdl, transaction, and worker).</td>
</tr>
<tr>
<td><strong>show details</strong></td>
<td>Specify after the status parameter to view a more detailed report about Q Capture program status, with the following information:</td>
</tr>
<tr>
<td></td>
<td>• Whether the Q Capture program is running</td>
</tr>
<tr>
<td></td>
<td>• Time since the program started</td>
</tr>
<tr>
<td></td>
<td>• Location of the Q Capture diagnostic log</td>
</tr>
<tr>
<td></td>
<td>• Number of active Q subscriptions</td>
</tr>
<tr>
<td></td>
<td>• Value of the CURRENT_LOG_TIME and CURRENT_MEMORY. These values might be newer than what is inserted into the IBMQREP_CAPMON control table.</td>
</tr>
<tr>
<td></td>
<td>• Logical log sequence number of the last transaction that the Q Capture program published to a send queue</td>
</tr>
<tr>
<td></td>
<td>• Amount of memory in megabytes that Q Capture used during the latest monitor interval to build transactions from log records</td>
</tr>
<tr>
<td></td>
<td><strong>Linux/UNIX/WINDOWS</strong></td>
</tr>
<tr>
<td></td>
<td>• Path to DB2 log files</td>
</tr>
<tr>
<td></td>
<td>• Oldest DB2 log file needed for a Q Capture restart</td>
</tr>
<tr>
<td></td>
<td>• Current DB2 log file captured</td>
</tr>
<tr>
<td><strong>stop</strong></td>
<td>Specify to stop the Q Capture program in an orderly way and commit the messages that it processed up to that point.</td>
</tr>
</tbody>
</table>
Table 42. Definitions for asncqcmand invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stopq=send_queue/all</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

```
asnqcqcmd 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:

```
asnqcqcmd 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:

```
asnqcqcmd 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:

```
asnqcqcmd 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 ...
```
- `applyupto=gmt_timestamp`
- `autostop=n`
- `arm=identifier`
- `buffered_inserts=n`
- `classic_load_file_sz=n`
- `commit_count=n`
- `caf=y`
- `deadlock_retries=n`
- `dftmodel=model_spill_queue_name`
- `ignbaddata=y`
- `insert_bidi_signal=n`
- `loadcopy_path=filepath`
- `load_data_buff_sz=n`
- `logreuse=n`
- `logstdout=y`
- `max_parallel_loads=n`
- `monitor_interval=n`
- `monitor_limit=n`
- `nickname_commit_ct=n`
- `p2p_2nodes=y`
- `prune_interval=n`
- `asnpwd.aut=pwdfile`
- `richklvl=n`
- `term=n`
- `skiptrans=skiptrans-clause`
- `spill_commit_count=n`
- `trace_limit=n`

**skiptrans-clause::**

- `receive_queue=receive_queue`
- `transaction_ID=transaction_ID`
- `begin_transaction_ID=begin_transaction_ID`
- `end_transaction_ID=end_transaction_ID`
Parameters

Table 43 defines the invocation parameters for the asnqapp command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_server=dbname</td>
<td>Specifies the name of the database or subsystem that contains the Q Apply control tables.</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Linux/UNIX/W indows</strong></td>
</tr>
<tr>
<td>apply_schema=schema</td>
<td>Specifies a name that identifies the Q Apply program that you want to start.</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong></td>
</tr>
<tr>
<td>apply_path=path</td>
<td>Specifies the location where you want a Q Apply program to write its log and work files. The default is the directory where you invoked the asnqapp command. This is an absolute path name. Use double quotation marks (&quot;&quot;&quot;) to preserve case.</td>
</tr>
</tbody>
</table>
### Table 43. Definitions for asnqapp invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
</table>
| applyupto=gmt_timestamp | Specifies that the Q Apply program processes transactions that were committed at the source up to this time and then stops. The timestamp must be specified as Greenwich mean time (GMT).  
You can specify the timestamp in the following formats:  
**Full timestamp**  
YYYY-MM-DD-HH.MM.SS.MMMMMM  
This is the full timestamp format. For example, 2007-03-13-13.44.33.555555 describes March 13th, 2007, 1:44 PM, 33 seconds, 555555 microseconds, GMT.  
**Partial timestamps**  
You can also specify a shorter version of the full timestamp by using one of the following formats:  
YYYY-MM-DD-HH.MM, SS  
YYYY-MM-DD-HH.MM  
YYYY-MM-DD-HH  
HH.MM  
HH  
The partial timestamps are converted to full timestamps. The HH.MM and HH partial timestamps are converted to the full timestamp of the current day in GMT.  
**Restriction:** You cannot specify both the **autostop** and **applyupto** parameters.  
You might want to set the heartbeat interval to a value greater than zero so that the Q Apply program can tell if the **applyupto** time has passed. |
| arm=identifier | Specifies a three-character alphanumeric string that is used to identify a single instance of the Q Apply program to the Automatic Restart Manager. The value that you supply is appended to the ARM element name that Q Apply generates for itself: ASNQAxxyyyyy (where xxyy 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 will pad the name with blanks to make a unique 16-byte name. |
| autostop=y/n | Specifies whether a Q Apply program terminates after all receive queues are emptied once.  
y | The Q Apply program stops when all receive queues have been emptied once. During the load phase, the Q Apply program ignores autostop=y until the Q subscription is loaded.  
n (default) | The Q Apply program continues running even if all receive queues have been emptied once.  
**Restriction:** You cannot specify both the **autostop** and **applyupto** parameters. |
Table 43. Definitions for qapp invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffered_inserts=y/n</td>
<td>Specifies whether the Q Apply program uses buffered inserts, which can improve performance in some partitioned databases. 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. 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.</td>
</tr>
<tr>
<td>caf=n/y</td>
<td>Specifies whether 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.</td>
</tr>
<tr>
<td></td>
<td>n (default)</td>
</tr>
<tr>
<td></td>
<td>The Q Apply program uses Resource Manager Services (RRS) connect.</td>
</tr>
<tr>
<td></td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>Specifies that the replication program overrides the default RRS connect and runs with CAF connect. 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 connect because RRS is not started. The program attempts to use CAF instead. The program runs correctly with CAF connect.</td>
</tr>
<tr>
<td>commit_count=n</td>
<td>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. <strong>Recommendation:</strong> 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.</td>
</tr>
<tr>
<td>deadlock_retries=n</td>
<td>Specifies the number of times that a Q Apply program tries to reapply changes to a table after an SQL deadlock or lock timeout. The default is three tries. You cannot lower this default value.</td>
</tr>
</tbody>
</table>
Table 43. Definitions for asnqapp invocation parameters  (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dftmodelq</strong></td>
<td>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 <strong>dftmodelq</strong> parameter. The following list summarizes the behavior of the parameter:</td>
</tr>
<tr>
<td></td>
<td><strong>If you specify dftmodelq when you start Q Apply</strong>:&lt;br&gt;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:&lt;br&gt;- If the value is NULL or IBMQREP.SPILL.MODELQ, then Q Apply will use the value that you specify for the <strong>dftmodelq</strong> parameter.&lt;br&gt;- 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 <strong>dftmodelq</strong> parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>If you do not specify dftmodelq when you start Q Apply</strong>:&lt;br&gt;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.</td>
</tr>
</tbody>
</table>
| **ignbaddata**  | **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). Specifies whether the Q Apply program checks for illegal characters in data from the source and continues processing even if it finds illegal characters.  
**y**<br>Q Apply checks for illegal characters and takes the following actions if any are found:<br>- 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.  
**n** (default)<br>Q Apply does not check for illegal characters and does 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. |
Table 43. Definitions for asnqapp invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert_bidi_signal</td>
<td>Specifies whether the Q Capture and Q Apply programs use P2PNORECAPTURE signal inserts to prevent recapture of transactions in bidirectional replication.</td>
</tr>
<tr>
<td></td>
<td><strong>Y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>N</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>loadcopy_path</td>
<td>Use with the DB2 High Availability Disaster Recovery (HADR) feature: Use this parameter when the primary server in a HADR configuration is loaded by the Q Apply program calling the DB2 LOAD utility. Setting this parameter 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.</td>
</tr>
<tr>
<td>load_data_buff_sz</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 43. Definitions for asnqapp invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>logreuse=y/n</strong></td>
<td>Specifies whether a Q Apply program reuses or appends to its diagnostic log file.</td>
</tr>
<tr>
<td></td>
<td><strong>Z/OS</strong> The log file name is <code>apply_server.apply_schema.QAPP.log</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Linux UNIX Windows</strong> The log file name is <code>db2instance.apply_server.apply_schema.QAPP.log</code>.</td>
</tr>
<tr>
<td></td>
<td><em>y</em> On restart, the Q Apply program reuses its log file by clearing the file then writing to the blank file.</td>
</tr>
<tr>
<td></td>
<td><strong>n (default)</strong> The Q Apply program appends new information to an existing Q Apply log file when it restarts.</td>
</tr>
<tr>
<td><strong>logstdout=y/n</strong></td>
<td>Specifies whether a Q Apply program sends log messages to both its log file and the console.</td>
</tr>
<tr>
<td></td>
<td><em>y</em> The Q Apply program sends messages to both the log file and the console (stdout).</td>
</tr>
<tr>
<td></td>
<td><strong>n (default)</strong> The Q Apply program directs most log messages to the log file only.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>max_parallel_loads=n</strong></td>
<td>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 <strong>max_parallel_loads</strong> differs depending on the platform of the target server:</td>
</tr>
<tr>
<td></td>
<td><strong>Z/OS</strong> 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 <strong>max_parallel_loads=1</strong>. If errors occur, reset the value to 1.</td>
</tr>
<tr>
<td></td>
<td><strong>Linux UNIX Windows</strong> On Linux, UNIX, and Windows the default is 15 parallel loads.</td>
</tr>
<tr>
<td><strong>monitor_interval=n</strong></td>
<td>Specifies how often, in milliseconds, a Q Apply program adds rows to the IBMQREP_APPLYMON table. The default is 60000 milliseconds (one minute).</td>
</tr>
<tr>
<td><strong>monitor_limit=n</strong></td>
<td>Specifies the number of minutes that rows remain in the IBMQREP_APPLYMON table before they can be pruned. At each pruning interval, a Q Apply program prunes rows in this table if they are older than <strong>monitor_limit</strong> based on the current timestamp. The default is 10080 minutes (seven days).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nickname_commit_ct=n</td>
<td><strong>Federated targets:</strong> 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, which must use the EXPORT and IMPORT utilities. The default is <code>nickname_commit_ct=10</code>. You might see improved loading performance by increasing the number of rows between commits.</td>
</tr>
</tbody>
</table>
| p2p_2nodes=y/n         | This 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 to the IBMQREP_DELTOMB table. Only use the setting `p2p_2nodes=y` for peer-to-peer replication with two active servers.  
   **n (default)**  
   The Q Apply program records conflicting DELETE operations in the IBMQREP_DELTOMB table.  
   **y**  
   The Q Apply program does not use the IBMQREP_DELTOMB table to record conflicting DELETE operations in peer-to-peer replication with two active servers.  
   **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. |
<p>| prune_interval=n       | Specifies how often, in seconds, a Q Apply program looks for rows that are old enough to prune in the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables. The default is 300 seconds (five minutes). |
| pwdfile=file name      | Specifies the name of the encrypted password file. The Q Apply program uses the password file to connect to the Q Capture server if you specified an automatic load that uses the EXPORT and IMPORT or EXPORT and LOAD utilities. When you use theasnpwd command to create a password file, the default file name is <code>asnpwd.aut</code>. The Q Apply program searches for the password file in the directory specified by the <code>apply_path</code> parameter. If you do not specify an <code>apply_path</code> parameter, the Q Apply program searches for the password file in the directory where you invoked the asnqapp command. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>richklvl=11</td>
<td>Specifies 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.</td>
</tr>
<tr>
<td>0</td>
<td>Q Apply does not check for RI-based dependencies.</td>
</tr>
<tr>
<td>2 (default)</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
</table>
| `skiptrans="transaction_ID"` | Specifies that the Q Apply program should not apply one or more transactions from one or more receive queues based on their transaction ID. The format of the command is as follows: <br> `asnqapp apply_server=server apply_schema=schema skiptrans="receive_queue_name;transaction_ID"`<br> Where `transaction_ID` can be a single transaction identifier or a range of IDs separated by a hyphen (-). You can also specify multiple receive queue-transaction ID pairs for different queues. Separate multiple `receive_queue_name;transaction_ID` pairs with a comma. <br> 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: <br> **z/OS** 0000:xxxx:xxxx:xxxx:mmmm<br> 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. <br> **Linux/UNIX/Windows** mmm:0000:xxxx:xxxx:xxxx<br> Where `xxxx:xxxx:xxxx` is the transaction ID, and `mmmm` is the partition identifier for partitioned databases (this value is 0000 for non-partitioned databases).<br> To specify a transaction ID, you can use uppercase A-E, lowercase a-f, numerics 0-9, and special characters: "-" and ";" only. For example, the following transaction IDs are allowed (the last example is a range of IDs):<br> 0000:0000:0000:51a1:0000<br> 0000000000051a10000<br> 0000:0000:0000:51a1-0000:0000:0000:0000:51a8<br> Follow these other guidelines in specifying the transaction to ignore:<br> - Leading zeros cannot be omitted in a transaction ID. The transaction ID can be specified with or without colons. For example, 0000:0000:0000:51a1:0001 is treated the same as 0000000000051a10001.<br> - The length of each unit between colons in the input transaction ID should be equal to 4. For example, 0000:1090:1234:5671:001 is not allowed because the length of the last unit is less than 4.<br> - Wild card characters such as (*) are not allowed.<br> - No spaces are allowed between the delimiters (semi-colon, hyphen, comma).
Table 43. Definitions for asnqapp invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>spill_commit_count=n</td>
<td>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.</td>
</tr>
<tr>
<td>term=y/n</td>
<td>Specifies whether the Q Apply program terminates if the target DB2 is unavailable (quiesced, failover HADR, stopped, etc.)</td>
</tr>
<tr>
<td></td>
<td><strong>y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Apply program terminates if DB2 is unavailable.</td>
</tr>
<tr>
<td></td>
<td><strong>n</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Apply program continues running if DB2 is unavailable. When DB2 is available, the Q Apply program begins applying transactions at the point where it left off.</td>
</tr>
<tr>
<td>trace_limit=n</td>
<td>The number of minutes that rows remain in the IBMQREP_APPLYTRACE table before they can be pruned. At each pruning interval, the Q Apply 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).</td>
</tr>
</tbody>
</table>

Example 1

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"
```

Example 2

To start a Q Apply program and temporarily shorten the default interval for inserting rows into the IBMQREP_APPLYMON table:

```
asnqapp apply_server=targetdb apply_schema="alpha" monitor_interval=100
```

Example 3

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
```

Example 4

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="2007-04-02-20"
```
This example uses the partial timestamp format to stop the Q Apply program when it reads a transaction with a timestamp after 8 PM on April 2nd, 2007 Greenwich mean time.

**Example 5**

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:

```
asqapp 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
  -prune
  -qryparms
  -status [show details]
  -stop [show details]
  -stopq=receive_queue
  -startq=receive_queue
  -startq="receive_queue;skiptrans=skiptrans-clause"
  -reinitq=receive_queue
  -resumesub=recv_queue:sub_name
  -spillsub=recv_queue:sub_name

Parameters:

- autostop=y/n
- deadlock_retries=n
- logreuse=y/n
- logstdout=y/n
- monitor_interval=n
- monitor_limit=n
- prune_interval=n
- term=n
- trace_limit=n
```

---

348 Replication and Event Publishing Guide and Reference
Parameters

Table 44 defines the invocation parameters for the asnqacmd command.

**Table 44. Definitions for the asnqacmd invocation parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>apply_server=db_name</strong></td>
<td>Specifies the name of the database or subsystem that contains the Q Apply control tables.</td>
</tr>
<tr>
<td><strong>apply_schema=schema</strong></td>
<td>Specifies a name that identifies the running Q Apply program that you want to work with.</td>
</tr>
</tbody>
</table>
| **chgparms** | Specify to change one or more of the following operational parameters of the Q Apply program while it is running:  
  - autostop  
  - logreuse  
  - logstdout  
  - monitor_interval  
  - monitor_limit  
  - prune_interval  
  - term  
  - trace_limit  
  - deadlock_retries  
  You can specify multiple parameters in one asnqacmd chgparms command, 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. "asnqapp: Starting a Q Apply program" on page 336 includes descriptions of the parameters that you can override with this command.  
  **Important:** The parameters that you are changing must immediately follow the chgparms parameter. |
<p>| <strong>prune</strong> | 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 <strong>prune_interval</strong> parameter. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>qryparms</strong></td>
<td>Specify if you want the current operational parameter values for the Q Apply program written to the standard output (stdout).</td>
</tr>
<tr>
<td><strong>status</strong></td>
<td>Specify to see a message about the state of each Q Apply thread (main, housekeeping, monitor, browsers, and agents).</td>
</tr>
</tbody>
</table>
| **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 336. |
### Table 44. Definitions for the asqacmd invocation parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reinitq=</strong> receive_queue</td>
<td>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.</td>
</tr>
<tr>
<td><strong>resumesub=</strong> recv_queue:sub_name</td>
<td>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 <strong>spillsub</strong> parameter is used to put the Q subscription in a spilling state (S) and the <strong>resumesub</strong> 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.</td>
</tr>
<tr>
<td><strong>spillsub=</strong> recv_queue:sub_name</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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:
```
asqacmd 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:
```
asqacmd 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:
```
asqacmd apply_server=targetdb apply_schema="alpha" status
```

### Example 4

To receive detailed messages about the Q Apply program and active receive queues:
```
asqacmd apply_server=targetdb apply_schema="alpha" status show details
```
Example 5

To prune the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables once:

```
asnacmd apply_server=targetdb apply_schema="alpha" prune
```

Example 6

To place a Q subscription in spill mode for performing maintenance on the target table:

```
asnacmd apply_server=targetdb apply_schema="BSN"
  spillsub="BSN.QM1_TO_QM2.recvq:EmployeeSub"
```

When you are finished maintaining the target table, use the asnacmd resumesub command 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:

```
asnacmd apply_server=targetdb apply_schema=ASN
  startq="Q2;skiptrans=0000:0000:0000:0000:51a1-0000:0000:0000:0000:51a8"
```

---

**asnmon: Starting a Replication Alert Monitor**

Use the asnmon command to start a Replication Alert Monitor 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.

The Replication Alert Monitor records the following information:
- The status of Q Capture, Q Apply, Capture, and Apply programs
- Error messages written to the control tables
- Threshold values

**Syntax**

```
|--asnmon
  |--monitor_server=server
  <|monitor_qual=mon_qual|
    |--monitor_interval=n
      |--runonce=y
      <|arm=identifier|
    <|autoprune=n
      |--logreuse=y
      |--logstdout=y
    <|term=y
      |--alert_prune_limit=n
      |--trace_limit=n
    <|max_notifications_per_alert=n
      |--max_notifications_minutes=n
```
Parameters

Table 45 defines the invocation parameters for the asnmon command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor_server=server</td>
<td>Specifies the name of the Monitor control server where the Replication Alert Monitor program runs and the monitor control tables reside. This must be the first parameter if entered.</td>
</tr>
<tr>
<td>monitor_qual=mon_qual</td>
<td>Specifies the monitor qualifier that the Replication Alert Monitor program uses. The monitor qualifier identifies the server to be monitored and the associated monitoring conditions. You must specify a monitor qualifier. The monitor qualifier name is case sensitive and can be a maximum of 18 characters.</td>
</tr>
<tr>
<td>monitor_interval=n</td>
<td>Specifies how frequently (in seconds) the Replication Alert Monitor program runs for this monitor qualifier. The default is 300 seconds (five minutes). This parameter is ignored by the Replication Alert Monitor if you set the runonce parameter to y. Important: This monitor_interval parameter affects the Replication Alert Monitor program only. This parameter does not affect Q Capture, Q Apply, Capture, and Apply programs.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>runonce=y/n</td>
<td>Specifies whether the Replication Alert Monitor program runs only one time for this monitor qualifier.</td>
</tr>
<tr>
<td></td>
<td><strong>n (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Replication Alert Monitor program runs at the frequency indicated by the <code>monitor_interval</code> parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>y</strong></td>
</tr>
<tr>
<td></td>
<td>The Replication Alert Monitor program runs only one monitor cycle.</td>
</tr>
<tr>
<td></td>
<td>If you set the runonce parameter to <code>y</code>, the <code>monitor_interval</code> parameter is ignored by the Replication Alert Monitor.</td>
</tr>
<tr>
<td>arm=identifier</td>
<td>Specifies a three-character alphanumeric string that is used to identify a single instance of the Replication Alert Monitor program to the Automatic Restart Manager. The value that you supply is appended to the ARM element name that the monitor program generates for itself: <code>ASNAMxxxxyyyy</code> (where <code>xxxx</code> is the data-sharing group attach name, and <code>yyyy</code> is the DB2 member name). You can specify any length of string for the <code>arm</code> parameter, but the monitor program will concatenate only up to three characters to the current name. If necessary, the monitor program will pad the name with blanks to make a unique 16-byte name.</td>
</tr>
<tr>
<td>autopruney/n</td>
<td>Specifies whether automatic pruning of the rows in the Replication Alert Monitor alerts (IBMSNAP_ALERTS) table is enabled.</td>
</tr>
<tr>
<td></td>
<td><strong>y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Replication Alert Monitor program automatically prunes the rows in the <code>IBMSNAP_ALERTS</code> table that are older than the <code>alert_prune_limit</code> parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>n</strong></td>
</tr>
<tr>
<td></td>
<td>Automatic pruning is disabled.</td>
</tr>
<tr>
<td>logreuse=y/n</td>
<td>Specifies whether the Replication Alert Monitor program reuses or appends messages to its diagnostic log file (<code>db2instance.monitor_server.mon_qual.MON.log</code>).</td>
</tr>
<tr>
<td></td>
<td><strong>n (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Replication Alert Monitor program appends messages to the log file.</td>
</tr>
<tr>
<td></td>
<td><strong>y</strong></td>
</tr>
<tr>
<td></td>
<td>The Replication Alert Monitor program reuses the log file by deleting it and then recreating it when the Replication Alert Monitor program is restarted.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>logstdout=y/n</code></td>
<td>Specifies where messages are sent by the Replication Alert Monitor program.</td>
</tr>
<tr>
<td>n (default)</td>
<td>The Replication Alert Monitor program sends messages to the log file only.</td>
</tr>
<tr>
<td>y</td>
<td>The Replication Alert Monitor program sends messages to both the log file and the standard output (stdout).</td>
</tr>
<tr>
<td><code>term=y/n</code></td>
<td>Specifies whether a monitor program keeps running when DB2 is quiesced.</td>
</tr>
<tr>
<td>y (default)</td>
<td>The monitor program stops when DB2 is quiesced.</td>
</tr>
<tr>
<td>n</td>
<td>The monitor program keeps running while DB2 is in quiesce mode and has forced all applications to disconnect (including the monitor program). When DB2 is taken out of quiesce mode, the monitor program goes back to monitoring replication. Regardless of the setting for the <code>term</code> parameter, a monitor program stops when DB2 shuts down. When DB2 starts again, you must restart the monitor program.</td>
</tr>
<tr>
<td><code>alert_prune_limit=n</code></td>
<td>Specifies how long (in minutes) rows are kept in the Replication Alert Monitor alerts (IBMSNAP_ALERTS) table. Any rows older than this value are pruned. The default is 10 080 minutes (seven days).</td>
</tr>
<tr>
<td><code>trace_limit=n</code></td>
<td>Specifies how long (in minutes) a row can remain in the Replication Alert Monitor trace (IBMSNAP_MONTRACE) table before it becomes eligible for pruning. All IBMSNAP_MONTRACE rows that are older than the value of this <code>trace_limit</code> parameter are pruned at the next pruning cycle. The default is 10 080 minutes (seven days).</td>
</tr>
<tr>
<td><code>max_notifications_per_alert=n</code></td>
<td>Specifies the maximum number of the same alerts that are sent to a user when the alerts occurred during the time period specified by the <code>max_notifications_minutes</code> parameter value. Use this parameter to avoid re-sending the same alerts to a user. The default is 3.</td>
</tr>
<tr>
<td><code>max_notifications_minutes=n</code></td>
<td>This parameter works with the <code>max_notifications_per_alert</code> parameter to indicate the time period when alert conditions occurred. The default is 60 minutes.</td>
</tr>
<tr>
<td><code>pwdfile=filepath</code></td>
<td>Specifies the fully qualified name of the password file. You define this file by using the asnpwd command. The default file name is asnpwd.aut.</td>
</tr>
<tr>
<td><code>monitor_path=path</code></td>
<td>Specifies the location of the log files used by the Replication Alert Monitor program. The default is the directory where the asnmon command was invoked.</td>
</tr>
</tbody>
</table>
Table 45. *asnmon* invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>monitor_errors=address</strong></td>
<td>Specifies the e-mail address to which notifications are sent if a fatal error is detected before the alert monitor connects to the Monitor control server. Use this parameter to send a notification that the Monitor control server connection failed because of invalid start parameters, an incorrect monitor qualifier, a down database, or other error. Type double quotation marks around the e-mail address text. You can enter multiple e-mail addresses. Separate the e-mail addresses with commas. You can type spaces before or after the commas.</td>
</tr>
<tr>
<td><strong>email_server=servername</strong></td>
<td>Specifies the e-mail server address. Enter this parameter only if you use the ASNMAIL exit routine with SMTP (Simple Mail Transfer Protocol).</td>
</tr>
<tr>
<td><strong>console=y/n</strong></td>
<td><strong>z/OS</strong> Specifies whether the Replication Alert Monitor program sends alert notifications to the z/OS console. If you set this parameter to Y (yes) and an e-mail server was already configured, alerts are sent to both the z/OS console and the e-mail server.</td>
</tr>
<tr>
<td></td>
<td><strong>n</strong> (default) The Replication Alert Monitor program does not send alert notifications to the z/OS console.</td>
</tr>
<tr>
<td></td>
<td><strong>y</strong> The Replication Alert Monitor program sends alert notifications to the z/OS console.</td>
</tr>
</tbody>
</table>

**Return codes**

The *asnmon* command returns a zero return code upon successful completion. A nonzero return code is returned if the command is unsuccessful.

**Examples for *asnmon***

The following examples illustrate how to use the *asnmon* command.

**Example 1**

To start the Replication Alert Monitor with the default parameters:

```
asnmon monitor_server=wsdb monitor_qual=monqual
```

**Example 2**

To start a Replication Alert Monitor that runs every 120 seconds (two minutes) for the specified monitor qualifier:

```
asnmon monitor_server=wsdb monitor_qual=monqual monitor_interval=120
```

**Example 3**

To start a Replication Alert Monitor and specify that it run only once for the specified monitor qualifier:
Example 4

To start a Replication Alert Monitor that sends e-mail notifications if it detects monitoring errors:

```
asnmon monitor_server=wsdb monitor_qual=monqual
monitor_errors="repladm@company.com, dbadmin@company.com"
```

Example 5

To start a Replication Alert Monitor that runs every 120 seconds (two minutes) and waits 1440 minutes (24 hours) before sending alerts:

```
asnmon monitor_server=wsdb monitor_qual=monqual
monitor_interval=120
max_notifications_per_alert=2
max_notifications_minutes=1440
```

This Replication Alert Monitor program sends a maximum of two alerts when the alerts occurred during the time period specified by the `max_notifications_minutes` parameter value (1440 minutes).

### asnmcmd: Working with a running Replication Alert Monitor

Use `asnmcmd` to send commands to a running Replication Alert Monitor 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 Replication Alert Monitor program on z/OS, see [Working with running SQL replication programs by using the MVS MODIFY command](#).

**Syntax**

```
asnmcmd

asnmcmd: Working with a running Replication Alert Monitor

Parameters:

- `monitor_server=server`
- `monitor_qual=mon_qual`
- `reinit`
- `status`
- `stop`
- `qryparms`
- `suspend`
- `resume`
- `monitor_interval=n`
- `autoprune=n`
- `alert_prune_limit=n`
- `trace_limit=n`
- `max_notifications_per_alert=n`
```

Chapter 23. System commands for Q replication and event publishing  357
Parameters

Table 46 defines the invocation parameters for the asnmcmd command.

Table 46. asnmcmd invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor_server</td>
<td>Specifies the name of the Monitor control server where the Replication Alert Monitor program runs and the monitor control tables reside. This must be the first parameter if entered.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a Monitor control server, this parameter defaults to the value from the DB2DBDFT environment variable.</td>
</tr>
<tr>
<td></td>
<td>The default is DSN.</td>
</tr>
<tr>
<td>monitor_qual</td>
<td>Specifies the monitor qualifier that the Replication Alert Monitor program uses. The monitor qualifier identifies the server to be monitored and the associated monitoring conditions.</td>
</tr>
<tr>
<td></td>
<td>You must specify a monitor qualifier. The monitor qualifier name is case sensitive and can be a maximum of 18 characters.</td>
</tr>
<tr>
<td>chgparms</td>
<td>Specify to change one or more of the following operational parameters of the Replication Alert Monitor while it is running:</td>
</tr>
<tr>
<td></td>
<td>• monitor_interval</td>
</tr>
<tr>
<td></td>
<td>• autoprun</td>
</tr>
<tr>
<td></td>
<td>• alert_prune_limit</td>
</tr>
<tr>
<td></td>
<td>• trace_limit</td>
</tr>
<tr>
<td></td>
<td>• max_notifications_per_alert</td>
</tr>
<tr>
<td></td>
<td>• max_notifications_minutes</td>
</tr>
</tbody>
</table>
|                 | You can specify multiple parameters in one chgparms subcommand, and you can change these parameter values as often as you want. The changes temporarily override the values in the IBMSNAP_MONPARMS table, but they are not saved in the table. When you stop and restart the Replication Alert Monitor, it uses the values in IBMSNAP_MONPARMS. |}

Important: The parameter that you are changing must immediately follow the chgparms subcommand.

reinit          | Specify to have the Replication Alert Monitor program read its control tables to refresh the data that it has for contacts, alert conditions, and parameters in its memory. When all values are read, the Monitor program begins its cycle of checking conditions on the servers. After this cycle is complete, the next monitor cycle begins after the time specified in monitor_interval has elapsed. |
Table 46. `asnmcmd` invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Specify to receive messages that indicate the state of each thread (administration, serialization, and worker) in the Replication Alert Monitor.</td>
</tr>
<tr>
<td>qryparms</td>
<td>Specify if you want the current operational parameter values for the Replication Alert Monitor written to the standard output (stdout).</td>
</tr>
<tr>
<td>suspend</td>
<td>Specify if you want the Replication Alert Monitor to stop checking conditions on servers temporarily until you issue the resume command.</td>
</tr>
<tr>
<td>resume</td>
<td>Specify if the Replication Alert Monitor has been suspended and you want the Monitor program to begin checking conditions on servers again.</td>
</tr>
<tr>
<td>stop</td>
<td>Specify to stop the Replication Alert Monitor in an orderly way.</td>
</tr>
</tbody>
</table>

Examples for `asnmcmd`

The following examples illustrate how to use the `asnmcmd` command.

Example 1

To stop the Replication Alert Monitor for the specified monitor qualifier:

```
asnmcmd monitor_server=wsdb monitor_qual=monqual stop
```

Example 2

To receive messages that indicate the status of the Replication Alert Monitor threads:

```
asnmcmd monitor_server=wsdb monitor_qual=monqual status
```

Example 3

To refresh the Replication Alert Monitor with current values from the monitor control tables:

```
asnmcmd monitor_server=wsdb monitor_qual=monqual reinit
```

Example 4

To reduce the maximum number of notifications that the Replication Alert Monitor sends during a specified time period from the default of 3:

```
asnmcmd monitor_server=wsdb monitor_qual=monqual chgparms max_notifications_per_alert=2
```

Example 5

To send the current operational parameters of the Replication Alert Monitor to the standard output:

```
asnmcmd monitor_server=wsdb monitor_qual=monqual qryparms
```
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 --add --modify --delete --list
```

Init parameters:

```
encrypt all password asnpwd.aut using filepath_name
```

Add parameters:

```
alias db_alias --id userid password password
```

Modify parameters:

```
alias db_alias --id userid password password
```

Delete parameters:

```
alias db_alias
```

List parameters:

```
```
Parameters

Table 47 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>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>init</em></td>
<td>Specify to create an empty password file. This command will fail if you specify the <em>init</em> parameter with a password file that already exists.</td>
</tr>
<tr>
<td><em>add</em></td>
<td>Specify to add an entry to the password file. There can only be one entry in the password file per <em>db_alias</em>. This command will fail if you specify the <em>add</em> parameter with an entry that already exists in the password file. Use the <em>modify</em> parameter to change an existing entry in the password file.</td>
</tr>
<tr>
<td><em>modify</em></td>
<td>Specify to modify the password or user ID for an entry in the password file.</td>
</tr>
<tr>
<td><em>delete</em></td>
<td>Specify to delete an entry from the password file.</td>
</tr>
<tr>
<td><em>list</em></td>
<td>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 <em>encrypt password</em> parameter. Passwords are never displayed by the list command.</td>
</tr>
<tr>
<td><em>encrypt</em></td>
<td>Specifies which entries in a file to encrypt.</td>
</tr>
<tr>
<td>all (default)</td>
<td>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.</td>
</tr>
<tr>
<td>password</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 47. asnpwd invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>using filepath</td>
<td>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:\sql11\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.</td>
</tr>
<tr>
<td>alias db_alias</td>
<td>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.</td>
</tr>
<tr>
<td>id userid</td>
<td>Specifies the user ID that has access to the database.</td>
</tr>
<tr>
<td>password password</td>
<td>Specifies the password for the specified user ID. This password is case sensitive and is encrypted in the password file.</td>
</tr>
</tbody>
</table>

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
```

asnsct: Creating a replication service

Windows
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 -A account -M password -C asncap_command -A asnqcap_command
```

**Parameters**

Table 48 defines the invocation parameters for the asnscrt command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-QC</td>
<td>Specifies that you are starting a Q Capture program.</td>
</tr>
<tr>
<td>-QA</td>
<td>Specifies that you are starting a Q Apply program.</td>
</tr>
<tr>
<td>-M</td>
<td>Specifies that you are starting a Replication Alert Monitor program.</td>
</tr>
<tr>
<td>-C</td>
<td>Specifies that you are starting a Capture program.</td>
</tr>
<tr>
<td>-A</td>
<td>Specifies that you are starting an Apply program.</td>
</tr>
<tr>
<td>db2_instance</td>
<td>Specifies the DB2 instance used to identify a unique DB2 replication service. The DB2 instance can be a maximum of eight characters.</td>
</tr>
<tr>
<td>account</td>
<td>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).</td>
</tr>
<tr>
<td>password</td>
<td>Specifies the password used with the account name. If the password contains special characters, type a backslash () before each special character.</td>
</tr>
<tr>
<td>asnqcap_command</td>
<td>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.</td>
</tr>
</tbody>
</table>

364 Replication and Event Publishing Guide and Reference
Table 48. `asncrt` invocation parameter definitions for Windows operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>asnqapp_command</code></td>
<td>Specifies the complete <code>asnqapp</code> command to start a Q apply program. Use the documented <code>asnqapp</code> command syntax with the appropriate <code>asnqapp</code> parameters.</td>
</tr>
<tr>
<td></td>
<td>If the DB2PATH environment variable is not defined, you must specify the location for the work files by including the <code>apply_path</code> parameter with the <code>asnqapp</code> command. If the DB2PATH variable is defined and you specify an <code>apply_path</code>, the <code>apply_path</code> parameter overrides the DB2PATH variable. The <code>asncrt</code> command does not validate the syntax of the <code>asnqapp</code> parameters that you enter.</td>
</tr>
<tr>
<td><code>asnmon_command</code></td>
<td>Specifies the complete <code>asnmon</code> command to start a Replication Alert Monitor program. Use the documented <code>asnmon</code> command syntax with the appropriate <code>asnmon</code> parameters.</td>
</tr>
<tr>
<td></td>
<td>If the DB2PATH environment variable is not defined, you must specify a location for the log files by including the <code>monitor_path</code> parameter with the <code>asnmon</code> command. If the DB2PATH variable is defined and you specify a <code>monitor_path</code>, the <code>monitor_path</code> parameter overrides the DB2PATH variable. The <code>asncrt</code> command does not validate the syntax of the <code>asnmon</code> parameters that you enter.</td>
</tr>
<tr>
<td><code>asncap_command</code></td>
<td>Specifies the complete <code>asncap</code> command to start a Capture program. Use the documented <code>asncap</code> command syntax with the appropriate <code>asncap</code> parameters.</td>
</tr>
<tr>
<td></td>
<td>If the DB2PATH environment variable is not defined, you must specify a location for the work files by including the <code>capture_path</code> parameter with the <code>asncap</code> command. If the DB2PATH variable is defined and you specify a <code>capture_path</code>, the <code>capture_path</code> parameter overrides the DB2PATH variable. The <code>asncrt</code> command does not validate the syntax of the <code>asncap</code> parameters that you enter.</td>
</tr>
<tr>
<td><code>asnapply_command</code></td>
<td>Specifies the complete <code>asnapply</code> command to start an Apply program. Use the documented <code>asnapply</code> command syntax with the appropriate <code>asnapply</code> parameters.</td>
</tr>
<tr>
<td></td>
<td>If the DB2PATH environment variable is not defined, you must specify the location for the work files by including the <code>apply_path</code> parameter with the <code>asnapply</code> command. If the DB2PATH variable is defined and you specify an <code>apply_path</code>, the <code>apply_path</code> parameter overrides the DB2PATH variable. The <code>asncrt</code> command does not validate the syntax of the <code>asnapply</code> parameters that you enter.</td>
</tr>
</tbody>
</table>

Examples for `asncrt`

The following examples illustrate how to use the `asncrt` 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=mc3
   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**

**Windows**

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
```

366  Replication and Event Publishing Guide and Reference
Parameters

Table 49 defines the invocation parameters for the asnsdrop command.

Table 49. asnsdrop invocation parameter definitions for Windows operating systems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_name</td>
<td>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.</td>
</tr>
<tr>
<td>ALL</td>
<td>Specifies that you want to drop all DB2 replication services.</td>
</tr>
</tbody>
</table>

Examples for asnsdrop

The following examples illustrate how to use the asnsdrop command.

Example 1

To drop a DB2 replication service:

```bash
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:

```bash
asnsdrop "DB2.SAMPLEDB.SAMPLEDB.CAP.A S N"
```

Example 3

To drop all DB2 replication services:

```bash
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

```bash
asnslist [DETAILS]
```
Parameters

Table 50 defines the invocation parameter for the asnslist command.

Table 50. asnslist invocation parameter definition for Windows operating systems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>details</td>
<td>Specifies that you want to list detailed data about all DB2 replication services on a system.</td>
</tr>
</tbody>
</table>

Examples for asnslist

The following examples illustrate how to use the asnslist command.

Example 1

To list the names of DB2 replication services on a system:

```bash
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:

```bash
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 APPLY_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 CAPTURE_SCHEMA=ASN QCAPTURE_PATH=C:\PROGRA~1\SQLLIB
    Dependency: DB4-0
```

asntdiff: Comparing data in source and target tables

Use the asntdiff command to compare a source table with a target table and generate a list of differences between the two. Run the asntdiff command on Linux, UNIX, Windows, or z/OS at an operating system prompt or in a shell script.

The asntdiff command compares DB2 tables on Linux, UNIX, Windows, z/OS, and System i operating systems.

For information on the asntdiff -f command option, which enables you to compare tables that are not part of replication by using an input file, see "asntdiff -f (input file) command option" on page 374.
Syntax

```
asntdiff -DB=server -DB2_SUBSYSTEM=subsystem
  [SCHEMA=schema]
```

```
[DIFF_SCHEMA=difference_table_schema] [DIFF_TABLESPACE=tablespace]
```

```
[DIFF_DROP=y] [MAXDIFF=difference_limit] [WHERE=WHERE_clause]
```

```
[DIFF_PATH=log_path] [PWDFILE=filename] [DIFF=table_name]
```

```
[RANGECOL=range_clause_option] [-SQLID=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
```

**Parameters**

Table 51 defines the invocation parameters for the asntdiff command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DB=server</code></td>
<td>Specifies the DB2 alias of the database that stores information about the source and target tables that will be compared. The value differs depending on whether you are using Q replication or SQL replication:</td>
</tr>
<tr>
<td></td>
<td><strong>Q replication</strong></td>
</tr>
<tr>
<td></td>
<td>The name of the Q Capture server, which contains the IBMQREP_SUBS table.</td>
</tr>
<tr>
<td></td>
<td>[z/OS] The location name of the Q Capture server, which contains the IBMQREP_SUBS table.</td>
</tr>
<tr>
<td></td>
<td><strong>SQL replication</strong></td>
</tr>
<tr>
<td></td>
<td>The name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table.</td>
</tr>
<tr>
<td></td>
<td>[z/OS] The location name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table.</td>
</tr>
<tr>
<td><code>DB2_SUBSYSTEM=subsystem</code></td>
<td>[z/OS] Specifies the name of the subsystem where you run the asntdiff utility.</td>
</tr>
</tbody>
</table>
Table 51. asntdiff invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA=\textit{schema}</td>
<td>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.</td>
</tr>
<tr>
<td>DIFF_SCHEMA=\textit{difference_table_schema}</td>
<td>Specifies the schema that qualifies the difference table. The default is ASN.</td>
</tr>
<tr>
<td>DIFF_TABLESPACE=\textit{tablespace}</td>
<td>Specifies the table space where the difference table will be placed. If this parameter is not specified, the table will be created in the default table space in the database or subsystem where the asntdiff command was run. This is a two-part name, \textit{dbname}.\textit{tablespace}, where \textit{dbname} is the logical database name and \textit{tablespace} is the table space name.</td>
</tr>
<tr>
<td>DIFF_DROP=\textit{y/n}</td>
<td>Specifies whether an existing difference table will be dropped and recreated before it is used to record differences. If the table does not exist, the asntdiff command creates it.</td>
</tr>
<tr>
<td>\textit{n} (default)</td>
<td>The difference table will be used as is and the existing rows will be deleted.</td>
</tr>
<tr>
<td>\textit{y}</td>
<td>The difference table will be dropped and recreated.</td>
</tr>
<tr>
<td>MAXDIFF=\textit{difference_limit}</td>
<td>Specifies the maximum number of differences that you want the asntdiff command to process before it stops. The default value is 10000.</td>
</tr>
<tr>
<td>WHERE=\textit{WHERE_clause}</td>
<td>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:</td>
</tr>
<tr>
<td>\textbf{Q replication}</td>
<td>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.</td>
</tr>
<tr>
<td>\textbf{SQL replication}</td>
<td>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.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DIFF_PATH = log_path</td>
<td>Specifies the location where you want the asntdiff 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 (&quot;&quot;&quot;) to preserve case.</td>
</tr>
<tr>
<td>PWDFILE = filename</td>
<td>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.</td>
</tr>
<tr>
<td>DIFF = table_name</td>
<td>Specifies the name of the table that will be created in the source database to store differences between the source and target tables. The table will have one row for each difference that is detected. If you do not include this parameter or the DIFF_SCHEMA parameter, the difference table will be named ASN.ASNTDIFF.</td>
</tr>
</tbody>
</table>
Table 51. asntdiff invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
</table>
| 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 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.nnnnnn. For example, 2008-03-10-10.35.30.55555 is the GMT timestamp for March 10, 2008, 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 51. asntdiff invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLID=authorization_ID</td>
<td>Specifies an authorization ID that can be used on z/OS 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.</td>
</tr>
</tbody>
</table>

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="'IBMQREPERTIME' FROM: '2008-03-10-00.00.00000' TO: '2007-04-12-00.00.00000'"
```

**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: '2008-03-10-12.00.00000' TO: '2008-03-11-12.00.00000'"
```
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

```bash
asndiff -f input_filename
```

Parameters

Table 52 defines the mandatory parameters to include in the input file for the asndiff -f command.

For descriptions of optional parameters that you can include in the input file (and which are shared by the standard asndiff command) see "asndiff: Comparing data in source and target tables" on page 368.

Table 52. asndiff -f invocation parameter definitions for Linux, UNIX, Windows, and z/OS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_filename</td>
<td>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 asndiff -f command.</td>
</tr>
<tr>
<td>SOURCE_SERVER=</td>
<td>Specifies the alias of the database where the source table exists.</td>
</tr>
<tr>
<td>source_server_name</td>
<td></td>
</tr>
<tr>
<td>TARGET_SERVER=</td>
<td>Specifies the alias of the database where the target table exists.</td>
</tr>
<tr>
<td>target_server_name</td>
<td></td>
</tr>
<tr>
<td>SOURCE_SELECT=</td>
<td>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 asndiff 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.</td>
</tr>
<tr>
<td>source_select_statement</td>
<td></td>
</tr>
<tr>
<td>TARGET_SELECT=</td>
<td></td>
</tr>
<tr>
<td>target_select_statement</td>
<td></td>
</tr>
</tbody>
</table>

The following examples show the mandatory parameters, SQL statements, and optional parameters that you put in the input file.
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=NO
```

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(HIST_CHAR1,1,5) as HIST_CHAR1, SUBSTR(HIST_CHAR2,1,10) as HIST_CHAR2,
HIST_INT3,HIST_DEC1,HIST_DEC2,HIST_DEC3,CAST(HIST_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,HIST_DEC1,HIST_DEC2,HIST_DEC3,INT1
FROM BISVT.THIST17 ORDER BY 4"
DB2_SUBSYSTEM=DSN7
DIFF_DROP=Y
DEBUG=NO
MAXDIFF=10000
```

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=NO
MAXDIFF=10000
PWDFILE="asnpwd.aut"
DIFF_PATH="C:\utils\"
```

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
```

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(HIST_CHAR1,1,5) as HIST_CHAR1, SUBSTR(HIST_CHAR2,1,10) as HIST_CHAR2,
HIST_INT3,HIST_DEC1,HIST_DEC2,HIST_DEC3,CAST(HIST_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,HIST_DEC1,HIST_DEC2,HIST_DEC3,INT1
FROM BISVT.THIST17 ORDER BY 4"
DB2_SUBSYSTEM=DSN7
DIFF_DROP=Y
DEBUG=NO
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=NO
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
```
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
On parameters:

- **b** buffer_size
- **fn** filename
- **fs** filesize

- **d** diag_mask
- **df** function_name|component_name diag_mask

Format parameters:

- **fn** filename
- **d** diag_mask
Change settings parameters:

- `diag_mask`
- `function_name|component_name`
- `holdlock`

**Parameters**

Table 53 defines the invocation parameters for the `asntrc` command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
| `-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. |
<p>| <code>-qcap</code> | Specifies that a Q Capture program is to be traced. The Q Capture program is identified by the <code>-schema</code> parameter. |
| <code>-schema qcapture_schema</code> | 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 <code>-qcap</code> parameter. |
| <code>-qapp</code> | Specifies that a Q Apply program is to be traced. The Q Apply program is identified by the <code>-schema</code> parameter. |
| <code>-schema qapply_schema</code> | 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 <code>-qapp</code> parameter. |
| <code>-cap</code> | Specifies that a Capture program is to be traced. The Capture program is identified by the <code>-schema</code> parameter. |
| <code>-schema capture_schema</code> | 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 <code>-cap</code> parameter. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-app</td>
<td>Specifies that an Apply program is to be traced. The Apply program is identified by the -qualifier parameter.</td>
</tr>
<tr>
<td>-qualifier apply_qualifier</td>
<td>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.</td>
</tr>
<tr>
<td>-mon</td>
<td>Specifies that a Replication Alert Monitor program is to be traced. The Replication Alert Monitor program is identified by the -qualifier parameter.</td>
</tr>
<tr>
<td>-qualifier monitor_qualifier</td>
<td>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.</td>
</tr>
<tr>
<td>off</td>
<td>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.</td>
</tr>
<tr>
<td>kill</td>
<td>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.</td>
</tr>
<tr>
<td>clr</td>
<td>Specify to clear a trace buffer. This parameter erases the contents of the trace buffer but leaves the buffer active.</td>
</tr>
<tr>
<td>diag</td>
<td>Specify to view the filter settings while the trace facility is running.</td>
</tr>
<tr>
<td>resetlock</td>
<td>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.</td>
</tr>
<tr>
<td>dmp filename</td>
<td>Specify to write the current contents of the trace buffer to a file.</td>
</tr>
<tr>
<td>-holdlock</td>
<td>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.</td>
</tr>
<tr>
<td>ftw</td>
<td>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.</td>
</tr>
<tr>
<td>fmt</td>
<td>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.</td>
</tr>
<tr>
<td>v7fmt</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 53. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>stat</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>statlong</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>-fn filename</strong></td>
<td>Specifies the file name containing the mirrored trace information, which includes all the output from the trace facility.</td>
</tr>
<tr>
<td><strong>-help</strong></td>
<td>Displays the valid command parameters with descriptions.</td>
</tr>
<tr>
<td><strong>-listsymbols</strong></td>
<td>Displays the valid function and component identifiers to use with the -df parameter.</td>
</tr>
<tr>
<td><strong>-b buffer_size</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>-fs filesize</strong></td>
<td>Specifies the size limit (in bytes) of the mirrored trace information file.</td>
</tr>
</tbody>
</table>
| **-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. |
Table 53. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-df function_name\component_name diag_mask</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

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
asintrc command needs to match the database specified by the capture_server
parameter with the asncap command:

```bash
asintrc -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
asintrc command needs to match the database specified by the control_server
parameter with the asnapply command:

```bash
asintrc -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 asintrc command needs to match the database specified by the
monitor_server parameter with the asnmon command:

```bash
asintrc -db DSN6 -qualifier MONQUAL -mon
asnmon monitor_server=DSN6 monitor_qual=MONQUAL
```

## asntrep: Repairing differences between source and target tables

<table>
<thead>
<tr>
<th>Platform</th>
<th>z/OS</th>
<th>Linux</th>
<th>UNIX</th>
<th>Windows</th>
</tr>
</thead>
</table>

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

```bash
asntrep --DB=server --DB2_SUBSYSTEM=subsystem --DIFF_SCHEMA=difference_table_schema --DIFF_TABLESPACE=tablespace --WHERE=WHEREClause --DIFF_PATH=log_path --PWDFILE=filename --DIFF=table_name
```

### Example

```bash
asntrep --DB=source_db --DB2_SUBSYSTEM=source_subsystem --DIFF_SCHEMA=target_schema --DIFF_TABLESPACE=target_tablespace
```

### Notes

- Replace `source_db`, `source_subsystem`, `target_schema`, `target_tablespace` with the actual database names and subsystems.
- Use the `--DIFF` parameter to specify the table(s) for synchronization.
- The `--PWDFILE` parameter is used to specify the password file location.
- The `--DIFF_PATH` parameter is used to specify the log file path for differences.

---

384  Replication and Event Publishing Guide and Reference
## Parameters

Table 54 defines the invocation parameters for the asntrep command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB=server</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>Q replication</strong></td>
</tr>
<tr>
<td></td>
<td>The value is the name of the Q Capture server, which contains the IBMQREP_SUBS table.</td>
</tr>
<tr>
<td></td>
<td><strong>SQL replication</strong></td>
</tr>
<tr>
<td></td>
<td>The value is the name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table.</td>
</tr>
<tr>
<td>z/OS</td>
<td>The value of this parameter is a location name.</td>
</tr>
<tr>
<td>DB2_SUBSYSTEM=subsystem</td>
<td>Specifies the name of the subsystem where you run the asntrep utility.</td>
</tr>
<tr>
<td>SCHEMA=schema</td>
<td>Specifies the schema of the Q Capture control tables for Q replication, or the Apply control tables for SQL replication.</td>
</tr>
<tr>
<td>DIFF_SCHEMA=difference_table_schema</td>
<td>Specifies the schema that qualifies the difference table. The default is ASN.</td>
</tr>
<tr>
<td>DIFF_TABLESPACE=tablespace</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 54. asntrep invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
</table>
| 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:**

```
<table>
<thead>
<tr>
<th>single-database-options</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple-database-options</td>
</tr>
</tbody>
</table>
```

**single-database-options:**

```
| DATABASE=dbname |
| CONFIGSERVER=srvrname  |
| FILE=filename   |
```
multiple-database-options:

\[ \text{-o output}\_file\_name} \]

database-options:

\[ \text{DATABASE=dbname} \]
\[ \text{CONFIGSERVER=srvrname} \]
\[ \text{FILE=filename} \]
\[ \text{SCHEMA=schema-ID=user\_ID} \]
\[ \text{PASSFILE=password\_file} \]

report-options:

\[ \text{REPORT} \]
\[ \text{XML\_input\_file} \]
\[ \text{ZIP=zip\_file\_name} \]
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
- GETCOLS=ON
- GETMONITOR=OFF

Parameters for gather command, single database

Table 55 defines the invocation parameters for the asnqanalyze gather command for the single_database_options.

Table 55. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE=dbname</td>
<td>The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed.</td>
</tr>
<tr>
<td>CONFIGSERVER=srvrname</td>
<td>The Q Capture server or Q Apply server to be analyzed when asnqanalyze uses a Type 4 connection to the server.</td>
</tr>
<tr>
<td>FILE=file_name</td>
<td>The configuration file where the port, hostname, and other connection information are located. This is required for databases that use a Type 4 connection.</td>
</tr>
<tr>
<td>INSTANCE=instance</td>
<td>The DB2 instance of the database to be analyzed. Default: DB2</td>
</tr>
<tr>
<td>SUBSYSTEM=instance</td>
<td>The DB2 subsystem to be analyzed. Default: DSN1</td>
</tr>
<tr>
<td>USERID=user_ID</td>
<td>The user ID to connect to the database.</td>
</tr>
<tr>
<td>PASSFILE=password_file</td>
<td>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. <strong>Tip:</strong> 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.</td>
</tr>
</tbody>
</table>
### Table 55. `asnqanalyze gather` invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SCHEMA</strong> = <em>schema</em></td>
<td>The schema of the Q Capture or Q Apply control tables to be analyzed. Default: ASN</td>
</tr>
<tr>
<td><strong>CAPLOGDIR</strong> = <em>capture_log_directory</em></td>
<td>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 <code>asnqanalyze</code> 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.</td>
</tr>
<tr>
<td><strong>APPLOGDIR</strong> = <em>apply_log_directory</em></td>
<td>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 <code>asnqanalyze</code> 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.</td>
</tr>
<tr>
<td><strong>PORT</strong> = <em>port</em></td>
<td>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 <code>asnqanalyze</code> command. Default: 1414</td>
</tr>
<tr>
<td><strong>CHANNEL</strong> = <em>channel</em></td>
<td>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 <code>asnqanalyze</code> command. Default: SYSTEM.DEF.SVRCONN</td>
</tr>
<tr>
<td><strong>HOSTNAME</strong> = <em>hostname</em></td>
<td>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.</td>
</tr>
<tr>
<td><strong>WARNERRONLY</strong> = <em>ON/OFF</em></td>
<td>The type of records to retrieve from the diagnostic log file.</td>
</tr>
<tr>
<td><strong>OFF (default)</strong></td>
<td>All records are retrieved, including error, warning, and informational messages and descriptions of program actions.</td>
</tr>
<tr>
<td><strong>ON</strong></td>
<td>Only error and warning messages are retrieved.</td>
</tr>
<tr>
<td><strong>GETCOLS</strong> = <em>ON/OFF</em></td>
<td>Specifies whether the Q Analyzer collects information about all columns or only the columns that are being replicated.</td>
</tr>
<tr>
<td><strong>ON (default)</strong></td>
<td>Information about all columns in the source and target tables is collected whether or not the columns are being replicated.</td>
</tr>
<tr>
<td><strong>OFF</strong></td>
<td>Information about replicated columns is collected.</td>
</tr>
</tbody>
</table>
Table 55. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETMONITOR=ON/OFF</td>
<td>Specifies whether data in the IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables is retrieved.</td>
</tr>
<tr>
<td></td>
<td><strong>ON (default)</strong></td>
</tr>
<tr>
<td></td>
<td>Data is retrieved.</td>
</tr>
<tr>
<td></td>
<td><strong>OFF</strong></td>
</tr>
<tr>
<td></td>
<td>Data is not retrieved.</td>
</tr>
<tr>
<td>LOGDAYS=Integer</td>
<td>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</td>
</tr>
<tr>
<td>ZIP=ON/OFF</td>
<td>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.</td>
</tr>
<tr>
<td>-o output_file_name</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>Compressed file</strong></td>
</tr>
<tr>
<td></td>
<td>database_name.schema.zip</td>
</tr>
<tr>
<td></td>
<td><strong>Uncompressed file</strong></td>
</tr>
<tr>
<td></td>
<td>database_name.schema.xml</td>
</tr>
</tbody>
</table>

Parameters for gather command, multiple databases

Table 56 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.

Table 56. asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST=(database_options)</td>
<td>Indicates that multiple databases are being provided. The database options (database_options) need to be specified within single parentheses following the parameter.</td>
</tr>
<tr>
<td>DATABASE=dbname</td>
<td>The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed.</td>
</tr>
<tr>
<td>CONFIGSERVER=srvrname</td>
<td>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.</td>
</tr>
<tr>
<td>SCHEMA=schema</td>
<td>The schema of the Q Capture or Q Apply control tables to be analyzed. Default: ASN</td>
</tr>
<tr>
<td>USERID=user_ID</td>
<td>The user ID to connect to the database.</td>
</tr>
<tr>
<td>INIFILE=file_name</td>
<td>The configuration file where the port, host name are located. This is required for databases that use a Type 4 connection.</td>
</tr>
</tbody>
</table>
Table 56. asqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSFILE</td>
<td>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 asqanalyze 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.</td>
</tr>
<tr>
<td>GETCOLS=ON/OFF</td>
<td>Specifies whether the Q Analyzer collects information about all columns or only the columns that are being replicated.</td>
</tr>
<tr>
<td></td>
<td>ON (default) Information about all columns in the source and target tables is collected whether or not the columns are being replicated.</td>
</tr>
<tr>
<td></td>
<td>OFF Information about replicated columns is collected.</td>
</tr>
<tr>
<td>GETMONITOR</td>
<td>Specifies whether data in the IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables is retrieved.</td>
</tr>
<tr>
<td></td>
<td>ON (default) Data is retrieved.</td>
</tr>
<tr>
<td></td>
<td>OFF Data is not retrieved.</td>
</tr>
<tr>
<td>ZIP=ON/OFF</td>
<td>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.</td>
</tr>
<tr>
<td>-o output_file_name</td>
<td>Specifies the name of the file that is produced by the asqanalyze gather command. If you do not specify a file name, the default name is as follows:</td>
</tr>
<tr>
<td></td>
<td>Compressed file database_name.schema.zip</td>
</tr>
<tr>
<td></td>
<td>Uncompressed file database_name.schema.xml</td>
</tr>
</tbody>
</table>

Parameters for report command

Table 57 on page 393 defines the invocation parameters for the asqanalyze report command.
Table 57. *asnqanalyze report invocation parameter definitions for Linux, UNIX, and Windows*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML_input_file</td>
<td>Specifies one or more space-separated XML files that were produced by the <em>asnqanalyze gather</em> command. Specify a path to the XML file if you run the <em>asnqanalyze report</em> 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.</td>
</tr>
<tr>
<td>-ZIP:zip_file</td>
<td>Specifies that a zipped file that was generated by the <em>asnqanalyze gather</em> command is provided as input. The zip file can have one or more XML files. The <em>REPORT</em> function unzips the file and analyzes each XML file.</td>
</tr>
</tbody>
</table>

**Parameters for generate html report command**

Table 58 defines the invocation parameters for the *asnqanalyze generate html* report command.

Table 58. *asnqanalyze gather invocation parameter definitions for Linux, UNIX, and Windows operating systems*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE=dbname</td>
<td>The Q Capture server or Q Apply server whose control tables, database catalogs, or log files are analyzed.</td>
</tr>
<tr>
<td>CONFIGSERVER=srvrname</td>
<td>Specifies the Q Capture server or Q Apply server to be analyzed when <em>asnqanalyze</em> uses a Type 4 connection to the server.</td>
</tr>
<tr>
<td>INIFILE=file_name</td>
<td>The configuration file where the port, host name are located. This is required for databases that use a Type 4 connection.</td>
</tr>
<tr>
<td>USERID=user_ID</td>
<td>The user ID to connect to the database.</td>
</tr>
<tr>
<td>PASSFILE=password_file</td>
<td>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. <em>Q Analyzer</em> cannot use encrypted password files that are created by the <em>asnpwd</em> command. You must specify a path to the password file if you run the <em>asnqanalyze</em> command from a directory other than where the file is stored. <strong>Tip:</strong> 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 <em>Q Analyzer</em>.</td>
</tr>
<tr>
<td>SCHEMA=schema</td>
<td>The schema of the Q Capture or Q Apply control tables to be analyzed. Default: ASN</td>
</tr>
<tr>
<td>GETCOLS=ON/OFF</td>
<td>Specifies whether the <em>Q Analyzer</em> collects information about all columns or only the columns that are being replicated.</td>
</tr>
<tr>
<td></td>
<td><strong>ON</strong> (default)</td>
</tr>
<tr>
<td></td>
<td>Information about all columns in the source and target tables is collected whether or not the columns are being replicated.</td>
</tr>
<tr>
<td></td>
<td><strong>OFF</strong></td>
</tr>
<tr>
<td></td>
<td>Information about replicated columns is collected.</td>
</tr>
</tbody>
</table>
Table 58. `asqnanalyze gather` invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETMONITOR=ON/OFF</td>
<td>Specifies whether data in the IBMQREP_CAPMON, IBMQREP_CAPQMON, and IBMQREP_APPLYMON tables is retrieved.</td>
</tr>
<tr>
<td></td>
<td><strong>ON (default)</strong> Data is retrieved.</td>
</tr>
<tr>
<td></td>
<td><strong>OFF</strong> Data is not retrieved.</td>
</tr>
<tr>
<td>-o output_file_name</td>
<td>Specifies the name of the file that is produced by the <code>asqnanalyze gather</code> command. If you do not specify a file name, the default name is as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>Compressed file</strong> <code>database_name.schema.zip</code></td>
</tr>
<tr>
<td></td>
<td><strong>Uncompressed file</strong> <code>database_name.schema.xml</code></td>
</tr>
</tbody>
</table>

**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.

```
asqnanalyze 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`.

```
asqnanalyze gather DATABASE=TARGET SUBSYSTEM=DSN4 USERID=db2admin PASSFILE=pwdfile.txt HOSTNAME="Z14SHG" 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:
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 event 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.

You can operate the message formatting program with JCL. You can find sample JCL in the sample qrhqlqual.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
```

Chapter 23. System commands for Q replication and event publishing
Parameters

Table 59. asnqmfmt invocation parameter definitions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the name of a WebSphere MQ queue whose messages you want to format, view, and optionally delete.</td>
</tr>
<tr>
<td>queue_manager_name</td>
<td>Specifies the name of a WebSphere MQ queue manager where the queue is defined.</td>
</tr>
<tr>
<td>-o filepath_name</td>
<td>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.</td>
</tr>
<tr>
<td>-hex</td>
<td>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.</td>
</tr>
<tr>
<td>-l number</td>
<td>Specifies the number of messages that you want to format.</td>
</tr>
<tr>
<td>-delmsg</td>
<td>Specifies that the messages will be deleted from the queue after they are formatted.</td>
</tr>
<tr>
<td>-mqmd</td>
<td>Specifies that you want to view the WebSphere MQ message descriptor for each message that is formatted.</td>
</tr>
<tr>
<td>-oenc output_encoding_name</td>
<td>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.</td>
</tr>
<tr>
<td>-help</td>
<td>Displays the valid command parameters with descriptions.</td>
</tr>
</tbody>
</table>

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:
asnqfmt 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:

asnqfmt 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):

asnqfmt Q2 QMGR2 -l 100 -o C:\qrepl\Q2_messages

asnqfmt: Formatting and viewing event publishing messages (z/OS)

Use the asnqfmt 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 asnqfmt requires XML4C 1.4 version (HXXML14A.SIXMMOD1).

You can operate the message formatting program with JCL. You can find sample JCL in the sample qrhlqual.SASNSAMP(ASNQXMFMT).

Syntax

```bash
asnqfmt queue_name queue_manager_name -o filepath_name

-hex

-l number

delmsg

mqmd

-oenc output_encoding_name

-help
```

Parameters

Table 60 describes the invocation parameters for asnqfmt:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>Specifies the name of a WebSphere MQ queue whose messages you want to format, view, and optionally delete.</td>
</tr>
<tr>
<td>queue_manager_name</td>
<td>Specifies the name of a WebSphere MQ queue manager where the queue is defined.</td>
</tr>
</tbody>
</table>
Table 60. asnqxmfmt invocation parameter definitions (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o filepath_name</td>
<td>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.</td>
</tr>
<tr>
<td>-hex</td>
<td>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.</td>
</tr>
<tr>
<td>-l number</td>
<td>Specifies the number of messages that you want to format.</td>
</tr>
<tr>
<td>-delmsg</td>
<td>Specifies that the messages will be deleted from the queue after they are formatted.</td>
</tr>
<tr>
<td>-mqmd</td>
<td>Specifies that you want to view the WebSphere MQ message descriptor for each message that is formatted.</td>
</tr>
<tr>
<td>-oenc output_encoding_name</td>
<td>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.</td>
</tr>
<tr>
<td>-help</td>
<td>Displays the valid command parameters with descriptions.</td>
</tr>
</tbody>
</table>

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 24. 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 61 describes the control tables at the Q Capture server.

Table 61. Control tables at the Q Capture server

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;IBMQREP_ADMINMSG table&quot; on page 400</td>
<td>An internal table that contains administrative messages received by a Q Capture program.</td>
</tr>
<tr>
<td>&quot;IBMQREP_CAPENQ table&quot; on page 400</td>
<td>Ensures that only one Q Capture program with a given schema is running per Q Capture server.</td>
</tr>
<tr>
<td>&quot;IBMQREP_CAPENVINFO table&quot; on page 401</td>
<td>Stores environment variables and other information that replication tools use to access remote programs.</td>
</tr>
<tr>
<td>&quot;IBMQREP_CAPMON table&quot; on page 402</td>
<td>Contains statistics about the performance of a Q Capture program.</td>
</tr>
<tr>
<td>&quot;IBMQREP_CAPPARMS table&quot; on page 404</td>
<td>Contains parameters that you can specify to control the operations of a Q Capture program.</td>
</tr>
<tr>
<td>&quot;IBMQREP_CAPQMON table&quot; on page 408</td>
<td>Contains statistics about the performance of a Q Capture program for each send queue.</td>
</tr>
<tr>
<td>&quot;IBMQREP_CAPTRACE table&quot; on page 410</td>
<td>Contains informational, warning, and error messages from a Q Capture program.</td>
</tr>
<tr>
<td>&quot;IBMQREP_COLVERSION table (z/OS)&quot; on page 410</td>
<td>Created in DB2 servers on z/OS operating systems to allow the Q Capture and Capture programs to keep track of different versions of a source table.</td>
</tr>
<tr>
<td>IBMQREP_IGNTRAN table</td>
<td>Can be used to inform the Q Capture program about transactions that you do not want to be captured from the DB2 recovery log.</td>
</tr>
<tr>
<td>IBMQREP_IGNTRANTRC table</td>
<td>Records information about transactions that were specified to be ignored.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SENDQUEUES table&quot; on page 414</td>
<td>Contains information about the WebSphere MQ queues that a Q Capture program uses to send transaction, row operation, large object, or informational messages.</td>
</tr>
</tbody>
</table>
Table 61. Control tables at the Q Capture server (continued)

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;IBMQREP_SIGNAL table&quot; on page 418</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SRC_COLS table&quot; on page 421</td>
<td>Identifies columns in the source table that are replicated or published for a Q subscription or publication.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SRCH_COND table&quot; on page 422</td>
<td>An internal table that a Q Capture program uses to evaluate the search condition that you specified for a Q subscription or publication.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SUBS table&quot; on page 423</td>
<td>Contains information about Q subscriptions and publications, including subscription type, source tables, search conditions, data sending options, target loading options, and states.</td>
</tr>
<tr>
<td>&quot;IBMQREP_TABVERSION table (z/OS)&quot; on page 427</td>
<td>Created in DB2 servers on z/OS operating systems to allow the Q Capture and Capture programs to keep track of different versions of a source table.</td>
</tr>
</tbody>
</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 62 provides a brief description of the columns in the IBMQREP_ADMINMSG table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQMSGID</td>
<td><strong>Data type:</strong> CHAR(24) FOR BIT DATA; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td>MSG_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No, with default</td>
</tr>
</tbody>
</table>

**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:

- For DB2 for Linux, UNIX, and Windows, only one Q Capture program with a given schema is running per database.
- For non-data-sharing DB2 for z/OS, only one Q Capture program with a given schema is running per subsystem.
- 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 63 provides a brief description of the column in the IBMQREP_CAPENQ table.

Table 63. Column in the IBMQREP_CAPENQ table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKNAME</td>
<td>Data type: INTEGER; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>This column contains no data.</td>
</tr>
</tbody>
</table>

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

Unique index: None

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 64 on page 402 provides a brief description of the columns in the IBMQREP_CAPENVINFO table.
### Table 64. Columns in the IBMQREP_CAPENVINFO table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Data type: VARCHAR(30); Nullable: No</td>
</tr>
<tr>
<td>STARTTIME</td>
<td>The timestamp when the Q Capture program started.</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>The TCP/IP host name of the server where the Q Capture program is running.</td>
</tr>
<tr>
<td>LOGFILE</td>
<td>The path and file name of the Q Capture diagnostic log file.</td>
</tr>
<tr>
<td>TMPDIR</td>
<td>The path of the directory where the Inter-Process Communication (IPC) key of the Q Capture program is located.</td>
</tr>
<tr>
<td>ASNUSEMQCLIENT</td>
<td>The value of the Q replication ASNUSEMQCLIENT environment variable.</td>
</tr>
<tr>
<td>MQSERVER</td>
<td>The value of the WebSphere MQ MQSERVER environment variable.</td>
</tr>
<tr>
<td>MQCHLLIB</td>
<td>The value of the WebSphere MQ MQCHLLIB environment variable.</td>
</tr>
<tr>
<td>MQCHLTAB</td>
<td>The value of the WebSphere MQ MQCHLTAB environment variable.</td>
</tr>
<tr>
<td>VALUE</td>
<td>Data type: VARCHAR(3800); Nullable: Yes</td>
</tr>
</tbody>
</table>

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 65 on page 403 provides a brief description of the columns in the IBMQREP_CAPMON table.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Capture server when the row was inserted into this</td>
</tr>
<tr>
<td></td>
<td>table.</td>
</tr>
<tr>
<td>CURRENT_LOG_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Capture server of the latest database commit that was</td>
</tr>
<tr>
<td></td>
<td>seen by the Q Capture log reader.</td>
</tr>
<tr>
<td>CAPTURE_IDLE</td>
<td>This column is deprecated.</td>
</tr>
<tr>
<td>CURRENT_MEMORY</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The amount of memory (in bytes) that the Q Capture program used to</td>
</tr>
<tr>
<td></td>
<td>reconstruct transactions from the log.</td>
</tr>
<tr>
<td>ROWS_PROCESSED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The number of rows (individual insert, update, or delete operations) that</td>
</tr>
<tr>
<td></td>
<td>the Q Capture program read from the log.</td>
</tr>
<tr>
<td>TRANS_SKIPPED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The number of transactions (containing changed rows) that were not put on</td>
</tr>
<tr>
<td></td>
<td>queues because the changes were to columns that are not part of a Q</td>
</tr>
<tr>
<td></td>
<td>subscription or publication (the ALL_CHANGED_ROWS parameter in the</td>
</tr>
<tr>
<td></td>
<td>IBMQREP_SUBS table was set to No, the default).</td>
</tr>
<tr>
<td>TRANS_PROCESSED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The number of transactions that the Q Capture program processed.</td>
</tr>
<tr>
<td>TRANS_SPILLED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The number of transactions that the Q Capture program spilled to a file after</td>
</tr>
<tr>
<td></td>
<td>exceeding the MEMORY_LIMIT threshold.</td>
</tr>
<tr>
<td>MAX_TRANS_SIZE</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The largest transaction, in bytes, that the Q Capture program processed.</td>
</tr>
<tr>
<td>QUEUES_IN_ERROR</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The number of queues that were not accepting messages.</td>
</tr>
<tr>
<td>RESTART_SEQ</td>
<td><strong>Data type:</strong> CHAR(10) FOR BIT DATA; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The logical log sequence number in the recovery log at which the Q Capture</td>
</tr>
<tr>
<td></td>
<td>program starts during a warm restart. This value represents the earliest log</td>
</tr>
<tr>
<td></td>
<td>sequence number that the Q Capture program found for which a commit or</td>
</tr>
<tr>
<td></td>
<td>abort record has not yet been found.</td>
</tr>
<tr>
<td>CURRENT_SEQ</td>
<td><strong>Data type:</strong> CHAR(10) FOR BIT DATA; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The most recent logical log sequence number in the recovery log that the Q</td>
</tr>
<tr>
<td></td>
<td>Capture program read.</td>
</tr>
<tr>
<td>LAST_EOL_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The time at the Q Capture control server when the Q Capture program reached</td>
</tr>
<tr>
<td></td>
<td>the end of the log.</td>
</tr>
<tr>
<td>LOGREAD_API_TIME</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number of milliseconds that the Q Capture program spent using the DB2 log</td>
</tr>
<tr>
<td></td>
<td>read application program interface (API) to retrieve log records.</td>
</tr>
</tbody>
</table>
Table 65. Columns in the IBMQREP_CAPMON table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_LOGREAD_CALLS</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number of log read API calls that Q Capture made.</td>
</tr>
<tr>
<td>NUM_END_OF_LOGS</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number of times that Q Capture reached the end of the log.</td>
</tr>
<tr>
<td>LOGRDR_SLEEPTIME</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number of seconds 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.</td>
</tr>
</tbody>
</table>

**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 66 provides a brief description of the columns in the IBMQREP_CAPPARMS table.

Table 66. Columns in the IBMQREP_CAPPARMS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMGR</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the WebSphere MQ queue manager that the Q Capture program uses.</td>
</tr>
<tr>
<td>REMOTE_SRC_SERVER</td>
<td><strong>Data type:</strong> VARCHAR(18); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>RESTARTQ</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>ADMINQ</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STARTMODE</td>
<td><strong>Data type</strong>: VARCHAR(6); <strong>Nullable</strong>: No, with default&lt;br&gt;The steps that the Q Capture program takes when it starts.&lt;br&gt;&lt;br&gt;<strong>cold</strong>&lt;br&gt;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 at the end.&lt;br&gt;&lt;br&gt;<strong>warmsi (default)</strong>&lt;br&gt;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.&lt;br&gt;&lt;br&gt;<strong>warmns</strong>&lt;br&gt;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.&lt;br&gt;&lt;br&gt;During warm starts, the Q Capture program will process only Q subscriptions or publications that are not in I (inactive) state.</td>
</tr>
<tr>
<td>MEMORY_LIMIT</td>
<td><strong>Data type</strong>: INTEGER; <strong>Nullable</strong>: No, with default&lt;br&gt;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&lt;br&gt;&lt;br&gt;<strong>z/OS</strong>&lt;br&gt;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.</td>
</tr>
<tr>
<td>COMMIT_INTERVAL</td>
<td><strong>Data type</strong>: INTEGER; <strong>Nullable</strong>: No, with default&lt;br&gt;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</td>
</tr>
<tr>
<td>AUTOSTOP</td>
<td><strong>Data type</strong>: CHAR(1); <strong>Nullable</strong>: No, with default&lt;br&gt;A flag that indicates whether the Q Capture program stops when it reaches the end of the active DB2 log.&lt;br&gt;&lt;br&gt;<strong>Y</strong>&lt;br&gt;The Q Capture program stops when it reaches the end of the active DB2 log.&lt;br&gt;&lt;br&gt;<strong>N (default)</strong>&lt;br&gt;The Q Capture program continues running when it reaches the end of the active DB2 log.</td>
</tr>
<tr>
<td>MONITOR_INTERVAL</td>
<td><strong>Data type</strong>: INTEGER; <strong>Nullable</strong>: No, with default&lt;br&gt;How often, in milliseconds, the Q Capture program adds rows to the IBMQREP_CAPMON and IBMQREP_CAPQMON tables. Default: 300000</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MONITOR_LIMIT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default&lt;br&gt;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: 10800</td>
</tr>
<tr>
<td>TRACE_LIMIT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default&lt;br&gt;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: 10800</td>
</tr>
<tr>
<td>SIGNAL_LIMIT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default&lt;br&gt;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: 10800</td>
</tr>
<tr>
<td>PRUNE_INTERVAL</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default&lt;br&gt;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</td>
</tr>
<tr>
<td>SLEEP_INTERVAL</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default&lt;br&gt;The number of milliseconds that the Q Capture program is idle after processing the active log and any transactions that remain in memory. Default: 5000</td>
</tr>
<tr>
<td>LOGREUSE</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default&lt;br&gt;A flag that indicates whether the Q Capture program reuses the Q Capture log file or appends to it.&lt;br&gt;&lt;br&gt;<strong>Y</strong> On restart, the Q Capture program reuses its log file by clearing the file then writing to the blank file.&lt;br&gt;&lt;br&gt;<strong>N (default)</strong> The Q Capture program appends new information to an existing Q Capture log file when it restarts.</td>
</tr>
<tr>
<td>LOGSTDOUT</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default&lt;br&gt;A flag that indicates whether the Q Capture program sends log messages to outputs other than its log file.&lt;br&gt;&lt;br&gt;<strong>Y</strong> The Q Capture program sends log messages to both the log file and console (stdout).&lt;br&gt;&lt;br&gt;<strong>N (default)</strong> The Q Capture program directs most log file messages to the log file only.&lt;br&gt;&lt;br&gt;Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TERM</td>
<td>Data type: CHAR(1); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether the Q Capture program stops if the source DB2 is unavailable (quiesced, failover HADR, stopped, etc.)</td>
</tr>
<tr>
<td></td>
<td>Y (default)</td>
</tr>
<tr>
<td></td>
<td>The Q Capture program stops if DB2 is unavailable.</td>
</tr>
<tr>
<td></td>
<td>N The Q Capture program keeps running if DB2 is unavailable.</td>
</tr>
<tr>
<td>CAPTURE_PATH</td>
<td>Data type: VARCHAR(1040); Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The path where the files that are created by the Q Capture program are stored.</td>
</tr>
<tr>
<td>ARCH_LEVEL</td>
<td>Data type: CHAR(4); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>The version of the control tables. For Version 9.7 the value is 0907. Other ARCH_LEVEL values are 0802, 0901, and 0905.</td>
</tr>
<tr>
<td></td>
<td>Attention: When updating the IBMQREP_CAPPARMS table, do not change the value in this column.</td>
</tr>
<tr>
<td>COMPATIBILITY</td>
<td>Data type: CHAR(4); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>The version of messages that the Q Capture program sends to a Q Apply program.</td>
</tr>
<tr>
<td></td>
<td>0907 (default)</td>
</tr>
<tr>
<td></td>
<td>Version 9.7 messages are sent.</td>
</tr>
<tr>
<td></td>
<td>0905 Version 9.5 messages are sent.</td>
</tr>
<tr>
<td></td>
<td>0901 Version 9.1 messages are sent.</td>
</tr>
<tr>
<td></td>
<td>0802 Version 8 messages are sent.</td>
</tr>
<tr>
<td>LOB_SEND_OPTION</td>
<td>Data type: CHAR(1); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates how the Q Capture program sends LOB data.</td>
</tr>
<tr>
<td></td>
<td>I (default)</td>
</tr>
<tr>
<td></td>
<td>Inline. The LOB values are sent within the transaction message. The inlined LOB values can improve performance.</td>
</tr>
<tr>
<td></td>
<td>S Separate. The LOB values are sent in one or more separate LOB messages that follow the transaction message.</td>
</tr>
<tr>
<td>QFULL_NUM_RETRIES</td>
<td>Data type: INTEGER; Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>This column specifies the number of times that the Q Capture program will retry an MQPUT for the send queue.</td>
</tr>
<tr>
<td></td>
<td>Default: 30; Maximum: 1000</td>
</tr>
<tr>
<td>QFULL_RETRY_DELAY</td>
<td>Data type: INTEGER; Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>This column specifies the amount of time that the Q Capture program will sleep before the next retry. The value is specified in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>Default: 250 (milliseconds); Minimum: 20; Maximum: 360000</td>
</tr>
</tbody>
</table>
### Table 66. Columns in the IBMQREP_CAPPARMS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_PERSISTENCE</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that specifies whether a Q Capture program writes persistent (logged) or nonpersistent (unlogged) messages to WebSphere MQ queues.</td>
</tr>
<tr>
<td></td>
<td><strong>Y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>Q Capture write persistent messages to all queues. The messages are logged by the queue manager and can be recovered.</td>
</tr>
<tr>
<td></td>
<td><strong>N</strong></td>
</tr>
<tr>
<td></td>
<td>Q Capture write nonpersistent messages to all queues. The messages are not logged and cannot be recovered.</td>
</tr>
<tr>
<td>LOGRDBUFMSZ</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>The size of the buffer 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.</td>
</tr>
</tbody>
</table>

**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 67 provides a brief description of the columns in the IBMQREP_CAPQMON table.

### Table 67. Columns in the IBMQREP_CAPQMON table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Capture server when the row was inserted into this table.</td>
</tr>
<tr>
<td>SENDQ</td>
<td><strong>Data type:</strong> VARCHAR(97); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the send queue that this row of monitor statistics tells you about.</td>
</tr>
<tr>
<td>ROWS_PUBLISHED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The number of rows (individual insert, update, or delete operations) that the Q Capture program put on this send queue.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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:** CHAR(10) FOR BIT DATA; **Nullable:** Yes  
The most recent logical log sequence number (LSN) 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. |
| RESTART_SEQ      | **Data type:** CHAR(10) FOR BIT DATA; **Nullable:** Yes  
The LSN from which the Q Capture program starts putting messages on this send queue during a warm restart. This value represents the earliest LSN 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. |
**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 68 provides a brief description of the columns in the IBMQREP_CAPTRACE table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATION</td>
<td>Data type: CHAR(8); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The type of message from the Q Capture program:</td>
</tr>
<tr>
<td></td>
<td>INFO Describe actions that the Q Capture program takes.</td>
</tr>
<tr>
<td></td>
<td>WARNING Describe conditions that could cause errors for the Q Capture program.</td>
</tr>
<tr>
<td></td>
<td>ERROR Describe errors encountered by the Q Capture program.</td>
</tr>
<tr>
<td>TRACE_TIME</td>
<td>Data type: TIMESTAMP; Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The time at the Q Capture server that the message was put on a send queue.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Data type: VARCHAR(1024); Nullable: No Data type: INTEGER; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The reason code for the replication or event publishing error message.</td>
</tr>
<tr>
<td></td>
<td>The ASN message ID followed by the message text. This column contains English-only text.</td>
</tr>
<tr>
<td>REASON_CODE</td>
<td>Data type: INTEGER; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The reason code for the replication or event publishing error message.</td>
</tr>
<tr>
<td>MQ_CODE</td>
<td>Data type: INTEGER; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The reason code for the WebSphere MQ error message.</td>
</tr>
</tbody>
</table>

**IBMQREP_COLVERSION table (z/OS)**

The IBMQREP_COLVERSION table is created on z/OS operating systems to allow the Q Capture and Capture programs to keep track of different versions of a source table.

**Server:** Q Capture server

**Default schema:** ASN
Unique index: LSN, TABLEID1, TABLEID2, POSITION

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 registration or Q subscription for a source table is first activated, and then each time the source table is altered.

ASN is the only allowed schema for this table.

Table 69 provides a brief description of the columns in the IBMQREP_TRG_COLS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| LSN         | Data type: CHAR(10) 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. |
| 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 |
### IBMQREP COLVERSION table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 timespan 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>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| EOL_TIMEOUT | Data type: TIMESTAMP; Nullable: No  
The timestamp at the Q Capture server when the row was inserted into this table. |

### 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

[Table 71](#) provides a brief description of the columns in the IBMQREP_IGNTRAN table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
Table 71. Columns in the IBMQREP_IGNTRAN table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| PLANNNAME   | **Data type:** CHAR(8); **Nullable:** Yes  
  The plan name for the transaction that you want to ignore. |
| 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:  
  
  **Y (default)**  
  Tracing is enabled. Each time a transaction is ignored, a row is inserted  
  into the IBMQREP_IGNTRANTRC table and a message is issued.  
  
  **N**  
  Tracing is disabled. |

**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

**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 72 provides a brief description of the columns in the IBMQREP_IGNTRANTRC table.

Table 72. Columns in the IBMQREP_IGNTRANTRC table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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  
  The authorization token (job name) for the transaction that was ignored. |
| PLANNAME    | **Data type:** CHAR(8); **Nullable:** Yes  
  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 72. Columns in the IBMQREP_IGNTRANTRC table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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_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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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)**  
Messages contain all the row (update, insert, or delete) operations within a transaction, and information about the transaction.  
**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)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_ACTION</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>S (default)</strong></td>
<td>The Q Capture program stops.</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>This value is deprecated and is treated the same as a value of S.</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>After you fix the send queue error, you must take one of the following actions:</td>
</tr>
<tr>
<td></td>
<td>• Issue a startq command to tell Q Capture to start putting messages on the queue (set its state to active).</td>
</tr>
<tr>
<td></td>
<td>• Stop and start the Q Capture program.</td>
</tr>
<tr>
<td></td>
<td>• Use the Q Capture reinit command to reload all Q subscriptions from the Q Capture control tables.</td>
</tr>
<tr>
<td>LOB_TOO_BIG_ACTION</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Q (default)</strong></td>
<td>The Q Capture program follows the error action that is defined for the send queue.</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>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.</td>
</tr>
<tr>
<td>XML_TOO_BIG_ACTION</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Q (default)</strong></td>
<td>The Q Capture program follows the error action that is defined for the send queue.</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 73. Columns in the IBMQREP_SENDQUEUES table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SENDRAW_IFERROR</td>
<td>Data type: CHAR(1); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>Specifies whether to send raw hex strings if code page conversion fails.</td>
</tr>
<tr>
<td></td>
<td>N (default)</td>
</tr>
<tr>
<td></td>
<td>No, the data is not sent.</td>
</tr>
<tr>
<td></td>
<td>Y Yes, the raw hex strings are sent.</td>
</tr>
</tbody>
</table>

**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 provides a brief description of the columns in the IBMQREP_SIGNAL table.

Table 74. Columns in the IBMQREP_SIGNAL table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL_TIME</td>
<td>Data type: TIMESTAMP; Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td>SIGNAL_TYPE</td>
<td>Data type: VARCHAR(30); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates the type of signal that was posted:</td>
</tr>
<tr>
<td>CMD</td>
<td>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.</td>
</tr>
<tr>
<td>USER</td>
<td>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).</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL_SUBTYPE</td>
<td>(Continued)</td>
</tr>
<tr>
<td>ADDCOL</td>
<td>Add one column to an active, unidirectional Q subscription or to a publication.</td>
</tr>
<tr>
<td>STARTQ</td>
<td>Start putting messages on a specified queue or all inactive queues.</td>
</tr>
<tr>
<td>STOPQ</td>
<td>Stop putting messages on a specified queue or all inactive queues.</td>
</tr>
<tr>
<td>P2PNEW2MEMB</td>
<td>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.</td>
</tr>
<tr>
<td>P2PMEMB2NEW</td>
<td>An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at active servers.</td>
</tr>
<tr>
<td>P2PMEMB2INIT</td>
<td>An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at active servers.</td>
</tr>
<tr>
<td>P2PSPOOLING</td>
<td>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.</td>
</tr>
<tr>
<td>P2PLOADDONE</td>
<td>An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at a new server.</td>
</tr>
<tr>
<td>P2PSUBSTOP</td>
<td>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.</td>
</tr>
<tr>
<td>P2PSUBSTOPPING</td>
<td>An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the remaining active servers.</td>
</tr>
<tr>
<td>P2PREADYSTOP</td>
<td>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.</td>
</tr>
<tr>
<td>P2PNORECAPTURE</td>
<td>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.</td>
</tr>
<tr>
<td>REPPLICATE_LOAD</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 74. Columns in the IBMQREP_SIGNAL table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL_INPUT_IN</td>
<td><strong>Data type:</strong> VARCHAR(500); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>If the SIGNAL_TYPE=USER, then this column contains user-defined input. If the</td>
</tr>
<tr>
<td></td>
<td>SIGNAL_TYPE=CMD, then this value depends on the SIGNAL_SUBTYPE value:</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + CAPSTART</strong></td>
</tr>
<tr>
<td></td>
<td>The Q subscription or publication name.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + CAPSTOP</strong></td>
</tr>
<tr>
<td></td>
<td>The Q subscription or publication name.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + LOADDONE</strong></td>
</tr>
<tr>
<td></td>
<td>The Q subscription or publication name.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + STOP</strong></td>
</tr>
<tr>
<td></td>
<td>NULL (no value is required).</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + IGNORETRANS</strong></td>
</tr>
<tr>
<td></td>
<td>NULL (no value is required).</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + QINERROR</strong></td>
</tr>
<tr>
<td></td>
<td>For a user application, the name of the send queue that is in error. For</td>
</tr>
<tr>
<td></td>
<td>the Q Apply program, the name of the send queue that is in error and</td>
</tr>
<tr>
<td></td>
<td>the ASN message number and space-separated tokens.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + REINIT_SUB</strong></td>
</tr>
<tr>
<td></td>
<td>The Q subscription or publication name.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + ADDCOL</strong></td>
</tr>
<tr>
<td></td>
<td>The Q subscription or publication name and the column name,</td>
</tr>
<tr>
<td></td>
<td>separated by a semicolon. For example, QSUB1;COL10.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + STARTQ</strong></td>
</tr>
<tr>
<td></td>
<td>The queue name or ALL.</td>
</tr>
<tr>
<td></td>
<td><strong>CMD + STOPQ</strong></td>
</tr>
<tr>
<td></td>
<td>The queue name or ALL.</td>
</tr>
<tr>
<td>SIGNAL_STATE</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates the status of the signal.</td>
</tr>
<tr>
<td>P (default)</td>
<td>The signal is pending; the Q Capture program did not receive it yet.</td>
</tr>
<tr>
<td>R</td>
<td>The Q Capture program received the signal.</td>
</tr>
<tr>
<td>C</td>
<td>The Q Capture program completed processing the signal.</td>
</tr>
<tr>
<td>F</td>
<td>The signal failed. For example, the Q Capture program cannot perform a</td>
</tr>
<tr>
<td></td>
<td>CAPSTART because the Q subscription or publication is faulty.</td>
</tr>
<tr>
<td>SIGNAL_LSN</td>
<td><strong>Data type:</strong> CHAR(10) FOR BIT DATA; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The logical log sequence number of the log record for the insert into the</td>
</tr>
<tr>
<td></td>
<td>SIGNAL table.</td>
</tr>
</tbody>
</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>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNAME</td>
<td><strong>Data type:</strong> VARCHAR(30); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the Q subscription or publication for this source table.</td>
</tr>
<tr>
<td>SRC_COLNAME</td>
<td><strong>Data type:</strong> VARCHAR(30); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the column in the source table for which changes will be captured.</td>
</tr>
<tr>
<td>IS_KEY</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>0 (default)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>n</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Restriction:</strong> Large-object (LOB) columns and LONG columns cannot be used in the replication or publishing key.</td>
</tr>
<tr>
<td>COL_OPTIONS_FLAG</td>
<td><strong>Data type:</strong> CHAR(10); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>The first character determines whether the before-image value of the column is published or replicated. The first character can have the following values:</td>
</tr>
<tr>
<td></td>
<td><strong>N</strong></td>
</tr>
<tr>
<td></td>
<td>No before-image values are sent for this column.</td>
</tr>
<tr>
<td></td>
<td><strong>D</strong></td>
</tr>
<tr>
<td></td>
<td>Before-image values are sent for delete operations.</td>
</tr>
<tr>
<td></td>
<td><strong>U</strong></td>
</tr>
<tr>
<td></td>
<td>Before-image values are sent for update operations.</td>
</tr>
<tr>
<td></td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td></td>
<td>Before-image values are sent for both delete and update operations.</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> NNNNNNNNNNN</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

**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>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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(30); VARCHAR(128) for DB2 for z/OS Version 8 new-function mode; 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(30); VARCHAR(128) for DB2 for z/OS Version 8 new-function mode; Nullable: Yes

The schema name or high-level qualifier of the target table or stored procedure for a Q subscription.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| TARGET_NAME    | **Data type:** VARCHAR(128); VARCHAR(18) for DB2 for z/OS Version 7 and Version 8 compatibility mode; **Nullable:** Yes  
    The name of the target table for a Q subscription. |
| 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. |
<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.                                                                                                                                                                                                                                  |
| 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 the LOAD from CURSOR utility, EXPORT and IMPORT utilities, or EXPORT and LOAD utilities, depending on the LOAD_TYPE 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)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 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 subscription is being deactivated within the peer-to-peer group. |
| 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 |
| 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 |
### Table 77. Columns in the IBMQREP_SUBS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_MEMBERS</td>
<td><strong>Data type</strong>: CHAR(254) FOR BIT DATA; <strong>Nullable</strong>: Yes</td>
</tr>
<tr>
<td></td>
<td>This column is updated by the Q Capture program when members join or leave a peer-to-peer group.</td>
</tr>
<tr>
<td>OPTIONS_FLAG</td>
<td><strong>Data type</strong>: CHAR(4) FOR BIT DATA; <strong>Nullable</strong>: No, with default</td>
</tr>
<tr>
<td></td>
<td>Reserved for future.</td>
</tr>
<tr>
<td>SUPPRESS_DELETES</td>
<td><strong>Data type</strong>: CHAR(1); <strong>Nullable</strong>: No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that tells the Q Capture program whether to send rows that were deleted from the source table:</td>
</tr>
<tr>
<td></td>
<td>N (default)</td>
</tr>
<tr>
<td></td>
<td>Send deleted rows.</td>
</tr>
<tr>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Do not send deleted rows.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td><strong>Data type</strong>: VARCHAR(200); <strong>Nullable</strong>: Yes</td>
</tr>
<tr>
<td></td>
<td>A user-supplied description of the Q subscription or publication.</td>
</tr>
<tr>
<td>TOPIC</td>
<td><strong>Data type</strong>: VARCHAR(256); <strong>Nullable</strong>: Yes</td>
</tr>
<tr>
<td></td>
<td>A user-supplied topic to be included in the JMS-compliant (MQRFH2) message header for each XML message that is sent for the publication.</td>
</tr>
<tr>
<td>CAPTURE_LOAD</td>
<td><strong>Data type</strong>: CHAR(1); <strong>Nullable</strong>: No, with default</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>W (default)</td>
</tr>
<tr>
<td></td>
<td>Q Capture issues a warning message after the load completes.</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

### IBMQREP_TABVERSION table (z/OS)

The ASN.IBMQREP_TABVERSION table is created on z/OS operating systems to allow 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 registration or Q subscription for a source table is first activated, and then each time the source table is altered.

**Server**: Q Capture server

**Default schema**: ASN

**Unique index**: LSN, TABLEID1, TABLEID2, VERSION

**Important**: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

[Table 78 on page 428](#) provides a brief description of the columns in the IBMQREP_TABVERSION table.
Table 78. Columns in the IBMQREP_TABVERSION table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSN</td>
<td>Data type: CHAR(10) FOR BIT DATA; Nullable: No</td>
</tr>
<tr>
<td>TABLEID1</td>
<td>Data type: SMALLINT; Nullable: No</td>
</tr>
<tr>
<td>TABLEID2</td>
<td>Data type: SMALLINT; Nullable: No</td>
</tr>
<tr>
<td>VERSION</td>
<td>Data type: INTEGER; Nullable: No</td>
</tr>
<tr>
<td>SOURCE_OWNER</td>
<td>Data type: VARCHAR(128); Nullable: No</td>
</tr>
<tr>
<td>SOURCE_NAME</td>
<td>Data type: VARCHAR(128); Nullable: No</td>
</tr>
</tbody>
</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 79 describes the control tables at the Q Apply server.

Table 79. Control tables at the Q Apply server

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;IBMQREP_APPENQ table&quot; on page 429</td>
<td>Ensures that only one Q Apply program with a given schema is running per Q Apply server.</td>
</tr>
<tr>
<td>&quot;IBMQREP_APPENVINFO table&quot; on page 430</td>
<td>Stores environment variables and other information that replication tools use to access remote programs.</td>
</tr>
<tr>
<td>&quot;IBMQREP_APPLYMON table&quot; on page 430</td>
<td>Contains statistics about the performance of a Q Apply program for each receive queue.</td>
</tr>
<tr>
<td>&quot;IBMQREP_APPLYPARMS table&quot; on page 434</td>
<td>Contains parameters that you can specify to control the operations of a Q Apply program.</td>
</tr>
<tr>
<td>&quot;IBMQREP_APPLYTRACE table&quot; on page 438</td>
<td>Contains informational, warning, and error messages from the Q Apply program.</td>
</tr>
<tr>
<td>&quot;IBMQREP_DELTOMB table&quot; on page 439</td>
<td>An internal table used by the Q Apply program to record conflicting deletes in peer-to-peer replication.</td>
</tr>
<tr>
<td>&quot;IBMQREP_DONEMSG table&quot; on page 440</td>
<td>An internal table used by the Q Apply program to record which messages were processed.</td>
</tr>
<tr>
<td>&quot;IBMQREP_EXCEPTIONS table&quot; on page 440</td>
<td>Contains row changes that could not be applied because of conflicts, errors, or rollbacks.</td>
</tr>
</tbody>
</table>
Table 79. Control tables at the Q Apply server (continued)

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;IBMQREP_RECVQUEUES table&quot; on page 443</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SAVERI table&quot; on page 445</td>
<td>An internal table that the Q Apply program uses to store referential integrity constraints that are dropped while targets are being loaded.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SPILEDROW table&quot; on page 447</td>
<td>An internal table that the Q Apply program uses to keep track of rows sent to a temporary spill queue.</td>
</tr>
<tr>
<td>&quot;IBMQREP_SPILLQS table&quot; on page 446</td>
<td>Identifies temporary spill queues that will hold changes to source tables before they are applied to targets.</td>
</tr>
<tr>
<td>&quot;IBMQREP_TRG_COLS table&quot; on page 454</td>
<td>Contains information about the mapping between source and target columns.</td>
</tr>
<tr>
<td>&quot;IBMQREP_TARGETS table&quot; on page 447</td>
<td>Contains information about target tables or stored procedures, and options for Q subscriptions.</td>
</tr>
</tbody>
</table>

**IBMQREP_APPENQ table**

The IBMQREP_APPENQ 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_APPENQ table ensures that:

- **For DB2 for Linux, UNIX, and Windows,** only one Q Apply program with a given schema is running per database.
- **For non-data-sharing DB2 for z/OS,** only one Q Apply program with a given schema is running per subsystem.
- **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 80** provides a brief description of the column in the IBMQREP_APPENQ table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKNAME</td>
<td>Data type: INTEGER; Nullable: Yes</td>
</tr>
</tbody>
</table>

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

**Unique index:** None

**Important:** Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Data type: VARCHAR(30); Nullable: No</td>
</tr>
<tr>
<td>STARTTIME</td>
<td>The timestamp when the Q Apply program started.</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>The TCP/IP host name of the server where the Q Apply program is running.</td>
</tr>
<tr>
<td>LOGFILE</td>
<td>The path and file name of the Q Apply diagnostic log file.</td>
</tr>
<tr>
<td>TMPDIR</td>
<td>The path of the directory where the Inter-Process Communication (IPC) key of the Q Apply program is located.</td>
</tr>
<tr>
<td>ASNUSEMQCLIENT</td>
<td>The value of the Q replication ASNUSEMQCLIENT environment variable.</td>
</tr>
<tr>
<td>MQSERVER</td>
<td>The value of the WebSphere MQ MQSERVER environment variable.</td>
</tr>
<tr>
<td>MQCHLLIB</td>
<td>The value of the WebSphere MQ MQCHLLIB environment variable.</td>
</tr>
<tr>
<td>MQCHLTAB</td>
<td>The value of the WebSphere MQ MQCHLTAB environment variable.</td>
</tr>
</tbody>
</table>

**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 82 provides a brief description of the columns in the IBMQREP_APPLYMON table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Apply server when the row was inserted into the IBMQREP_APPLYMON table.</td>
</tr>
<tr>
<td>RECVQ</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue that this row of Q Apply performance statistics pertains to.</td>
</tr>
<tr>
<td>QSTART_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Apply server when the receive queue was started.</td>
</tr>
<tr>
<td>CURRENT_MEMORY</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The amount of memory in bytes that the Q Apply browser thread used for reading transactions from this queue.</td>
</tr>
<tr>
<td>QDEPTH</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The queue depth (number of messages on the queue). The Q_PERCENT_FULL column expresses the fullness of the queue as a percentage.</td>
</tr>
<tr>
<td>END2END_LATENCY</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>QLATENCY</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>APPLY_LATENCY</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>TRANS_APPLIED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The total number of transactions from this receive queue that the Q Apply committed to the target.</td>
</tr>
<tr>
<td>ROWS_APPLIED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The total number of insert, update, and delete operations from this receive queue that the Q Apply program applied to the target.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OLDEST_TRANS</strong></td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No \n\nA 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: \n\n- 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.) \n- The latest heartbeat time, if no transactions are being processed and the heartbeat message arrived after the oldest applied transaction. \n- The value 1900-01-01-00.00.00.000000 if the Q Apply program has not seen any messages (transaction or heartbeat).</td>
</tr>
<tr>
<td><strong>OLDEST_INFLT_TRANS</strong></td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No \n\nAt each monitor interval OLDEST_INFLT_TRANS represents: \n\n- 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. \n- The value NULL, if Q Apply is not processing transactions. \n- The value NULL, if Q Apply has not seen any transaction messages so far. \n\nOLDEST_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.</td>
</tr>
<tr>
<td><strong>OKSQLSTATE_ERRORS</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No \n\nThe 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.</td>
</tr>
<tr>
<td><strong>HEARTBEAT_LATENCY</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No \n\nThe 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.</td>
</tr>
<tr>
<td><strong>KEY_DEPENDENCIES</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No \n\nThe total number of replication key constraints that were detected, forcing transactions to be serialized.</td>
</tr>
<tr>
<td><strong>UNIQ_DEPENDENCIES</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No \n\nThe total number of unique index constraints that were detected, forcing transactions to be serialized.</td>
</tr>
<tr>
<td><strong>UNIQ_RETRIES</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No \n\nThe number of times that the Q Apply program tried to re-apply rows that were not applied in parallel because of unique index constraints.</td>
</tr>
<tr>
<td><strong>JOB_DEPENDENCIES</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No \n\nThe number of transactions that are delayed because of correlation ID dependencies.</td>
</tr>
</tbody>
</table>
### Table 82. Columns in the IBMQREP_APPLYMON table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPTURE_LATENCY</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The average elapsed milliseconds between the time the Q Capture program reads transaction commit statements in the DB2 recovery log and the time Q Capture puts the last message for the transactions on the send queue.</td>
</tr>
<tr>
<td>OLDEST_COMMIT_LSN</td>
<td><strong>Data type:</strong> CHAR(10) FOR BIT DATA; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>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'.</td>
</tr>
<tr>
<td></td>
<td>You can use the OLDEST_COMMIT_LSN value on z/OS as the value for the <strong>maxcmtnseq</strong> parameter when you need to restart the Q Capture program from a known point in the recovery log.</td>
</tr>
<tr>
<td>ROWS_PROCESSSED</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number of rows that were read from receive queues and applied but not yet committed to the target.</td>
</tr>
<tr>
<td>Q_PERCENT_FULL</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>OLDEST_COMMIT_SEQ</td>
<td><strong>Data type:</strong> CHAR(10) FOR BIT DATA; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>You can use the OLDEST_COMMIT_SEQ value on Linux, UNIX, and Windows as the value for the <strong>maxcmtnseq</strong> parameter when you need to restart the Q Capture program from a known point in the recovery log.</td>
</tr>
</tbody>
</table>

### 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 83 provides a brief description of the columns in the IBMQREP_APPLYPARMS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMGR</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the WebSphere MQ queue manager that the Q Apply program works with.</td>
</tr>
<tr>
<td>MONITOR_LIMIT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td>TRACE_LIMIT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td>MONITOR_INTERVAL</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>How often, in milliseconds, the Q Apply program adds a row to the IBMQREP_APPLYMON table. Default: 300000</td>
</tr>
<tr>
<td>PRUNE_INTERVAL</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>How often, in seconds, the Q Apply program automatically prunes rows in the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables. Default: 300</td>
</tr>
<tr>
<td>AUTOSTOP</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that tells the Q Apply program whether to stop when all receive queues have been emptied once.</td>
</tr>
<tr>
<td></td>
<td><strong>N (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Apply program continues running after all receive queues have been emptied once.</td>
</tr>
<tr>
<td></td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Apply program stops when all receive queues have been emptied once.</td>
</tr>
<tr>
<td>LOGREUSE</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether the Q Apply program reuses the Q Apply log file or appends to it.</td>
</tr>
<tr>
<td></td>
<td><strong>N (default)</strong></td>
</tr>
<tr>
<td></td>
<td>The Q Apply program appends new information to an existing Q Apply log file when it restarts.</td>
</tr>
<tr>
<td></td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td></td>
<td>On restart, the Q Apply program reuses its log file by clearing the file then writing to the blank file.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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 Version 9.7 the value is 0907. Other ARCH_LEVEL values are 0802, 0901, and 0905.  
**Attention:** When updating the IBMQREP_APPLYARMS 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 is unavailable (quiesced, failover HADR, stopped, etc.)  
**Y (default)**  
The Q Apply program stops if DB2 is unavailable.  
**N**  
The Q Apply program continues running if DB2 is unavailable. |
| PWDFILE        | **Data type:** CHAR(1); **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. |
| 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. 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. |
**Table 83. Columns in the IBMQREP_APPLYPARMS table (continued)**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOADCOPY_PATH</td>
<td><strong>Data type:</strong> VARCHAR(1040); <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>NICKNAME_COMMIT_COUNT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>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, which must use the EXPORT and IMPORT utilities.</td>
</tr>
<tr>
<td></td>
<td>The default is <code>nickname_commit_count=10</code>.</td>
</tr>
<tr>
<td></td>
<td>This parameter can be used to tune the performance of the DB2 IMPORT utility by reducing the number of commits for federated targets.</td>
</tr>
<tr>
<td>SPILL_COMMIT_COUNT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>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 <code>spill_commit_count=10</code>. 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.</td>
</tr>
<tr>
<td>LOAD_DATA_BUFF_SIZE</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>CLASSICLOAD_FILE_SIZE</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>This parameter applies only to automatic loads of z/OS target tables from Classic sources.</td>
</tr>
</tbody>
</table>
Table 83. Columns in the IBMQREP_APPLYPARMS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_PARALLEL_LOADS</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>z/OS</strong></td>
</tr>
<tr>
<td></td>
<td>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 <strong>max_parallel_loads=1</strong>. If errors occur, reset the value to 1.</td>
</tr>
<tr>
<td></td>
<td><strong>Linux UNIX Windows</strong></td>
</tr>
<tr>
<td></td>
<td>On Linux, UNIX, and Windows the default is 15 parallel loads.</td>
</tr>
<tr>
<td>COMMIT_COUNT</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>Specifies the number of transactions that each Q Apply agent thread applies to the target table within a commit scope. By default (<strong>commit_count=1</strong>), the agent threads commit after each transaction that they apply. By increasing <strong>commit_count</strong> and grouping more transactions within the commit scope, you might see improved performance. <strong>Recommendation:</strong> Use a higher value for <strong>commit_count</strong> 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.</td>
</tr>
<tr>
<td>INSERT_BIDI_SIGNAL</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether the Q Capture and Q Apply programs use P2PNORECAPTURE signal inserts to prevent recapture of transactions in bidirectional replication.</td>
</tr>
<tr>
<td></td>
<td><strong>Y (default)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>N</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 84 provides a brief description of the columns in the IBMQREP_APPLYTRACE table.

### Table 84. Columns in the IBMQREP_APPLYTRACE table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATION</td>
<td><strong>Data type:</strong> CHAR(8); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The type of message from the Q Apply program:</td>
</tr>
<tr>
<td></td>
<td><strong>INFO</strong> Describe actions that the Q Apply program takes.</td>
</tr>
<tr>
<td></td>
<td><strong>WARNING</strong> Describe conditions that could cause errors for the Q Apply program.</td>
</tr>
<tr>
<td></td>
<td><strong>ERROR</strong> Describe errors encountered by the Q Apply program.</td>
</tr>
<tr>
<td>TRACE_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The time at the Q Apply server when the row was inserted into this table.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td><strong>Data type:</strong> VARCHAR(1024); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The ASN message ID followed by the message text. This column contains English-only text.</td>
</tr>
<tr>
<td>REASON_CODE</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The reason code for the replication error message.</td>
</tr>
<tr>
<td>MQ_CODE</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The reason code for the WebSphere MQ error message.</td>
</tr>
</tbody>
</table>

### 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 85 provides a brief description of the columns in the IBMQREP_DELTOMB table.

### Table 85. Columns in the IBMQREP_DELTOMB table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_OWNER</td>
<td><strong>Data type:</strong> VARCHAR(30), VARCHAR(128) for DB2 for z/OS Version 8 new-function mode; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The owner name of the target table for which the conflicting delete was recorded.</td>
</tr>
</tbody>
</table>
Table 85. Columns in the `IBMQREP_DELTOMB` table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_NAME</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the table for which the conflicting delete was recorded</td>
</tr>
<tr>
<td>VERSION_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp of the conflicting delete at the originating server.</td>
</tr>
<tr>
<td>VERSION_NODE</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>Identifies the server in a peer-to-peer group where the conflicting delete</td>
</tr>
<tr>
<td></td>
<td>originated.</td>
</tr>
<tr>
<td>KEY_HASH</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The hash value of the key for the conflicting delete.</td>
</tr>
<tr>
<td>PACKED_KEY</td>
<td><strong>Data type:</strong> VARCHAR(4096); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The packed key value of the conflicting delete.</td>
</tr>
</tbody>
</table>

**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 86 provides a brief description of the columns in the IBMQREP_DONEMSG table.

Table 86. Columns in the `IBMQREP_DONEMSG` table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECVQ</td>
<td><strong>Data type:</strong> VARCHAR(97); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue where the message arrived.</td>
</tr>
<tr>
<td>MGMSGID</td>
<td><strong>Data type:</strong> CHAR(24) FOR BIT DATA; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The WebSphere MQ message identifier of the message.</td>
</tr>
</tbody>
</table>

**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

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 87 provides a brief description of the columns in the IBMQREP_EXCEPTIONS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCEPTION_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Apply server when the error or conflict occurred.</td>
</tr>
<tr>
<td></td>
<td>Default: Current timestamp</td>
</tr>
<tr>
<td>RECVQ</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue where the transaction message arrived.</td>
</tr>
<tr>
<td>SRC_COMMIT_LSN</td>
<td><strong>Data type:</strong> VARCHAR(48) FOR BIT DATA, VARCHAR(10) FOR BIT DATA for z/OS Version 7; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The logical log sequence number at the Q Capture server for the transaction.</td>
</tr>
<tr>
<td>SRC_TRANS_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Capture server for the transaction.</td>
</tr>
<tr>
<td>SUBNAME</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the Q subscription that the transaction belonged to.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>REASON</td>
<td>A description of the error or conflict that caused the transaction to be logged into the IBMQREP_EXCEPTIONS table.</td>
</tr>
<tr>
<td>NOTFOUND</td>
<td>An attempt to delete or update a row that did not exist.</td>
</tr>
<tr>
<td>DUPLICATE</td>
<td>An attempt to insert a row that was already present.</td>
</tr>
<tr>
<td>CHECKFAILED</td>
<td>The conflict detection rule was to check all values or check changed values, and a nonkey value was not as expected.</td>
</tr>
<tr>
<td>LOBXMLTOOBIG</td>
<td>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.</td>
</tr>
<tr>
<td>SQLERROR</td>
<td>An SQL error occurred, and it was not on the list of acceptable errors in the OKSQLSTATES column of the IBMQREP_Targets table.</td>
</tr>
<tr>
<td>OKSQLSTATE</td>
<td>An SQL error occurred, and it was on the list of acceptable errors in the OKSQLSTATES column of the IBMQREP_Targets table.</td>
</tr>
<tr>
<td>P2PDUPKEY</td>
<td>In peer-to-peer replication, a key update failed because a target row with the same key already existed, but was newer.</td>
</tr>
<tr>
<td>P2PNOTFOUND</td>
<td>In peer-to-peer replication, a delete or update failed because the target row didn't exist.</td>
</tr>
<tr>
<td>P2PVERLOSER</td>
<td>In peer-to-peer replication, a delete or update failed because the target row was newer than the row in the change message.</td>
</tr>
<tr>
<td>P2PINSERTED</td>
<td>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.</td>
</tr>
<tr>
<td>SQLCODE</td>
<td>The SQL code returned by DB2 for the transaction.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>The SQL state number returned by DB2 for the transaction.</td>
</tr>
<tr>
<td>SQLERRMC</td>
<td>The error message tokens from the SQLCA structure used for executing the transaction.</td>
</tr>
<tr>
<td>OPERATION</td>
<td>The type of SQL operation that failed. Possible values are INSERT, INSERT(LOAD), DELETE, DELETE(LOAD), UPDATE, UPDATE(LOAD), KEY UPDATE, KEY UPDATE(LOAD).</td>
</tr>
</tbody>
</table>
Table 87. Columns in the IBMQREP_EXCEPTIONS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Data type: CLOB; Nullable: No</td>
</tr>
<tr>
<td></td>
<td>A SQL statement that describes the row that caused an error.</td>
</tr>
<tr>
<td>IS_APPLIED</td>
<td>Data type: CHAR(1); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether the row was applied to the target table even though it was entered into the IBMQREP_EXCEPTIONS table.</td>
</tr>
<tr>
<td></td>
<td>Y The row was applied because the CONFLICT_ACTION specified for the Q subscription was F (force).</td>
</tr>
<tr>
<td></td>
<td>N The transaction was not applied.</td>
</tr>
<tr>
<td>CONFLICT_RULE</td>
<td>Data type: CHAR(1); Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The type of conflict detection that resulted in the row being entered in the IBMQREP_EXCEPTIONS table.</td>
</tr>
<tr>
<td></td>
<td>K Only key values were checked.</td>
</tr>
<tr>
<td></td>
<td>C Changed nonkey values as well as key values were checked.</td>
</tr>
<tr>
<td></td>
<td>A All values were checked.</td>
</tr>
<tr>
<td>SRC_TRANS_ID</td>
<td>Data type: VARCHAR(48); Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The identifier for the transaction that the row that could not be applied belongs to.</td>
</tr>
</tbody>
</table>

**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 88 provides a brief description of the columns in the IBMQREP_RECVQUEUES table.

Table 88. Columns in the IBMQREP_RECVQUEUES table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPQMAPNAME</td>
<td>Data type: VARCHAR(128); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the replication queue map that includes this receive queue.</td>
</tr>
<tr>
<td>RECVQ</td>
<td>Data type: VARCHAR(48); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue for this Q subscription.</td>
</tr>
</tbody>
</table>
### Table 88. Columns in the IBMQREP_RECVQUEUES table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SENDQ</strong></td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The name of the send queue that is used by the Q Capture program for this Q subscription.</td>
</tr>
<tr>
<td><strong>ADMINQ</strong></td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NUM_APPLY_AGENTS</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>MEMORY_LIMIT</strong></td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>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. Default: 32 MB</td>
</tr>
<tr>
<td><strong>CAPTURE_SERVER</strong></td>
<td><strong>Data type:</strong> VARCHAR(18); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the database where the Q Capture program that uses this receive queue runs.</td>
</tr>
<tr>
<td></td>
<td>This is a location name.</td>
</tr>
<tr>
<td><strong>CAPTURE_ALIAS</strong></td>
<td><strong>Data type:</strong> VARCHAR(8); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The DB2 database alias that corresponds to the Q Capture server that is named in the CAPTURE_SERVER column.</td>
</tr>
<tr>
<td><strong>CAPTURE_SCHEMA</strong></td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>The schema of the Q Capture program that uses this receive queue. Default: ASN</td>
</tr>
<tr>
<td><strong>STATE</strong></td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that shows the receive queue’s current status.</td>
</tr>
<tr>
<td></td>
<td><strong>A (default)</strong></td>
</tr>
<tr>
<td></td>
<td>Active: The Q Apply program is processing and applying the transactions from this queue.</td>
</tr>
<tr>
<td></td>
<td>I Inactive: A severe error was encountered on the queue.</td>
</tr>
<tr>
<td><strong>STATE_TIME</strong></td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Apply server of the last state change for this receive queue. Default: Current timestamp</td>
</tr>
<tr>
<td><strong>STATE_INFO</strong></td>
<td><strong>Data type:</strong> CHAR(8); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number for the ASN message about the queue state. For details, see the IBMQREP_APPLYTRACE table, or the Q Apply diagnostic log.</td>
</tr>
<tr>
<td><strong>DESCRIPTION</strong></td>
<td><strong>Data type:</strong> VARCHAR(254); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>A user-supplied description of the replication queue map that contains this receive queue.</td>
</tr>
</tbody>
</table>
Table 88. Columns in the IBMQREP_RECVQUEUES table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |

**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 89 provides a brief description of the columns in the IBMQREP_SAVERI table.

Table 89. Columns in the IBMQREP_SAVERI table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNAME</td>
<td><strong>Data type:</strong> VARCHAR(132); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the Q subscription to which the target tables belong.</td>
</tr>
<tr>
<td>RECVQ</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue that is specified for the Q subscription.</td>
</tr>
<tr>
<td>CONSTNAME</td>
<td><strong>Data type:</strong> VARCHAR (128), VARCHAR(18) for z/OS Version 7; <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The unique name of the constraint.</td>
</tr>
</tbody>
</table>
### Table 89. Columns in the IBMQREP_SAVERI table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| TABSCHEMA   | **Data type:** VARCHAR(30), VARCHAR(128) for DB2 for z/OS Version 8 new-function mode; **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 90 provides a brief description of the columns in the IBMQREP_SPILLQS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPILLQ</td>
<td>Data type: VARCHAR(48); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the temporary spill queue that is used for this Q subscription.</td>
</tr>
<tr>
<td>SUBNAME</td>
<td>Data type: VARCHAR(132); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the Q subscription.</td>
</tr>
<tr>
<td>RECVQ</td>
<td>Data type: VARCHAR(48); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue that is used for this Q subscription.</td>
</tr>
</tbody>
</table>

**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 91 provides a brief description of the columns in the IBMQREP_SPILLEDROW table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPILLQ</td>
<td>Data type: VARCHAR2(48); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the spill queue where the message was temporarily stored.</td>
</tr>
<tr>
<td>MQMSGID</td>
<td>Data type: CHAR(24) FOR BIT DATA; Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The WebSphere MQ message identifier of the message.</td>
</tr>
</tbody>
</table>

**IBMQREP_TARGETS table**

The IBMQREP_TARGETS table contains information about target tables or stored procedures and the Q subscriptions to which they belong. This table stores details about Q subscription 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

Important: Do not alter this table by using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

Table 92 provides a brief description of the columns in the IBMQREP_TARGETS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 92. Columns in the IBMQREP_TARGETS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_NICKNAME</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The nickname that is assigned to the source table for automatic loads that</td>
</tr>
<tr>
<td></td>
<td>uses the LOAD from CURSOR utility when the Q Apply program is running on a</td>
</tr>
<tr>
<td></td>
<td>non-z/OS platform.</td>
</tr>
<tr>
<td>TARGET_OWNER</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The schema name or high-level qualifier of the target table or stored</td>
</tr>
<tr>
<td></td>
<td>procedure for this Q subscription.</td>
</tr>
<tr>
<td>TARGET_NAME</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the target table for this Q subscription.</td>
</tr>
<tr>
<td>TARGET_TYPE</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates the type of replication target.</td>
</tr>
<tr>
<td></td>
<td>1 <em>(default)</em></td>
</tr>
<tr>
<td></td>
<td>User table</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Consistent-change-data (CCD) table</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Stored procedure</td>
</tr>
<tr>
<td>FEDERATED_TGT_SRVR</td>
<td><strong>Data type:</strong> VARCHAR(18); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The name of the non-DB2 relational database that contains the Q subscription</td>
</tr>
<tr>
<td></td>
<td>target.</td>
</tr>
</tbody>
</table>
### Table 92. Columns in the IBMQREP_TARGETS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STATE</strong></td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that is inserted by the Q Apply program to describe the current state of the Q subscription.</td>
</tr>
<tr>
<td>I (default)</td>
<td>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.</td>
</tr>
<tr>
<td>L</td>
<td>The Q Capture program has begun activating the Q subscription by sending a subscription schema message, and is sending changes from the source table.</td>
</tr>
<tr>
<td>E</td>
<td>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.</td>
</tr>
<tr>
<td>D</td>
<td>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.</td>
</tr>
<tr>
<td>F</td>
<td>The Q Apply program is applying messages from the spill queue.</td>
</tr>
<tr>
<td>T</td>
<td>The Q Apply program is terminating because of an error. It deactivates the Q subscription, then empties and deletes the spill queue.</td>
</tr>
<tr>
<td>A</td>
<td>The Q Apply program is applying changes to the target.</td>
</tr>
<tr>
<td>R</td>
<td>The Q Apply program is resuming operations after a Q subscription was placed in the spilled state (S) by the <code>spillsub</code> parameter of the MODIFY or asqnacmd command. The <code>resumesub</code> parameter places the Q subscription in the resuming state (R) after target table maintenance is complete. 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.</td>
</tr>
<tr>
<td>S</td>
<td>The Q Apply program is placing rows for the Q subscription in a temporary spill queue. Specifying the <code>spillsub</code> parameter places the Q subscription in the spilling state so that you can perform maintenance on the target table.</td>
</tr>
<tr>
<td>W</td>
<td><strong>For peer-to-peer configurations with more than two servers and a load phase:</strong> 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.</td>
</tr>
<tr>
<td><strong>STATE_TIME</strong></td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>The timestamp at the Q Apply server of the last change in state for this Q subscription. Default: Current timestamp</td>
</tr>
<tr>
<td><strong>STATE_INFO</strong></td>
<td><strong>Data type:</strong> CHAR(8); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>The number for the ASN message about the Q subscription state. For details, see the IBMQREP_APPLYTRACE table or the Q Apply diagnostic log.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SUBTYPE</td>
<td>Data type: CHAR(1); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates the type of replication that the Q subscription is involved in.</td>
</tr>
<tr>
<td></td>
<td><strong>U</strong> (default)</td>
</tr>
<tr>
<td></td>
<td><strong>B</strong></td>
</tr>
<tr>
<td></td>
<td><strong>P</strong></td>
</tr>
<tr>
<td>CONFLICT_RULE</td>
<td>Data type: CHAR(1); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>K</strong> (default)</td>
</tr>
<tr>
<td></td>
<td><strong>C</strong></td>
</tr>
<tr>
<td></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td></td>
<td><strong>V</strong></td>
</tr>
<tr>
<td>CONFLICT_ACTION</td>
<td>Data type: CHAR(1); Nullable: No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that tells the Q Apply program what to do when a row change conflicts:</td>
</tr>
<tr>
<td></td>
<td><strong>I</strong> (default)</td>
</tr>
<tr>
<td></td>
<td><strong>F</strong></td>
</tr>
<tr>
<td></td>
<td><strong>D</strong></td>
</tr>
<tr>
<td></td>
<td><strong>S</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Q</strong></td>
</tr>
</tbody>
</table>

All conflicting rows are inserted into the IBMQREP_EXCEPTIONS table.
### Table 92. Columns in the IBMQREP_TARGETS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</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. 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.  
| 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.  
|                 | **Note:** 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.  
| 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.  

452  Replication and Event Publishing Guide and Reference
### Table 92. Columns in the IBMQREP_Targets table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAS_LOADPHASE</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether the target table will be loaded with data from the source.</td>
</tr>
<tr>
<td></td>
<td><strong>N (default)</strong>  The target will not be loaded.</td>
</tr>
<tr>
<td></td>
<td><strong>I</strong> An automatic load. The Q Apply program loads the target table.</td>
</tr>
<tr>
<td></td>
<td><strong>E</strong> A manual load. An application other than the Q Apply program loads the target table.</td>
</tr>
<tr>
<td>LOAD_TYPE</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A flag to indicate which utility is called to load the target table when HAS_LOADPHASE is <strong>I</strong> (automatic load).</td>
</tr>
<tr>
<td></td>
<td><strong>0 (default)</strong> The Q Apply program selects the load utility from among the options below.</td>
</tr>
<tr>
<td></td>
<td><strong>1</strong> Use the LOAD from CURSOR utility.</td>
</tr>
<tr>
<td></td>
<td><strong>2</strong> Use the EXPORT and IMPORT utilities.</td>
</tr>
<tr>
<td></td>
<td><strong>3</strong> Use the EXPORT and LOAD utilities.</td>
</tr>
<tr>
<td></td>
<td><strong>4</strong> Select from a Classic replication source and use the LOAD utility with the replace option.</td>
</tr>
<tr>
<td></td>
<td><strong>104</strong> Select from a Classic replication source and use the LOAD utility with resume option (z/OS targets) or insert option (Linux, UNIX, and Windows targets).</td>
</tr>
<tr>
<td></td>
<td><strong>5</strong> <strong>Linux, UNIX, and Windows targets:</strong> Select from a Classic replication source and use the IMPORT utility with replace option.</td>
</tr>
<tr>
<td></td>
<td><strong>105</strong> <strong>Linux, UNIX, and Windows targets:</strong> Select from a Classic replication source and use the IMPORT utility with insert option.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td><strong>Data type:</strong> VARCHAR(254); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>A user-supplied description of the Q subscription.</td>
</tr>
<tr>
<td>SEARCH_CONDITION</td>
<td><strong>Data type:</strong> VARCHAR(2048); <strong>Nullable:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>MODELQ</td>
<td><strong>Data type:</strong> VARCHAR(36); <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>The name of the model queue that the Q Apply program uses to create spill queues during the target loading process. Default: IBMQREPSPI.LMODELQ</td>
</tr>
</tbody>
</table>
Table 92. Columns in the IBMQREP_Targets table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCD_CONDENSED</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether a CCD target table is condensed or noncondensed.</td>
</tr>
<tr>
<td></td>
<td><strong>N (default)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

| 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 |

**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

**Non-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 93 provides a brief description of the columns in the IBMQREP_TRG_COLS table.

Table 93. Columns in the IBMQREP_TRG_COLS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECVQ</td>
<td><strong>Data type:</strong> VARCHAR(48); <strong>Nullable:</strong> No</td>
</tr>
<tr>
<td></td>
<td>The name of the receive queue that is used for this Q subscription.</td>
</tr>
</tbody>
</table>
Table 93. Columns in the IBMQREP_TRG_COLS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNAME</td>
<td>Data type: VARCHAR(132); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the Q subscription.</td>
</tr>
<tr>
<td>SOURCE_COLNAME</td>
<td>Data type: VARCHAR(1024); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>This column can contain one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• The name of the source column that is being replicated or published</td>
</tr>
<tr>
<td></td>
<td>• An SQL or XML expression that is used to create the target column contents</td>
</tr>
<tr>
<td></td>
<td>• An auditing column in a consistent-change-data (CCD) table</td>
</tr>
<tr>
<td>TARGET_COLNAME</td>
<td>Data type: VARCHAR(128); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>The name of the target column. If the target is a stored procedure, this</td>
</tr>
<tr>
<td></td>
<td>column contains the name of the parameter to which the Q Apply program passes</td>
</tr>
<tr>
<td></td>
<td>the source column value.</td>
</tr>
<tr>
<td>TARGET_COLNO</td>
<td>Data type: INTEGER; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>A number assigned to a target column. If the target is a stored procedure,</td>
</tr>
<tr>
<td></td>
<td>this column contains a number assigned to the parameter to which the Q Apply</td>
</tr>
<tr>
<td></td>
<td>program passes the source column value.</td>
</tr>
<tr>
<td>MSG_COL_CODEPAGE</td>
<td>Data type: INTEGER; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>An identifier for the code page that is used to encode the value of the</td>
</tr>
<tr>
<td></td>
<td>source column.</td>
</tr>
<tr>
<td>MSG_COL_NUMBER</td>
<td>Data type: SMALLINT; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The source column’s order of appearance in a change message, starting from 0</td>
</tr>
<tr>
<td>MSG_COL_TYPE</td>
<td>Data type: SMALLINT; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The DB2 data type of the target column.</td>
</tr>
<tr>
<td>MSG_COL_LENGTH</td>
<td>Data type: SMALLINT; Nullable: Yes</td>
</tr>
<tr>
<td></td>
<td>The maximum data length defined on the target column.</td>
</tr>
<tr>
<td>IS_KEY</td>
<td>Data type: CHAR(1); Nullable: No</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether the source column is part of the key for the</td>
</tr>
<tr>
<td></td>
<td>source table. If the value of this flag does not match the target table key</td>
</tr>
<tr>
<td></td>
<td>definition, the Q Apply program rejects the schema message and invalidates</td>
</tr>
<tr>
<td></td>
<td>the Q subscription:</td>
</tr>
<tr>
<td></td>
<td>Y                          The column is part of the source table key.</td>
</tr>
<tr>
<td></td>
<td>N                          The column is not part of the source table key.</td>
</tr>
<tr>
<td></td>
<td>Restriction: Large-object (LOB) columns and LONG columns cannot be used in</td>
</tr>
<tr>
<td></td>
<td>the replication or publishing key.</td>
</tr>
<tr>
<td>MAPPING_TYPE</td>
<td>Data type: CHAR(1); Nullable: Yes, with default</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates the type of mapping between the source column and the</td>
</tr>
<tr>
<td></td>
<td>target column.</td>
</tr>
<tr>
<td></td>
<td>R                          A regular mapping where the value in the source</td>
</tr>
<tr>
<td></td>
<td>column corresponds to the value in the target column.</td>
</tr>
<tr>
<td></td>
<td>E                          An SQL expression is used to generate the target column</td>
</tr>
<tr>
<td></td>
<td>contents.</td>
</tr>
<tr>
<td></td>
<td>A                          An auditing column in a CCD table. The column does not</td>
</tr>
<tr>
<td></td>
<td>exist at the source table.</td>
</tr>
<tr>
<td></td>
<td>Default value: NULL</td>
</tr>
</tbody>
</table>
Table 93. Columns in the IBMQREP_TRG_COLS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_COL_MAP</td>
<td><strong>Data type:</strong> VARCHAR(2000); <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>The column position, data type, length, and code page for all of the columns</td>
</tr>
<tr>
<td></td>
<td>that are used in an SQL expression. Default value: NULL</td>
</tr>
<tr>
<td>BEF_TARG_COLNAME</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> Yes, with default</td>
</tr>
<tr>
<td></td>
<td>Specifies the before-image column name, if one exists (for CCD targets only).</td>
</tr>
<tr>
<td></td>
<td>Default value: NULL</td>
</tr>
</tbody>
</table>

Control tables at the Monitor control server

The control tables at the Monitor control server contain information about when, how, and whom you want the Replication Alert Monitor to contact when an alert condition occurs. For Linux, UNIX, Windows, and z/OS, you build these control tables to your specifications by using the ASNCLP command-line program or Replication Center. Replication on System i does not have Monitor control tables.

Table 94 describes the control tables at the Monitor control server.

Table 94. Control tables at the Monitor control server

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;IBMSNAP_ALERTS table&quot; on page 457</td>
<td>Contains a record of all the alerts issued by the Replication Alert Monitor.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_CONDITIONS table&quot; on page 458</td>
<td>Contains the alert conditions for which the Replication Alert Monitor will contact someone, and contains the group or individual’s name to contact if a particular condition occurs.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_CONTACTGRP table&quot; on page 463</td>
<td>Contains the individual contacts that make up the contact groups.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_CONTACTS table&quot; on page 464</td>
<td>Contains information on how the Replication Alert Monitor notifies each person or group when an alert condition that is associated with that contact name occurs.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_GROUPS table&quot; on page 465</td>
<td>Contains the name and description of each contact group.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_MONENQ table&quot; on page 466</td>
<td>Used to ensure that only one Replication Alert Monitor program is running per Monitor qualifier.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_MONPARMS table&quot; on page 465</td>
<td>Contains parameters that you can modify to control the operations of the Monitor program.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_MONSERVERS table&quot; on page 467</td>
<td>Contains the latest time that a server was monitored by a Replication Alert Monitor program (identified by a Monitor qualifier).</td>
</tr>
<tr>
<td>&quot;IBMSNAP_MONTRACE table&quot; on page 469</td>
<td>Contains messages from the Monitor program.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_MONTRAIL table&quot; on page 469</td>
<td>Contains information about each monitor cycle.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_SUSPENDS table&quot; on page 471</td>
<td>Contains information about temporary suspensions of the monitor program.</td>
</tr>
<tr>
<td>&quot;IBMSNAP_TEMPLATES table&quot; on page 472</td>
<td>Contains information about how often and how long the monitor program is suspended.</td>
</tr>
</tbody>
</table>
**IBMSNAP_ALERTS table**

The IBMSNAP_ALERTS table contains a record of all the alerts issued by the Replication Alert Monitor. The table records what alert conditions occur, at which servers they occur, and when they were detected.

**Server:** Monitor control server

**Non-unique index:** MONITOR_QUAL, COMPONENT, SERVER_NAME, SCHEMA_OR_QUAL, SET_NAME, CONDITION_NAME, ALERT_CODE

Table 95 provides a brief description of the columns in the IBMSNAP_ALERTS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_QUAL</td>
<td><strong>Data type:</strong> CHAR(18); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The Monitor qualifier that identifies which Replication Alert Monitor program issued the alert.</td>
</tr>
<tr>
<td>COMPONENT</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The replication component that is being monitored:</td>
</tr>
<tr>
<td></td>
<td>C CAPTURE PROGRAM</td>
</tr>
<tr>
<td></td>
<td>A APPLY PROGRAM</td>
</tr>
<tr>
<td></td>
<td>S Q CAPTURE PROGRAM</td>
</tr>
<tr>
<td></td>
<td>R Q APPLY PROGRAM</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td><strong>Data type:</strong> CHAR(18); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server where the alert condition occurred.</td>
</tr>
<tr>
<td>SERVER_ALIAS</td>
<td><strong>Data type:</strong> CHAR(8); <strong>Nullable:</strong> Yes.</td>
</tr>
<tr>
<td></td>
<td>The DB2 alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server where the alert condition occurred.</td>
</tr>
<tr>
<td>SCHEMA_OR_QUAL</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The Capture schema, Apply qualifier, Q Capture schema, or Q Apply schema that is being monitored.</td>
</tr>
<tr>
<td>SET_NAME</td>
<td><strong>Data type:</strong> CHAR(18); <strong>Nullable:</strong> No, with default; <strong>Default:</strong> Current subscription set.</td>
</tr>
<tr>
<td></td>
<td>If you set an alert condition for the Apply program, this column specifies the name of the subscription set that is being monitored. If you do not specify a set name, then monitoring is done at the Apply-qualifier level, meaning that every set within the given Apply qualifier is monitored.</td>
</tr>
<tr>
<td></td>
<td>If you set an alert condition for the Q Apply receive queue depth or spill queue depth, this column specifies the name of the receive queue or spill queue that is being monitored.</td>
</tr>
<tr>
<td>CONDITION_NAME</td>
<td><strong>Data type:</strong> CHAR(18); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The condition code that was tested when the alert was triggered.</td>
</tr>
</tbody>
</table>
Table 95. Columns in the IBMSNAP_ALERTS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCCURRED_TIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The time that the alert condition occurred at the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>ALERT_COUNTER</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The number of times that this alert has been previously detected in consecutive monitor cycles.</td>
</tr>
<tr>
<td>ALERT_CODE</td>
<td><strong>Data type:</strong> CHAR(10); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The message code that was issued when the alert occurred.</td>
</tr>
<tr>
<td>RETURN_CODE</td>
<td><strong>Data type:</strong> INT; <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The integer value returned by a user condition.</td>
</tr>
<tr>
<td>NOTIFICATION_SENT</td>
<td><strong>Data type:</strong> CHAR(1); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>A flag that indicates whether a notification message was sent:</td>
</tr>
<tr>
<td></td>
<td>Y A notification message was sent.</td>
</tr>
<tr>
<td></td>
<td>E A notification was not sent because the email_server parameter was not specified.</td>
</tr>
<tr>
<td></td>
<td>N A notification was not sent because the number of notifications already reached the limit set by the max_notifications_per_alert parameter.</td>
</tr>
<tr>
<td>ALERT_MESSAGE</td>
<td><strong>Data type:</strong> VARCHAR(1024); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The text of the message that was sent, including the message code.</td>
</tr>
</tbody>
</table>

**IBMSNAP_CONDITIONS table**

The IBMSNAP_CONDITIONS table contains the alert conditions for which the Replication Alert Monitor will contact someone, and it contains the group or individual’s name to contact if a particular condition occurs. The Replication Alert Monitor can monitor a combination of conditions on Capture control servers, Apply control servers, Q Capture servers, and Q Apply servers.

**Server:** Monitor control server

**Non-unique index:** MONITOR_QUAL, COMPONENT, SERVER_NAME, SCHEMA_OR_QUAL, SET_NAME, CONDITION_NAME

Table 96 provides a brief description of the columns in the IBMSNAP_CONDITIONS table.

Table 96. Columns in the IBMSNAP_CONDITIONS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER_NAME</td>
<td><strong>Data type:</strong> CHAR(18); <strong>Nullable:</strong> No.</td>
</tr>
<tr>
<td></td>
<td>The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server where this condition is being monitored.</td>
</tr>
</tbody>
</table>
Table 96. Columns in the IBMSNAP_CONDITIONS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT</td>
<td>The replication component that is being monitored:</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>SCHEMA_OR_QUAL</td>
<td>The Capture schema, Apply qualifier, Q Capture schema, or Q Apply schema that is being monitored.</td>
</tr>
<tr>
<td>SET_NAME</td>
<td>If you set an alert condition for the Apply program, this column specifies the name of the subscription set that is being monitored. If you do not specify a set name, then monitoring is done at the Apply-qualifier level, meaning that every set within the given Apply qualifier is monitored.</td>
</tr>
<tr>
<td>MONITOR_QUAL</td>
<td>The Monitor qualifier that identifies which Replication Alert Monitor program is monitoring the Capture control server, Apply control server, Q Capture server, or Q Apply server for this condition.</td>
</tr>
<tr>
<td>SERVER_ALIAS</td>
<td>The DB2 alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server where this condition is being monitored.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>A flag that indicates whether the Replication Alert Monitor will process this condition during the next monitoring cycle:</td>
</tr>
<tr>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>
### Table 96. Columns in the `IBMSNAP_CONDITIONS` table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **CONDITION_NAME** | **Data type:** CHAR(18); **Nullable:** No.  
The name of the condition that the Replication Alert Monitor is monitoring at the given Capture control server, Apply control server, Q Capture server, or Q Apply server. Conditions for the Capture program begin with CAPTURE. Conditions for the Apply program begin with APPLY. Conditions for the Q Capture program begin with QCAPTURE. Conditions for the Q Apply program begin with QAPPLY.  
**CAPTURE_STATUS**  
The status of the Capture program.  
**CAPTURE_ERRORS**  
Whether the Capture program posted any error messages.  
**CAPTURE_WARNINGS**  
Whether the Capture program posted any warning messages.  
**CAPTURE_LASTCOMMIT**  
The last time the Capture program committed data during the last monitor cycle.  
**CAPTURE_CLATENCY**  
The Capture program’s current latency.  
**CAPTURE_HLATENCY**  
Whether the Capture program’s latency is greater than a certain number of seconds.  
**CAPTURE_MEMORY**  
The amount of memory (in megabytes) that the Capture program is using. |
<p>| <strong>APPLY_STATUS</strong> | The status of the Apply program. |
| <strong>APPLY_SUBSFAILING</strong> | Whether any subscription sets failed. |
| <strong>APPLY_SUBSINACT</strong> | Whether any subscription sets either failed or are inactive. |
| <strong>APPLY_ERRORS</strong> | Whether the Apply program posts any error messages. |
| <strong>APPLY_WARNINGS</strong> | Whether the Apply program posts any warning messages. |
| <strong>APPLY_FULLREFRESH</strong> | Whether a full refresh occurred. |
| <strong>APPLY_REJTRANS (update anywhere)</strong> | Whether the Apply program rejects transactions in any subscription set. |
| <strong>APPLY_SUBSDELAY</strong> | Whether the Apply program delays more than the time that you specified in the PARM_INT parameter. |
| <strong>APPLY_REWORKED</strong> | Whether the Apply program reworked any rows at the target table. |
| <strong>APPLY_LATENCY</strong> | Whether the end-to-end latency of the Apply program exceeds a threshold. |</p>
<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCAPTURE_STATUS</td>
<td>Whether the Q Capture program is down.</td>
</tr>
<tr>
<td>QCAPTURE_ERRORS</td>
<td>Whether the Q Capture program posted any error messages.</td>
</tr>
<tr>
<td>QCAPTURE_WARNINGS</td>
<td>Whether the Q Capture program posted any warning messages.</td>
</tr>
<tr>
<td>QCAPTURE_LATENCY</td>
<td>Whether the Q Capture latency (the difference between the last insert into the IBMQREP_CAPMON table and the timestamp of the last transaction that the Q Capture program read in the DB2 log) exceeds a threshold.</td>
</tr>
<tr>
<td>QCAPTURE_MEMORY</td>
<td>Whether the memory amount that the Q Capture program used exceeds a threshold.</td>
</tr>
<tr>
<td>QCAPTURE_TRANSIZE</td>
<td>Whether a transaction exceeds the MAX_TRANS_SIZE (maximum transaction size) that is set in the IBMQREP_CAPMON table.</td>
</tr>
<tr>
<td>QCAPTURE_SUBSINACT</td>
<td>Whether a Q subscription changed to I (inactive) state.</td>
</tr>
<tr>
<td>QAPPLY_STATUS</td>
<td>Whether the Q Apply program is down.</td>
</tr>
<tr>
<td>QAPPLY_ERRORS</td>
<td>Whether the Q Apply program posted any error messages.</td>
</tr>
<tr>
<td>QAPPLY_WARNINGS</td>
<td>Whether the Q Apply program posted any warning messages.</td>
</tr>
<tr>
<td>QAPPLY_LATENCY</td>
<td>Whether the queue latency (the time it takes for a message to go from the send queue to the receive queue) exceeds a threshold.</td>
</tr>
<tr>
<td>QAPPLY_EELATENCY</td>
<td>Whether the end-to-end latency (the time it takes for a transaction to replicate from the source to the target) exceeds a threshold.</td>
</tr>
<tr>
<td>QAPPLY_EXCEPTIONS</td>
<td>Whether the Q Apply inserted a row in the IBMQREP_EXCEPTIONS table because of a SQL error or conflict.</td>
</tr>
<tr>
<td>QAPPLY_MEMORY</td>
<td>Whether the amount of memory that the Q Apply program used to read messages from a particular receive queue exceeds a threshold.</td>
</tr>
<tr>
<td>QAPPLY_RECVQINACT</td>
<td>Whether a receive queue changed to I (inactive) state.</td>
</tr>
<tr>
<td>QAPPLY_SPILLQDEPTH</td>
<td>Whether the number of messages on a spill queue exceeds a threshold.</td>
</tr>
<tr>
<td>QAPPLY_QDEPTH</td>
<td>Whether the number of messages on a receive queue exceeds a threshold.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PARM_INT</td>
<td>Data type: INT; Nullable: Yes. The integer parameter for the condition. The value of this column depends on the value of the CONDITION_NAME column.</td>
</tr>
<tr>
<td>CAPTURE_LASTCOMMIT</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>CAPTURE_CLATENCY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>CAPTURE_HLATENCY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>CAPTURE_MEMORY</td>
<td>Threshold in megabytes.</td>
</tr>
<tr>
<td>APPLY_SUBSDELAY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>APPLY_REWORKED</td>
<td>Threshold in rows reworked.</td>
</tr>
<tr>
<td>APPLY_LATENCY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>QCAPTURE_LATENCY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>QCAPTURE_MEMORY</td>
<td>Threshold in megabytes</td>
</tr>
<tr>
<td>QCAPTURE_TRANSIZE</td>
<td>Threshold in megabytes</td>
</tr>
<tr>
<td>QAPPLY_EELATEENCY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>QAPPLY_LATENCY</td>
<td>Threshold in seconds.</td>
</tr>
<tr>
<td>QAPPLY_MEMORY</td>
<td>Threshold in megabytes</td>
</tr>
<tr>
<td>QAPPLY_SPILLQDEPTH</td>
<td>Threshold in number of messages.</td>
</tr>
<tr>
<td>QAPPLY_QDEPTH</td>
<td>Threshold in number of messages.</td>
</tr>
</tbody>
</table>
Table 96. Columns in the IBMSNAP_CONDITIONS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| PARM_CHAR   | **Data type:** VARCHAR(128); **Nullable:** Yes.  
The character parameter for the condition. This column holds additional strings used by the condition.  
The CAPTURE_STATUS and APPLY_STATUS conditions use the value of this column. The value of this column is a string concatenating three parameters separated by commas:  
- Capture server or Apply control server.  
- **z/OS** This is the DB2 subsystem name.  
- Remote DB2 instance name (only when the server is remote).  
- Remote hostname.  
If the value is NULL or a zero length string, the Monitor program uses the following defaults:  
- The CURRENT SERVER value from the Capture or Apply control server.  
- The remote DB2 instance name value:  
  - **Linux UNIX** This value is the name of the user ID that was used when the UNIX server was connected.  
  - **Windows** This value is DB.  
- The value of the hostname in the DB2 node directory. |
| CONTACT_TYPE| **Data type:** CHAR(1); **Nullable:** No.  
A flag that indicates whether to contact an individual or a group if this condition occurs:  
C Individual contact  
G Group of contacts  
**z/OS** Z  
Send alert to z/OS console. |
| CONTACT     | **Data type:** VARCHAR(127); **Nullable:** No.  
The name of the individual contact or group of contacts to be notified if this condition occurs. For alerts that are sent to the z/OS console, the value of this column is CONSOLE. |

**IBMSNAP_CONTACTGRP table**

The IBMSNAP_CONTACTGRP table contains the individual contacts that make up contact groups. You can specify for the Replication Alert Monitor to contact these groups of individuals if an alert condition occurs. An individual can belong to multiple contact groups (the columns are not unique).

**Server:** Monitor control server

**Non-unique index:** GROUP_NAME, CONTACT_NAME

[Table 97 on page 464 provides a brief description of the columns in the IBMSNAP_CONTACTGRP table.]

Chapter 24. Control tables for Q replication and event publishing 463
### IBMSNAP_CONTACTS table

The IBMSNAP_CONTACTS table contains the necessary information for the Replication Alert Monitor to use to notify individuals when an alert condition that is associated with the individuals (or their group) occurs. One individual per row is specified.

**Server**: Monitor control server

**Non-unique index**: CONTACT_NAME

Table 98 provides a brief description of the columns in the IBMSNAP_CONTACTS table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTACT_NAME</td>
<td><strong>Data type</strong>: VARCHAR(127); <strong>Nullable</strong>: No. The name of the contact. Only an individual contact is allowed. Group names are not supported. For alerts that are sent to the z/OS console, the value of this column is CONSOLE.</td>
</tr>
<tr>
<td>EMAIL_ADDRESS</td>
<td><strong>Data type</strong>: VARCHAR(128); <strong>Nullable</strong>: No. The main e-mail or pager address for this contact. For alerts that are sent to the z/OS console, this column is ignored.</td>
</tr>
</tbody>
</table>
| ADDRESS_TYPE    | **Data type**: CHAR(1); **Nullable**: Yes. A flag that indicates whether the e-mail address for this contact is an e-mail account or a pager address:  
E The e-mail address is for an e-mail account.  
P The e-mail address is for a pager.  
Z Alerts are sent to the z/OS console.     |
| DELEGATE        | **Data type**: VARCHAR(127); **Nullable**: Yes. The contact name to receive the notifications in a delegation period. Only an individual contact name is allowed. Group names are not supported.                           |
| DELEGATE_START  | **Data type**: DATE; **Nullable**: Yes. The start date of a delegation period when notifications will be sent to the individual named in the DELEGATE column.                                             |
| DELEGATE_END    | **Data type**: DATE; **Nullable**: Yes. The end date of a delegation period.                                                                                                                                  |
Table 98. Columns in the IBMSNAP_CONTACTS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td><strong>Data type</strong>: VARCHAR(1024); <strong>Nullable</strong>: Yes.</td>
</tr>
<tr>
<td></td>
<td>A description of the contact.</td>
</tr>
</tbody>
</table>

**IBMSNAP_GROUPS table**

The IBMSNAP_GROUPS table contains the name and description of each contact group. One group per row is specified.

**Server**: Monitor control server

**Non-unique index**: GROUP_NAME

Table 99 provides a brief description of the columns in the IBMSNAP_GROUPS table.

Table 99. Columns in the IBMSNAP_GROUPS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_NAME</td>
<td><strong>Data type</strong>: VARCHAR(127); <strong>Nullable</strong>: Yes.</td>
</tr>
<tr>
<td></td>
<td>The name of the contact group.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td><strong>Data type</strong>: VARCHAR(1024); <strong>Nullable</strong>: Yes.</td>
</tr>
<tr>
<td></td>
<td>A description of the contact group.</td>
</tr>
</tbody>
</table>

**IBMSNAP_MONENQ table**

The IBMSNAP_MONENQ table is reserved for future use by replication.

**Server**: Monitor control server

**Non-unique index**: MONITOR_QUAL

Table 100 provides a brief description of the column in the IBMSNAP_MONENQ table.

Table 100. Columns in the IBMSNAP_MONENQ table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_QUAL</td>
<td><strong>Data type</strong>: CHAR(18); <strong>Nullable</strong>: No.</td>
</tr>
<tr>
<td></td>
<td>Reserved for future use by replication.</td>
</tr>
</tbody>
</table>

**IBMSNAP_MONPARMS table**

The IBMSNAP_MONPARMS table contains parameters that you can modify to control the operations of the Replication Alert Monitor.

You can define these parameters to set values such as the number of notification messages that the Monitor program will send when an alert condition is met. If you make changes to the parameters in this table, the Monitor program reads your modifications only during startup.

**Server**: Monitor control server
Index: MONITOR_QUAL

Default schema: ASN

This table contains information that you can update by using SQL.

Table 101 provides a brief description of the columns in the IBMSNAP_MONPARMS table.

Table 101. Columns in the IBMSNAP_MONPARMS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_QUAL</td>
<td>Data type: CHAR(18); Nullable: No. The Monitor qualifier matches the parameters to the Replication Alert Monitor program to which these parameters apply.</td>
</tr>
<tr>
<td>ALERT_PRUNE_LIMIT</td>
<td>Data type: INT; Nullable: No, with default; Default: 10080 minutes (7 days). A flag that indicates how old the data is before it will be pruned from the table.</td>
</tr>
<tr>
<td>AUTOPRUNE</td>
<td>Data type: CHAR(1); Nullable: No, with default; Default: Y. A flag that indicates whether the Monitor program automatically prunes rows that are no longer needed from the IBMSNAP_ALERTS, IBMSNAP_MONTRACE, and IBMSNAP_MONTRAIL control tables: Y Autoprune is on. N Autoprune is off.</td>
</tr>
<tr>
<td>EMAIL_SERVER</td>
<td>Data type: INT(128); Nullable: Yes. The address of the e-mail server that is using the SMTP protocol.</td>
</tr>
<tr>
<td>LOGREUSE</td>
<td>Data type: CHAR(1); Nullable: No, with default; Default: N. A flag that indicates whether the Monitor program overwrites the Monitor log file or appends to it. Y The Monitor program reuses the log file by first deleting it and then recreating it when the Monitor program is restarted. N The Monitor program appends new information to the Monitor log file.</td>
</tr>
<tr>
<td>LOGSTDOUT</td>
<td>Data type: CHAR(1); Nullable: No, with default; Default: N. A flag that indicates where the Monitor program directs the log file messages: Y The Monitor program directs log file messages to both the standard out (STDOUT) and the log file. N The Monitor program directs most log file messages to the log file only. Initialization messages go to both the standard out (STDOUT) and the log file.</td>
</tr>
<tr>
<td>NOTIF_PER_ALERT</td>
<td>Data type: INT; Nullable: No, with default; Default: 3. The number of notification messages that will be sent when an alert condition is met.</td>
</tr>
<tr>
<td>NOTIF_MINUTES</td>
<td>Data type: INT; Nullable: No, with default; Default: 60. The number of minutes that you will receive notification messages when an alert condition is met.</td>
</tr>
</tbody>
</table>
Table 101. Columns in the IBMSNAP_MONPARMS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_ERRORS</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> Yes. Specifies the e-mail address where notification messages are sent whenever an error occurs that is related to the operation of the Replication Alert Monitor.</td>
</tr>
<tr>
<td>MONITOR_INTERVAL</td>
<td><strong>Data type:</strong> INT; <strong>Nullable:</strong> No, with default; <strong>Default:</strong> 300 (5 minutes). How often, in seconds, the Replication Alert Monitor runs to monitor the alert conditions that were selected.</td>
</tr>
<tr>
<td>MONITOR_PATH</td>
<td><strong>Data type:</strong> VARCHAR(1040); <strong>Nullable:</strong> Yes. The path where the output from the Monitor program is sent.</td>
</tr>
</tbody>
</table>
| RUNONCE             | **Data type:** CHAR(1); **Nullable:** No, with default; **Default:** N. A flag that indicates whether the Monitor program will check for the alert conditions that were selected:  
  Y  The Monitor program checks for any alert conditions.  
  N  The Monitor program does not check for any alert conditions.  
If RUNONCE is set to y, then the MONITOR_INTERVAL is ignored. |
| TERM                | **Data type:** CHAR(1); **Nullable:** No, with default; **Default:** N. A flag that indicates whether the Monitor program terminates when DB2 is placed in quiesce mode:  
  N  The Monitor program stays active when DB2 is quiesced and waits for DB2 to be unquiesced.  
  Y  The Monitor program terminates when DB2 is quiesced.  
Regardless of the value of TERM, a monitor program stops when DB2 shuts down. When DB2 starts again, you must restart the monitor program. |
| TRACE_LIMIT         | **Data type:** INT; **Nullable:** No, with default; **Default:** 10080. The number of minutes that rows remain in the IBMSNAP_MONTRACE table before they are eligible for pruning. During the pruning process, the rows in the Monitor trace table are pruned if the number of minutes (current timestamp - the time a row was inserted in the IBMSNAP_MONTRACE table) exceeds the value of TRACE_LIMIT. |
| ARCH_LEVEL          | **Data type:** CHAR(8); **Nullable:** No, with default; **Default:** 0905. The level of the monitor control tables. For Version 9.5 the level is 0905. Other values are 0802 and 0901.  
  **Important:** When updating the IBMSNAP_MONPARMS table, do not change the value in this column. |
| DELAY               | **Data type:** CHAR(1); **Nullable:** Yes **Default:** N. A flag that indicates whether the monitor program should delay the start of monitoring by waiting until the specified monitor interval has elapsed, even for the first iteration. |

**IBMSNAP_MONSERVERS table**

The IBMSNAP_MONSERVERS table contains information about the last time that the Replication Alert Monitor monitored a Capture control server, Apply control server, Q Capture server, or Q Apply server.
Server: Monitor control server

Non-unique index: MONITOR_QUAL, SERVER_NAME

Table 102 provides a brief description of the columns in the IBMSNAP_MONSERVERS table.

Table 102. Columns in the IBMSNAP_MONSERVERS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_QUAL</td>
<td>Data type: CHAR(18); Nullable: No.</td>
</tr>
<tr>
<td></td>
<td>The Monitor qualifier that identifies which Replication Alert Monitor was monitoring the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>Data type: CHAR(18); Nullable: No.</td>
</tr>
<tr>
<td></td>
<td>The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.</td>
</tr>
<tr>
<td>SERVER_ALIAS</td>
<td>Data type: CHAR(8); Nullable: Yes.</td>
</tr>
<tr>
<td></td>
<td>The DB2 alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.</td>
</tr>
<tr>
<td>LAST_MONITOR_TIME</td>
<td>Data type: TIMESTAMP; Nullable: Yes.</td>
</tr>
<tr>
<td></td>
<td>The time (at the Capture control server, Apply control server, Q Capture server, or Q Apply server) that the Replication Alert Monitor program last connected to this server. This value is used as a lower bound value to fetch messages from the control tables and is the same value that START_MONITOR_TIME from the last successful monitor cycle.</td>
</tr>
<tr>
<td>START_MONITOR_TIME</td>
<td>Data type: TIMESTAMP; Nullable: Yes.</td>
</tr>
<tr>
<td></td>
<td>The time (at the Capture control server, Apply control server, Q Capture server, or Q Apply server) that the Replication Alert Monitor connected to the Capture control server, Apply control server, Q Capture server, or Q Apply server. This value is used as a upper bound value to fetch alert messages from the control tables.</td>
</tr>
<tr>
<td>END_MONITOR_TIME</td>
<td>Data type: TIMESTAMP; Nullable: Yes.</td>
</tr>
<tr>
<td></td>
<td>The time (at the Capture control server, Apply control server, Q Capture server, or Q Apply server) that the Replication Alert Monitor ended monitoring this server.</td>
</tr>
<tr>
<td>LASTRUN</td>
<td>Data type: TIMESTAMP; Nullable: No.</td>
</tr>
<tr>
<td></td>
<td>The last time (at the Monitor control server) when the Replication Alert Monitor started to process the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>LASTSUCCESS</td>
<td>Data type: TIMESTAMP; Nullable: Yes.</td>
</tr>
<tr>
<td></td>
<td>The value from the LASTRUN column of the last time (at the Monitor control server) when the Replication Alert Monitor successfully completed processing the Capture control server, Apply control server, Q Capture server, or Q Apply server. If the monitoring of this server keeps failing, the value could be the same (the history of this columns is stored in the IBMSNAP_MONTRAIL table).</td>
</tr>
</tbody>
</table>
Table 102. Columns in the IBMSNAP_MONSERVERS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| STATUS      | Data type: SMALLINT; Nullable: No.  
  A flag that indicates the status of the monitoring cycle:  
  -1 The Replication Alert Monitor failed to process this server successfully.  
  0 The Replication Alert Monitor processed this server successfully.  
  1 The Replication Alert Monitor is currently processing this server. |

IBMSNAP_MONTRACE table

The IBMSNAP_MONTRACE table contains audit trail information for the Replication Alert Monitor. Everything that the Monitor program does is recorded in this table, which makes it one of the best places for you to look if a problem with the Monitor program occurs.

Server: Monitor control server

Non-unique index: MONITOR_QUAL, TRACE_TIME

Table 103 provides a brief description of the columns in the IBMSNAP_MONTRACE table.

Table 103. Columns in the IBMSNAP_MONTRACE table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
</table>
| MONITOR_QUAL     | Data type: CHAR(18); Nullable: No.  
  The Monitor qualifier that identifies which Replication Alert Monitor issued the message. |
| TRACE_TIME       | Data type: TIMESTAMP; Nullable: No.  
  The timestamp when the message was inserted into this table. |
| OPERATION        | Data type: CHAR(8); Nullable: No.  
  A value used to classify messages:  
  ERROR       
    An error message  
  WARNING     
    A warning message  
  INFO        
    An informational message |
| DESCRIPTION      | Data type: VARCHAR(1024); Nullable: No.  
  The message code and text. |

IBMSNAP_MONTRAIL table

The IBMSNAP_MONTRAIL table contains information about each monitor cycle. The Replication Alert Monitor inserts one row for each Capture control server, Apply control server, Q Capture server, and Q Apply server that it monitors.

Server: Monitor control server

Non-unique index: None
Table 104 provides a brief description of the columns in the IBMSNAP_MONTRAIL table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_QUAL</td>
<td>CHAR(18)</td>
<td>No</td>
<td>The Monitor qualifier that identifies which Replication Alert Monitor was monitoring the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>CHAR(18)</td>
<td>No</td>
<td>The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.</td>
</tr>
<tr>
<td>SERVER_ALIAS</td>
<td>CHAR(8)</td>
<td>Yes</td>
<td>The DB2 alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.</td>
</tr>
<tr>
<td>STATUS</td>
<td>SMALLINT</td>
<td>No</td>
<td>A flag that indicates the status of the monitoring cycle:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 The Replication Alert Monitor failed to process this server successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0  The Replication Alert Monitor processed this server successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1  The Replication Alert Monitor is currently processing this server.</td>
</tr>
<tr>
<td>LASTRUN</td>
<td>TIMESTAMP</td>
<td>No</td>
<td>The time (at the Monitor control server) when the Replication Alert Monitor program last started to process the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>LASTSUCCESS</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>The last time (at the Monitor control server) when the Replication Alert Monitor successfully completed processing the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>ENDTIME</td>
<td>TIMESTAMP</td>
<td>No, with default</td>
<td>The time when this row was inserted into this table. Default: Current timestamp.</td>
</tr>
<tr>
<td>LAST_MONITOR_TIME</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>The time (at the Capture control server, Apply control server, Q Capture server, or Q Apply server) when the Replication Alert Monitor last connected to the Capture control server, Apply control server, Q Capture server, or Q Apply server. This value is used as a lower bound value to fetch messages from the control tables and is the same value as START_MONITOR_TIME from the previous successful monitor cycle.</td>
</tr>
<tr>
<td>START_MONITOR_TIME</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>The last time when the Replication Alert Monitor started to monitor the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
<tr>
<td>END_MONITOR_TIME</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>The last time when the Replication Alert Monitor finished monitoring the Capture control server, Apply control server, Q Capture server, or Q Apply server.</td>
</tr>
</tbody>
</table>
### Table 104. Columns in the IBMSNAP_MONTRAIL table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLCODE</td>
<td>Data type: INT; Nullable: Yes. The SQLCODE of any errors that occurred during this monitor cycle.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Data type: CHAR(5); Nullable: Yes. The SQLSTATE of any errors that occurred during this monitor cycle.</td>
</tr>
<tr>
<td>NUM_ALERTS</td>
<td>Data type: INT; Nullable: No. The number of alert conditions that occurred during this monitor cycle.</td>
</tr>
<tr>
<td>NUM_NOTIFICATIONS</td>
<td>Data type: INT; Nullable: No. The number of notifications that were sent during this monitor cycle.</td>
</tr>
<tr>
<td>SUSPENSION_NAME</td>
<td>Data type: VARCHAR(128); Nullable: Yes. The name of the suspension that is used to stop the operation of the monitor for defined periods.</td>
</tr>
</tbody>
</table>

#### IBMSNAP_SUSPENDS table

The IBMSNAP_SUSPENDS table stores information about temporary suspensions of the monitor program.

- **Server:** Monitor control server
- **Default schema:** ASN
- **Primary key:** SUSPENSION_NAME
- **Unique index:** SERVER_NAME, TEMPLATE_NAME, START

Table 105 provides a brief description of the columns in the IBMSNAP_SUSPENDS table.

### Table 105. Columns in the IBMSNAP_SUSPENDS table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSPENSION_NAME</td>
<td>Data type: VARCHAR(128); Nullable: No. The name of the monitor suspension.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>Data type: CHAR(18); Nullable: No. The name of the Q Capture server, Q Apply server, Capture control server, or Apply control server where monitoring is suspended.</td>
</tr>
<tr>
<td>SERVER_ALIAS</td>
<td>Data type: CHAR(18); Nullable: Yes. The alias of the server where monitoring is suspended.</td>
</tr>
<tr>
<td>TEMPLATE_NAME</td>
<td>Data type: VARCHAR(128); Nullable: Yes. The name of the monitor suspension template. If the value of this column does not exist in the IBMSNAP_TEMPLATES control table, the monitor suspends one time from the START timestamp until the STOP timestamp.</td>
</tr>
</tbody>
</table>
Table 105. Columns in the IBMSNAP_SUSPENDS table (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No &lt;br&gt;The time to start using the template. If no template is specified, this is the start time of the suspension.</td>
</tr>
<tr>
<td>STOP</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No &lt;br&gt;The time to stop using the template. If no template is specified, this is the end time of the suspension.</td>
</tr>
</tbody>
</table>

**IBMSNAP_TEMPLATES table**

The IBMSNAP_TEMPLATES table stores information about how often and how long the monitor program is suspended. This information is called a monitor suspension template.

**Server:** Monitor control server

**Default schema:** ASN

**Unique index:** TEMPLATE_NAME

Table 106 provides a brief description of the columns in the IBMSNAP_TEMPLATES table.

Table 106. Columns in the IBMSNAP_TEMPLATES table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPLATE_NAME</td>
<td><strong>Data type:</strong> VARCHAR(128); <strong>Nullable:</strong> No &lt;br&gt;The name of the monitor suspension template.</td>
</tr>
<tr>
<td>START_TIME</td>
<td><strong>Data type:</strong> TIME; <strong>Nullable:</strong> No &lt;br&gt;The time of the day to start the suspension. Default: 00:00:00</td>
</tr>
<tr>
<td>WDAY</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> Yes &lt;br&gt;The day of the week in which the suspension begins, starting with 0 for Sunday and continuing to 6 for Saturday. A null values means the suspension can begin on any day.</td>
</tr>
<tr>
<td>DURATION</td>
<td><strong>Data type:</strong> INTEGER; <strong>Nullable:</strong> No &lt;br&gt;The duration of the suspension in minutes.</td>
</tr>
</tbody>
</table>

**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 107 provides a brief description of the extra columns in user tables that are required for peer-to-peer replication.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMQREPVERNODE</td>
<td><strong>Data type:</strong> SMALLINT; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A number that identifies the database or subsystem that contains the table within a peer-to-peer group. Default: 0</td>
</tr>
<tr>
<td>IBMQREPVERTIME</td>
<td><strong>Data type:</strong> TIMESTAMP; <strong>Nullable:</strong> No, with default</td>
</tr>
<tr>
<td></td>
<td>A timestamp that records when a change occurs in the table. Default: 0001-01-01-00.00.00</td>
</tr>
</tbody>
</table>
Chapter 25. 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 108 describes the three types of messages.

<table>
<thead>
<tr>
<th>Type of message</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.
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 109 provides a quick reference to
the types of data messages.

Informational messages
Report on the status of a Q Capture program or publication. Table 110 on
page 477 provides a quick reference to the types of informational messages.

Table 109. Data messages from the Q Capture program to a user application

<table>
<thead>
<tr>
<th>Message type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 109. Data messages from the Q Capture program to a user application (continued)

<table>
<thead>
<tr>
<th>Message type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row operation</td>
<td>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.</td>
</tr>
<tr>
<td>Large object (LOB)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 110. Informational messages from the Q Capture program to a user application

<table>
<thead>
<tr>
<th>Message type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscription deactivated</td>
<td>Tells the user application that the Q Capture program deactivated a publication.</td>
</tr>
<tr>
<td>Load done received</td>
<td>Acknowledges that the Q Capture program received the message that the target table is loaded.</td>
</tr>
<tr>
<td>Error report</td>
<td>Tells the user application that the Q Capture program encountered a publication error.</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>Tells the user application that the Q Capture program is still running when it has no data messages to send.</td>
</tr>
<tr>
<td>Subscription schema</td>
<td>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.</td>
</tr>
<tr>
<td>Add column</td>
<td>Contains information about a column that was added to an existing publication.</td>
</tr>
</tbody>
</table>

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 111 describes the msg element.

Table 111. Element description for the msg element (Q Capture program to a user application)

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

Structure

```xml
<?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
<?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 480
- "Column element (col)" on page 482
- "Elements for a single-column value" on page 483
- "Elements for a double-column value" on page 485
- "Before-value and after-value elements (beforeVal and afterVal)" on page 486

## 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 112 describes the trans element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

## Structure

```xml
<trans isLast="is_last_indicator" segmentNum="segment_number"
   cmiLSN="commit_logical_sequence_number" cmiTime="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
The correlation ID (normally a job name) that ran the source update. XML data type: string.

plan_name
The plan name that is associated with the transaction. XML data type: string.

Example
The following example shows a transaction element that contains one or more of the insert row, update row, or delete row elements.

```xml
<?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 113 describes the insertRow, deleteRow, and updateRow elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>insertRow</td>
<td>Not empty, complex type, complex content</td>
</tr>
<tr>
<td>deleteRow</td>
<td>Not empty, complex type, complex content</td>
</tr>
<tr>
<td>updateRow</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

Structure

```xml
<insertRow subName="publication_name" srcOwner="source_owner"
           srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">
  elements
</insertRow>
```
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
<msg 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="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>
    </trans>
</msg>
```
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 114 describes the col element.

Table 114. Element description for col

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

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.
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
<?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 485.

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 115 on page 484 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 115. Element descriptions for single column value

<table>
<thead>
<tr>
<th>Name</th>
<th>XML data type</th>
<th>Value’s data format</th>
</tr>
</thead>
<tbody>
<tr>
<td>smallint</td>
<td>short</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>bigint</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>float (32 bits)</td>
<td>[-d.dddE[-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[-d.dddE[-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>double (64 bits)</td>
</tr>
<tr>
<td>real</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>decimal</td>
<td>decimal</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>date</td>
<td>YYYY-MM-DD</td>
</tr>
<tr>
<td>time</td>
<td>time</td>
<td>HH:MM:SS.SSS</td>
</tr>
<tr>
<td>timestamp</td>
<td>dateTime</td>
<td>YYYY-MM-DDTHH:MM:SS.SSS</td>
</tr>
<tr>
<td>char</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>varchar</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>long varchar</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>bitchar</td>
<td>hexBinary</td>
<td></td>
</tr>
<tr>
<td>bitvarchar</td>
<td>hexBinary</td>
<td></td>
</tr>
<tr>
<td>bitlongvarchar</td>
<td>hexBinary</td>
<td></td>
</tr>
<tr>
<td>graphic</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>vargraphic</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>longvargraphic</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>rowid</td>
<td>hexBinary</td>
<td></td>
</tr>
<tr>
<td>blob</td>
<td>hexBinary</td>
<td></td>
</tr>
<tr>
<td>clob</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dbcllob</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**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 dbcllob 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
<?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 483.

#### Structure

```xml
<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
<?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="0000:0000:06d6:87ab"
        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 116 describes the beforeVal and afterVal elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeVal</td>
<td>Nullable, complex type, simple content</td>
</tr>
<tr>
<td>afterVal</td>
<td>Nullable, complex type, simple content, optional</td>
</tr>
</tbody>
</table>

### Structure

```xml
<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
<?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 117 describes the rowOp element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowOp</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

**Structure**

```xml
<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.
Where `insertRow, updateRow, or deleteRow` represents the elements that are explained in [“Transaction message” on page 478](#).

**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 118 describes the lob element.

**Table 118. Element description for lob**

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

**Structure**

```xml
<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
<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 119 describes the LOB column value elements.

**Table 119. Element description for LOB column values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob</td>
<td>Nullable, complex type, simple content</td>
</tr>
<tr>
<td>clob</td>
<td>Nullable, complex type, simple content</td>
</tr>
<tr>
<td>dblob</td>
<td>Nullable, complex type, simple content</td>
</tr>
</tbody>
</table>

**Structure**

```xml
<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
<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>
</xml>
```

**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 120 describes the subDeactivated element.

**Table 120. Element description for subDeactivated**

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>subDeactivated</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>
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 121 describes the loadDoneRcvd element.

Table 121. Element description for loadDoneRcvd

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadDoneRcvd</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

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
<?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 122 describes the errorRpt element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorRpt</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

Structure

```xml
<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.
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
<?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 123 describes the heartbeat element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>heartbeat</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

**Structure**

```xml
<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
<?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">
```

494   Replication and Event Publishing Guide and Reference
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 497

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 124 describes the subSchema element.

Table 124. Element description for subSchema

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>subSchema</td>
<td>Not empty, complex type, simple content</td>
</tr>
</tbody>
</table>

Structure

```xml
<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 125 provides additional details about two XML data types used in attributes for the subSchema element.

Table 125. Additional data type descriptions for subSchema attributes

<table>
<thead>
<tr>
<th>Type name</th>
<th>Base Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadPhaseEnumType</td>
<td>string</td>
<td>none, external</td>
</tr>
<tr>
<td>dbServerTypeEnumType</td>
<td>string</td>
<td>QDB2, QDB2/6000, QDB2/HPUX, QDB2/NT, QDB2/SUN, QDB2/LINUX, QDB2/Windows</td>
</tr>
</tbody>
</table>

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
<?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>
</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 126 describes the col element within a schema message.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

**Structure**

```xml
<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 127 on page 498 XML data type: `dataTypeEnumType`. 

Chapter 25. Structure of XML messages for event publishing 497
Table 127. Additional data type description for dataTypeEnumType

<table>
<thead>
<tr>
<th>Type name</th>
<th>Base type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataTypeEnumType</td>
<td>string</td>
<td>smallint, integer, bigint, float, real, double, decimal,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>char, varchar, longvarchar, bitchar, bitvarchar,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bitlongvarchar, graphic, vargraphic, longvargraphic, time, timestamp,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>date, rowid, blob, clob, dbclob</td>
</tr>
</tbody>
</table>

**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
<?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="Q082/6000"
            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 128 describes the addColumn element.

Table 128. Element description for addColumn

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>addColumn</td>
<td>Not empty, complex type, simple content</td>
</tr>
</tbody>
</table>

**Structure**

```xml
<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
<?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>
</addColumn>
```

Where **column_element** represents the column element that is explained in “Column element (col) in a subscription schema message” on page 497.

**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 129 describes the four types of control messages from a user application to a Q Capture program.

<table>
<thead>
<tr>
<th>Message type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidate send queue</td>
<td>Requests that a Q Capture program invalidate a send queue by performing the queue error action that you specified.</td>
</tr>
<tr>
<td>Load done</td>
<td>Tells a Q Capture program that the target table for a publication is loaded.</td>
</tr>
<tr>
<td>Activate subscription</td>
<td>Requests that a Q Capture program activate a publication.</td>
</tr>
<tr>
<td>Deactivate subscription</td>
<td>Requests that a Q Capture program deactivate a publication.</td>
</tr>
</tbody>
</table>

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 130 describes the msg element.

Table 130. Element description for msg (user application to a Q Capture program)

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>Not empty, complex type, complex content</td>
</tr>
</tbody>
</table>

Structure

```xml
<?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 msgsub.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.
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
<?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 131] describes the invalidateSendQ element.

**Table 131. Element description for invalidateSendQ**

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalidateSendQ</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

#### Structure

```xml
<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
<?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 132 describes the loadDone element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadDone</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

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
<?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 133 describes the activateSub element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>activateSub</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

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.
Deactivate subscription message

A deactivate subscription message tells the Q Capture program to stop capturing changes for a publication.

Table 134 describes the deactivateSub element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>deactivateSub</td>
<td>Empty, complex type</td>
</tr>
</tbody>
</table>

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>
Chapter 26. Samples

Use these samples to help set up and operate a Q replication or event publishing environment.

Sample programs (Linux, UNIX, Windows)

The sample programs for the Linux, UNIX, and Windows operating systems are located in the sql1ib/samples/repl directory.

Samples to operate Q replication programs (Linux, UNIX, Windows)

Table 135 describes sample programs that can help you start the Q Capture program, the Q Apply program, and the Replication Alert Monitor.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qcapture_api.c</td>
<td>This sample program contains code to start the Q Capture program.</td>
</tr>
<tr>
<td>qapply_api.c</td>
<td>This sample program contains code to start the Q Apply program.</td>
</tr>
<tr>
<td>monitor_api.c</td>
<td>This sample program contains code to start the Replication Alert Monitor.</td>
</tr>
</tbody>
</table>

Table 136 describes the makefiles that you need to build, compile, and link the programs that are listed in Table 135.

<table>
<thead>
<tr>
<th>Makefile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qcapture_api_nt.mak</td>
<td>This makefile builds the code for the qcapture_api.c sample program on Windows operating systems.</td>
</tr>
<tr>
<td>qcapture_api_unix.mak</td>
<td>This makefile builds the code for the qcapture_api.c sample program on Linux and UNIX operating systems.</td>
</tr>
<tr>
<td>qapply_api_nt.mak</td>
<td>This makefile builds the code for the qapply_api.c sample program on Windows operating systems.</td>
</tr>
<tr>
<td>qapply_api_unix.mak</td>
<td>This makefile builds the code for the qapply_api.c sample program on Linux and UNIX operating systems.</td>
</tr>
</tbody>
</table>

Samples to set up Q replication and event publishing (Linux, UNIX, Windows)

Table 137 on page 506 describes sample programs that can help you set up your Q replication and event publishing environment by providing examples of:
• A program to set up Q replication objects.
• A program to create control tables for replication.
• A program that uses event publishing in a business environment.

Table 137. Sample programs for the Linux, UNIX, and Windows operating systems.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnqdefq</td>
<td>This sample script shows which WebSphere MQ objects you need and how to create them. The script includes commands to create queue managers and queues for two servers in unidirectional, remote replication. Use this sample script to quickly create and set up the WebSphere MQ objects for this type of Q replication environment.</td>
</tr>
<tr>
<td>asnqctlw.sql</td>
<td>This sample script creates control tables for replication.</td>
</tr>
<tr>
<td>asnqwxml.zip</td>
<td>This file shows an example of a web-based application for event publishing. The sample demonstrates how to use a publication in a business scenario.</td>
</tr>
<tr>
<td>asnqspC.SQC</td>
<td>This sample program is an example of a stored procedure that is written in C.</td>
</tr>
<tr>
<td>asnqspcreate.sql</td>
<td>This sample script can create a stored procedure that is written in SQL.</td>
</tr>
<tr>
<td>asnqspSQL.sql</td>
<td>This sample script is an example of a stored procedure that is written in SQL.</td>
</tr>
</tbody>
</table>

AsnqspC.mak is a makefile that you can use to compile the asnqspC.SQC sample program. It is located in the same directory as the other sample programs.

Sample programs (z/OS)

The SASNSAMP partitioned data set (PDS) contains the sample programs for the z/OS operating system.

The following sections describe sample programs to help you set up and operate your Q replication and event publishing environment for the z/OS operating system:

Samples to set up Q replication and event publishing (z/OS)

The following samples are designed to help you set up a Q replication and event publishing environment on the z/OS platform.

Table 138 on page 507 and Table 139 on page 507 describe the samples.
Table 138. Sample programs for the z/OS operating system.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnqdefq</td>
<td>This sample script shows which WebSphere MQ objects you need and how to create them. The script includes commands to create queue managers and queues for two servers in unidirectional, remote replication. Use this sample script to quickly create and set up the WebSphere MQ objects for this type of Q replication environment.</td>
</tr>
<tr>
<td>asnqcntz.sql</td>
<td>This sample script contains code to create control tables for replication.</td>
</tr>
<tr>
<td>asnqbnds</td>
<td>This sample program contains code to bind the Q Capture program and the Q Apply program at a local server.</td>
</tr>
<tr>
<td>asnqbndl</td>
<td>This sample program contains code to bind the Q Apply program at a remote Windows server.</td>
</tr>
</tbody>
</table>

Table 139. Sample stored procedures for the z/OS operating system.

<table>
<thead>
<tr>
<th>Stored procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnqdspc</td>
<td>A sample C++ stored procedure. See &quot;Sample JCL for Q replication and event publishing (z/OS)&quot; for the asnq001 sample JCL to prepare for this procedure.</td>
</tr>
<tr>
<td>asnqsps</td>
<td>A sample SQL stored procedure. See &quot;Sample JCL for Q replication and event publishing (z/OS)&quot; for the asnq002 sample JCL to prepare for this procedure.</td>
</tr>
</tbody>
</table>

Sample JCL for Q replication and event publishing (z/OS)

The following samples show methods of using JCL to operate the replication and event publishing programs and utilities.

Table 140 describes the JCL samples.

Table 140. Sample JCL to run replication programs.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnstpa</td>
<td>This sample contains JCL to stop a Q Apply program by using the asnacmd command.</td>
</tr>
<tr>
<td>asnstpc</td>
<td>This sample contains JCL to stop a Q Capture program by using the asnccmd command.</td>
</tr>
<tr>
<td>asnstra</td>
<td>This sample contains JCL to start Q Apply program.</td>
</tr>
<tr>
<td>asnstrc</td>
<td>This sample contains JCL to start a Q Capture program.</td>
</tr>
<tr>
<td>asnstrm</td>
<td>This sample contains JCL to start the Replication Alert Monitor.</td>
</tr>
<tr>
<td>asnqtrc</td>
<td>This sample contains JCL to run the asntrc program for the Q Capture program or the Q Apply program.</td>
</tr>
<tr>
<td>asntrdmp</td>
<td>This sample contains JCL to write the buffer that is generated by the asntrc program to a file.</td>
</tr>
<tr>
<td>asntrflw</td>
<td>This sample contains JCL to generate asntrc flow trace report.</td>
</tr>
<tr>
<td>asntrfmt</td>
<td>This sample contains JCL to generate asntrc fmt trace report.</td>
</tr>
<tr>
<td>asnqtoff</td>
<td>This sample contains JCL to stop the asntrc program.</td>
</tr>
<tr>
<td>Program</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>asnqton</td>
<td>This sample contains JCL to start the asntrc program.</td>
</tr>
<tr>
<td>asntdiff</td>
<td>This sample contains JCL to run the asntdiff command.</td>
</tr>
<tr>
<td>asnqj001</td>
<td>This sample contains JCL to precompile, compile, link-edit, and bind the asnqdspe sample of a C++ stored procedure. See “Samples to set up Q replication and event publishing (z/OS)” on page 506 for stored procedure samples.</td>
</tr>
<tr>
<td>asnqj002</td>
<td>This sample contains JCL to precompile, compile, link-edit, and bind the asnqps sample of a SQL stored procedure. See “Samples to set up Q replication and event publishing (z/OS)” on page 506 for stored procedure samples.</td>
</tr>
</tbody>
</table>
Product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product interface, in a suite-wide information center, and in PDF file books.

You can also order IBM publications in hardcopy format online or through your local IBM representative.

To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order.

You can send your comments about documentation in the following ways:
• Online reader comment form: www.ibm.com/software/data/rcf/
• E-mail: comments@us.ibm.com

Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You can also provide feedback on products and documentation.

Customer support

For customer support for IBM products and for product download information, go to the support and downloads site at www.ibm.com/support/us/.

You can open a support request by going to the software support service request site at www.ibm.com/software/support/probsub.html.

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/us/.

Software services

For information about software, IT, and business consulting services, go to the solutions site at www.ibm.com/bussinesssolutions/us/en.

Information Management product support

For Information Management product support, news, and other product information, go to the Information Management support site at www.ibm.com/software/data/support/.

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/

General information

To find general information about IBM, go to www.ibm.com.

Product feedback

You can provide general product feedback through the Consumability Survey at

Documentation feedback

You can click the feedback link in any topic in the information center to comment on the information center.

You can also send your 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.

```
---required_item---optional_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.

```
---required_item---default_choice---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).
Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere™ Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:

- IBM InfoSphere Business Glossary
- IBM InfoSphere Business Glossary Anywhere
- InfoSphere DataStage and QualityStage Administrator
- IBM InfoSphere FastTrack
- IBM InfoSphere Information Analyzer
- IBM InfoSphere Information Services Director
- IBM InfoSphere Metadata Workbench

For more information about IBM product accessibility status, go to http://www.ibm.com/able/product_accessibility/index.html

Accessible documentation

Accessible documentation for IBM Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to set display preferences in your browser. It also allows you to use screen readers and other assistive technologies to access the documentation.
Notices

This information was developed for products and services offered in the U.S.A.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not 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.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

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

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not
been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

**Trademarks**

IBM trademarks and certain non-IBM trademarks are marked on their first occurrence in this information with the appropriate symbol.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACSLink, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACSLink licensee of the United States Postal Service.

Other company, product, or service names may be trademarks or service marks of others.
CLOB (character large object) data types
   column values 489
   large object (LOB) message 489
   memory 6
   message buffer size 6
   replicating 160

code pages
   for event publishing
      description 10
      overview 9
   for Q replication
      description 9
      overview 9
   Q Apply program 9
   Q Capture program
      for event publishing 10
      for Q replication 9
   XML messages for event publishing 10

cold starts
   preventing 173
   startmode parameter 176

column (vertical) subsetting
   event publishing 152
   publications 152
   Q replication 72
   Q subscriptions 72

key columns
   event publishing 156
   publications 156
   Q replication 88
   Q subscriptions 88

mapping
   in Q replication 77
   in Q subscriptions 77

published in publications 156

publishing
   event publishing 152
   publications 152
   replicating
      Q replication 72
      Q subscriptions 72

subsetting
   event publishing 152
   publications 152
   Q replication 72
   Q subscriptions 72

columns
   CREATE PUB command 150
   CREATE PUBQMAP command 148
   CREATE QSUB command 70
   CREATE REPLQMAP command 66

commands
   running or saving (Replication Center) 167
   commit_count parameter, Q Apply 211, 336
   commit_interval parameter, Q Capture 176, 324

compression dictionaries 290
computed columns 78, 82
CONCURRENT_ACCESS_RESOLUTION 137

configuring
   Replication Alert Monitor
      for Linux 272
      for UNIX 272
      for Windows 272

conflict detection
   bidirectional replication 97
   conflict detection (continued)
      peer-to-peer replication 99
      conflict detection, bidirectional
         replication 108

conflicts
   in IBMQREP_EXCEPTIONS
      table 260
   value-based and version-based 260
   consistent-change-data (CCD) tables 85
   contacts
      description 263
      contacts and contact groups for
         monitoring, defining 274

conflicts (continued)
   contacts and contact groups, defining 274
   control messages 475
   control tables 251
      analyzing 287
      dropping 243, 251
      event publishing
         overview 399
      maintaining 291

Monitor control server
   IBMMSNAP_ALERTS 457
   IBMMSNAP_CONDITIONS 458
   IBMMSNAP_CONTACTGRP 463
   IBMMSNAP_CONTACTS 464
   IBMMSNAP_GROUPS 465
   IBMMSNAP_MONENQ 465
   IBMMSNAP_MONNAMES 465
   IBMMSNAP_MONSERVERS 468
   IBMMSNAP_MONTRACE 469
   IBMMSNAP_MONTRAIL 469
   IBMMSNAP_SUSPENDS 471
   IBMMSNAP_TEMPLATES 472
   list 456
   pruning 291
   Q Apply
      dropping 243

Q Apply server
   IBMQREP_APPENQ 429
   IBMQREP_APPENVINFO 430
   IBMQREP_APPLYMON 430
   IBMQREP_APPLYPARMS 434
   IBMQREP_APPLYTRACE 438
   IBMQREP_DELTOMB 439
   IBMQREP_DONEMSG 440
   IBMQREP_EXCEPTIONS 441
   IBMQREP_RECVQUEUES 443
   IBMQREP_SAVRMS 445
   IBMQREP_SPILLEDROW 447
   IBMQREP_SPILLQS 446
   IBMQREP_TARGETS 448
   IBMQREP_TRG_COLS 454
   list 428
   Q Capture
      dropping 243, 251
   Q Capture for Oracle server
      IBMQREP_EOLFLUSH 412

Q Capture server
   IBMQREP_ADMINMSG 400
   IBMQREP_CAPENQ 401
   IBMQREP_CAPENVINFO 401
   IBMQREP_CAPMON 402
   IBMQREP_CAPPARMS 404
   IBMQREP_CAPQMON 408

control tables (continued)
   Q Capture server (continued)
      IBMQREP_CAPTRACE 410
      IBMQREP_COLVERSION 410
      IBMQREP_IGNTRAN 412
      IBMQREP_IGNTRANTRC 413
      IBMQREP_SENDQUEUES 414
      IBMQREP_SIGNAL 418
      IBMQREP_SRC_COLS 421
      IBMQREP_SRCH_COND 422
      IBMQREP_SUBS 423
      IBMQREP_TABVERSION 427
      list 399
   Q replication
      overview 399
   reorganizing 293
   Replication Alert Monitor,
      creating 273
   RUNSTATS utility 292
   control tables, creating 55
   control tables, creating at different
      version 57

converting
   data for event publishing
      description 10
   data for Q replication
      description 9
   CREATE PUB command 150
   CREATE PUBQMAP command 148
   CREATE QSUB command 70
   CREATE REPLQMAP command 66

creating
   publications
      overview 150
   Q subscriptions
      peer-to-peer replication,
         overview 114
      unidirectional replication,
         overview 68
   WebSphere MQ objects for
      replication 17
   creating monitors 275
   CURRENTLY COMMITTED access
      behavior 137
   customer support 509

D

data conversion
   event publishing, description 10
   Q replication, description 9

data exceptions, overview 260

data messages 475

data transformation 78, 82

data types
   BLOB, replicating 160
   CLOB, replicating 160
   considerations
      when publishing 159
      when replicating 159

DBCLOB, replicating 160

LOB, replicating 160

restrictions 159

database
   configuring for z/OS 53

DATE data, replicating 161
older DB2 databases
replication 58
operating
Q Apply program 207, 348
Q Capture program 169, 333
Replication Alert Monitor 357

P
p2p_2nodes parameter, Q Apply 211, 336
packages
binding
overview 50
Q Apply program (Linux, UNIX, Windows) 51
Q Capture program (Linux, UNIX, Windows) 50
Replication Alert Monitor (Linux, UNIX, Windows) 52
parallel loads on z/OS 135
parameters
CAPTURE_PATH, specifying on z/OS 303
Q Apply program
apply_path 211
apply_schema 211
apply_server 211
autostop 211
changing 224
changing while running 348
classic_load_file_sz 211, 336
commit_count 211, 336
deadlock_retries 211
default values 209
load_data_buff_sz 336
load_data_buff_sze 211
loadcopy_path 211, 336
logreuse 211
logstdout 211
max_parallel_loads 211, 336
monitor_interval 211
monitor_limit 211
nickname_commit_ct 211, 336
pwdfile 211
qmgr 211
skiptrans 211, 336
spill_commit_count 211, 336
term 211
trace_limit 211
Q Capture program
add_partition 176
adminq 176
autostop 176
caf 176
capture_path 176
capture_schema 176
capture_server 176
changing 191
changing while running 333
commit_interval 176
default values 174
descriptions 176
logreuse 176
logstdout 211
max_parallel_loads 211, 336
parameters (continued)
Q Capture program (continued)
message_persistence 176
methods of changing 191
monitor_interval 176
monitor_limit 176
overview 174
prune_interval 176
qfull_num_retries 176
qfull_retry_delay 176
qmgr 176
restarted 176
signal_limit 176
sleep_interval 176
startmode 176
term 176
trace_limit 176
Replication Alert Monitor
alert_prune_limit 281
autoprune 281
default values 281
description 281
e-mail_server 281
max_notification_minutes 281
max_notifications_per_alert 281
monitor_errors 281
monitor_limit 281
monitor_path 281
runonce 281
trace_limit 281
WebSphere MQ
parameters, invocation
Q Apply program 336
Q Capture program 324
Replication Alert Monitor
for UNIX 352
for Windows 352
for z/OS 352
partitioned tables, replicating 54
password file 13
creating 360, 377
passwords
setting up, overview 11
passwords, storing 13
peer-to-peer replication 114
adding columns to replicate 236
conflict detection 99
description 99
error options 93
Q subscriptions
overview 114
referential integrity 99
replication objects
three or more servers 99
two servers 99
starting 118, 119, 121
stopping 121
three or more remote servers
WebSphere MQ objects 23
three or more servers
load options 141
objects 99
two remote servers
WebSphere MQ objects 21
peer-to-peer replication (continued)
two servers
load options 141
objects 99
versioning columns 472
performance, Q Capture and Q Apply programs 257
planning
storage requirements, WebSphere MQ 38
printing
Monitor program
messages 281
problem rows 260
product accessibility
accessibility 513
profiles
target object profiles for Q replication 69
prune parameter
Q Capture 333
prune parameter, Q Apply 348
prune threads 255
prune_interval parameter
Q Apply 336
Q Capture 176, 324
prunings
Q Apply program
asqacmd command 348
publications
adding columns to replicate 246
changing properties 245
columns included in message 156
creating
overview 150
deleting 248
filtering rows 153
grouping with publishing queue maps 147
including before values 157
key columns 156
naming rules 317
Q Capture messages 153
reinitializing 245
search conditions 153
starting 198
subsetting source columns 152
WHERE clauses 153
Publications
creating 150
publishing
before values in publications 157
converted data 10
event publishing
columns included in message 156
when Q Capture sends a message 153
grouping publishing queue maps and publications 147
publications
columns included in message 156
when Q Capture sends a message 153
setting up event publishing 147
subset of data
filtering rows in publications 153
publishing (continued)
subset source columns for publications 152
with the Q Capture program columns included in publication 156
when Q Capture sends a message 153
XML publications before values 157
publishing queue maps 148
changing 249
deleting 250
grouping with publications 147
pwdfile parameter
Q Apply 336
Q Capture 324
pwdfile parameter, Q Apply 211
Q Apply control tables
dropping 243
list 428
Q Apply control tables, creating 55
Q Apply latency 258
Q Apply program
agent threads
defined 255
alert conditions 265
asnmcmd command 348
authorization requirements 11
bidirectional replication 97
browser threads
defined 255
checking run-time parameter values 348
checking status 253, 348
code pages 9
cold starts preventing 173
startmode parameter 176
commit_count 211, 336
deleting 250
classic_load_file_sz 211, 336
deadlock_retries 211
default values 209
Q Apply program (continued)
parameters (continued)
description 209, 211
dftmodelq 211, 336
ignbaddrda 211, 336
insert_bidi_signal 211, 336
load_data_buff_size 211
load_data_buff_sz 336
loadcopy_path 211, 336
logreuse 211
logstdout 211
max_parallel_loads 211, 336
monitor_limit 211
nickname_commit_ct 211, 336
p2p_2nodes 211, 336
pray presented 243
Q Apply server
federated configuration 126
IBMREP_APPENQ control table 429
Q Apply server (continued)
IBMREP_APPENVINFO control table 430
IBMREP_APPLYMON control table 430
IBMREP_APPLYPARMS control table 434
IBMREP_APPLYTRACE control table 438
IBMREP_DELTOMB control table 439
IBMREP_DONEMSG control table 440
IBMREP_EXCEPTIONS control table 441
IBMREP_RECVQUEUES control table 443
IBMREP_SAVERI control table 445
IBMREP_SPILLEDROW control table 447
IBMREP_SPILLQS control table 446
IBMREP_TARGETS control table 448
IBMREP_TRG_COLS control table 454
list of control tables 428
Q Apply throughput 257
Q Capture
parameters
Linux, startallq 200
Q Capture control tables
dropping 243, 251
list 399
Q Capture control tables, creating 55
Q Capture for Oracle server
IBMREP_EOLFLUSH control table 412
Q Capture latency 258
Q Capture messages
before values in publications 157
columns included in message 156
when Q Capture sends a message for publications 153
for Q subscriptions 73
Q Capture program 171
alert conditions 265
authorization requirements 11
before values in publications 157
bidirectional replication 97
checking run-time parameter values 333
cold starts preventing 173
startmode parameter 176
columns sent in publications 156
data conversion
for event publishing 10
default parameter values 209
for Q replication 9
finding oldest required log file 289
handling table structure changes 238
memory 1, 7
memory on AIX 46
messages 500
Q Capture program (continued)
MODIFY command 306
operating 169
packages, binding 50
parameters 174
add_partition 176
adming 176
autostop 176
capture_path 176
capture_schema 176
change 176
changing 191
changing dynamically 192
changing saved 194
commit_interval 176
description 176
logreuse 176
logstout 176
memory_limit 176
method of changing 191
monitor_interval 176
prune_interval 176
restartq 176
signal_limit 176
sleep_interval 176
startmode 176
term 176
trace_limit 176
peer-to-peer replication 99
prune command 333
reinitializing 233, 245, 333
reinitializing a send queue 333
running in batch mode 306
scheduling 314, 315
Linux, UNIX 314
Q Capture messages 73
Windows 315
schema
rules for bidirectional replication 97
rules for peer-to-peer replication 99
sending commands to a running 333
started task 309
starting 169
asqcap command 324
starting with JCL 305
starting, peer-to-peer (three servers) 119, 121
starting, peer-to-peer (two servers) 118
stopping 185, 333
stopping, peer-to-peer (two servers) 121
trace files, storage requirements 8
unidirectional replication 63
viewing historical data 257
when Q Capture sends a message for publication 153
working with send queues in event publishing 147
in Q replication 64
XML messages 476
z/OS 315

Q Capture schema
naming rules 317
rules for bidirectional replication 97
rules for peer-to-peer replication 99
Q Capture schemas
dropping 243, 251
Q Capture server
code pages for event publishing 10
IBMQREP_ADMINMSG control table 400
IBMQREP_CAPENQ control table 401
IBMQREP_CAPENVINFO control table 401
IBMQREP_CAPMON control table 402
IBMQREP_CAPPARMS control table 404
IBMQREP_CAPQMcontrol table 408
IBMQREP_CAPQM control table 402
IBMQREP_COLVERSION control table 410
IBMQREP_COLVERSION control table 410
IBMQREP_IGNTRAN control table 412
IBMQREP_IGNTRANTRC control table 413
IBMQREP_SENDQUEUES control table 414
IBMQREP_SIGNAL control table 418
IBMQREP_SRC_COLS control table 421
IBMQREP_SRCH_COND control table 422
IBMQREP_SUBS control table 423
IBMQREP_TABVERSION control table 427
list of control tables 399
Q Capture throughput
checking memory usage 1
monitoring 257
Q Capture transaction latency 257, 258
Q replication
authorization requirements
for administration 12
for Q Apply 11
for Q Capture 11
for Replication Alert Monitor 272
overview 11
bidirectional replication
conflict detection 97, 108
description 97
changing 233
code pages 9
compression dictionaries 290
computed columns 78, 82
conflict detection
bidirectional replication 97
peer-to-peer replication 99
conflict detection, bidirectional replication 108
connectivity requirements 13
control tables
list, Q Apply 428
list, Q Capture 399
list, Replication Alert Monitor 456
Q replication (continued)
control tables (continued)
maintaining 291
overview 399
pruning 291
data conversion
description 9
databases, configuring
for z/OS 53
error options 93
errors, recovering 293
expressions 78
federated targets
overview 123
restrictions 124
software requirements 124
task overview 124
grouping replication queue maps and Q subscriptions 64
ignoring transactions 229
maintaining
cold starts, preventing 173
overview 287
source logs, manually finding
oldest required (z/OS) 288
source logs, retaining 287
source objects 287
source systems 287
memory
Q Apply program 4
Q Capture program 1
multidirectional replication
bidirectional replication 97
choosing between bidirectional and peer-to-peer replication 104
peer-to-peer replication 99
setting up 97
multiple database partitions 53
object names
case 320
naming rules 317
packages
Replication Alert Monitor (Linux, UNIX, Windows) 52
packages, binding
overview 50
Q Apply (Linux, UNIX, Windows) 51
Q Capture (Linux, UNIX, Windows) 50
peer-to-peer replication
conflict detection 99
creating Q subscriptions, overview 114
description 99
versioning columns 472
Q Apply program
bidirectional replication 97
peer-to-peer replication 99
unidirectional replication 63
Q Capture program
bidirectional replication 97
peer-to-peer replication 99
unidirectional replication 63
Q subscriptions
error options 93
filtering rows 74
Q replication (continued)
Q subscriptions (continued)
  naming rules 317
  search conditions 74
  WHERE clauses 74
queue manager clustering 44
rebinding packages and plans 295
recommendations for loading target tables 133
REORG command 293
restrictions
  data types 159
  IDENTITY columns 70, 105, 114, 115
RUNSTATS utility 292
sample programs
  Linux, UNIX, Windows 505
  overview 505
z/OS 506
samples
  JCL 507
setting up passwords 11
storage requirements
  DB2 logs 6
  trace files 8
storage requirements, WebSphere MQ 38
storing user IDs and passwords 13
system services 303
target object profiles 69
unidirectional replication
  creating Q subscriptions, overview 68
  description 63
  filtering rows 74
  index columns for targets 88
  key columns for targets 88
  mapping source and target columns 77
  Q Capture messages 73
  search conditions 74
  setting up 63
  source columns for Q subscriptions 72
  subsetting source columns 72
  unexpected conditions in targets 89
  viewing reports about 253
WebSphere MQ objects 17
setting up, overview 17
Q Replication Analyzer
  description 367
parameters 387
Q replication commands
asntrep 384
list 323
road map 323
Q subscriptions 197
  adding columns to replicate 235, 236
  bidirectional
    creating 105
  bidirectional replication 97
  CCD targets 85
  changing properties 233
  conflict detection, bidirectional replication 108
  creating 70
  federated replication 130
  peer-to-peer replication
    overview 114
  unidirectional replication
    overview 68
deleting 241
error options 93
filtering rows 74
grouping with replication queue maps 64
in bidirectional replication
  conflict detection 108
  index columns for targets 88
  key columns for targets 88
  naming rules 317
  options
    mapping source and target columns 77
    Q Capture messages 73
    peer-to-peer replication
      creating 114
    reinitializing 233
    restarting without redefining 241
    restrictions
      IDENTITY columns 70, 105, 114, 115
      search conditions 74
      starting 196
      subsetting source columns 72
      target object profiles 69
      unexpected conditions in targets 89
      unidirectional replication 63
      WHERE clauses 74
Q subscriptions, data type restrictions 159
qmgr parameter
  Q Capture 176
  qmgr parameter, Q Apply 336
  qmgr parameter, Q Capture 324
queryparms parameter
  Q Capture 333
  queryparms parameter, Q Apply 348
queue depth for large object (LOB) values 43
queue latency 257, 258
queue manager
description 30
required settings 25
queue manager clustering 44
QUEUE_DEPTH alert condition 32
queues
  administration
    for Q Apply 30
    for Q Capture 30
receive
description 30
restart 30
send 30
spill 30
R
range-partitioned tables, replicating 54
recapture, avoiding 106
receive queues
description 30
memory for Q replication 4
naming rules 317
reinitializing 239
required settings 25
starting 228
stopping 227
testing 35
working with the Q Apply program 64
referential constraints
  peer-to-peer replication 99
  referential integrity
    peer-to-peer replication 99
reinit parameter
  Q Capture 333
  reinitializing 279
reinitq parameter
  Q Apply 348
  Q Capture 333
remote DB2 databases
  replication 58
REORG command 293
replicating source table loads 144
replication
  bidirectional replication
    conflict detection 108
    description 97
  BLOB data types 160
  CLOB data types 160
  converted data 9
  DBCLOB data types 160
  mapping source and target columns Q replication 77
  multidirectional replication, setting up 97
  peer-to-peer replication 99
Q subscriptions
  creating for peer-to-peer replication 114
  creating for unidirectional replication, overview 68
  messages from Q Capture 73
ROWID columns 160
subset of data
  filtering rows in Q subscriptions 74
source columns for Q replication 72
unidirectional replication
description 63
setting up 63
replication (continued)
unidirectional replication (continued)
unexpected conditions 89
replication administration stored procedure 32
Replication Alert Monitor 274, 278, 279
alert conditions
e-mail notifications 268
events 263
list 265
overview 265
sending alerts to z/OS console 270
status 263
thresholds 263
alerts 263
authorization requirements 272
changing alert conditions 276
changing runtime parameters 284
Classic replication 263
contact groups 263
contacts 263
control tables
IBMSNAP_ALERTS 457
IBMSNAP_CONDITIONS 458
IBMSNAP_CONTACTGRP 463
IBMSNAP_CONTACTS 464
IBMSNAP_GROUPS 465
IBMSNAP_MONENQ 465
IBMSNAP_MONPARMS 465
IBMSNAP_MONSERVERS 468
IBMSNAP_MONTRACE 469
IBMSNAP_MONTRAIL 469
creating control tables 273
description 263
for Linux binding 272
for UNIX binding 272
operating 357
for Windows binding 272
operating 357
for z/OS
operating 357
memory 271
MODIFY command 306
monitoring replication, overview 263
operational errors 285
packages, binding 52
parameters
alert_prune_limit 281
autoprune 281
default values 281
descriptions 281
e-mail_server 281
max_notification_minutes 281
max_notifications_per_alert 281
monitor_errors 281
monitor_interval 281
monitor_limit 281
monitor_path 281
runonce 281
trace_limit 281
pruning control tables 286
running in batch mode 306
scheduling 314
Replication Alert Monitor (continued)
Linux, UNIX 314
Windows 315
selecting alert conditions 275
setting up 271
specifying notification criteria 285
specifying run frequency 285
started task 309
starting with JCL 305
stopping 280
viewing messages 257
Replication Alert Monitor program
checking status 253
Replication Center
running or saving commands 167
saving SQL scripts 165
software prerequisites 53
target object profiles 69
replication commands
asnqacmd 348
asncanalyze 387
asncapp 336
asnqcap 324
asnqcmm 333
asnqmfmt 395
asnqlist 367
asnqiff 368, 374
asntrep 384
for UNIX
asnmcmd 357
for Windows
asnmcmd 357
for z/OS
asnmcmd 357
replication programs
checking performance 253
scheduling 314
replication queue maps
bidirectional replication 97
changing 239
creating 66
deleting 242
grouping with Q subscriptions 64
peer-to-peer replication 99
sending test messages 35
unidirectional replication 63
replication services
creating 312
description 311
display name 311
dropping 314
listing 367
name 311
starting 313
stopping 313
viewing 314
restart message 200
restart queues
description 30
restart, Q Capture program 171
restart parameter, Q Capture 176, 324
restrictions
abstract data types 159
data types 159
DB2 Extenders large objects 160
distinct data types 159
EDITPROC clauses 159
restrictions (continued)
FIELDPROC clauses 159
LONG VARCHAR data types 159
spatial data types 159
user-defined data types 159
VALIDPROC clauses 159
resumesub parameter, Q Apply 348
resuming 279
richklvl parameter 211, 336
richklvl parameter, Q Apply 211
row (horizontal) subsetting 74
row operation XML message 487
ROWID columns, replicating 160
rows
filtering
event publishing 153
Q replication 74
runonce parameter, Replication Alert Monitor 281
RUNSTATS utility 292
S
sample commands for WebSphere MQ
objects 30
sample programs
event publishing
Linux, UNIX, Windows 505
overview 505
z/OS 506
Q replication
Linux, UNIX, Windows 505
overview 505
z/OS 506
samples
Q replication
JCL 507
schemas
for the Q Apply program
rules for bidirectional replication 97
rules for peer-to-peer replication 99
for the Q Capture program
rules for bidirectional replication 97
rules for peer-to-peer replication 99
naming rules
for bidirectional replication 97
for peer-to-peer replication 99
SCM (Service Control Manager)
creating replication services 312
description 311
dropping replication services 314
starting replication services 313
stopping replication services 313
viewing replication services 314
screen readers 509
send queues
description 30
managing activity by queue 200
naming rules 317
reinitializing 239, 249
required settings 25
restart points 200
send queues (continued)
  starting 203
  stopping 204
  testing 35
  working with the Q Capture program in event publishing 147
  in Q replication 64
  serialization threads 255
  server definitions for federated replication 127
Service Control Manager (SCM)
  creating replication services 312
  description 311
  dropping replication services 314
  starting a replication service 313
  stopping a replication service 313
  viewing replication services 314
setting up
  multidirectional replication 97
  passwords 11
  Replication Alert Monitor 272
  unidirectional replication 63
  user IDs 11
SHARE parameter, WebSphere MQ 25
signal_limit parameter, Q Capture 176, 324
signals
  ADDCOL 235, 236, 246
  skiptrans parameter, Q Apply 211, 336, 348
  sleep_interval parameter, Q Capture 176, 324
  software requirements
    Replication Center 53
    software services 509
source columns
  mapping to target columns
    Q replication 77
  published in publication 156
  publishing a subset
    event publishing 152
    publications 152
  replicating a subset 72
source columns, adding 235, 236, 246
source database
  enabling for replication 48
source logs, manually finding oldest required (z/OS) 288
source logs, retaining 287
source servers
  code pages
    for event publishing 10
    for Q replication 9
  when Q Capture sends a message
    for publications 153
  for Q subscriptions 73
source systems
  definition 287
  maintaining 287
source table
  adding columns to replicate 235, 236, 246
  replicating loads 144
  source table structure changes 238
  source tables
    for bidirectional replication 97
    for peer-to-peer replication 99
source tables (continued)
  for unidirectional replication
    description 63
  in publications
    filtering rows 153
    search conditions 153
  when Q Capture sends a message 153
  WHERE clauses 153
  in Q subscriptions
    filtering rows 74
    mapping source and target columns 77
    Q Capture messages 73
    search conditions 74
    maintaining 287
    mapping to targets, Q replication 77
    naming rules 317
source-target pairs
  for bidirectional replication
    description 97
  for peer-to-peer replication
    description 99
  for unidirectional replication 63
  mapping columns, Q replication 77
  spatial data types
    restrictions 159
spill queues
  description 30
  required settings 25
spill_commit_count parameter, Q Apply 211, 336
spillsub parameter, Q Apply 348
SQL errors
  acceptable 260
  unexpected 260
SQL expressions 78, 82
SQL replication commands
  asnpwd 360
  asnsct 364
  asnsdrop 366
  asnsrct 377
SQL scripts
  saving with ASNCLP 165
  saving with Replication Center 165
SQL states
  acceptable in Q replication
    setting for peer-to-peer 99
  SQLSTATE
    acceptable in Q replication
      setting for peer-to-peer 99
STATFILE
  acceptable in Q replication
    setting for peer-to-peer 99
START PUB command 198
START QSUB command 196
started tasks 309
starting 278
  Q Apply program 336
  starting from known point in log 171
startmode parameter, Q Capture 176, 324
starting 278
  Q Apply program 336
  starting from known point in log 171
startmode parameter, Q Capture 176, 324
started tasks 309
starting 278
  Q Apply program 336
  starting from known point in log 171
statistics
  Q Apply program 257
  Q Capture program 257
status of replication programs, checking 253
status parameter
  Q Apply 348
  Q Capture 333
stop parameter
  Q Capture 333
stop parameter, Q Apply 348
stopping 197, 199
  Replication Alert Monitor
    for UNIX 357
    for Windows 357
    for z/OS 357
stopq command (Q Apply) 227
stopq command (Q Capture) 204
stopq parameter, Q Apply 348
storage
  DB2 logs 6
  WebSphere MQ 38
storage requirements
  for diagnostic files 7
  Q Capture program 7
  trace files 8
stored procedures
  Q replication
    error options 93
    parameters for key columns 88
    unexpected conditions in targets 89
  Q subscriptions
    error options 93
    parameters for key columns 88
    unexpected conditions in targets 89
subscription deactivated message 491
subscription schema message 495
subsetting
  source columns
    publications 152
    Q subscriptions 72
  source rows
    publications 153
    Q subscriptions 74
support, customer 509
suspending 279
Sybase targets
  binding Q Apply packages 51, 126
  synchronization
    asntdiff and asntrepair utilities 297
system commands
  asnpwd 360
  asnqcmd 348
  asnqanalyze 387
  asnqapp 336
  asnsct 324
  asnsqcmd 333
  asnqcf 305
  asnl 397
  asnsct 364
  asnsqcf 397
  asnsrct 367
  asntdiff 368, 374
  asnsrct 377
  asntrep 384

T

  table differencing utility 297, 368, 374
  table repair utility 302, 384

Index 529
tables
IBMQREP_ADMINMSG 400
IBMQREP_APPENQ 429
IBMQREP_APPENVINFO 430
IBMQREP_APPLYMON 430
IBMQREP_APPLYPARMS 434
IBMQREP_APPLYTRACE 438
IBMQREP_CAPENQ 401
IBMQREP_CAPENVINFO 401
IBMQREP_CAPMON 402
IBMQREP_CAPPARMS 404
IBMQREP_CAPQMON 408
IBMQREP_CAPTRACE 410
IBMQREP_COLVERSION 410
IBMQREP_DELTOMB 439
IBMQREP_DONEMSG 440
IBMQREP_EOLFLUSH 412
IBMQREP_EXCEPTIONS 441
IBMQREP_IGNTRAN 412
IBMQREP_IGNTRANTRC 413
IBMQREP_RECVQUEUES 443
IBMQREP_SAVERI 445
IBMQREP_SENDQUEUES 414
IBMQREP_SIGNAL 418
IBMQREP_SPILLEDROW 447
IBMQREP_SPILLEDTABLES 447
IBMSNAP_ALERTS 457
IBMSNAP_CONDITIONS 458
IBMSNAP_CONTACTGRP 463
IBMSNAP_CONTACTS 464
IBMSNAP_GROUPS 465
IBMSNAP_MONENQ 465
IBMSNAP_MONPARMS 465
IBMSNAP_MONSERVERS 468
IBMSNAP_MONTRACE 469
IBMSNAP_MONTRAIL 469
IBMSNAP_SUSPENDS 471
IBMSNAP_TEMPLATES 472
target database configuring 49
key columns
publications 156
Q subscriptions 88
load options
automatic 134
automatic load 133
bidirectional replication 141
manual load 133
no load 133, 140
overview 133
peer-to-peer replication 141
recommendations 133
unidirectional replication 141
maintaining 294
mapping to sources
Q replication 77
recovery 317
repairing 302
storage space for changes 6
target key
publications 156
Q subscriptions 88
temporary directory, specifying 46
term parameter 211
Q Apply 336
Q Capture 176, 324
test messages, sending 35
threads
Q replication
administration 255
agent 255
browser 255
checking status 255
hold 255
housekeeping 255
prune 255
serialization 255
worker 255
throughput
Q Apply program 257
Q Capture program 257
TIMESTAMP data, replicating 161
TMPDIR environment variable 46
trace files, storage requirements 8
trace_limit parameter
Q Apply 211, 336
Q Capture 176, 324
Replication Alert Monitor 281
use with asmon command 352
validates 517
transaction latency, Q Capture 258
transaction message 478
transactions, ignoring 229
troubleshooting
checking program status 255
troubleshooting commands
asqanalyzer 387
unidirectional replication
adding columns to replicate 235
description 63
error options 93
load options 141
Q subscriptions
creating, overview 68
filtering rows 74
index columns for targets 88
key columns for targets 88
mapping source and target
columns 77
Q Capture messages 73
search conditions 74
source columns 72
unexpected conditions in the
targets 89
WHERE clauses 74
setting up
description 63
grouping replication queue maps
and Q subscriptions 64
WebSphere MQ objects required
remote 17
same system 18
user applications
list of messages from 500
user IDs, storing 13
user mappings for federated
replication 127
user-defined types (UDTs)
Q replication 159
utilities
EXPORT and IMPORT 134
EXPORT and LOAD 134
LOAD from CURSOR 134
message formatter 395, 397
Q Replication Analyzer 387
table differencing 368, 374
table repair 384
V
VALIDATE WSMQ ENVIRONMENT
FOR command 66, 148
VALIDPROC clauses 159
value-based conflicts 260
version-based conflicts 260
versioning columns 472
vertical (column) subsetting
event publishing 152
publications 152
Q replication 72
Q subscriptions 72
W
warm start
effect on log data 287
startmode parameter 174, 176
startmode parameter, three types 324
WebSphere MQ
authorization requirements
Q Apply program 37
Q Capture program 37
Replication Alert Monitor 37
WebSphere MQ (continued)
  dead-letter queues 42
  message segmentation 39
  message size limit
    Q Apply 39
    Q Capture 39
objects
  administration queue 30
  authorization requirements 37
  bidirectional replication 21
  connectivity requirements 37
  event publishing, remote 20
  list of 30
  peer-to-peer replication (three or
    more remote servers) 23
  peer-to-peer replication (two
    remote servers) 21
  receive queue 30
  required settings 25
  restart queue 30
  send queue 30
  spill queue 30
  unidirectional replication,
    remote 17
  unidirectional replication, same
    system 18
parameters
  CHLTYPE 25
  DEFPBIST 25
  DEFSOPT 25
  DEFTYPE 25
  DISCINT 25
  HBINT 25
  MAXDEPTH 25
  MAXMSGL 25
  MSGDLVSQ 25
  SHARE 25
  setting up event publishing 17
  setting up Q replication 17
  storage requirements 38
WebSphere MQ clients 32
WHERE clause
  filtering data
    rows in publications 153
    rows in Q subscriptions 74
Windows service
  creating 364, 366
  Windows Service Control Manager (SCM)
    asnslist command 367
    description 311
    listing replication services 367
  worker threads 255
  wrappers for federated replication 127

X
XML expressions 82
XML messages
  event publishing
    delimiters in character data 476
    technical requirements 475
  heartbeat message 494
  msg: root element 477, 500
  overview of types 475
  row operation message 487
  structure
    overview 475

Z
z/OS
  load considerations 135
  specifying CAPTURE_PATH 303
z/OS console
  sending monitor alerts 270