IBM DB2 Information Integrator

Replication and Event Publishing Guide and Reference

Version 8.2

SC18-7568-00

IBM DB2 Information Integrator

Replication and Event Publishing Guide and Reference

Version 8.2

SC18-7568-00

Contents

About this book xi
Who should read this book xi
Text conventions used in this book. xi
Terminology used in this book xi
Where to find more information xii
How to read syntax diagrams xiii

Part 1. Introduction to Q replication and event publishing 1

Chapter 1. Introduction to Q replication 3
Introduction to Q replication—Overview 3
Q replication 3
  Unidirectional replication 4
  Bidirectional replication. 4
  Peer-to-peer replication 4
Q subscriptions 5
Replication queue maps 6

Chapter 2. Introduction to event publishing. 9
Introduction to event publishing—Overview. . . . 9
Event publishing 9
  Objects for event publishing 10
  Controlling message content 11
XML publications 11
Publishing queue maps 12

Chapter 3. Introduction to the programs that are used for replicating and publishing 15
Introduction to the programs that are used for replicating and publishing—Overview 15
Q Capture program. 15
  Capturing changes 15
  Filtering data 16
  Message formats. 16
  Committing messages 16
  Changing parameters 17
  Communicating with Q Capture 17
  Restart messages 18
Q Apply program 18
  Browser and agent threads 19
  Applying transactions in parallel 19
  Changing parameters 19
Schemas for the Q Apply and Q Capture programs 20

Part 2. Preparing for Q replication and event publishing 23

Chapter 4. Planning memory and storage requirements 25
Planning memory and storage requirements for Q replication and event publishing 25
Planning memory requirements for Q replication and event publishing 25
  Planning memory requirements for Q replication and event publishing 25
  Memory used by the Q Capture program . . . 26
  Memory used by the Q Apply program 29
  Memory used by the Replication Alert Monitor 30
  Memory for LOB data types for Q replication and event publishing 31
Planning storage requirements for Q replication and event publishing. 31
  Planning storage requirements for Q replication and event publishing 31
  Storage requirements for DB2 UDB logs for Q replication and event publishing 32
  Storage for diagnostic files for Q replication and event publishing. 32
  Storage requirements for when the Q Capture program exceeds its memory limit. 33
  Storage requirements for traces for Q replication and event publishing 34

Chapter 5. Data conversion 35
Data conversion for Q replication and event publishing—Overview. 35
Data conversion for Q replication 35
Data conversion for event publishing. 36

Chapter 6. Setting up user IDs and passwords 37
Setting up user IDs and passwords—Overview . . 37
Authorization requirements for Q replication and event publishing. 37
  Authorization requirements for Q replication and event publishing—Overview 37
  Authorization requirements for the Q Capture program 37
  Authorization requirements for the Q Apply program 38
  Authorization requirements for the Replication Alert Monitor. 39
  Authorization requirements to administer Q replication and event publishing 40
Storing user IDs and passwords for remote servers 41

Chapter 7. Setting up WebSphere MQ for Q replication and event publishing . 43
Setting up WebSphere MQ for Q replication and event publishing—Overview 43
WebSphere MQ objects required for Q replication and event publishing 43
  WebSphere MQ objects required for Q replication and event publishing—Overview 43

Required WebSphere MQ objects at a glance . . 44
WebSphere MQ objects required for unidirectional replication on the same system . . 45
WebSphere MQ objects required for unidirectional replication (remote) 47
WebSphere MQ objects required for event publishing 48
WebSphere MQ objects required for bidirectional or peer-to-peer replication (two remote servers) . 50
WebSphere MQ objects required for peer-to-peer replication (three or more remote servers) . . . 53
Required settings for WebSphere MQ objects . . . 56
WebSphere MQ objects at the source 56
WebSphere MQ objects at the target 57
WebSphere MQ channel objects. 58
WebSphere MQ message size 60
Storage requirements for WebSphere MQ for Q replication and event publishing 61
Storage requirements for spill queues for the Q Apply program 62
Connectivity and authorization requirements for WebSphere MQ objects 62
Queue depth considerations for large object (LOB) values 64
Queue manager clustering in Q replication and event publishing. 64

Chapter 8. Configuring servers for Q replication and event publishing. . . . 67
Configuring servers for Q replication and event publishing—Overview. 67
Connectivity requirements for Q replication and event publishing. 67
Configuring databases for Q replication and event publishing (Linux, UNIX, Windows) 68
Configuring databases for Q replication and event publishing (Linux, UNIX, Windows) . . . 68
Setting environment variables (Linux, UNIX, Windows) 68
Configuring the source database to work with the Q Capture program (Linux, UNIX, Windows). 69
Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows) . . 71
Optional: Binding the program packages (Linux, UNIX, Windows) 72
Configuring databases for Q replication and event publishing (z/OS) 75
Software prerequisites for the Replication Center . . 76
Creating control tables for the Q Capture and Q Apply programs. 76

Part 3. Setting up Q replication and event publishing. 79

Chapter 9. Setting up replication from sources to targets (unidirectional) . . . 81
Setting up replication from sources to targets (unidirectional)—Overview 81
Unidirectional replication. 81
Grouping replication queue maps and Q subscriptions 83
How the Q Capture program works with the send queue 83
How the Q Apply program works with the receive queue. 83
Suggestions for grouping similar Q subscriptions with replication queue maps. 84
Creating replication queue maps 84
Creating Q subscriptions for unidirectional replication 86
Creating Q subscriptions for unidirectional replication—Overview. 86
Target object profiles for Q replication 87
Creating a single Q subscription for unidirectional replication 88
Creating multiple Q subscriptions for unidirectional replication 90
Source columns for Q subscriptions (unidirectional replication) 92
How often the Q Capture program sends a message (unidirectional replication) 93
Search conditions to filter rows (unidirectional replication) 94
How source columns map to target columns (unidirectional replication) 98
Index or key columns for targets (unidirectional replication) 99
Options for unexpected conditions in the target table (unidirectional replication) 100
Error options for Q replication 103
Using stored procedures to manipulate the data that is applied to targets. 106
Using stored procedures to manipulate the data that is applied to targets for Q replication—Overview 106
Stored procedures for manipulating source data for Q replication 107
Writing stored procedures to manipulate source data for Q replication. 108
Stored procedure parameter that identifies the type of operation for Q replication 110
Stored procedure parameter that identifies whether each source column was suppressed. . 112
Stored procedure parameters that identify the transaction for Q replication 112
Stored procedure parameters that map to source columns for Q replication 113

Chapter 10. Setting up replication from sources to targets (multidirectional) 117
Setting up replication from sources to targets (multidirectional)—Overview 117
Bidirectional replication 117
Replication objects for bidirectional replication 118
Conflict detection in bidirectional replication 119
Peer-to-peer replication 120
Replication objects for peer-to-peer replication with two servers 121

Replication objects for peer-to-peer replication with three or more servers 122
Conflict resolution and referential integrity . . 123
Bidirectional replication versus peer-to-peer replication 125
Scenarios that work best with bidirectional replication 125
Scenarios that work best with peer-to-peer replication 125
Bidirectional replication versus peer-to-peer replication for high-availability scenarios . . . 125
Creating Q subscriptions for bidirectional replication 127
Creating Q subscriptions for bidirectional replication 127
Options for conflict detection (bidirectional replication) 129
Creating Q subscriptions for peer-to-peer replication 135
Creating Q subscriptions for peer-to-peer replication—Overview 135
Creating Q subscriptions for peer-to-peer replication with two servers 135
Creating Q subscriptions for peer-to-peer replication with three or more servers 137
Starting bidirectional or peer-to-peer replication with two servers 139
Stopping bidirectional or peer-to-peer replication with two servers 140
Starting replication in a peer-to-peer group with three or more servers. 140
Stopping replication in a peer-to-peer group with three or more servers. 141

Chapter 11. Options for loading target tables for Q replication 143
Options for loading target tables for Q replication—Overview 143
Recommendations for loading target tables for Q replication 143
Automatic load option for Q replication 144
Automatic load option for Q replication—Overview 144
Utilities used for automatic load option for Q replication 145
Automatic load considerations for z/OS platform 146
Specifying nicknames for the automatic load option for Q replication 146
Manual load option for Q replication 148
No load option for Q replication 150
Load options for different types of Q replication 150

Chapter 12. Setting up publishing from sources (event publishing) . . . 155
Setting up publishing from sources (event publishing)—Overview 155
Grouping publishing queue maps and XML publications 155
How the Q Capture program works with the send queue 155
Suggestions for grouping similar XML publications with publishing queue maps . . . 156
Creating publishing queue maps 156
Creating XML publications 158
Creating XML publications—Overview 158
Creating a single XML publication 159
Creating multiple XML publications. 161
Source columns for XML publications 163
When the Q Capture program publishes a message for XML publications. 163
Search conditions to filter rows in XML publications 164
Key columns for XML publications 167
Options for including unchanged columns in messages for XML publications 168
Options for including before values in messages for XML publications. 169

Chapter 13. Considerations for replicating and publishing data types . 171
Considerations for replicating and publishing data types for Q replication and event publishing—Overview 171
Considerations for general data types for Q replication and event publishing 171
Considerations for large object (LOB) data types for Q replication and event publishing 172

Chapter 14. Running SQL scripts and operational commands from the Replication Center 173
Running SQL scripts and operational commands from the Replication Center—Overview 173
SQL scripts and operational commands that the Replication Center generates 173
Running and saving SQL scripts from the Replication Center. 174
Running and saving commands from the Replication Center. 175

Part 4. Administering and monitoring your Q replication and event publishing environment. . . 177

Chapter 15. Changing a Q replication environment 179
Changing a Q replication environment—Overview 179
Changing attributes of Q subscriptions 179
Adding columns to existing Q subscriptions . . . 181
Changing acceptable SQL states for Q subscriptions 183
Changing attributes of replication queue maps . . 184
Deleting Q subscriptions 186
Deleting replication queue maps 188
Dropping Q Capture schemas or Q Apply schemas 188

Chapter 16. Changing an event publishing environment 191
Changing an event publishing environment—Overview. 191
Changing attributes of XML publications 191
Adding columns to existing XML publications . . 193
Changing attributes of publishing queue maps . . 195
Deleting XML publications 196
Deleting publishing queue maps 197
Dropping Q Capture schemas 198

Chapter 17. Operating a Q Capture program 201
Operating a Q Capture program—Overview . . . 201
Starting a Q Capture program 201
Considerations for cold starting a Q Capture program 203
Parameters of a Q Capture program 205
  Parameters of a Q Capture program—Overview 205
  Default values for Q Capture operating parameters 205
  Descriptions of Q Capture parameters 206
Changing the Q Capture parameters 217
  Changing the Q Capture parameters—Overview 217
  Methods of changing the Q Capture operating parameters 217
  Changing saved Q Capture parameters in the IBMQREP_CAPPARMS table 219
  Dynamically changing parameters while a Q Capture program is running 220
Activating Q subscriptions or XML publications 221
Deactivating Q subscriptions or XML publications 222
Stopping a Q Capture program 224

Chapter 18. Operating a Q Apply program 227
Operating a Q Apply program—Overview . . . 227
Starting a Q Apply program 227
Parameters of a Q Apply program 229
  Parameters of a Q Apply program—Overview 229
  Default values for Q Apply operating parameters 229
  Descriptions of Q Apply parameters. 231
Changing the Q Apply parameters 237
  Changing the Q Apply parameters—Overview 237
  Changing saved Q Apply parameters in the IBMQREP_APPLYPARMS table 237
  Dynamically changing parameters while a Q Apply program is running 238
Starting message processing on a receive queue 239
Stopping message processing on a receive queue 241
Stopping a Q Apply program 242

Chapter 19. Viewing reports about the Q replication and event publishing programs 245
Viewing reports about the Q replication and event publishing programs—Overview 245
Checking the status of the Q replication and event publishing programs 245
Threads of the Q Capture, Q Apply, and Replication Alert Monitor programs 246
Historical and performance data for Q replication and event publishing programs 248
Latency 250
Exceptions 252

Chapter 20. Monitoring replication with the Replication Alert Monitor . . 255
Monitoring replication with the Replication Alert Monitor—Overview 255
The Replication Alert Monitor 255
Alert conditions and notifications for the Replication Alert Monitor 257
  Alert conditions and notifications for the Replication Alert Monitor—Overview 257
  Alert conditions for the Replication Alert Monitor 258
  E-mail notifications for replication alert conditions 261
  The ASNMAIL exit routine for sending alerts in replication 262
Setting up the Replication Alert Monitor 263
  Setting up the Replication Alert Monitor . . . 263
  Creating control tables for the Replication Alert Monitor 263
  Defining contact information for the Replication Alert Monitor 264
Operating the Replication Alert Monitor 265
  Operating the Replication Alert Monitor . . . 265
  Creating monitors for replication or publishing 266
  Selecting alert conditions for the Replication Alert Monitor 267
  Starting monitors 268
  Reinitializing monitors 268
  Default values of the parameters that are used to operate the Replication Alert Monitor . . . 269
  Descriptions of the parameters that are used to operate the Replication Alert Monitor 270
  Setting parameters for the Replication Alert Monitor 272
  Stopping a monitor 275

Chapter 21. Maintaining a Q replication and publishing environment 277
Maintaining a Q replication and event publishing environment—Overview. 277
Considerations for maintaining Q replication and event publishing source systems 277
  Considerations for maintaining Q replication and event publishing source systems—Overview 277
  Maintaining source tables in a Q replication and event publishing environment 278
  Retaining log files for Q replication and event publishing 278

Considerations for managing compression dictionaries in Q replication and event publishing environment (z/OS) 282
Maintaining control tables in Q replication and event publishing 283
Maintaining control tables in Q replication and event publishing—Overview 283
Pruning control tables in Q replication and event publishing 284
Considerations for using the RUNSTATS utility on control tables for Q replication and event publishing 285
Reorganizing Q replication and event publishing control tables 285
When replication programs cannot connect to their DB2 UDB server 286
Maintaining target tables in Q replication and event publishing 286
Considerations for rebinding packages and plans for Q replication and event publishing 287

Chapter 22. Detecting and repairing differences between source and target tables. 289
Detecting and repairing differences between source and target tables 289
Tdiff: Table difference utility 289
Trepair: Table repair utility 292

Chapter 23. Using system services to operate the replication programs . . . 293
Using system services to operate the Q replication and event publishing programs—Overview . . . 293
Operating the replication programs by using system services (z/OS) 293
Operating the Q replication and event publishing programs by using system services (z/OS)—Overview 293
Running the replication programs by using JCL 294
Running the Q replication and event publishing programs with system-started tasks 296
The Automatic Restart Manager (ARM) recovery system 297
Setting up the Automatic Restart Manager (ARM) to restart Q replication and event publishing programs 298
Operating the replication programs by managing replication services (Windows) 299
Managing replication services with the Windows Service Control Manager (SCM) —Overview 299
Replication services for Q replication and event publishing (Windows) 299
Creating a replication service 300
Starting a replication service 301
Stopping a replication service 302
Viewing a list of replication services. 302
Dropping a replication service. 303
Scheduling replication programs 303
Scheduling the replication programs—Overview 303
Scheduling the replication programs (Linux, UNIX). 303
Scheduling the replication programs (Windows) 304
Scheduling the replication programs (z/OS) . . 305

Part 5. Reference information for Q replication and event publishing 307

Chapter 24. Naming rules and guidelines. 309
Naming rules and guidelines for Q replication and event publishing—Overview 309
Naming rules for Q replication and event publishing objects 309
How lowercase object names are handled for Q replication and event publishing 312

Chapter 25. System commands for Q replication and event publishing . . . 313
System commands for Q replication and event publishing—Overview 313
Road map: Q replication and event publishing system commands. 313
asnqcap: Starting a Q Capture program 315
asnqccmd: Working with a running Q Capture program 320
asnqapp: Starting a Q Apply program 323
asnqacmd: Working with a running Q Apply program 326
asnmon: Starting a Replication Alert Monitor. . . 329
asnmcmd: Working with a running Replication Alert Monitor 333
asnqanalyze: Operating the Q Replication Analyzer 336
asnscrt: Creating a DB2 replication service to start the replication programs. 339
asnslist: Listing DB2 replication services 342
asnsdrop: Dropping DB2 replication services . . . 343
asnpwd: Creating and maintaining password files 344
asntrc: Operating the replication trace facility. . . 348
asntdiff: Comparing data in source and target tables 355
asntrep: Repairing differences between source and target tables 357
asnqmfmt: Formatting and viewing Q replication and event publishing messages 358

Chapter 26. Control tables for Q replication and event publishing . . . 361
Control tables for Q replication and event publishing—Overview 361
Control tables at a glance 361
List of control tables at the Q Capture server . . . 367
List of control tables at the Q Apply server . . . 368
List of tables at the Monitor control server . . . 369
Detailed structure of the Q Capture control tables 370
Detailed structure of the Q Capture control tables—Overview 370
IBMQREP_ADMINMSG table 371
IBMQREP_CAPENQ table 371

IBMQREP_CAPMON table 372
IBMQREP_CAPPARMS table 374
IBMQREP_CAPQMON table 377
IBMQREP_CAPTRACE table 378
IBMQREP_SRCH_COND table 379
IBMQREP_SENDQUEUES table 380
IBMQREP_SIGNAL table 382
IBMQREP_SRC_COLS table 386
IBMQREP_SUBS table 387
Detailed structure of the Q Apply control tables 391
Detailed structure of the Q Apply control tables—Overview 391
IBMQREP_APPENQ table 392
IBMQREP_APPLYMON table 393
IBMQREP_APPLYPARMS table 395
IBMQREP_APPLYTRACE table 397
IBMQREP_DELTOMB table. 398
IBMQREP_DONEMSG table 399
IBMQREP_EXCEPTIONS table 400
IBMQREP_RECVQUEUES table 402
IBMQREP_SAVERI table. 404
IBMQREP_SPILLEDROW table 405
IBMQREP_SPILLQS table 406
IBMQREP_TRG_COLS table 406
IBMQREP_TARGETS table 408
Detailed structure of the Monitor control tables . . 414
Detailed structure of the Monitor control tables—Overview 414
IBMSNAP_ALERTS table 414
IBMSNAP_CONDITIONS table 416
IBMSNAP_CONTACTGRP table 421
IBMSNAP_CONTACTS table 422
IBMSNAP_GROUPS table 423
IBMSNAP_MONENQ table. 423
IBMSNAP_MONPARMS table 423
IBMSNAP_MONSERVERS table 425
IBMSNAP_MONTRACE table 426
IBMSNAP_MONTRAIL table 427
Detailed structure of versioning columns for peer-to-peer replication 429

Chapter 27. Structure of XML messages for event publishing 431
Structure of XML messages for event publishing—Overview 431
XML message types and requirements 431
XML message types and requirements—Overview 431
XML message types 431
Technical requirements for XML messages . . . 432
How XML delimiters are handled in character data 432
Structure of XML messages from Q Capture to a user application 433
Structure of XML messages from Q Capture to a user application—Overview 433
List of XML messages from Q Capture to a user application 434
msg: Root element for XML messages from Q Capture to a user application 435
Transaction message 436
Row operation message 445
Large object (LOB) message 446
Subscription deactivated message 449
Load done received message 450
Error report message 451
Heartbeat message 453
Subscription schema message (subSchema) . . 454
Add column message 458
Structure of XML messages from a user application to Q Capture 459
Structure of XML messages from a user application to Q Capture—Overview 459
List of XML messages from a user application to Q Capture 459
msg: Root element for XML messages from a user application to Q Capture 460
Invalidate send queue message 461
Load done message 462
Activate subscription message. 463
Deactivate subscription message 463

Appendix A. Sample programs to use for Q replication and event publishing 465
Sample programs to use for Q replication and event publishing—Overview 465
Sample programs (Linux, UNIX, Windows) . . . 465
Sample programs to use for Q replication and event publishing (Linux, UNIX, Windows)—Overview 465
Samples to set up Q replication and event publishing (Linux, UNIX, Windows). 465
Samples to operate Q replication programs (Linux, UNIX, Windows) 466
Sample programs (z/OS) 467
Sample programs to use for Q replication and event publishing (z/OS)—Overview. 467
Samples to set up Q replication and event publishing (z/OS) 467
Sample JCL programs for Q replication and event publishing (z/OS). 468

Appendix B. ASNCLP: Command-line interface for administering replication and event publishing 469

Glossary 471
Glossary 471

Accessibility. 483
Keyboard input and navigation 483
Accessible display 483
Font settings. 483
Nondependence on color 483
Alternative alert cues 483
Compatibility with assistive technologies 483
Accessible documentation 484

Notices 485
Trademarks 487

Index 489

Contacting IBM 503

Product information 503

Comments on the documentation. 503

About this book

This book describes how to plan, set up, maintain, and monitor a data replication and publishing environment using Q replication and event publishing. You can use Q replication and event publishing with other products in the IBM® replication solution to tailor a replication and publishing environment that suits your business needs. This book contains the guidance and reference information for the Q replication and event publishing components that are introduced in *IBM DB2 Information Integrator Introduction to Replication and Event Publishing*.

Most sections in this book pertain to all supported operating-system environments. For example, the section entitled *Starting the Q Apply program* explains how to start the Q Apply program from DB2 Universal Database for Linux and for all UNIX platforms, DB2 Universal Database for Windows, or DB2 Universal Database for z/OS and OS/390. Information that pertains to specific operating systems is clearly labeled.

Who should read this book

This book is for database administrators, data replication specialists, and others who set up and maintain a Q replication or event publishing environment. This book assumes that you are familiar with:

- Standard database terminology.
- Database design, database administration, database security, server connectivity, and networking.
- The concepts about Q replication and event publishing that are described in *IBM DB2 Information Integrator Introduction to Replication and Event Publishing*.
- The operating systems that will be involved in your Q replication or event publishing environment.
- The data that you want to replicate or publish.
- The applications that you want to receive messages in event publishing.
- WebSphere MQ concepts and objects. (This book refers you to other sources for more detailed information about configuring and using WebSphere MQ.)

Text conventions used in this book

This book uses these highlighting conventions:

- **Boldface type** indicates commands or user interface controls such as names of fields, folders, icons, or menu choices.
- `Monospace type` indicates examples of text that you enter exactly as shown.
- *Italic type* indicates variables that you should replace with a value. It is also used to indicate book titles and to emphasize words.

Terminology used in this book

This book uses standard terminology for database, connectivity, copying, SQL, and LAN concepts. All of the replication and publishing concepts used in this book are defined in the "Glossary" on page 471. The main concepts used in this book are introduced and explained in *IBM DB2 Information Integrator Introduction to Replication and Event Publishing*.

Unless otherwise specified, the following meanings are assumed:

Linux Linux refers to Q replication and event publishing for all Linux platforms (such as Linux Intel and Linux z/Series).

UNIX UNIX refers to Q replication and event publishing for all UNIX platforms (such as HP UX, Solaris Operating Environment, and AIX).

Windows
Windows refers to Q replication and event publishing for Windows platforms (such as 2000 and XP).

z/OS z/OS refers to Q replication and event publishing for z/OS and OS/390. z/OS is the next generation of the OS/390 operating system, and it also includes UNIX System Services (USS) on z/OS.

Where to find more information

This section identifies other sources of information about DB2 replication that you might find useful.

If you want to ...	Refer to ...
Learn about the DB2 Information Integrator replication solution	DB2 DataPropagator at www.ibm.com/software/data/dpropr/
Learn basic concepts of replication and event publishing	Either of the following places: • The DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/ • *IBM DB2 Information Integrator Introduction to Replication and Event Publishing*
Learn how to set up and maintain SQL replication	Either of the following places: • The DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/ • *IBM DB2 Information Integrator SQL Replication Guide and Reference*
Learn about last-minute changes to the product	The Installation Notes on the CD-ROM or the Release Notes that are installed with the products.
Learn how to use the Replication Analyzer command-line tool to produce an HTML report about your replication environment	DB2 DataPropagator product documentation at www.ibm.com/software/data/dpropr/library.html
Download the ASNCLP command-line processor and supporting documentation	http://www.ibm.com/software/data/dpropr/library.html
Debug error messages	ASN messages in either of the following places: • The DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/ • DB2 UDB *Message Reference Volume 1*
Find technical support resources and customer support options	www.ibm.com/software/data/integration/db2ii/
View the information set for DB2 Universal Database and DB2 Information Integrator	See the DB2 Information Center http://publib.boulder.ibm.com/infocenter/db2help/

How to read syntax diagrams

The following rules apply to the syntax diagrams used in this book:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement.

  The ──► symbol indicates that the statement syntax is continued on the next line.

  The ►── symbol indicates that a statement is continued from the previous line.

  The ──►◄ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ►── symbol and end with the ──► symbol.
- Keywords, their allowable synonyms, and reserved parameters, are either shown in uppercase or lowercase, depending on the operating system. These items must be entered exactly as shown. Variables appear in lowercase italics (for example, *column-name*). They represent user-defined parameters or suboptions.

  When entering commands, separate the parameters and keywords by at least one space if there is no intervening punctuation.
- Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs, and so on) and numbers exactly as given.
- Footnotes are shown by a number in parentheses, for example, (1).
- Required items appear on the horizontal line (the main path).

```
►►──required_item──────────────────────────────────────────────────►◄
```

- A parameter's default value is displayed above the path:

```
                 ┌─default_value─┐
►►──required_item┴───────────────┴─────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►──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─┘
```

Part 1. Introduction to Q replication and event publishing

This part of the book contains the following chapters:

- Chapter 1, "Introduction to Q replication," on page 3 provides an overview of how Q replication works and an overview of different configurations, including unidirectional and multidirectional replication. The chapter also describes the objects involved in Q replication, including replication queue maps, Q subscriptions, and target object profiles.
- Chapter 2, "Introduction to event publishing," on page 9 provides an overview of how event publishing works and describes the objects involved in event publishing, including publishing queue maps and XML publications.
- Chapter 3, "Introduction to the programs that are used for replicating and publishing," on page 15 describes the Q Capture program, which is the program in Q replication and event publishing that captures data from the source tables, and the Q Apply program, which is the program in Q replication that applies data to the target tables or passing data to a stored procedure for data manipulation.

Chapter 1. Introduction to Q replication

Introduction to Q replication—Overview

The following topics provide an introduction to the main concepts in Q replication:

- "Q replication"
- "Q subscriptions" on page 5
- "Replication queue maps" on page 6

Related concepts:

- "Introduction to the programs that are used for replicating and publishing—Overview" on page 15
- "Introduction to event publishing—Overview" on page 9

Q replication

Q replication is a high-volume, low-latency replication solution that uses WebSphere® MQ message queues to transmit transactions between source and target databases or subsystems. The Q Capture program reads the DB2® recovery log for changes to a source table that you specify. The program then sends transactions as messages over queues, where they are read and applied to targets by the Q Apply program.

This type of replication offers several advantages:

Minimum latency
Changes are sent as soon as they are committed at the source and read from the log.

High-volume throughput
The Q Capture program can keep up with rapid changes at the source, and the multithreaded Q Apply program can keep up with the speed of the communication channel.

Minimum network traffic
Messages are sent using a compact format, and data-sending options allow you to transmit the minimum amount of data.

Asynchronous
The use of message queues allows the Q Apply program to receive transactions without having to connect to the source database or subsystem. If either of the replication programs is stopped, messages remain on queues to be processed whenever the program is ready. Because the messages are persistent, the source and target remain synchronized even in the event of a system or device failure.

Q replication allows many different configurations. You can replicate between remote servers or within a single server. You can replicate changes in a single direction or in multiple directions. Replicating in multiple directions can be bidirectional (useful for managing standby or backup systems) or peer-to-peer (useful for synchronizing data on production systems).

To use Q replication, you create the following types of objects:

- Replication queue maps, which identify the WebSphere MQ queues for sending and receiving data.
- Q subscriptions, which identify options such as which rows and columns are replicated or published and options for loading target tables.

The following sections provide a quick overview of the three types of Q replication:
- "Unidirectional replication"
- "Bidirectional replication"
- "Peer-to-peer replication"

Unidirectional replication

Unidirectional replication is a configuration that has the following characteristics:
- Changes 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.
- Changes that occur at the target table are not replicated back to the source table.
- The target table typically is read-only, or is updated only by the Q Apply program.

Bidirectional replication

Bidirectional replication is a configuration that has the following characteristics:
- Replication occurs between tables on two servers. 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.
- Updates on either of the servers are replicated to the other server.
- Applications on any of the servers 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 of data, and the other server updates the other columns.
- You can choose which copy of the table wins if a conflict occurs.

Peer-to-peer replication

Peer-to-peer replication (also known as multimaster replication) is a configuration that has the following characteristics:
- Replication occurs between tables on two or more servers.
- Updates on any one server are replicated to all other associated servers that are involved in the peer-to-peer configuration.
- 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.

Related concepts:
- "Bidirectional replication" on page 117
- "Peer-to-peer replication" on page 120
- "Q Apply program" on page 18
- "Q Capture program" on page 15

- "Introduction to Q replication—Overview" on page 3
- "Q subscriptions" on page 5
- "Replication queue maps" on page 6
- "Unidirectional replication" on page 81

Q subscriptions

In Q replication, you create objects called *Q subscriptions* to define how data from a single source table is replicated to a single target table or is passed to parameters in a stored procedure for data manipulation. The Q subscription tells the replication programs which changes to capture from the source table, what queues to use for sending and receiving change messages, and how to process the messages. Figure 1 shows how a Q subscription connects a source table to a target table.

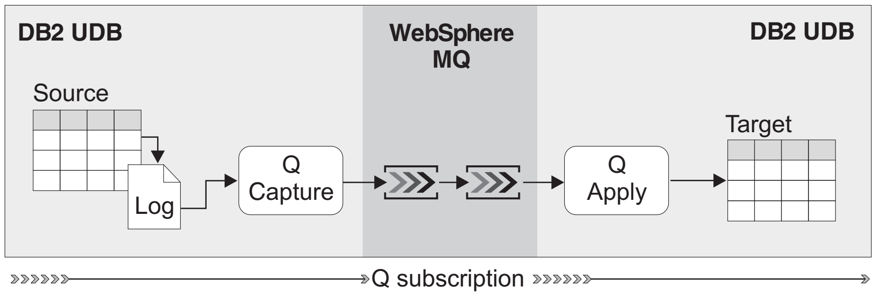


Figure 1. Q subscription. Changes from a source table are replicated over WebSphere MQ queues to a target table.

You must create a Q subscription for each source-to-target pair. 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

For Q subscriptions, you can specify the following options:
- Which columns to replicate and how they map to columns at the target table (or to parameters in a stored procedure)
- A search condition to determine which rows from the source table are replicated
- How the Q Apply program detects and responds to conflicts between rows at the source and target database

Q subscriptions for either bidirectional or peer-to-peer replication must replicate all columns in the source table. These types of replication also require specific types of conflict detection and conflict handling.

If you make changes to a Q subscription after you create it, you need to reinitialize it so that the Q Capture program knows about those changes.

Important: Q subscriptions are separate objects from XML publications; Q subscriptions do not replicate data that is published in XML publications. Q subscriptions are for replicating data, and XML publications are for publishing data.

Related concepts:
- "Q replication" on page 3
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "XML publications" on page 11

Related tasks:
- "Changing attributes of Q subscriptions" on page 179
- "Deleting Q subscriptions" on page 186

Replication queue maps

In Q replication, a *replication queue map* identifies the WebSphere® MQ queues that a Q Capture program and a Q Apply program use to transport data and communicate. Each replication queue map identifies one of each of the following WebSphere MQ queues:

Send queue
The WebSphere MQ queue where the Q Capture program sends source data and informational messages. Messages that are sent using a send queue that is identified by a replication queue map must be in compact format, which is the message format that the Q Apply program reads.

Receive queue
The WebSphere MQ queue where the Q Apply program receives source transactions before applying them to the target table or passing them to a stored procedure.

Administration queue
The Q Apply program uses this administration queue to send control messages to the Q Capture program.

You can use a single replication queue map to transport data for one or more Q subscriptions. Also, 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.

In addition to identifying WebSphere MQ queues, a replication queue map also contains replication attributes that involve the send queue and the receive queue. For each replication queue map, you specify the following attributes that involve the send queue:
- The WebSphere MQ name of the send queue.
- The maximum size of a message that the Q Capture program can put on this send queue. (This limit is independent of the WebSphere MQ Series maximum message length, but this limit must be equal to or less than the WebSphere MQ Series maximum message length.)
- How the Q Capture program responds if an error occurs at the WebSphere MQ queue.
- How often 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 changes to replicate.

You specify the following attributes that involve the receive queue:

- The WebSphere MQ name of the receive queue.
- The number of threads for each Q Apply browser to be created to apply transactions to target tables or pass transactions to stored procedures to manipulate the data.
- The amount of memory that the Q Apply program can use to process messages from the receive queue.
- The schema that identifies the Q Capture program that is sending messages that contain transactions.
- The name of the WebSphere MQ message queue to use for the administration queue, which is the queue where the Q Apply program sends messages when it needs to communicate with the Q Capture program.

Related concepts:

- "Publishing queue maps" on page 12
- "Introduction to Q replication—Overview" on page 3
- "Grouping replication queue maps and Q subscriptions" on page 83

Related tasks:

- "Creating replication queue maps" on page 84

Chapter 2. Introduction to event publishing

Introduction to event publishing—Overview

The following topics provide an introduction to the main concepts in event publishing:

- "Event publishing"
- "XML publications" on page 11
- "Publishing queue maps" on page 12

Related concepts:

- "Introduction to the programs that are used for replicating and publishing—Overview" on page 15
- "Introduction to Q replication—Overview" on page 3

Event publishing

In *event publishing*, changes to source tables are translated into XML messages and sent over WebSphere® MQ queues to a user application of your choice.

The following examples show reasons that you might want to use event publishing:

To update a Web site
Changes to a database or subsystem that records stock prices are captured from the DB2® log, and published as XML messages to a JSP (Java Server Pages) application that runs on an application server (such as WebSphere Application Server). The XML messages are then used to update HTML pages that display the up-to-date stock information.

To feed a central integration broker
When a customer updates an address, the transaction is published as an XML message to a central WebSphere Business Integration Message Broker. The broker translates the XML message into formats that can be understood by different applications throughout the business, its partners, and suppliers.

Tip: The sample program for setting up Q replication and event publishing provides an example of a Web-based application that consumes the XML messages that the Q Capture program publishes. The sample demonstrates how to use XML publications in a business scenario.

Event publishing uses only the Q Capture program, not the Q Apply program. The Q Capture program captures changes that you specify when you create an object called an XML publication. These transactions or row-level changes are then sent to a queue. You specify which queue to use when you create an object called a publishing queue map.

Event publishing gives you the flexibility to use transactional data that is published in XML format for a wide variety of uses. If you want to replicate changes to a target using the Q Apply program, use Q replication rather than event publishing.

Objects for event publishing

In event publishing, the following objects exist between servers:

Publishing queue maps
: You must create at least one publishing queue map to transport data from the Q Capture program on each source server.

XML publications
: For each source table that you want to publish changes from, you must create at least one XML publication. The XML publication specifies the table, as well as the rows and columns that you want to publish. You can have multiple XML publications that specify the same source table. For example, if you wanted to publish changes from the EMPLOYEE table to Application A and Application B, and changes from the DEPARTMENT table to Application A, you would create three XML publications, with two publishing queue maps:

- XML publication EMPLOYEE0001 from the EMPLOYEE table to Application A using publishing queue map A.
- XML publication DEPARTMENT0001 from the DEPARTMENT table to Application A using publishing queue map A.
- XML publication EMPLOYEE0002 from the EMPLOYEE table to Application B using publishing queue map B.

Application A reads the messages that the Q Capture program sends to the send queue in publishing queue map A, and Application B reads the messages that the Q Capture program sends to the send queue in publishing queue map B.

Figure 2 shows event publishing for three source tables on the same server. You need three XML publications and at least one publishing queue map.

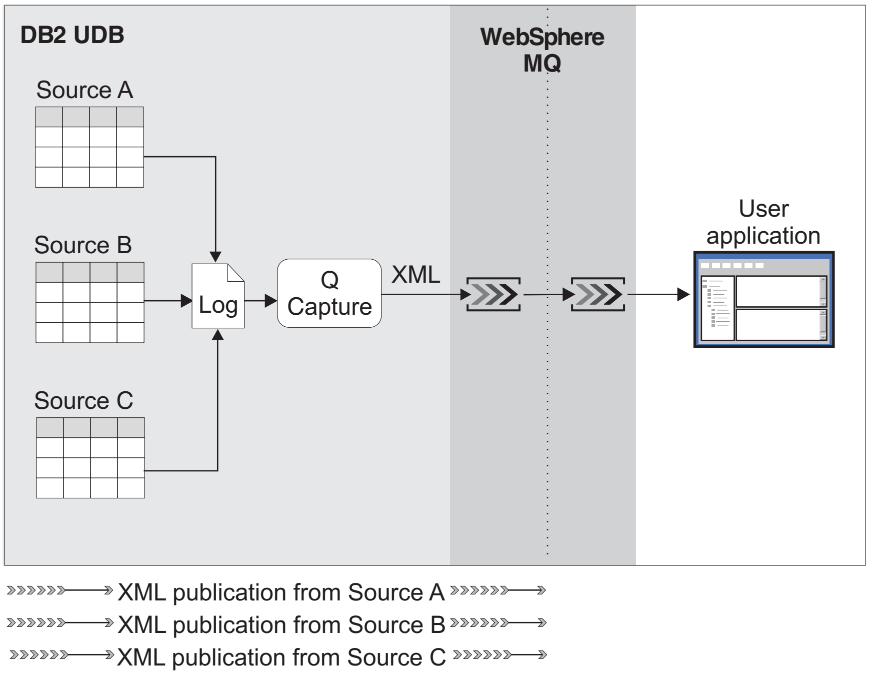

DB2 UDB
WebSphere MQ
Source A
Source B
Source C
Log
Q Capture
XML
User application
XML publication from Source A
XML publication from Source B
XML publication from Source C

Figure 2. Event publishing. Changes from source tables are published to one or more WebSphere MQ send queues in XML format so that a user application can retrieve and use those messages.

Controlling message content

You can choose whether the XML messages that are sent by the Q Capture program contain only a single row change from the source table, or an entire transaction. You make this choice when you create a publishing queue map. The Q Capture program will use the message format that you choose for all XML publications that specify this publishing queue map.

For example, you might specify that the messages published from the EMPLOYEE table to Application A contain individual row changes, and the messages from the DEPARTMENT table to Application A contain entire transactions. In that case, you would use the following configuration:

- XML publication EMPLOYEE0001 from the EMPLOYEE table to Application A using publishing queue map A (where publishing queue map A specifies individual row changes for messages).
- XML publication DEPARTMENT0001 from the DEPARTMENT table to Application A using publishing queue map C (where publishing queue map C specifies entire transactions for messages).

Related concepts:

- "Publishing queue maps" on page 12
- "Q Capture program" on page 15
- "Q replication" on page 3
- "Introduction to event publishing—Overview" on page 9
- "XML publications" on page 11

Related reference:

- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)" on page 465

XML publications

With event publishing, you create objects called *XML publications* to define how changes from a single source table are published in XML format to a WebSphere® MQ send queue. You can then have a user application of your choice retrieve and use those XML messages. An XML publication tells the Q Capture program which columns and rows in a source table you want to publish changes from. The transactions or row-level changes that are captured by the Q Capture program are sent as XML messages to a send queue that can be monitored by a user application of your choice. Figure 3 on page 12 shows how an XML publication connects a source table to a WebSphere MQ send queue.

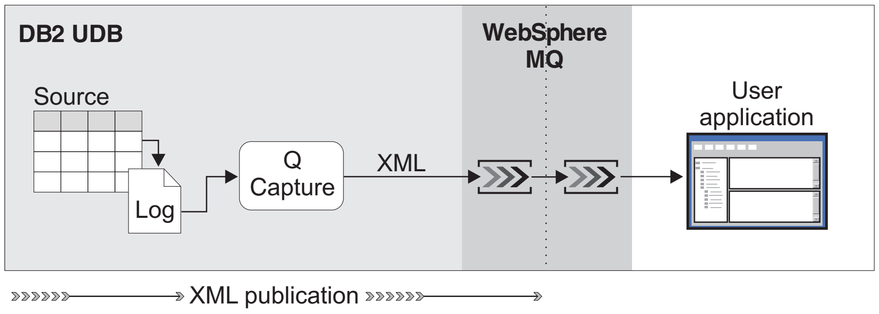


Figure 3. XML publication. Changes from a source table are published to a WebSphere MQ queue in XML format so that a user application can retrieve and use those messages.

For each XML publication, you specify the following information:

- The source table from which you want to publish changes.
- The columns and rows that you want published from the source table.
- The Q Capture program that publishes the source changes.
- The publishing queue map to use.
- Whether the XML publication, when it is first created, is activated automatically when you start or reinitialize the Q Capture program (this is the default behavior). You can also choose to create the XML publication in an inactive state, which requires you to activate the XML publication for the Q Capture program to begin capturing changes.

If you change an XML publication after you create it, you must reinitialize it so that the Q Capture program knows about those changes.

Important: XML publications are separate objects from Q subscriptions. Q subscriptions do not replicate data that is published in XML publications.

Related concepts:

- "Creating XML publications—Overview" on page 158
- "Q subscriptions" on page 5
- "Event publishing" on page 9

Related tasks:

- "Changing attributes of XML publications" on page 191
- "Deleting XML publications" on page 196

Publishing queue maps

In event publishing, a *publishing queue map* identifies the WebSphere® MQ queue that a Q Capture program uses to transport data and communicate with the user application that is receiving the XML messages. Each publishing queue map identifies one send queue, which is the WebSphere MQ queue where the Q Capture program sends source data and informational messages.

In event publishing, the Q Capture program publishes the source data in XML format to the send queue. The Q Apply program does not retrieve those messages. You can configure WebSphere MQ or an application of your choice to handle the XML messages that are on the send queue.

You can use a single publishing queue map to transport data for one or more XML publications. Also, a single Q Capture program can work with multiple publishing queue maps.

In addition to identifying the WebSphere MQ queue that the Q Capture program sends changes to, a publishing queue map also contains publishing attributes that involve the send queue. For each publishing queue map, you specify the following attributes that involve the send queue:

- The WebSphere MQ name of the send queue.
- The type of message content that the send queue transports. Message queues that are used for XML publications can transport either row-level changes or transactions.
- The maximum size of a message that the Q Capture program can put on this send queue. (This limit is independent of the WebSphere MQ Series maximum message length, but this limit must be less than or equal to the WebSphere MQ Series maximum message length.)
- How the Q Capture program responds if an error occurs at the WebSphere MQ queue.
- How often the Q Capture program sends messages to a send queue to tell the user application that the Q Capture program is still running when there are no changes to publish.

Related concepts:

- “Grouping publishing queue maps and XML publications” on page 155
- “Replication queue maps” on page 6
- “Introduction to event publishing—Overview” on page 9

Related tasks:

- “Creating publishing queue maps” on page 156

Chapter 3. Introduction to the programs that are used for replicating and publishing

Introduction to the programs that are used for replicating and publishing—Overview

The following topics provide an overview of the programs that are used for replicating and publishing data using Q replication and event publishing:

- "Q Capture program"
- "Q Apply program" on page 18
- "Schemas for the Q Apply and Q Capture programs" on page 20

Related concepts:

- "Introduction to Q replication—Overview" on page 3
- "Introduction to event publishing—Overview" on page 9

Q Capture program

The *Q Capture program* is a program that reads the DB2® recovery log for changes that occur in source tables, turns the changes into messages, and sends the messages over WebSphere® MQ queues where the messages are processed by a Q Apply program or user application. The Q Capture program lets you specify the data that you want to publish or replicate from the source table:

- You control which source changes are sent by specifying source tables, or even rows and columns within source tables.
- You control the speed and amount of data that flows over queues by setting the interval that the program commits messages, among other parameters.

You can define multiple Q Capture programs to independently capture data on a single source server. Each Q Capture program is identified on a server by a unique schema name.

The Q Capture program also gives you the option of sending data in a compact format to the Q Apply program, or in XML format to a user application. You can change the Q Capture program's operating parameters in several ways, including dynamically while it is running. The following sections provide details about the Q Capture program:

- "Capturing changes"
- "Filtering data" on page 16
- "Message formats" on page 16
- "Committing messages" on page 16
- "Changing parameters" on page 17
- "Communicating with Q Capture" on page 17
- "Restart messages" on page 18

Capturing changes

When a row changes in a source table, the Q Capture program reads the log record to see if the table is part of an active Q subscription or XML publication. If so, the

Q Capture program adds the row to the corresponding database transaction in memory. If a row change involves columns with large object (LOB) data, the Q Capture program copies the LOB data directly from the source table to the send queue.

If you define a search condition, the Q Capture program uses it to evaluate each row in memory. Rows that meet the search condition are assembled into messages when the transaction that they belong to is committed at the source database. The Q Capture program then puts messages on the send queues that you specify.

Filtering data

The Q Capture program allows you to speed up the process of replication and minimize network traffic by filtering rows with a search condition, limiting which column values are sent, or not propagating deletes.

Search condition
You can specify a search condition for each Q subscription or XML publication. The search condition is a restricted SQL WHERE clause. Row changes that do not meet the search condition are not included in messages.

Column options
For XML publications, you can limit which column values are added to messages by choosing from the following options. (For Q subscriptions, these column options are selected automatically for you based on other choices that you make.)

- **All changed rows:** By default, the Q Capture program sends a row only when a column that is part of a Q subscription or XML publication changes. You can choose to have the program send a row when any column changes.
- **Before values:** By default, the Q Capture program does not send before values of non-key columns that are updated. You can choose to have the program send before values of non-key columns.
- **Changed columns only:** By default, the Q Capture program sends only subscribed columns that have changed. You can choose to have the program also send subscribed columns that did not change.

Suppress deletes
By default, the Q Capture program captures deletes from the log and publishes or replicates them. You can, however, chose to have deletes suppressed.

Message formats

A Q Capture program can translate changes from a source table into two different message formats. Q replication uses a compact format that the Q Apply program can read. Event publishing uses messages in XML format. For XML messages only, you can choose whether messages contain a single row operation only, or all the row operations within a transaction.

Committing messages

After the Q Capture program puts messages on one or more send queues, it issues a commit call to the queue manager instructing it to make the messages on send queues available to the Q Apply program or user applications. You decide how often the Q Capture program commits messages. All of the DB2 transactions grouped within each commit are considered to be a WebSphere MQ transaction. Typically, each WebSphere MQ transaction contains several DB2 transactions.

You can adjust the time between commits by changing the value of the COMMIT_INTERVAL parameter. A shorter commit interval can lower end-to-end latency (the delay between the time transactions are committed at the source and target servers).

Changing parameters

The Q Capture operating parameters govern how much memory the program allocates for building transactions, the actions that it takes when starting, how often it deletes old data from its control tables, and other behaviors. You can change these parameters in three ways:

- By updating the control table where the Q Capture program reads its parameter values.
- By temporarily overriding the saved values when you start the program.
- By changing the parameters dynamically while the program is running.

If you use either of the latter two methods, your changes last only while the Q Capture program is running. When it stops and restarts, it uses the values that are saved in the control table, unless you override the values again.

Communicating with Q Capture

A Q Capture program responds to commands, SQL signals, and XML and compact messages.

You can use the Replication Center or system commands to control the following behaviors of the program:

- Start a Q Capture program and optionally change startup parameters
- Change parameter values while a Q Capture program is running
- Reinitialize a Q Capture program
- Reinitialize a send queue
- Stop a Q Capture program

The Replication Center issues SQL signals to communicate the following requests to a Q Capture program (you can also manually insert SQL signals to perform these tasks):

- Request that the Q Capture program activate or deactivate a Q subscription or XML publication
- Report that a target table is loaded
- Tell the Q Capture program to reinitialize a Q subscription or XML publication

You can manually insert SQL signals to perform the following tasks:

- Add a column to an active unidirectional Q subscription or XML publication
- Execute the error action that is defined for a send queue
- Stop the Q Capture program
- Ignore a transaction

The Q Apply program and user applications communicate with the Q Capture program by sending compact and XML messages respectively.

Restart messages

The Q Capture program uses a local WebSphere MQ queue to store restart information. The restart queue contains a single message that tells the Q Capture program where to start reading in the DB2 recovery log when the Q Capture program restarts.

Each time that the Q Capture program reaches its commit interval, it checks to see whether or not it needs to update its restart information. If so, the Q Capture program replaces the message on the restart queue with a new message that contains relevant restart information including, among other things, the earliest point in the log at which it needs to start processing log records upon restart. If you use the cold start option, the Q Capture program replaces the restart message with a message that indicates for the program to start processing log records at the current point in the log.

Related concepts:

- "Introduction to the programs that are used for replicating and publishing—Overview" on page 15
- "Operating a Q Capture program—Overview" on page 201
- "Changing the Q Capture parameters—Overview" on page 217
- "Memory for LOB data types for Q replication and event publishing" on page 31
- "Q Apply program" on page 18
- "Introduction to Q replication—Overview" on page 3
- "Schemas for the Q Apply and Q Capture programs" on page 20
- "Introduction to event publishing—Overview" on page 9

Q Apply program

The *Q Apply program* takes messages from WebSphere® MQ queues, rebuilds the transactions that the messages contain, and applies the transactions to target tables or stored procedures. The Q Apply program is designed to keep up with rapid changes at multiple sources by applying transactions to multiple targets at the same time. The program can apply changes in parallel while maintaining referential integrity between related target tables. You can define multiple Q Apply programs to independently apply data to a single target server. Each Q Apply program is identified on a server by a unique schema name.

To allow the Q Apply program to track what changes it has already applied to the target, each target table must have some mechanism to enforce that rows are unique. This uniqueness can come from a primary key, unique constraint, or unique index at the target.

You can specify whether the program applies transactions in parallel, change its operating behavior in several ways, and set options for how the program detects and handles conflicts.

The following sections provide more details about the Q Apply program:

- "Browser and agent threads" on page 19
- "Applying transactions in parallel" on page 19
- "Changing parameters" on page 19

Browser and agent threads

A Q Apply program can handle transactions from multiple receive queues. For each receive queue, the Q Apply program launches a single thread known as a *browser thread*.

The browser thread gets transaction messages from the receive queue and keeps track of dependencies between transactions. The browser thread also tracks which transactions it has already applied. To maintain referential integrity between dependent transactions, each replication queue map, which identifies the receive queue, must be processed by a single browser thread, and Q subscriptions that involve dependent tables must use the same replication queue map. Therefore, two Q Apply programs cannot get transactions from the same replication queue map.

Each browser thread launches one or more agent threads. The agent thread takes transaction messages from the browser thread, rebuilds the SQL statements for all row changes, applies the changes to targets, and issues the commit statement for transactions.

Applying transactions in parallel

WebSphere MQ ensures that transaction messages arrive in a receive queue in the same order that they were committed at the source. By default, the Q Apply program applies transactions in parallel using multiple agent threads. Such parallelism is important when many changes are replicated from the source server. You can set the number of agent threads to one if you want the Q Apply program to apply transactions in their strict arrival order.

The Q Apply program takes special precautions when it is applying transactions in parallel, because rows are not always applied in the same order in which the changes occurred at the source.

For example, assume that a new department is created by inserting a row into the DEPARTMENTS table at the source. Following this, an employee is added to the new department by inserting a row into the EMPLOYEES table at the source. A referential integrity constraint at the source requires that any row inserted into the EMPLOYEES table must have a matching record in the DEPARTMENTS table. The same parent-child dependency exists between the replicated copies of these two tables.

When messages that contain these source transactions arrive on the receive queue, the browser thread detects the dependency between these two transactions. If the Q Apply program is using multiple agents to apply transactions in parallel, the browser thread delays the insert of the employee record into the target table until the matching department row is applied. Meanwhile, agent threads continue to apply other transactions in parallel.

Changing parameters

The Q Apply operating parameters let you set how often the program saves or prunes performance data, where it stores its diagnostic log, and how often it retries to apply changes to targets after deadlocks or lock time-outs. You can change these parameters in three ways:

- By updating the control table where the Q Apply program reads its parameter values.
- By temporarily overriding the saved values when you start the program.
- By changing the parameters dynamically while the program is running.

If you use either of the latter two methods, your changes will last only while the Q Apply program is running. When it stops and restarts, it uses the values saved in the control table, unless you override the values again.

Two important operating parameters, the number of apply agents and the memory limit, determine a Q Apply program's behavior for each receive queue that it works with. You specify the value of these parameters when you create a replication queue map. If the Q Apply program is currently getting messages from a receive queue, you can change the number of Q Apply agents and memory limit for that receive queue without stopping the Q Apply program.

Related concepts:

- "Introduction to the programs that are used for replicating and publishing—Overview" on page 15
- "Operating a Q Apply program—Overview" on page 227
- "Changing the Q Apply parameters—Overview" on page 237
- "Q Capture program" on page 15
- "Introduction to Q replication—Overview" on page 3
- "Schemas for the Q Apply and Q Capture programs" on page 20
- "Introduction to event publishing—Overview" on page 9

Schemas for the Q Apply and Q Capture programs

A Q Capture or Q Apply *schema* is the name given to the set of control tables used by a Q Capture or Q Apply program. Q replication uses schemas to identify the Q Capture or Q Apply program that uses a specific set of control tables. For example, if you create a set of Q Capture control tables in the schema SCHEMA_GREEN, the Q Capture program that uses that set of control tables is identified by that schema.

Before a server appears in the Replication Center object tree, it must contain at least one Q Capture or Q Apply schema. You can create more than one schema on a server, though. You might want to create multiple Q Capture schemas for the following reasons:

Increasing parallelism
Use multiple Q Capture programs to parallelize traffic. You can set up multiple Q Capture schemas so that horizontal subsets of rows from a table, tables, or groups of tables are replicated using independent Q Capture programs and log readers. Each program can use a separate set of send queues, or even a separate queue manager, to speed the transfer of change messages. This configuration can improve performance and achieve higher throughput, for example, with very large source tables or on a large sysplex. The trade-offs are the additional CPU cost of running multiple log readers and the additional connections to DB2® UDB that would be required.

Meeting different replication requirements
Create multiple Q Capture schemas to handle different workload characteristics to direct the flow of source changes toward different uses. With independent Q Capture schemas, you can tune the Q Capture operational parameters independently for each workload. For example, two Q Capture programs with different schemas can read from the same source table: one publishing the changes in an XML message format and the other replicating the changes to the Q Apply program. For a Q Capture program

for low latency replication, you can set the commit interval to 500 milliseconds. And for a Q Capture program for publishing, you can set the commit interval to 5 seconds. You could achieve this same scenario by defining a single Q Capture program, as identified by a single Q Capture schema, that processes both XML publications as well as Q subscriptions, but in this case the operational parameters that Q Capture would utilize would be the same.

Although you can create multiple Q Apply schemas on a server, in most cases one Q Apply program per server is sufficient, even for replication configurations with a high volume of transactions being replicated to a large number of tables.

Part 2. Preparing for Q replication and event publishing

This part of the book contains the following chapters:

- Chapter 4, "Planning memory and storage requirements," on page 25 describes the amount of memory and storage that is required for Q replication and event publishing and helps you plan the appropriate amount of memory and storage to allocate.
- Chapter 5, "Data conversion," on page 35 describes how to plan for code pages in your configuration.
- Chapter 6, "Setting up user IDs and passwords," on page 37 describes how to set up user IDs and passwords and the authority requirements for the various programs.
- Chapter 7, "Setting up WebSphere MQ for Q replication and event publishing," on page 43 describes how to set up the WebSphere MQ environment and the necessary objects for WebSphere MQ.
- Chapter 8, "Configuring servers for Q replication and event publishing," on page 67 describes how to set up your servers, ensure that the servers can connect to each other, and create the necessary DB2 UDB objects.

Chapter 4. Planning memory and storage requirements

Planning memory and storage requirements for Q replication and event publishing

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. 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.

Procedure:

To determine how much memory and storage you need:

- Plan how much memory is required for Q replication and event publishing. See "Planning memory requirements for Q replication and event publishing" for details.
- Plan how much storage space is required for Q replication and event publishing. See "Planning storage requirements for Q replication and event publishing" on page 31 for details.

Planning memory requirements for Q replication and event publishing

Planning memory requirements for Q replication and event publishing

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. If the replication programs run out of memory and cannot get storage, they will stop. The programs have a memory limit parameter that you can use to control how much memory the programs consume.

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.

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.

Procedure:

To determine your memory requirements, you need to plan for the following things:

- How much memory to anticipate for the Q Capture program. See "Memory used by the Q Capture program" for details.
- How much memory to anticipate for the Q Apply program. See "Memory used by the Q Apply program" on page 29 for details.
- How much memory to anticipate for the Replication Alert Monitor. See "Memory used by the Replication Alert Monitor" on page 30 for details.
- How much memory to anticipate for LOB columns that you want to replicate or publish. See "Memory for LOB data types for Q replication and event publishing" on page 31 for details.

Related tasks:

- "Planning storage requirements for Q replication and event publishing" on page 31

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® UDB 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 XML publications while the Q Capture program is running. Each Q subscription or XML publication consumes a maximum of 1000 bytes of memory.

Memory for reconstructing transactions
When the Q Capture program reads the DB2 UDB 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 of the buffer on Linux, UNIX®, and Windows® operating systems is 200 1–KB pages. The default size on z/OS™ operating systems is 66 1–KB pages, and the memory is ECSA (extended common service area) storage.

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 UDB 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 32 MB. The larger the **memory_limit**, the less likely the Q Capture program will write transactions to spill files.

Use Table 1 on page 28 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**:

```
txr/1000 * ci = 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:

```
txci * mtxs = 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

Transaction rate (txr)	Maximum transaction size (mtxs)	Commit_interval in milliseconds (ci)	Transactions per commit_interval (txci)	Memory_limit in bytes (ml)
730	1 340	1 000	730	978 200
1 033	12 000	3 000	3 099	37 188 000
5 000	2 340	500	2 500	5 850 000
12 000	7 000	5000	60 000	420 000 000

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:

```
ml + (mas * 1000) + 2000000 = 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

Memory_limit in bytes (ml)	Maximum active subscriptions (mas)	Memory quota in bytes (q)
37 188 000	500	39 688 000
420 000 000	300	422 300 000

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**. See the online help for details.
- 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. If there is not enough storage, the Q Capture program will terminate.

Related concepts:

- "Changing the Q Capture parameters—Overview" on page 217
- "Memory used by the Q Apply program" on page 29

Related tasks:

- "Planning memory requirements for Q replication and event publishing" on page 25
- "Planning storage requirements for Q replication and event publishing" on page 31

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 Q replication. Consider adding more frequent commits to such transactions.

You can set the **memory_limit** parameter for each receive queue for the Q Apply program. Set the **memory_limit** value when you start the Q Apply program or while it is running. This value controls how much memory the Q Apply program can use to buffer transactions that it receives from a given receive queue until the Q Apply program applies them to the target.

Use Table 3 on page 30 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:

```
(mtxs + 44000) * (a + 1) = ml
```

Where:

mtxs
: Maximum transaction size, in bytes

a Number of agents

ml Memory limit for a receive queue, in bytes

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

Maximum transaction size (`mtxs`)	Number of agents (`a`)	Memory_limit (`ml`)
1 340	128	5 848 860
12 000	16	952 000

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:

```
sml + (mas * 1000) + 500000 = q
```

`sml` Sum of memory limit values

`mas` Maximum active subscriptions

`q` Memory quota in bytes

Table 4. Approximate memory quota for a Q Apply program

Sum of memory_limit values for all receive queues (`sml`)	Maximum active subscriptions (`mas`)	Memory quota (`q`)
5 848 860 + 952 000 = 6 800 860	100	7 400 860

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 storage, diagnostic information cannot be saved. The Q Apply program keeps running.

Related concepts:

- “Changing the Q Apply parameters—Overview” on page 237
- “Memory used by the Q Capture program” on page 26
- “Q Apply program” on page 18

Related tasks:

- “Planning memory requirements for Q replication and event publishing” on page 25
- “Planning storage requirements for Q replication and event publishing” on page 31

Memory used by the Replication Alert Monitor

Very little memory is used by the Replication Alert Monitor

Recommendation: Do not set a memory quota for the Replication Alert Monitor; however, if you need to set one, set it to 3 MB.

Related tasks:

- "Planning memory requirements for Q replication and event publishing" on page 25

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 the size of the LOB value exceeds the LOB message buffer size for the Q Capture program, then the LOB message is divided into two or more smaller messages. The LOB message buffer size can be up to the maximum message length that is defined for a send queue. (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.

Related concepts:

- "Queue depth considerations for large object (LOB) values" on page 64
- "Considerations for large object (LOB) data types for Q replication and event publishing" on page 172
- "WebSphere MQ message size" on page 60

Related tasks:

- "Planning memory requirements for Q replication and event publishing" on page 25

Planning storage requirements for Q replication and event publishing

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.

Procedure:

To determine your storage requirements, you need to plan for the following things:

- "Storage requirements for DB2 UDB logs for Q replication and event publishing" on page 32
- "Storage for diagnostic files for Q replication and event publishing" on page 32

- “Storage requirements for when the Q Capture program exceeds its memory limit” on page 33
- “Storage requirements for traces for Q replication and event publishing” on page 34

Related concepts:

- “Storage requirements for WebSphere MQ for Q replication and event publishing” on page 61

Related tasks:

- “Planning memory requirements for Q replication and event publishing” on page 25

Storage requirements for DB2 UDB logs for Q replication and event publishing

Q replication requires storage in the DB2® UDB logs for the changes that occur in the source servers and the target servers.

DB2 UDB log requirements at the source server

At the source server, you need to allocate enough log space for the changes that occur in the source tables. DB2 UDB 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. Unlike SQL replication, which stages captured data to staging tables, Q replication log requirements are limited to only 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.

DB2 UDB log requirements at the target server

At the 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.

Related tasks:

- “Planning storage requirements for Q replication and event publishing” on page 31

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 re-creates it.

Related concepts:

- "Changing the Q Capture parameters—Overview" on page 217

Related tasks:

- "Planning storage requirements for Q replication and event publishing" on page 31

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 biggest transaction is not necessarily the one that exceeded the memory limit.

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® UDB 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.

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

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.

Related concepts:
- "Memory used by the Q Capture program" on page 26

Related tasks:
- "Planning storage requirements for Q replication and event publishing" on page 31

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

Related tasks:
- "Planning storage requirements for Q replication and event publishing" on page 31

Related reference:
- "asntrc: Operating the replication trace facility" on page 348

Chapter 5. Data conversion

Data conversion for Q replication and event publishing—Overview

In Q replication and event publishing, data passes between various servers and programs, and sometimes the data must be converted. The following topics explain how Q replication and event publishing handle data conversion:

- "Data conversion for event publishing" on page 36
- "Data conversion for event publishing" on page 36

Data conversion 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. In Q replication, 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:

- 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 the Q Capture and Q Apply programs pass administrative messages back and forth, the data format of those messages might need to be converted. You must ensure that the Q Capture and Q Apply programs that process Q subscriptions together have compatible code pages.

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.

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.

Recommendation for z/OS™: 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 *Administration Guide: Planning* to determine if the code pages that you are using are compatible.

Related concepts:

- "Data conversion for event publishing" on page 36

Related reference:

- "Supported territory codes and code pages" in the *Administration Guide: Planning*

Data conversion for event publishing

In event publishing, the data is converted from the source server's code page to an XML message as UTF-8 (code page 1208), which is a standard XML encoding scheme.

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 code page of the Q Capture program.

Related concepts:

- "Data conversion for Q replication" on page 35

Chapter 6. Setting up user IDs and passwords

Setting up user IDs and passwords—Overview

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

The following topics explain the authorities and privileges that you need, and how you can store necessary user IDs and passwords in an encrypted password file that the replication and publishing programs can share.

- “Authorization requirements for Q replication and event publishing—Overview”
- “Storing user IDs and passwords for remote servers” on page 41

Authorization requirements for Q replication and event publishing

Authorization requirements for Q replication and event publishing—Overview

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. The authorities and privileges are explained in the following topics:

- “Authorization requirements for the Q Capture program”
- “Authorization requirements for the Q Apply program” on page 38
- “Authorization requirements for the Replication Alert Monitor” on page 39
- “Authorization requirements to administer Q replication and event publishing” on page 40

Related concepts:

- “Setting up user IDs and passwords—Overview” on page 37

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. The following list summarizes the DB2 UDB requirements and operating system requirements by platform.

Requirements for Linux, UNIX®, and Windows®

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.

In a massively parallel processing (MPP) configuration, the user IDs must be able to connect to database partitions and read the password file.

Requirements for z/OS™

User IDs that run a Q Capture program must:

- Be registered with access to UNIX System Services (USS).
- Be APF-authorized.
- Be defined to use z/OS UNIX or OS/390® UNIX (must have an OMVS segment). An OMVS segment is the portion of a RACF® profile that contains OMVS logon information.
- Have read and write access permission to either the /tmp directory or the directory that is specified by the TMPDIR environment variable.
- Have SYSADM authority, or the following specific privileges:
  - SELECT, UPDATE, INSERT, and DELETE for all control tables on the Q Capture server
  - SELECT for the DB2 catalog (SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, SYSIBM.SYSTABCONST, SYSIBM.SYSKEYCOLUSE, SYSIBM.SYSINDEXES, and SYSIBM.SYSKEYS)
  - TRACE
  - MONITOR1 and MONITOR2
  - EXECUTE for the Q Capture program packages
- Have WRITE access to the directory that is specified by the **capture_path** parameter (USS) or high-level qualifier (z/OS).

Related concepts:
- "Authorization requirements for Q replication and event publishing—Overview" on page 37

Related tasks:
- "Storing user IDs and passwords for remote servers" on page 41

Related reference:
- "Descriptions of Q Capture parameters" on page 206

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. The following list summarizes the DB2 UDB requirements and operating system requirements by platform.

Requirements for Linux, UNIX®, Windows®

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.

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 z/OS™
User IDs that run a Q Apply program must:
- Be registered with access to UNIX System Services (USS).
- Be APF-authorized in order for the Q Apply program to use the Automatic Restart Manager (ARM) or produce SVCDUMPs.
- Be defined to use z/OS UNIX or OS/390® UNIX (must have an OMVS segment). An OMVS segment is the portion of a RACF® profile that contains OMVS logon information.
- Have read and write access permission to either the /tmp directory or the directory that is specified by the TMPDIR environment variable.
- Have SYSADM authority, or the following specific privileges:
  - SELECT, UPDATE, INSERT, and DELETE for all control tables on the Q Apply server
  - SELECT on the source tables if the Q Apply program will be used to load target tables
  - SELECT for the DB2 catalog (SYSIBM.SYSRELS, SYSIBM.SYSTABLES and SYSIBM.SYSDUMMY1)
  - EXECUTE for the Q Apply program packages
- Have WRITE access to the directory that is specified by the **apply_path** parameter (USS) or high-level qualifier (z/OS).

Related concepts:
- "Authorization requirements for Q replication and event publishing—Overview" on page 37
- "Utilities used for automatic load option for Q replication" on page 145

Related reference:
- "Descriptions of Q Apply parameters" on page 231

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 (Linux, UNIX®, and Windows® only)

Related concepts:
- "Authorization requirements for Q replication and event publishing—Overview" on page 37
- "The Replication Alert Monitor" on page 255

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 XML 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**. See the online help for details.

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).

Related concepts:
- "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" on page 72
- "Authorization requirements for Q replication and event publishing—Overview" on page 37

Related tasks:

- “Creating control tables for the Q Capture and Q Apply programs” on page 76
- “Running SQL scripts and operational commands from the Replication Center—Overview” on page 173

Storing user IDs and passwords for remote servers

On Linux, UNIX, and Windows operating systems, the following Q replication and event publishing programs require that you create a password file to store user IDs and passwords for connecting to remote servers:

- 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.
- Q Capture programs that run on DB2 UDB Enterprise Server Edition require a password file to connect to multiple partition databases.
- The Replication Alert Monitor requires a password file to connect to any Q Capture server or Q Apply server that you want to monitor.
- The Q Replication Analyzer requires a password file to connect to any Q Capture server or Q Apply server that you want to analyze.

Procedure:

To store user IDs and passwords for remote servers, create an encrypted password file for Q 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:

Q Apply program
: **apply_path**

Q Capture program
: **capture_path**

Replication Alert Monitor
: **monitor_path**

Q Replication Analyzer
: The analyzer looks for the password file in the directory where the program was invoked, or from the location that is specified by the **password_filepath** parameter.

If the Q Apply program, Replication Alert Monitor, and Replication Analyzer are running on the same system, they can share the same password file. If you want the three programs to share a password file, specify the same path and file name for all three programs, or use symbolic links to share the same password file in the different directories.

Note: 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**. See the online help for details.

Related concepts:

- “Setting up user IDs and passwords—Overview” on page 37

- “Utilities used for automatic load option for Q replication” on page 145

Related reference:

- “asnqanalyze: Operating the Q Replication Analyzer” on page 336

Chapter 7. Setting up WebSphere MQ for Q replication and event publishing

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.

This section describes the WebSphere MQ objects that are required for the Q Capture and Q Apply programs to operate, provides checklists of which objects are needed for different replication or publishing configurations, and describes required settings for queues, queue managers, and channel objects. It also discusses memory, storage, and authorization requirements for WebSphere MQ.

This section contains the following topics:
- "WebSphere MQ objects required for Q replication and event publishing—Overview"
- "WebSphere MQ message size" on page 60
- "Storage requirements for WebSphere MQ for Q replication and event publishing" on page 61
- "Storage requirements for spill queues for the Q Apply program" on page 62
- "Connectivity and authorization requirements for WebSphere MQ objects" on page 62
- "Required settings for WebSphere MQ objects" on page 56
- "Queue depth considerations for large object (LOB) values" on page 64
- "Queue manager clustering in Q replication and event publishing" on page 64

WebSphere MQ objects required for Q replication and event publishing

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.

The following topics describe the WebSphere MQ objects that are used in Q replication and event publishing. They also provide checklists of the objects that are required for different replication and publishing scenarios, and they provide required settings for the objects:
- "Required WebSphere MQ objects at a glance" on page 44
- "WebSphere MQ objects required for unidirectional replication on the same system" on page 45
- "WebSphere MQ objects required for unidirectional replication (remote)" on page 47
- "WebSphere MQ objects required for event publishing" on page 48

- “WebSphere MQ objects required for bidirectional or peer-to-peer replication (two remote servers)” on page 50
- “WebSphere MQ objects required for peer-to-peer replication (three or more remote servers)” on page 53

For detailed information about creating WebSphere MQ objects, see the *WebSphere MQ System Administration Guide* for your operating system, *WebSphere MQ Intercommunication*, and *WebSphere MQ Script (MQSC) Command Reference*.

Related concepts:

- “Setting up WebSphere MQ for Q replication and event publishing—Overview” on page 43

Required WebSphere MQ objects at a glance

Here is a quick summary of the WebSphere® MQ objects that are required by the Q Capture and Q Apply programs, followed by a summary of how you define the objects for these programs.

Queue manager
A program that manages queues for Q Capture programs, Q Apply programs, and user applications. One queue manager is required on each system.

Send queue
A queue that directs data messages from a Q Capture program to a Q Apply program or user application. In remote configurations, this is the local definition on the source system of the receive queue on the target system. Each send queue should be used by only one Q Capture program.

Receive queue
A queue that receives data and informational messages from a Q Capture program to a Q Apply program or user application. This is a local queue on the target system.

Administration queue (for Q Capture)
A queue that receives control messages from a Q Apply program or a user application to the Q Capture program. This is a local queue on the source system. Each administration queue should be read by only one Q Capture program.

Administration queue (for Q Apply)
A queue that directs control messages from the Q Apply program or user application to a Q Capture program. In remote configurations, this is the local definition on the target system of the Q Capture administration queue on the source system.

Restart queue
A queue that holds a single message that tells the Q Capture program where to start reading in the DB2® recovery log after a restart. This is a local queue on the source system. Each Q Capture program must have its own restart queue.

Spill queue
A model queue that you define on the target system to hold transaction messages from a Q Capture program while a target table is being loaded. The Q Apply program creates these dynamic queues during the loading process based on your model queue definition, and then deletes them. The spill queue must have a specific name, IBMQREP.SPILL.MODELQ.

How you define WebSphere MQ objects for Q replication and event publishing

After you create the WebSphere MQ objects that are required for Q replication or event publishing, you need to tell the Q Capture and Q Apply programs about the objects.

You define these objects when you create control tables, and when you create replication queue maps or publishing queue maps.

When you create control tables
: For the Q Capture control tables, you provide the name of a queue manager on the system where the Q Capture program runs, and the name of a local administration queue and local restart queue. For the Q Apply control tables, you provide the name of a queue manager on the system where the Q Apply program runs.

When you create replication queue maps
: You provide the name of a send queue on the system where the Q Capture program runs, and a receive queue and administration queue on the system where the Q Apply program runs.

When you create publishing queue maps
: You provide the name of a send queue on the system where the Q Capture program runs.

You do not need to define WebSphere MQ channel objects such as transmission queues and channels for the Q replication or event publishing programs. You only need to define these objects within the source and target queue managers.

Related concepts:
- "Required settings for WebSphere MQ objects" on page 56
- "Publishing queue maps" on page 12
- "Replication queue maps" on page 6
- "Setting up WebSphere MQ for Q replication and event publishing—Overview" on page 43

Related tasks:
- "Creating control tables for the Q Capture and Q Apply programs" on page 76
- "Creating publishing queue maps" on page 156
- "Creating replication queue maps" on page 84

WebSphere MQ objects required for unidirectional replication on the 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 checklist 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 4 shows the WebSphere MQ objects that are required for unidirectional Q replication on the same system.

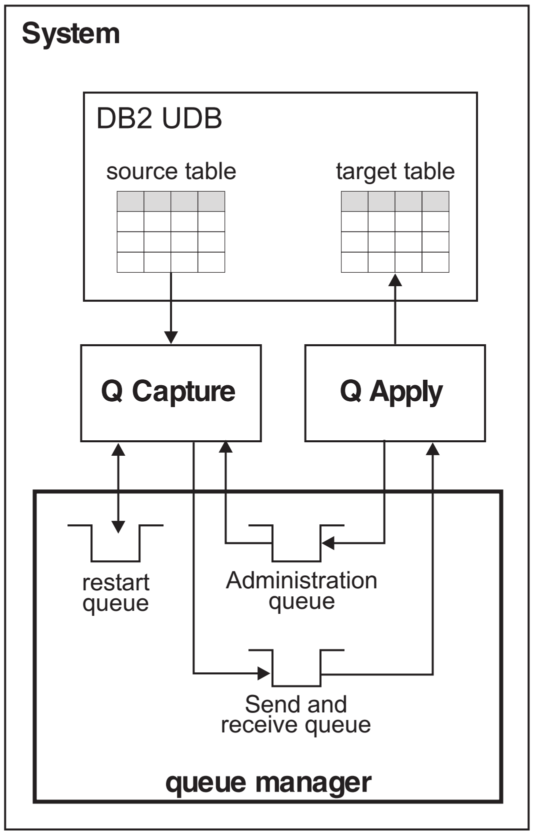


Figure 4. WebSphere MQ objects that are required for unidirectional Q replication on the same system. When the Q Capture program and Q Apply program run on the same system, only one queue manager is required. One local queue serves as both send queue and receive queue, and another local queue serves 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.

Related concepts:

- “WebSphere MQ objects required for Q replication and event publishing—Overview” on page 43
- “Required settings for WebSphere MQ objects” on page 56
- “Setting up replication from sources to targets (unidirectional)—Overview” on page 81

- “Required WebSphere MQ objects at a glance” on page 44

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 checklists 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 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).

___ 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 5 on page 48 shows the WebSphere® MQ objects that are required for unidirectional Q replication between remote servers.

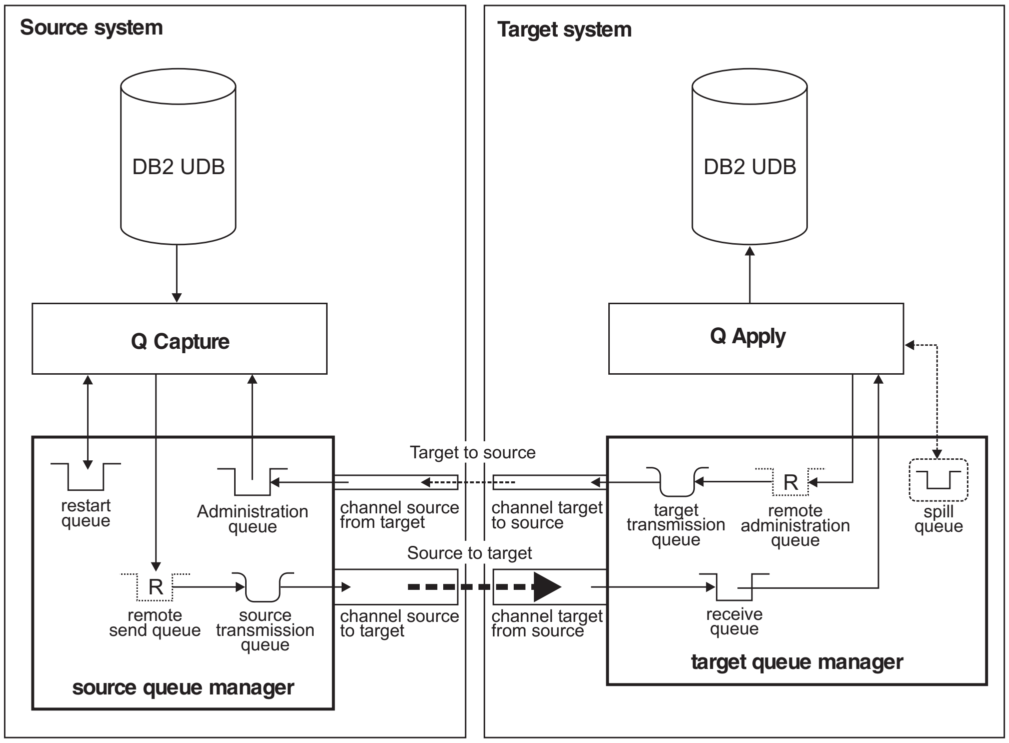


Figure 5. WebSphere MQ objects that are required for unidirectional Q replication between remote servers. Objects that are required for the Q Capture program are defined within the queue manager on the source system. Objects that are required for the Q Apply program are defined within the queue manager 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.

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

Related concepts:

- "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43
- "Required settings for WebSphere MQ objects" on page 56
- "Setting up replication from sources to targets (unidirectional)—Overview" on page 81
- "Required WebSphere MQ objects at a glance" on page 44

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 checklists 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 6 on page 50 shows the WebSphere® MQ objects that are required for event publishing between remote servers.

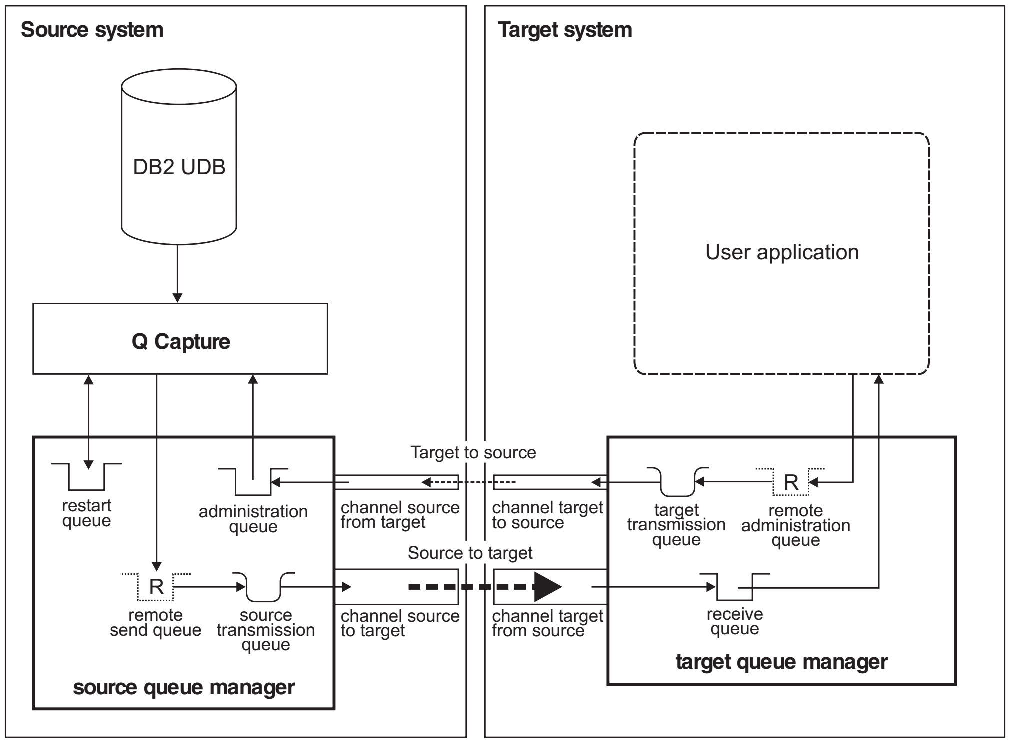


Figure 6. WebSphere MQ objects that are required for event publishing between remote servers. Objects that are required for the Q Capture program are defined within the queue manager on the source system. Objects that are required for the user application are defined within the queue manager on the target system. Two channel objects are required to create a transmission path between the source and target systems for data messages and informational messages from the Q Capture program. Two channel objects are also required to create a transmission path from the target system to the source system for control messages from the user application

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

Related concepts:

- "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43
- "Required settings for WebSphere MQ objects" on page 56
- "Setting up publishing from sources (event publishing)—Overview" on page 155
- "Required WebSphere MQ objects at a glance" on page 44

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

To replicate transactions in both directions between two servers, you have a paired Q Capture program and Q Apply program at each server. 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 would create the WebSphere MQ objects that link the Q Capture program at Server A with the Q Apply program at Server B. You would 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 would each connect to a single queue manager on the systems where they run.

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

Non-channel objects at System A

___ A queue manager.

___ A remote queue 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 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 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 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 Q Capture at System A to Q Apply at 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 Q Apply at System B to Q Capture at 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.

Channel objects from Q Capture at System B to Q Apply at 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.

Channel from Q Apply at System A to Q Capture at 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.

Figure 7 shows the WebSphere MQ objects that are required for bidirectional or peer-to-peer Q replication between two remote servers.

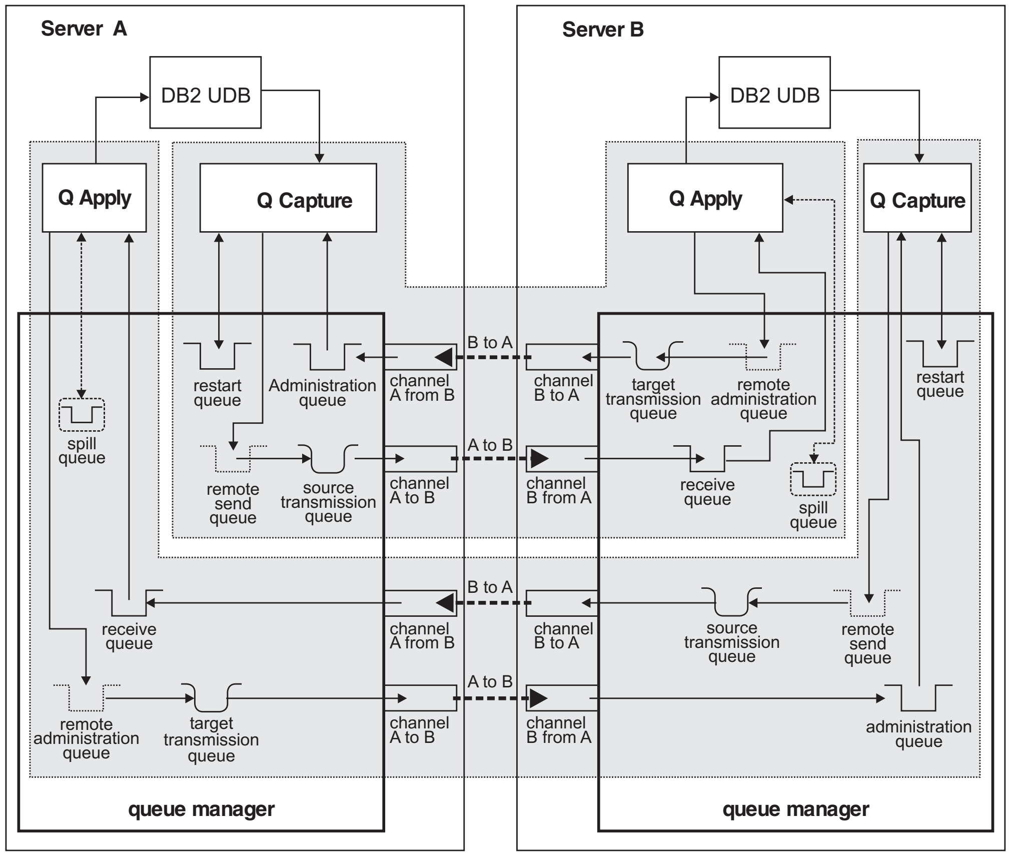


Figure 7. WebSphere MQ objects required for bidirectional or peer-to-peer Q replication between two remote servers. You must create two sets of the same WebSphere MQ objects that are required to connect a Q Capture program and a Q Apply program in unidirectional Q replication. One set of objects handles 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.

Related concepts:

- "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43

- “Required settings for WebSphere MQ objects” on page 56
- “Setting up replication from sources to targets (multidirectional)—Overview” on page 117
- “Required WebSphere MQ objects at a glance” on page 44

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 how many logical tables are being replicated, and on the number of servers in the group.

For example, if one logical table is replicated between three remote servers, the Q Capture program at Server A would need 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 would need two receive queues, one for transactions that are coming from Server B and one for transactions that are coming from Server C.

If two logical tables are being replicated, you must double the number of send queues and receive queues.

The following checklists show the objects that are required at each system in peer-to-peer replication with three or more servers. Because the queue manager is not part of the replication server but runs on the same system, the objects are grouped by system:

Non-channel objects at each system

___ One queue manager.

___ One remote send queue for each outgoing channel, and one for each logical table that is being replicated.

___ 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, and one for each logical table that is being replicated.

___ 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 8 on page 55 shows the WebSphere® MQ objects that are required at one server that is involved in peer-to-peer between three remote servers, with one logical table being replicated.

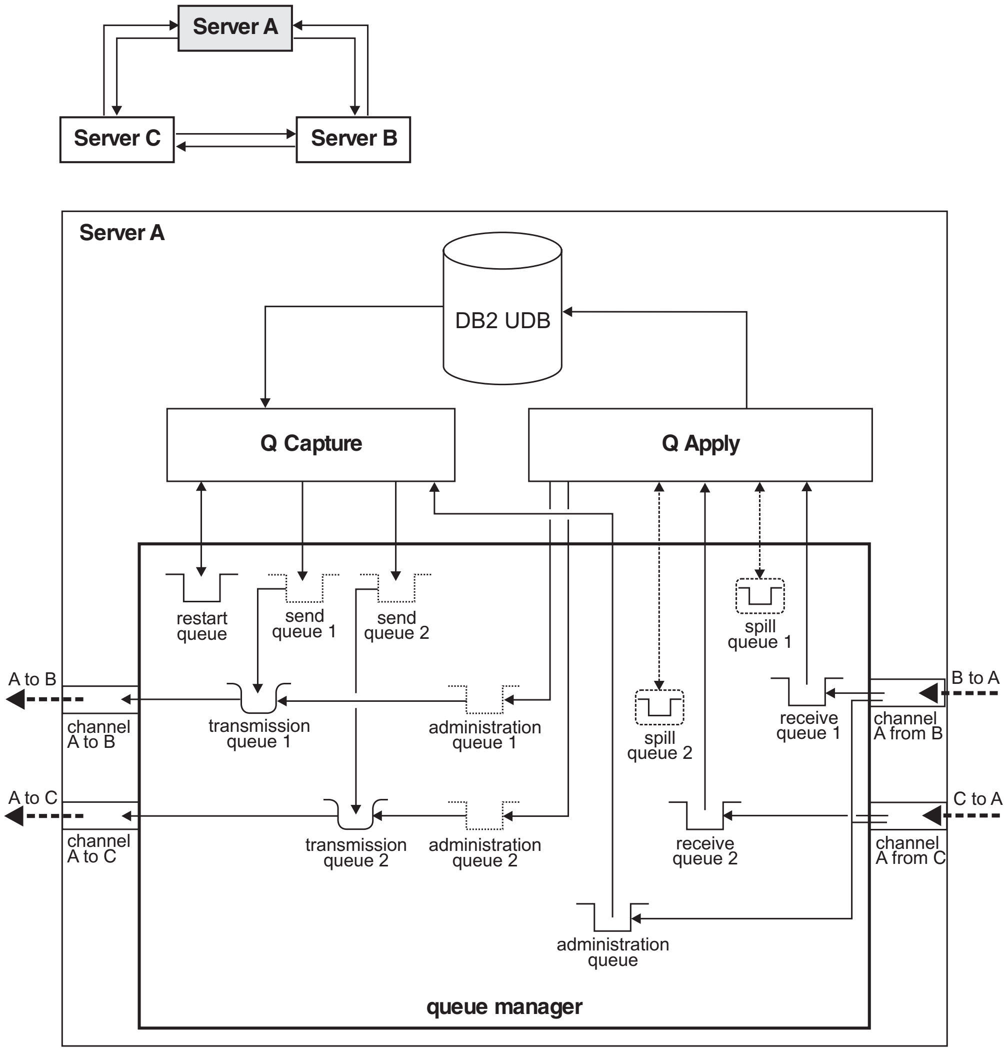


Figure 8. WebSphere MQ objects that are required at one server that is involved in peer-to-peer replication with two other remote servers. At each system, you create one queue manager. The Q Capture program requires one administration queue and one restart queue. You create one remote send queue for each outgoing channel (assuming a single logical table is being replicated). The Q Apply program requires a remote administration queue for each outgoing channel, and a local receive queue for each incoming channel (assuming one logical table is being replicated). You create one outgoing channel and one incoming channel for each additional server in the group.

Related concepts:

- "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43
- "Required settings for WebSphere MQ objects" on page 56
- "Setting up replication from sources to targets (multidirectional)—Overview" on page 117
- "Required WebSphere MQ objects at a glance" on page 44

Required settings for WebSphere MQ objects

This topic provides required settings for WebSphere® MQ objects used in various Q replication and event publishing scenarios. It has the following sections:

- "WebSphere MQ objects at the source"
- "WebSphere MQ objects at the target" on page 57
- "WebSphere MQ channel objects" on page 58

For more detail about configuring WebSphere MQ objects, see the *WebSphere MQ System Administration Guide* for your operating system, *WebSphere MQ Intercommunication*, and *WebSphere MQ Script (MQSC) Command Reference*.

WebSphere MQ objects at the source

Table 5 provides required values for selected parameters for WebSphere MQ objects at the source.

Table 5. Required parameter values for WebSphere MQ objects at the source

Object name	Required settings
Queue manager	**MAXMSGL** (The maximum size of messages allowed on queues for this queue manager.) This value must be at least as large as the MAX_MESSAGE_SIZE that you define when you create a replication queue map or publishing queue map. The MAX_MESSAGE_SIZE defines the message buffer that a Q Capture program allocates for each send queue. The value of MAXMSGL should also be at least as large as the MAXMSGL that is defined for each send queue, transmission queue, and the administration queue. **Note:** This parameter is not valid on z/OS™.
Send queue	**PUT (ENABLED)** Allows the Q Capture program to put data messages and informational messages on the queue. **DEFPSIST(YES)** The queue is persistent and survives a restart of the queue manager. Messages are logged, and can be recovered.
Administration queue	**GET (ENABLED)** Allows the Q Capture program to get messages from the queue. **DEFPSIST(YES)** The queue is persistent and survives a restart of the queue manager. Messages are logged, and can be recovered.

Table 5. Required parameter values for WebSphere MQ objects at the source (continued)

Object name	Required settings
Restart queue	**PUT (ENABLED)** Allows the Q Capture program to put a restart message on the queue. **GET (ENABLED)** Allows the Q Capture program to get the restart messages from the queue. **DEFPSIST(YES)** The queue is persistent and survives a restart of the queue manager. The message on the queue is logged and can be recovered.

WebSphere MQ objects at the target

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

Table 6. Required parameter values for WebSphere MQ objects at the target

Object name	Required settings
Queue manager	**MAXMSGL** (The maximum size of messages allowed on queues for this queue manager.) This value must be at least as large as the MAX_MESSAGE_SIZE and MEMORY_LIMIT that you define when you create a replication queue map. The MAX_MESSAGE_SIZE defines the message buffer that a Q Capture program allocates for each send queue. The MEMORY_LIMIT defines the message buffer that a Q Apply program uses for buffering transactions from each receive queue. **Note:** This parameter is not valid on z/OS.
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. **DEFPSIST(YES)** The queue is persistent and survives a restart of the queue manager. Messages are logged and can be recovered. **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. It is recommended that you accept this default for the receive queue even though Q replication messages are not prioritized.

Table 6. Required parameter values for WebSphere MQ objects at the target (continued)

Object name	Required settings
Administration queue	**PUT(ENABLED)** Allows the Q Apply program or user application to put control messages on the queue. **DEFPSIST(YES)** The queue is persistent and survives a restart of the queue manager. Messages are logged, and can be recovered. **SHARE** Allows multiple Q Apply threads to work with this queue.
Spill queue	**Queue name** The spill queue must have a specific name, IBMQREP.SPILL.MODELQ **DEFTYPE(PERMDYN)** Specifies that spill queues are permanent dynamic queues. They are created and deleted at the request of the Q Apply program, but they survive a restart. Messages are logged and can be recovered. **DEFSOPT(SHARED)** Allows more than one thread (different agent threads and the spill agent thread) to access messages on the spill queue at the same time. **MAXDEPTH(500000)** This is a recommended upper limit for the number of messages on the spill queue. Adjust this number based on the number of changes that are expected at the source table while the target table is being loaded. **MAXMSGL(100000)** 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. **MSGDLVSQ(FIFO)** Specifies that messages on the spill queue are delivered in first-in, first-out order.

WebSphere MQ channel objects

Table 7 on page 59 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 7. Required parameter values for WebSphere MQ objects used in channels

Object name	Description	Required settings
Transmission queue **(Q Capture to Q Apply or user application)**	Holds transaction messages and informational messages from the Q Capture program that are bound for a remote Q Apply program or user application.	**USAGE(XMITQ)** Transmission queue. **MAXDEPTH** 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. **MAXMSGL** Ensure that the maximum size of messages for the queue is not less than the MAXMSGL defined for the receive queue on the target system, and the MAX_MESSAGE_SIZE and MEMORY_LIMIT that you set when you create a replication queue map. **DEFPSIST(YES)** The queue is persistent and survives a restart of the queue manager. Messages are logged and can be recovered.

Transmission queue
 	
(Q Apply or user application to Q Capture)
	Holds control messages from the Q Apply program or user application that are destined for the Q Capture administration queue.	**USAGE(XMITQ)**
		Transmission queue.
		
		DEFPSIST(YES)
		The queue is persistent and survives a restart of the queue manager. Messages are logged and can be recovered.

Channel **(Q Capture to Q Apply or user application)**	Defined within the source queue manager for outbound data and informational messages.	**CHLTYPE(SDR)** A sender channel. **DISCINT** Ensure that the disconnect interval is large enough to keep this channel from timing out during periods when there are no transactions to replicate. **HBINT** Coordinate this value with the Q Capture **heartbeat_interval** parameter. If you use the HBINT parameter to send heartbeat flows, consider setting **heartbeat_interval**=0 to eliminate heartbeat messages.
Channel **(Q Apply or user application from Q Capture)**	Defined within the target queue manager for inbound transaction or informational messages.	**CHLTYPE(RCVR)** A receiver channel. **DISCINT** Ensure that the disconnect interval is large enough to keep this channel from timing out during periods when there are no transactions to replicate. **HBINT** Coordinate this value with the Q Capture **heartbeat_interval** parameter. If you use the HBINT parameter to send heartbeat flows, consider setting **heartbeat_interval**=0 to eliminate heartbeat messages.

Table 7. Required parameter values for WebSphere MQ objects used in channels (continued)

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

Related concepts:
- "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43
- "Setting up WebSphere MQ for Q replication and event publishing—Overview" on page 43
- "Required WebSphere MQ objects at a glance" on page 44

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 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. In most cases, the safest strategy is to use the same value for the Q Capture MAX_MESSAGE_SIZE and the MAXMSGL that you define for the queues that will hold the messages.

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 define this MEMORY_LIMIT when you create a replication queue map. The default is 2 MB.

  Important: Ensure that the MAXMSGL for the local queue that will serve as a receive queue is not larger than the MEMORY_LIMIT for 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 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 Q Capture program.

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

Related concepts:
- “Queue depth considerations for large object (LOB) values” on page 64
- “Considerations for large object (LOB) data types for Q replication and event publishing” on page 172
- “Publishing queue maps” on page 12
- “Replication queue maps” on page 6

Related tasks:
- “Creating publishing queue maps” on page 156
- “Planning memory requirements for Q replication and event publishing” on page 25
- “Creating replication queue maps” on page 84

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 will be used at the source. If messages cannot be applied, more resource will be used at the target.

All messages used in Q replication 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 persistent messages, as necessary.

For more information about WebSphere MQ log files on AIX®, HP-UX, Linux, Sun™ Solaris™, and Windows® systems, see *WebSphere MQ System Administration Guide*; for z/OS™ see *WebSphere MQ for z/OS Concepts and Planning Guide*; for other platforms, see the appropriate *System Management Guide*.

Related concepts:

- "Storage requirements for spill queues for the Q Apply program" on page 62

Related tasks:

- "Planning storage requirements for Q replication and event publishing" on page 31

Storage requirements for spill queues for the Q Apply program

A temporary spill queue is created for each Q subscription when the Q Apply program loads the target. The spill queue is removed after the load is completed. If you expect many updates to occur while the target is being loaded, you will need larger spill queues. You must allow enough storage for the spill queues, which the Q Apply program uses when it exceeds its memory limit. The maximum message size and queue depth for the spill queue must be large enough to handle the spilled data.

Recommendation: Initially load the targets at off-peak hours to limit the number of updates that the Q Apply program spills to the spill queue.

Related concepts:

- "Q Apply program" on page 18

Related tasks:

- "Planning storage requirements for Q replication and event publishing" on page 31

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 via 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 that it is managing queues for, are not 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.

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).

Authorization requirements for the Q Apply program

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

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

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, *WebSphere MQ Security*.

Related concepts:

- "Connectivity requirements for Q replication and event publishing" on page 67
- "The Replication Alert Monitor" on page 255
- "Q Apply program" on page 18
- "Q Capture program" on page 15
- "Setting up WebSphere MQ for Q replication and event publishing—Overview" on page 43

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® UDB, LOB values can be up to 2 gigabytes. Therefore, LOB values will frequently be divided into multiple messages.

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.

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.

Related concepts:

- "Required settings for WebSphere MQ objects" on page 56
- "Considerations for large object (LOB) data types for Q replication and event publishing" on page 172
- "Setting up WebSphere MQ for Q replication and event publishing—Overview" on page 43

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*.

Related concepts:

- "Q Apply program" on page 18
- "Q Capture program" on page 15
- "Setting up WebSphere MQ for Q replication and event publishing—Overview" on page 43

Chapter 8. Configuring servers for Q replication and event publishing

Configuring servers for Q replication and event publishing—Overview

Before you can replicate or publish data, you must create and configure your servers and ensure that they can connect to each other. The following topics explain how to set up and configure your environment:

- "Connectivity requirements for Q replication and event publishing"
- "Configuring databases for Q replication and event publishing (Linux, UNIX, Windows)" on page 68
- "Configuring databases for Q replication and event publishing (z/OS)" on page 75
- "Software prerequisites for the Replication Center" on page 76
- "Creating control tables for the Q Capture and Q Apply programs" on page 76

For information on configuring servers for DB2® Information Integrator Replication for z/OS™, see *IBM DB2 Information Integrator Replication Installation and Customization Guide for z/OS*.

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 Replication Center or Replication Alert Monitor, to load target tables, or to insert signals to activate or deactivate Q subscriptions or XML publications.

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

- The 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 data from the source 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 on DB2 UDB Extended Server Edition 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 Replication Center, 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.

Related concepts:

- "Configuring servers for Q replication and event publishing—Overview" on page 67

- “The Replication Alert Monitor” on page 255
- “Q Apply program” on page 18
- “Q Capture program” on page 15
- “Connectivity and authorization requirements for WebSphere MQ objects” on page 62

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

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

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

Procedure:

To configure databases for Q replication and event publishing:

1. Set environment variables. See “Setting environment variables (Linux, UNIX, Windows)” for details.
2. Configure the source database to work with the Q Capture program. See “Configuring the source database to work with the Q Capture program (Linux, UNIX, Windows)” on page 69 for details.
3. Configure the target database to work with the Q Apply program (Q replication only). See “Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows)” on page 71 for details.
4. Optional: Bind the program packages. See “Optional: Binding the program packages (Linux, UNIX, Windows)—Overview” on page 72 for details.

Related concepts:

- “Configuring servers for Q replication and event publishing—Overview” on page 67
- “Configuring databases for Q replication and event publishing (z/OS)” on page 75

Setting environment variables (Linux, UNIX, Windows)

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

Setting environment variables is part of the larger task of configuring databases.

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, as shown:

Windows:

```
SET DB2INSTANCE=db2_instance_name
```

Linux and UNIX:

```
export 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 DB2DBDFT to the Q Capture server or Q Apply server.
4. **Linux and UNIX:** Make sure that the system variables include the directory where the Q replication libraries and executables are installed. The default library path is LIBPATH=SQLLIB/LIB, and the default executable path is PATH=SQLLIB/BIN in the DB2 instance home directory. If you moved the 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/home/*db2inst*/sqllib/profile.env file and adding or modifying the line:

   ```
   DB2ENVLIST='EXTSHM'
   ```

   Where *db2inst* is the name of the DB2 instance that contains the target database.

   Also add the following line to the /home/*db2inst*/sqllib/userprofile file:

   ```
   export EXTSHM=ON
   ```

Next task:

- Configure the source database to work with the Q Capture program. See "Configuring the source database to work with the Q Capture program (Linux, UNIX, Windows)" for details.

Related concepts:

- "The Replication Alert Monitor" on page 255
- "Q Apply program" on page 18
- "Q Capture program" on page 15

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 so that the Q Capture program can read log files for changes to source tables. For this change to take effect, you must also perform an offline backup of the source database.

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.

Configuring the source database to work with the Q Capture program is part of the larger task of configuring databases.

Procedure:

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

Replication Center

Use the Turn On Archive Logging window to enable archive logging at the source database. To open the window, right-click the Q Capture server that you want to enable and select **Turn On Archive Logging**. See the online help for details.

Command line

To configure a DB2 database to run the Q Capture program:

1. Check your current setting by entering:

   ```
   db2 get database manager configuration for database
   ```

   Where *database* is the database that contains your source data.

   If the output shows `(LOGRETAIN) = RECOVERY` then your database is already configured to run the Q Capture program. If the output shows `(LOGRETAIN) = OFF`, proceed to the next step.
2. Issue the following command:

   ```
   update database configuration for database using LOGRETAIN RECOVERY
   ```

   If you want to use an exit routine to manage archived logs, you must set the USEREXIT database configuration parameter to ON.
3. Optional: Use the **update database configuration** command to increase values for the source database depending on your replication or publishing needs. The following database configuration values are adequate for many large replication scenarios (if the database is configured to work with other applications, your values might already be larger than these): APPLHEAPSZ 4096, LOGFILSIZ 2000, LOGPRIMARY 20, LOCKLIST 200, DBHEAP 1000, STMTHEAP 4096, LOGBUFSZ 64, MAXAPPLS 300, LOCKTIMEOUT 30.

   Use this formula to determine the value for the MAXLOCKS parameter:

   ```
   n >= 200 / average number of concurrent connections
   ```

   Where *n* is the value for MAXLOCKS.

   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 UDB Version 8.1 or newer to avoid most log space problems.
4. Issue the **backup database** command using appropriate parameters for the source database.

Next task:

- Configure the target database to work with the Q Apply program (Q replication only). See "Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows)" for details.

Related concepts:

- "Q Capture program" on page 15

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.

Configuring the target database to work with the Q Apply program is part of the larger task of configuring databases.

Procedure:

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

1. Issue the following command:

   ```
   update database configuration for database using MAXAPPLS n
   ```

   where *database* is the target database and *n* is the maximum number of applications that will be allowed to run on the target database at the same time.

   To determine *n*, use the following formula:

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

   Here is an example of the results for this calculation based on the following scenario:

   - Two applications other than the Q Apply program
   - Two Q Apply programs, one that uses 3 agents to process a single receive queue, and one that uses 12 agents to process four receive queues (so a total of five browser threads to process the five receive queues)

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

     In this scenario, you would set MAXAPPLS to at least 28.

2. Optional: If you plan to have the Q Apply program automatically load targets using the LOAD from CURSOR option of the LOAD utility, issue the following command:

   ```
   update dbm cfg using FEDERATED YES
   ```

3. Optional: Use the **update database configuration** command to increase values for the target database depending on your replication needs. The following database configuration values are adequate for many large replication scenarios (if the database is configured to work with other applications, your values might already be larger than these): APPLHEAPSZ 4096, LOGFILSIZ 2000, LOGPRIMARY 20, LOCKLIST 200, DBHEAP 1000, STMTHEAP 4096, LOCKTIMEOUT 30.

   Use this formula to determine the value for the MAXLOCKS parameter:

   ```
   n >= 200 / average number of concurrent connections
   ```

   Where *n* is the value for MAXLOCKS.

   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 UDB Version 8.1 or newer to avoid most log space problems.

   For LOGBUFSZ, a value between 64 and 512 is recommended.

Next task:

- Optional: Bind the program packages. See "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" for details.

Related concepts:

- "Q Apply program" on page 18

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

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

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. If you want, 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.

The following topics explain the manual binding process for each program:

- "Optional: Binding the Q Capture program packages (Linux, UNIX, Windows)" on page 73
- "Optional: Binding the Q Apply program packages (Linux, UNIX, Windows)" on page 73
- "Optional: Binding the Replication Alert Monitor packages (Linux, UNIX, Windows)" on page 74

Related concepts:

- "Considerations for rebinding packages and plans for Q replication and event publishing" on page 287

Related tasks:

- "Configuring databases for Q replication and event publishing (Linux, UNIX, Windows)" on page 68

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.

Binding the Q Capture program packages is part of the larger task of configuring databases for Q replication and event publishing (Linux, UNIX, Windows).

Procedure:

To bind the Q Capture program packages:

1. Connect to the Q Capture server by entering:
   ```
   db2 connect to database
   ```
   Where *database* is the Q Capture server.
2. Change to the directory where the Q Capture bind files are located.

   Windows:
   drive:\...\sqllib\bnd

   Linux and UNIX:
   db2homedir/sqllib/bnd

   Where *db2homedir* is the DB2 instance home directory.
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.

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

Related concepts:
- "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" on page 72
- "Q Capture program" on page 15

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 choose to specify bind options, or bind the packages manually during a time when you expect less contention at these databases. This procedure explains how to manually bind the Q Apply program packages.

Binding the Q Apply program packages is part of the larger task of configuring databases for Q replication and event publishing (Linux, UNIX, Windows).

Procedure:

To bind the Q Apply program packages:

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

   Windows:
      drive:\...\sqllib\bnd

   Linux and UNIX:
      db2homedir/sqllib/bnd

   Where *db2homedir* is the DB2 instance home directory.
2. For both the source and target databases, do the following steps:
   a. Connect to the database by entering:
      ```
      db2 connect to database
      ```
      Where *database* is the source or target database. 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 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.

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

Related concepts:

- "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" on page 72
- "Q Apply program" on page 18

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 choose to specify bind options, or bind the packages manually during a time when you expect less contention at these databases. This procedure explains how to manually bind the Replication Alert Monitor packages.

Binding the Replication Alert Monitor packages is part of the larger task of configuring databases for Q replication and event publishing (Linux, UNIX, Windows).

Procedure:

To bind the Replication Alert Monitor packages:

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

   Windows:
      drive:\...\sqllib\bnd

   Linux and UNIX:
      db2homedir/sqllib/bnd

Where *db2homedir* is the DB2 instance home directory.

2. For each Monitor control server, do the following steps:
   a. Connect to the Monitor control server database by entering:
      ```
      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 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 connects, do the following steps:
   a. Connect to the database by entering:
      ```
      db2 connect to database
      ```
      Where *database* is the monitored 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 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.

Related concepts:

- "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" on page 72
- "The Replication Alert Monitor" on page 255

Configuring databases for Q replication and event publishing (z/OS)

You must set up and customize the replication programs when you install DB2® Information Integrator Replication for z/OS™. See the instructions in *IBM DB2 Information Integrator Replication Installation and Customization Guide for z/OS*.

Related concepts:

- "Configuring servers for Q replication and event publishing—Overview" on page 67

Related tasks:

- "Configuring databases for Q replication and event publishing (Linux, UNIX, Windows)" on page 68

Software prerequisites for the Replication Center

To use the Replication Center, your system must have the correct Java™ Runtime Environment (JRE) installed. When you install DB2® UDB, 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.

z/OS™: To operate the Q Capture, Q Apply, or Replication Alert Monitor programs from the Replication Center, you must install the DB2 Administration Server (DAS) for z/OS. To display z/OS buffer pools, you must install the 390 Enablement package:

- DB2 for OS/390® and z/OS Version 7 FMID for the DB2 Administration Server is HDAS810.
- DB2 for OS/390 and z/OS Version 7 FMID for the 390 Enablement package is JDB771D. This package includes stored procedures that must be installed in DB2.
- DB2 for OS/390 Version 6 FMID for the 390 Enablement package is JDB661D.

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 the local system that is running the Replication Center and 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. After the DAS is installed, it can also be used with DB2 for OS/390 Version 6 subsystems.

Related concepts:

- "DB2 Administration Server" in the *Administration Guide: Implementation*
- "Configuring servers for Q replication and event publishing—Overview" on page 67

Related tasks:

- "Create a DB2 Administration Server" in the *Administration Guide: Implementation*

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 XML publications, message queues, operational parameters, and user preferences. You create the control tables in the database or z/OS subsystem that will be used as the Q Capture server or Q Apply server.

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 QC1 would be named QC1.IBMQREP_CAPPARMS.

For peer-to-peer replication, the Q Capture program and Q Apply program run as a pair at each server. You can use the Replication Center to create both sets of control tables at the same time on a server. In this case, both sets of control tables must have the same schema.

Prerequisites:

- The Replication Center must be able to connect to the server where you want to create the control tables.
- You must have the names of the following WebSphere MQ objects:

  For the Q Capture program

  - A queue manager at the system where the Q Capture program runs.
  - A local, persistent queue to serve as the administration queue.
  - A local, persistent queue to serve as the restart queue.

  For the Q Apply program

  - A queue manager at the system where the Q Apply program runs.

  Important: The Replication Center does not validate the WebSphere MQ queue manager and queue names that you specify when you create control tables. Make sure that the names that you specify match the WebSphere MQ object names. Otherwise, the Q Capture or Q Apply program will not run.

Restriction:

For multiple partitioned databases, all of the table spaces that are used by the control tables must be on the catalog partition. If you use an existing table space, the table space must be non-partitioned and it must be on the catalog partition.

Procedure:

To create control tables for a Q Capture program, Q Apply program, or both:

Use the Create Control Tables wizard in the Replication Center to create control tables for a Q Capture program, a Q Apply program, or both. To open the wizards, right-click the **Q Capture Servers** folder and select **Create Q Capture control tables**, or right-click the **Q Apply Servers** folder and select **Create Q Apply control tables**.

By default, the control tables are placed in one table space for Linux, UNIX, and Windows, and two table spaces for z/OS. You can customize where each control table is created, and you can specify existing table spaces or create new table spaces.

You can save the SQL script that is generated by the Create Control Tables wizard and use it to create additional sets of control tables. Use the Run Now or Save SQL window that displays after you finish the wizard to run the script immediately, save it as a file, or to schedule it as a task in the Task Center (except for z/OS).

See the online help for details.

Related concepts:

- “Configuring servers for Q replication and event publishing—Overview” on page 67
- “Q Apply program” on page 18

- "Q Capture program" on page 15
- "Schemas for the Q Apply and Q Capture programs" on page 20
- "Control tables for Q replication and event publishing—Overview" on page 361

Related tasks:

- "Running and saving SQL scripts from the Replication Center" on page 174

Part 3. Setting up Q replication and event publishing

This part of the book contains the following chapters:

- Chapter 9, “Setting up replication from sources to targets (unidirectional),” on page 81 describes how to create Q subscriptions for unidirectional replication so that changes are replicated in one direction: from a source table to a target table. It explains how to specify the WebSphere MQ queues that are used to transport the data and the options for specifying what data to replicate.
- Chapter 10, “Setting up replication from sources to targets (multidirectional),” on page 117 describes how to create Q subscriptions so that changes are replicated in both directions between tables. The chapter explains bidirectional replication and peer-to-peer replication, and helps you choose which type of replication might better meet your business needs. It explains how to specify the WebSphere MQ queues that are used to transport the data and the options for specifying what data to replicate.
- Chapter 11, “Options for loading target tables for Q replication,” on page 143 describes options for initially loading target tables with data from the source tables.
- Chapter 12, “Setting up publishing from sources (event publishing),” on page 155 describes how to create XML publications so that changes from source tables are published in XML format for use by a user application of your choice. The chapter also explains how to specify the WebSphere MQ queues that transport the changes.
- Chapter 13, “Considerations for replicating and publishing data types,” on page 171 describes general data restrictions for replicating and publishing, and how large objects are handled.
- Chapter 14, “Running SQL scripts and operational commands from the Replication Center,” on page 173 describes how to modify, save, and run SQL scripts that the Replication Center generates.

Chapter 9. Setting up replication from sources to targets (unidirectional)

Setting up replication from sources to targets (unidirectional)—Overview

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 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.

The following topics describe unidirectional replication, how to specify the queues that you need, how to set up unidirectional replication, and how to manipulate source data:

- “Unidirectional replication”
- “Creating replication queue maps” on page 84
- “Creating Q subscriptions for unidirectional replication—Overview” on page 86
- “Using stored procedures to manipulate the data that is applied to targets for Q replication—Overview” on page 106

Related concepts:

- “Setting up replication from sources to targets (multidirectional)—Overview” on page 117
- “Setting up publishing from sources (event publishing)—Overview” on page 155

Unidirectional replication

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 or passes them to a stored procedure.

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

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.

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 9 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.

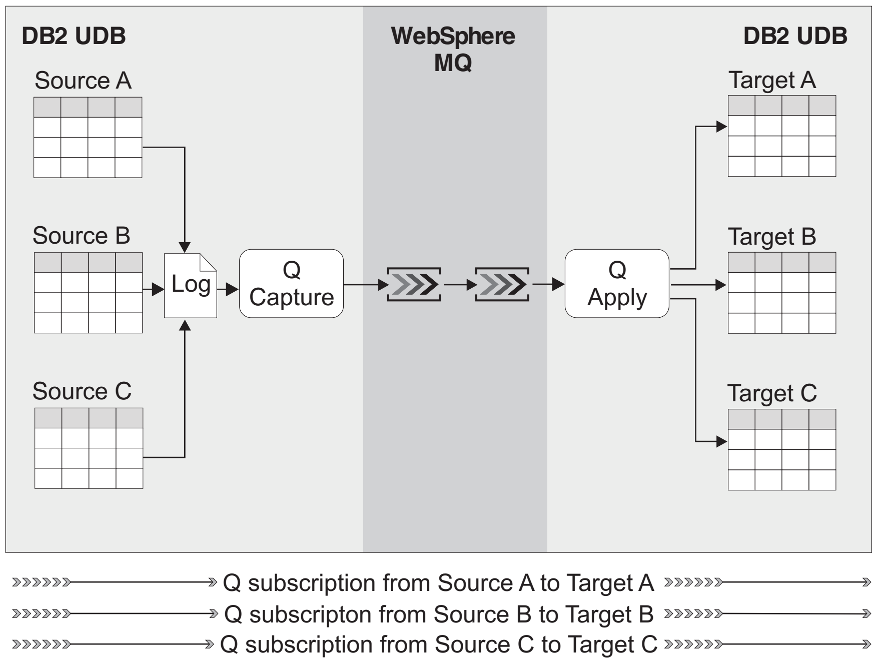

DB2 UDB
WebSphere MQ
DB2 UDB
Source A
Source B
Source C
Log
Q Capture
Q Apply
Target A
Target B
Target C
Q subscription from Source A to Target A
Q subscripton from Source B to Target B
Q subscription from Source C to Target C

Figure 9. 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 from the database log 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. Because the Q Apply browser at the receive queue detects dependencies between transactions, defining Q subscriptions so that dependent transactions use separate replication queue maps could produce undesirable results. 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.

Related concepts:
- “Creating Q subscriptions for peer-to-peer replication—Overview” on page 135
- “Replication queue maps” on page 6
- “Creating Q subscriptions for unidirectional replication—Overview” on page 86

Related tasks:
- “Creating Q subscriptions for bidirectional replication” on page 127
- “Creating replication queue maps” on page 84

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 while you are creating Q subscriptions.

Each replication queue map identifies the following WebSphere MQ queues:

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 Q Apply program uses this administration queue 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 it messages, initiating a full refresh of a target table, or telling the Q Capture program to stop replicating certain Q subscriptions.

The replication queue map also identifies the following options:

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 resulting action if an error occurs at one of the WebSphere MQ queues that is involved in this replication queue map. You must decide whether the Q Capture program:

- Deactivates the Q subscriptions that use the queue
- Stops running

The queue error action is used when a WebSphere MQ error occurs that is specific to this queue. Regardless of which option that you choose, the Q Capture program logs the error message in its diagnostic log and in the IBMQREP_CAPTRACE table.

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.

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

Prerequisites:

- Plan how you want to group replication queue maps and Q subscriptions. See "Grouping replication queue maps and Q subscriptions" on page 83 for details.
- On the server that contains the source tables for the Q subscriptions, create the control tables for the Q Capture program, and on the server that contains the target tables for the Q subscriptions, create the control tables for the Q Apply program. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- Ensure that you have defined the appropriate objects in WebSphere MQ. See "Required settings for WebSphere MQ objects" on page 56 for details.

Restrictions:

The same send queue cannot be used for both Q replication and event publishing because there is a queue-level attribute that specifies whether it transports compact messages (for Q replication) or XML messages (for event publishing).

Procedure:

To define a replication queue map, use the Create Replication Queue Map window in the Replication Center. To open the window, do one of the following:

- Expand the Q Capture schema that identifies the Q Capture program that you want to use the queue map. Right-click the **Replication Queue Maps** folder and select **Create**.
- Expand the Q Apply schema that identifies the Q Apply program that you want to use the queue map. Right-click the **Replication Queue Maps** folder and select **Create**.

See the online help for details.

Related concepts:

- "Replication queue maps" on page 6

Related tasks:

- "Changing attributes of replication queue maps" on page 184
- "Deleting replication queue maps" on page 188
- "Creating publishing queue maps" on page 156

Creating Q subscriptions for unidirectional replication

Creating Q subscriptions for unidirectional replication—Overview

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. 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

In Q replication, you can create one or multiple Q subscriptions at one time.

Important: Q subscriptions are separate objects from XML publications. XML publications do not publish data to the Q Apply program, but to an application of your choice. Q subscriptions are for replicating data, and XML 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 an XML publication.

The following topics explain how to create one or multiple Q subscriptions at one time for unidirectional replication and the options that you can choose if you want to further customize the Q subscriptions:

- "Target object profiles for Q replication"
- "Creating a single Q subscription for unidirectional replication" on page 88
- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Source columns for Q subscriptions (unidirectional replication)" on page 92
- "How often the Q Capture program sends a message (unidirectional replication)" on page 93
- "Search conditions to filter rows (unidirectional replication)" on page 94
- "How source columns map to target columns (unidirectional replication)" on page 98
- "Index or key columns for targets (unidirectional replication)" on page 99
- "Options for unexpected conditions in the target table (unidirectional replication)" on page 100
- "Error options for Q replication" on page 103

Related concepts:

- "Q subscriptions" on page 5
- "Setting up replication from sources to targets (unidirectional)—Overview" on page 81

Related tasks:

- "Creating Q subscriptions for bidirectional replication" on page 127
- "Creating Q subscriptions for peer-to-peer replication with two servers" on page 135
- "Creating Q subscriptions for peer-to-peer replication with three or more servers" on page 137

Target object profiles for Q replication

You can set up a profile in the Replication Center that the Replication Center uses when you create objects on the target server. Those objects might be tables (target tables), table spaces, indexes, or nicknames. A target object profile lets you specify your own default naming convention for all target tables, nicknames, and their associated database objects that are created on a target server. The Replication Center uses the naming rules that are contained in the target object profile to name objects that it creates. You can modify a target object profile so that it contains rules to ensure that all objects that the Replication Center creates are named according to your company's naming rules.

The Replication Center has a default target object profile; however, you can modify the default target object profile. You can also override the target object profile when you create new Q subscriptions with the Replication Center. 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 existing table spaces or in new table spaces. Furthermore, 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® UDB subsystem on z/OS™.

Finally, you can 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 a single Q subscription for unidirectional replication

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

Prerequisites:

Before you create a Q subscription for unidirectional replication, you must perform the following actions:
- Plan how you want to group replication queue maps and Q subscriptions. See "Grouping replication queue maps and Q subscriptions" on page 83 for details.
- On the server that contains the source tables for the Q subscriptions, create the control tables for the Q Capture program, and on the server that contains the target tables for the Q subscriptions, create the control tables for the Q Apply program. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- 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 define a Q subscription.) See "Creating replication queue maps" on page 84 for details.

- If you want the Q Apply program to pass source changes to a stored procedure instead of to a target table, write the stored procedure that you want to use. See "Writing stored procedures to manipulate source data for Q replication" on page 108 for details.

Restrictions:

A source or target for a Q subscription cannot be a view.

Procedure:

To create a Q subscription for unidirectional replication from one source table to one target table or stored procedure:

1. Use the wizard to specify that you are creating a Q subscription for unidirectional replication:
   a. Open the Create Q Subscriptions wizard from the Replication Center.

      To open the wizard, do one of the following:
      - Expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.
      - Expand the Q Apply schema that identifies the Q Apply program that you want to apply changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.

      See the online help for details.
   b. On the Replication page, specify that you want unidirectional replication.
2. On the Servers page:
   a. Specify information about the source server.
   b. Specify information about the target server.
   c. Specify the replication queue map that you want to use. If an appropriate replication queue map does not exist, create a new one. See "Creating replication queue maps" on page 84 for details.
3. On the Source Tables page, select the source table that you want to replicate from.
4. On the Target page, specify the type of target that you want to replicate to. You can specify one of the following options:
   - Have a new target table created for you.
   - Use an existing table as the target.
   - Manipulate source data using a stored procedure.

   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.
5. On the Rows and Columns page, specify what data to replicate:
   a. Specify the columns that you want to replicate:
      1) Use the Select Columns window to select a subset of the columns that you want to replicate. See "Source columns for Q subscriptions (unidirectional replication)" on page 92 for details.
      2) If you choose to publish a subset of the columns that are in the source table, specify when the Q Capture program sends a message to the Q

Apply program. See “How often the Q Capture program sends a message (unidirectional replication)” on page 93 for details.

   b. Specify the rows that you want to replicate. See “Search conditions to filter rows (unidirectional replication)” on page 94 for details.
   c. Specify how you want source columns to map to target columns, or manually choose which source column maps to which target column. See “How source columns map to target columns (unidirectional replication)” on page 98 for details.
   d. Specify the target index or key. See “Index or key columns for targets (unidirectional replication)” on page 99 for details.
6. On the Errors page, specify how the Q Apply program responds to errors:
   a. Specify how the Q Apply program responds to unexpected conditions in the target. See “Options for unexpected conditions in the target table (unidirectional replication)” on page 100 for details.
   b. Specify how the Q Apply program responds to errors. See “Error options for Q replication” on page 103 for details.
7. On the Loading Target Table page:
   a. Specify the options for loading the target table. See “Options for loading target tables for Q replication—Overview” on page 143 for details.
   b. Specify whether the Q subscription is active as soon as it is created.
8. On the Review Q Subscriptions page, confirm that the Q subscription is valid:
   - If you want to change anything about the Q subscription, modify the properties of the Q subscription.
   - If the Q subscription is missing information, modify its properties so that it is complete.
9. On the Summary page, click **Finish**.

Related concepts:
- “Creating Q subscriptions for unidirectional replication—Overview” on page 86
- “Using stored procedures to manipulate the data that is applied to targets for Q replication—Overview” on page 106
- “Target object profiles for Q replication” on page 87

Related tasks:
- “Changing attributes of Q subscriptions” on page 179
- “Deleting Q subscriptions” on page 186
- “Creating multiple Q subscriptions for unidirectional replication” on page 90

Creating multiple Q subscriptions for unidirectional replication

To save time, you can define multiple Q subscriptions for unidirectional replication at one time. After you define the attributes for all of the Q subscriptions that you want to create, you then have the option to change the properties of individual Q subscriptions before the Replication Center creates them.

For example, you can map 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 can use the same replication queue map, Q Capture program, and Q Apply program.

Tip: Create multiple Q subscriptions at once if you have many to define and most of them will share the same attributes.

Prerequisites:

Before you create Q subscriptions for unidirectional replication, you must perform the following actions:

- Plan how you want to group replication queue maps and Q subscriptions. See "Grouping replication queue maps and Q subscriptions" on page 83 for details.
- On the server that contains the source tables for the Q subscriptions, create the control tables for the Q Capture program, and on the server that contains the target tables for the Q subscriptions, create the control tables for the Q Apply program. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- Specify the queues for replicating and their attributes by creating replication queue maps. (You can do this task before you create Q subscriptions or while you define Q subscriptions.) See "Creating replication queue maps" on page 84 for details.

Restrictions:

A source or target for a Q subscription cannot be a view.

Procedure:

To create multiple Q subscriptions at one time for unidirectional replication:

1. Use the wizard to specify that you are creating Q subscriptions for unidirectional replication:
   a. Open the Create Q Subscriptions wizard from the Replication Center.

      To open the wizard, do one of the following:

      - Expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.
      - Expand the Q Apply schema that identifies the Q Apply program that you want to apply changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.

      See the online help for details.
   b. On the Replication page, specify that you want unidirectional replication.
2. On the Servers page:
   a. Specify information about the source server.
   b. Specify information about the target server.
   c. Specify the replication queue map that you want to use. If an appropriate replication queue map does not exist, create a new one. See "Creating replication queue maps" on page 84 for details.
3. On the Source Tables page, select the source tables that you want to replicate from.

   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. At the end of the wizard, before the Replication Center builds the Q subscriptions, you can modify individual Q subscriptions so that only a subset of the source columns and rows are replicated.

4. 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, and, if one does not exist, then the object is created.
5. On the Mapping Columns page, specify how you want source columns to map to target columns. See "How source columns map to target columns (unidirectional replication)" on page 98 for details.
6. On the Errors page, specify how the Q Apply program responds to errors:
   a. Specify how the Q Apply program responds to unexpected conditions in the target. See "Options for unexpected conditions in the target table (unidirectional replication)" on page 100 for details.
   b. Specify how the Q Apply program responds to errors. See "Error options for Q replication" on page 103 for details.
7. On the Loading Target Table page:
   a. Specify the options for loading the target tables. See "Options for loading target tables for Q replication—Overview" on page 143 for details.
   b. Specify whether the Q subscriptions are active as soon as they are created.
8. On the Review Q Subscriptions page, confirm that each Q subscription is valid:
   - If you want to change anything about an individual Q subscription, modify the properties of that Q subscription. For example, you can choose to replicate only a subset of the rows or columns from the source table for a particular Q subscription.
   - If any of the Q subscriptions are missing information, modify the properties to add the needed information.
9. On the Summary page, click **Finish**.

Related concepts:
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "Target object profiles for Q replication" on page 87

Related tasks:
- "Changing attributes of Q subscriptions" on page 179
- "Deleting Q subscriptions" on page 186
- "Creating a single Q subscription for unidirectional replication" on page 88

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 10 shows how a subset of columns are replicated.

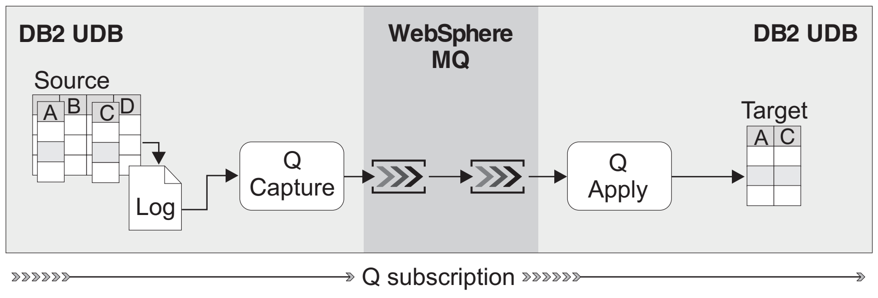


Figure 10. Column subsetting in a Q subscription for unidirectional replication. Only columns A and C are selected for replication. When a row is replicated from the source table to the target table, the Q Capture program replicates only the values from columns A and C.

To replicate a subset of the columns, select only the source columns that you want to be replicated to the target. If you are creating a single Q subscription, the Create Q Subscriptions wizard in the Replication Center gives you options for how to replicate a subset of the columns 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 that you want to subset columns in and edit the properties for that Q subscription.

Important for key columns: Because targets must contain sufficient key data to ensure uniqueness, make sure that you select the source columns that make up the key columns in the target table or the parameters in the stored procedure that map to key columns.

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.

Related concepts:
- “Source columns for XML publications” on page 163
- “Creating Q subscriptions for unidirectional replication—Overview” on page 86

Related tasks:
- “Creating multiple Q subscriptions for unidirectional replication” on page 90
- “Creating a single Q subscription for unidirectional replication” on page 88

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”

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.

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.

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.

Related concepts:
- “Creating Q subscriptions for unidirectional replication—Overview” on page 86

Related tasks:
- “Creating multiple Q subscriptions for unidirectional replication” on page 90
- “Creating a single Q subscription for unidirectional replication” on page 88

Search conditions to filter rows (unidirectional replication)

By default when you create Q subscriptions for unidirectional replication, all rows from the source table are replicated to the target table or stored procedure. However, when you create a Q subscription for unidirectional replication, you can specify a WHERE clause with a search condition to identify the rows that you want to be replicated. When the Q Capture program detects a change in the DB2® recovery log that is associated with a source table, the Q Capture program evaluates the change against the search condition to determine whether to replicate the change to the target table or stored procedure.

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

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

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

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

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

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

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

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

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

Suppose that the following change occurs at the source table:

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

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

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

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

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

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

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

Complex search conditions:

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

Example of WHERE clause with a subselect:

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

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

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

Restrictions for search conditions:
- Search conditions cannot contain column functions, unless the column function appears within a subselect statement.

  Invalid WHERE clause with column functions:

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

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

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

Invalid WHERE clause with GROUP BY:

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

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

Valid WHERE clause with GROUP BY:

```
WHERE :COL2 = (SELECT COL2 FROM T2 WHERE COL1=1 GROUP BY COL1, COL2)
```

- Search conditions cannot reference the actual name of the source table that you are replicating changes from. Do not use the `schema.tablename` notation in a WHERE clause for the actual name of the source table. However, you can reference another table name in a subselect by using `schema.tablename` notation.

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

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

  WHERE :ADMINISTRATOR.SALES > 100000
  ```

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

  Valid WHERE clause with column name:

  ```
  WHERE :SALES > 100000
  ```

  In the above WHERE clause, `SALES` is the column name.
- Search conditions cannot reference values that were in columns before a change occurred; they can reference values only after a change occurred.
- Search conditions cannot contain `IN` predicates that use a full select.

  Invalid WHERE clause with IN predicate:

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

  WHERE :COL1 IN (SELECT .... )
  ```

  Valid WHERE clause with IN predicate:

  ```
  WHERE :COL1 IN ('CA', 'IL')
  ```
- 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.

Related concepts:

- "Search conditions to filter rows in XML publications" on page 164
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86

Related tasks:

- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Creating a single Q subscription for unidirectional replication" on page 88

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 with the same columns that the source table has.)

Important: 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.

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.

Related concepts:
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86

Related tasks:
- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Creating a single Q subscription for unidirectional replication" on page 88

Index or key columns for targets (unidirectional replication)

The Q Apply program needs some mechanism to enforce that each row is unique when it applies the row to target tables or parameters in stored procedures. The Replication Center tries to select which columns should be used to identify uniqueness at the target. When you map a source table to a target table or stored procedure, you can override the columns that the Replication Center chose to uniquely identify rows in the target. The Q Apply program uses these columns, which make up a primary key, unique constraint, or unique index at the target, to track changes that it already applied.

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. If the Replication Center cannot find uniqueness at the source, then you must specify the columns that you want as the unique index for the target. You can accept the recommendation or specify the columns that you want as the primary key, unique constraint, or unique index for the target.

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"

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, then you must select the key 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.

Related concepts:

- "Creating Q subscriptions for unidirectional replication—Overview" on page 86

Related tasks:

- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Creating a single Q subscription for unidirectional replication" on page 88

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.

Recommendations for 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.

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 101
- "Ignore the unexpected condition" on page 101
- "Deactivate the corresponding Q subscription" on page 101

- "Have the Q Apply program stop reading from the corresponding receive queue" on page 102

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, an XML version of the row in error is inserted into the IBMQREP_EXCEPTIONS table.

Force the change

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

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

Ignore the unexpected condition

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

Recommendations: 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.

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 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 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.

Related concepts:

- "Creating Q subscriptions for unidirectional replication—Overview" on page 86

Related tasks:

- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Creating a single Q subscription for unidirectional replication" on page 88

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”
- “Stop the Q Apply program” on page 105

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 an XML version of any rows that were not applied and the details about the error are inserted into the IBMQREP_EXCEPTIONS table.

Deactivate the corresponding Q subscription

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

Recommendations:

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

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

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

Recommendations:

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

Stop the Q Apply program

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

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

Recommendations:

- You might want to stop the Q Apply program from applying transactions from the receive queue when an error occurs under the following circumstances:
  - Errors are not tolerated in your scenario and you do not expect them to occur often.
  - You have multiple Q subscriptions that are defined over related tables and the data is transported over multiple replication queue maps. When one Q subscription has a error, you want all of the Q subscriptions to stop. You can check what went wrong with the one Q subscription and then, after the Q subscription is fixed, restart the receive queue. If few changes are being

replicated to the target table, then using this option might be a good choice because it helps to preserve transactions and helps related tables maintain referential integrity.
- You might not want to stop the Q Apply program from applying transactions from the receive queue if this result is too severe in your scenario if an error occurs.

Related tasks:
- “Creating Q subscriptions for bidirectional replication” on page 127
- “Creating Q subscriptions for peer-to-peer replication with two servers” on page 135
- “Creating Q subscriptions for peer-to-peer replication with three or more servers” on page 137
- “Creating multiple Q subscriptions for unidirectional replication” on page 90
- “Creating a single Q subscription for unidirectional replication” on page 88

Using stored procedures to manipulate the data that is applied to targets

Using stored procedures to manipulate the data that is applied to targets for Q replication—Overview

When you create a Q subscription for unidirectional replication, you can specify that the changes be replicated from the source table to a stored procedure instead of directly to a target table. You can use the stored procedure to manipulate data that is captured from the source before the data is applied to the target table.

The following topics describe how to write stored procedures:
- “Stored procedures for manipulating source data for Q replication” on page 107
- “Writing stored procedures to manipulate source data for Q replication” on page 108
- “Stored procedure parameter that identifies the type of operation for Q replication” on page 110
- “Stored procedure parameter that identifies whether each source column was suppressed” on page 112
- “Stored procedure parameters that identify the transaction for Q replication” on page 112
- “Stored procedure parameters that map to source columns for Q replication” on page 113

Related concepts:
- “Setting up replication from sources to targets (unidirectional)—Overview” on page 81

Related tasks:
- “Creating a single Q subscription for unidirectional replication” on page 88

Stored procedures for manipulating source data for Q replication

Typically in Q subscriptions, data from a source table is mapped to a target table; however, for Q subscriptions in unidirectional replication, you can instead have the Q Apply program call a stored procedure and pass the source data as input parameters to the stored procedure. The source columns map to parameters in the stored procedure instead of to target columns. By mapping source columns directly to parameters in a stored procedure, you avoid the need to parse the incoming data and have a clean, simple programming model.

When you specify that you want the target of a Q subscription to be a stored procedure, you still have all of the options that you have for a Q subscription with a target table. For example, with a stored procedure, you still choose error options.

The stored procedure returns a return code to the Q Apply program that indicates whether the data was applied to the target table. The developer that creates the stored procedure must ensure that the stored procedure returns an appropriate SQL return code to the Q Apply program. The stored procedure is responsible for getting the source data to its final destination.

Example: The following example shows a signature of a stored procedure that accepts values from five columns in the source table, two of which are key columns.

```
CREATE PROCEDURE storedprocedure_name(
                                  INOUT operation integer,
                                  IN suppression_ind VARCHAR(size),
                                  IN src_commit_lsn CHAR(10),
                                  IN src_trans_time TIMESTAMP,
                                  XParm1,
                                  Parm1,
                                  XParm2
                                  Parm2,
                                  Parm3,
                                  Parm4,
                                  Parm5
                                 )
```

This example shows the four mandatory parameters:

`operation`
: This INOUT parameter identifies the type of operation.

`suppression_ind`
: Thus IN parameter identifies the parameters that have been suppressed.

`src_commit_lsn`
: This IN parameter identifies the log sequence number of when the source server issued the `COMMIT` for the transaction.

`src_trans_time`
: This IN parameter identifies the timestamp of when the source server issued the `COMMIT` for the transaction.

The signature also contains the before and after values for key columns and only the after values for non-key columns. The following parameters accept values from source columns:

- Parameters `Parm1` and `Parm2` map to the key columns in the source table.

- Parameters `XParm1` and `XParm2` accept the before values of key columns in the source table.
- Parameters `Parm3`, `Parm4`, and `Parm5` map to non-key columns in the source table.

Tip: See the sample programs for examples of stored procedures for Q replication that are written in C and in SQL.

Important: The stored procedure must not perform COMMIT or ROLLBACK functions because the Q Apply program commits or rolls back transactions.

Important for LOB data: The Q Apply program handles LOB data for stored procedures similarly to how it handles LOB data for target tables. For Q subscriptions with stored procedures, LOB changes are replicated by sending them in multiple messages, depending on the size of the LOB and the message size that you allowed the Q Capture program to send. The stored procedure call that involves the first LOB message for a single LOB change is the original row operation. All of the remaining LOB data for the single LOB change is sent by additional calls to the stored procedure. The Q Apply program transforms the operation into an update. The `suppression_ind` index marks all parameters as suppressed, except for the parameter that maps to the LOB column in the source table. The stored procedure is called each time a LOB message is on the receive queue.

Related concepts:
- “Considerations for large object (LOB) data types for Q replication and event publishing” on page 172
- “Memory for LOB data types for Q replication and event publishing” on page 31
- “Using stored procedures to manipulate the data that is applied to targets for Q replication—Overview” on page 106

Related tasks:
- “Creating a single Q subscription for unidirectional replication” on page 88

Related reference:
- “Samples to set up Q replication and event publishing (Linux, UNIX, Windows)” on page 465
- “Samples to set up Q replication and event publishing (z/OS)” on page 467

Writing stored procedures to manipulate source data for Q replication

This topic discusses how to write a stored procedure so that you can use it as a target in a Q subscription and how to set up the stored procedure. You must write the stored procedure before you can declare it as the target in a Q subscription.

The stored procedure consists of four mandatory parameters (one that identifies the operation and three that identify the transaction) and additional parameters that map to source columns.

Restrictions:
- The stored procedure must not perform COMMIT or ROLLBACK functions.
- You cannot send the before value from a non-key column to a parameter in the stored procedure.

- Because the Q Apply program knows only about the stored procedure and not about the table that the stored procedure might pass the manipulated data into, Q subscriptions that have stored procedures associated with them cannot maintain referential integrity.
- If the stored procedure maintains a target table and you want the Q Apply program to resolve conflicts, then do not create secondary unique indexes on the target table.

Procedure:

To write a stored procedure to be the target in a Q subscription and to set up the stored procedure:

1. Write a stored procedure that contains four mandatory parameters and additional parameters that map to source columns. The parameters in the stored procedure are positional. The first parameter must have the parameter mode INOUT. All other parameters must have parameter mode IN. See the following topics:
   - "Stored procedure parameter that identifies the type of operation for Q replication" on page 110
   - "Stored procedure parameter that identifies whether each source column was suppressed" on page 112
   - "Stored procedure parameters that identify the transaction for Q replication" on page 112, which include the following two mandatory parameters:
     - The log sequence number of when the source server issued the commit statement for the transaction
     - The timestamp of when the source server issued the commit statement for the transaction
   - "Stored procedure parameters that map to source columns for Q replication" on page 113

   Tip: See the sample programs for examples of stored procedures for Q replication that are written in C and in SQL.
2. Run the `CREATE PROCEDURE` statement runs properly so that the name and appropriate parameters of the stored procedure are registered with the DB2 database.

   Tips:
   - Use the GENERAL WITH NULLS parameter style when you declare the stored procedure. This style increases portability of the stored procedure across the DB2 Universal Database family.
   - **Linux, UNIX, Windows**: Make sure that the stored procedure exists in the sqllib/function directory.
3. Compile and bind the stored procedure at the target server.

   Tip for z/OS: Include the DBRM of the stored procedure into the Q Apply package list.
4. Ensure that the Q Apply program has the proper authority to call the stored procedure.

   After the call to the stored procedure, the Q Apply program always checks the SQL code before checking the return code to validate whether the stored procedure is successful.
5. If the stored procedure maintains a target, ensure that the target exists.
6. Declare the stored procedure as the target in a Q subscription.

Related concepts:

- "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" on page 72
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "Using stored procedures to manipulate the data that is applied to targets for Q replication—Overview" on page 106
- "Stored procedures for manipulating source data for Q replication" on page 107

Related reference:

- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)" on page 465
- "Samples to set up Q replication and event publishing (z/OS)" on page 467

Stored procedure parameter that identifies the type of operation for Q replication

Declare the first parameter in the stored procedure as an INOUT parameter that the Q Apply program uses to pass in the type of operation (`INSERT`, `UPDATE`, `DELETE`, or `KEY UPDATE`) and that the stored procedure uses to pass a return code back to the Q Apply program about whether the operation was successful. The attributes that you specify for the Q subscription determine how the Q Apply program handles unexpected conditions in the target (in this case, in the stored procedure).

Tip: See the sample programs for examples of the `operation` parameter in stored procedures that are written in C and SQL. Note how the `operation` parameter is used to pass the return code back to the Q Apply program.

Table 8 shows the operation values that the Q Apply program passes to the stored procedure and what each value means.

Table 8. SQL return codes that the Q Apply program passes to the stored procedure

Operation value	Type of operation
16	Delete
32	Insert
64	Update to non-key columns
128	Update to key columns

Set the `INOUT operation integer` parameter to the SQL code for the Q Apply program to evaluate to see if the change was successfully applied to the target. If the stored procedure does not return a SQL code to the Q Apply program, then the Q Apply program does not know what happened inside the stored procedure.

Important: The stored procedure must not perform COMMIT or ROLLBACK functions because the Q Apply program commits or rolls back transactions.

If other applications are also making changes to the table that the stored procedure is applying changes to, then the stored procedure might encounter unexpected conditions in that table. For example, the stored procedure might try to update a row that another application already deleted. You have the following choices for how to resolve unexpected conditions that might occur when the stored procedure attempts to insert the manipulated source data into the target table:

- "The Q Apply program handles the unexpected condition"
- "The stored procedure handles the unexpected condition"

The Q Apply program handles the unexpected condition

If you want the Q Apply program to handle the unexpected condition in the target that the stored procedure maintains, then write the stored procedure so that it returns one of the SQL codes listed below (Table 9) to the Q Apply program, and define the Q subscription for the Q Apply program to force changes into the target that the stored procedure maintains. The Q Apply program forces the source change. In the case of certain SQL return codes, the Q Apply program transforms the row operation, logs the exception in the IBMQREP_EXCEPTIONS table, and passes the transformed row operation back to the stored procedure.

The stored procedure handles the unexpected condition

If you do not want the Q Apply program to handle unexpected conditions in the target by forcing source changes into the target, you can build other error-handling logic into the stored procedure. The stored procedure must always return an SQL code to the Q Apply program, but the stored procedure can return 0 (zero) so that the Q Apply program does not know about unexpected conditions or actual failures in the target table. After the stored procedure passes SQL return codes back to the Q Apply program in the `operation` parameter, the Q Apply program interprets each SQL code and starts the appropriate action (depending on how you specified that the Q Apply program should handle unexpected conditions). The Q Apply program handles return codes of +100 and −803 as conflicts, and any other return code as an error. Table 9 shows the types of +100 and −803 return codes that the stored procedure outputs, how the Q Apply program interprets that type of return code, and what action the Q Apply program takes as a result. The information in the table below assumes that you specified for the Q Apply program to force target changes.

Table 9. SQL return codes that the stored procedure passes to the Q Apply program and how the Q Apply program responds.

SQL return code	Type of operation	What the return code means	How Q Apply reacts
0	Insert	The row was inserted successfully.	Q Apply processes the next row.
0	Update	The row was updated successfully.	Q Apply processes the next row.
0	Delete	The row was deleted successfully.	Q Apply processes the next row.
+100	Delete	The row was not found in the target.	Q Apply does not retry the call.
+100	Update	The row was not found in the target.	If you specified for Q Apply to force changes, Q Apply changes the update into an insert and tries the call again.
−803	Insert	The row already exists in the target	If you specified for Q Apply to force changes, Q Apply transforms the insert into an update and tries the call again.
−803	Key update	The new key already exists in the target.	If you specified for Q Apply to force changes, Q Apply transforms the operation to an update operation that has the new keys and tries the call again.

Related concepts:
- "Options for unexpected conditions in the target table (unidirectional replication)" on page 100

Related tasks:
- "Writing stored procedures to manipulate source data for Q replication" on page 108

Related reference:
- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)" on page 465
- "Samples to set up Q replication and event publishing (z/OS)" on page 467

Stored procedure parameter that identifies whether each source column was suppressed

After the parameter that identifies the operation (`operation`), declare the second parameter (`suppression_ind`) in the stored procedure as an IN parameter that identifies whether each source column was suppressed.

The IN parameter `suppression_ind` is a character array (`colsupind`) that holds a character for each parameter except for the four mandatory parameters. The character indicates whether the value of the source column was suppressed:

1
: The value of the source column that correlates with the parameter was suppressed.

0
: The value of the source column that correlates with the parameter was not suppressed

The array must be used inside the stored procedure for evaluating the functions parameters. The Q Apply program checks to ensure that exactly one before value exists for each parameter that maps to a key column.

Tip: See the sample programs for examples of the parameter that identifies suppressed columns in stored procedures that are written in C and SQL.

Related tasks:
- "Writing stored procedures to manipulate source data for Q replication" on page 108

Related reference:
- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)" on page 465
- "Samples to set up Q replication and event publishing (z/OS)" on page 467

Stored procedure parameters that identify the transaction for Q replication

After the parameter that identifies whether any source columns were (`suppression_ind`), declare the next two parameters in the stored procedure as IN parameters that take in information from the message header from the Q Apply program that identifies the transaction. Because the Q Apply program passes rows and not transactions to the stored procedure, the transactional parameters give you information to help you identify which transaction this row change belongs to, and the time stamps and log sequence numbers also help with error reporting or diagnostics. The following two parameters identify the transaction:

IN src_commit_lsn CHAR(10)
: The log sequence number of when the source server issued the commit statement for the transaction.

IN src_trans_time TIMESTAMP
: The timestamp of when the source server issued the commit statement for the transaction.

Tip: See the sample programs for examples of the two transactional parameters in stored procedures that are written in C and SQL.

Related tasks:

- "Writing stored procedures to manipulate source data for Q replication" on page 108

Related reference:

- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)" on page 465
- "Samples to set up Q replication and event publishing (z/OS)" on page 467

Stored procedure parameters that map to source columns for Q replication

After the parameters in the stored procedure that identify the transaction (`src_commit_lsn` and `src_trans_time`), declare the next parameters in the stored procedure as IN parameters that take in the source data from each column. You can list the additional parameters that map to columns in any order.

After the operational and transactional parameters, include only parameters in the stored procedure that map to the TARGET_COLNAME values in the IBMQREP_TRG_COLS table. No other additional parameters are allowed in the stored procedure, even if they are declared as OUT or INOUT parameters.

The names that are listed in the TARGET_COLNAME column must be the same names as the stored procedure parameters. Table 10 shows an example of how the source column names correspond to parameter names in the IBMQREP_TRG_COLS table. The table shows only the three relevant columns from the control table.

Table 10. Source column names and names of parameters in the stored procedure that are stored in the IBMQREP_TRG_COLS table

...	SOURCE_COLNAME	TARGET_COLNAME	IS_KEY	...
	Col1	Parm1	Y	
	Col2	Parm2	Y	
	Col3	Parm3	N	
	Col4	Parm4	N	
	Col5	Parm5	N	

You must declare some parameters in the stored procedure to be part of the key. The parameters that you declare as target key columns must map to key columns at the source; the source and target keys must match. The Q Apply program uses these key columns to correctly order transactions that deal with the same rows.

If the parameter maps to a key column in the source table, then you must perform the following actions:

- Declare the corresponding before-value column as a parameter.
- Ensure that the parameter for the key's before value appears directly in front of the parameter for the key's after value.
- Start the name of the parameter for the key's before value with 'X' and then the parameter name. For example, for the key column named `Parm1`, the parameter for the before value is `XParm1`.

Do not declare parameters for before values of non-key columns.

Tip: If you have many source columns that you are mapping to parameters in the stored procedure, consider naming the parameters with the names of the source columns. If the parameters are named identically to the source columns to which they correspond, then the Replication Center is able to automatically map the source column names to the target parameter names.

Example of a valid signature of a stored procedure with key parameters: In this example, `Parm1` and `Parm2` are parameters that map to key columns.

```
CREATE PROCEDURE storedprocedure_name(
                                      INOUT operation integer,
                                      IN suppression_ind VARCHAR(size),
                                      IN src_commit_lsn CHAR(10),
                                      IN src_trans_time TIMESTAMP,
                                      XParm1,
                                      Parm1,
                                      XParm2,
                                      Parm2,
                                      Parm3,
                                      Parm4,
                                      Parm5
                                     )
```

Example of an invalid stored procedure with key parameters: In this invalid example, `Parm1` and `Parm2` are parameters that map to key columns. This example is invalid because there is no parameter that accepts the before value of the key parameter `Parm2`.

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

CREATE PROCEDURE storedprocedure_name(
                                     INOUT operation integer,
                                     IN suppression_ind VARCHAR(size) ,
                                     IN src_commit_lsn CHAR(10),
                                     IN src_trans_time TIMESTAMP,
                                     XParm1,
                                     Parm1,
                                     Parm2,
                                     XParm3,
                                     Parm3,
                                     Parm4,
                                     Parm5
                                    )
```

Tip: See the sample programs for examples of parameters that map to source columns in stored procedures that are written in C and SQL.

Related tasks:

- "Writing stored procedures to manipulate source data for Q replication" on page 108

Related reference:

- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)" on page 465
- "Samples to set up Q replication and event publishing (z/OS)" on page 467

Chapter 10. Setting up replication from sources to targets (multidirectional)

Setting up replication from sources to targets (multidirectional)—Overview

With Q replication, you can replicate data between tables on two or more servers.

The following topics describe the two types of multidirectional replication (bidirectional or peer-to-peer) and how to set up bidirectional or peer-to-peer replication:

- "Bidirectional replication"
- "Peer-to-peer replication" on page 120
- "Bidirectional replication versus peer-to-peer replication" on page 125
- "Creating Q subscriptions for bidirectional replication" on page 127
- "Creating Q subscriptions for peer-to-peer replication—Overview" on page 135
- "Starting bidirectional or peer-to-peer replication with two servers" on page 139
- "Stopping bidirectional or peer-to-peer replication with two servers" on page 140
- "Starting replication in a peer-to-peer group with three or more servers" on page 140
- "Stopping replication in a peer-to-peer group with three or more servers" on page 141

Related concepts:

- "Setting up replication from sources to targets (unidirectional)—Overview" on page 81

Bidirectional replication

Bidirectional replication is a Q replication configuration that has the following characteristics:

- Replication occurs between tables on two servers. 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.
- 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.

In bidirectional replication, you replicate copies of tables between two servers. 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.

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.

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 11 on page 119 shows bidirectional replication of one logical table between two servers. For one logical table, you need two Q subscriptions and two replication queue maps.

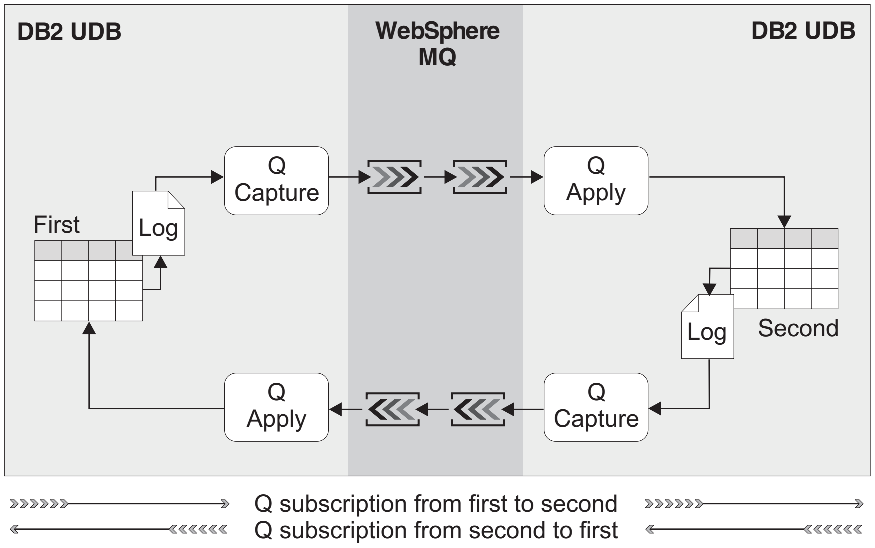


Figure 11. Q subscriptions between two copies of a table for bidirectional replication. Changes are replicated from each copy of the table to the other copy of that table over WebSphere MQ queues.

Conflict detection in bidirectional replication

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

For example, imagine a scenario in which applications on one system make changes to tables in a server (SERVER_RED) and that server replicates those changes to identical tables in a server (SERVER_GREEN) on a standby system. The first system fails, at which time your applications start using the tables on SERVER_GREEN. When the first system comes back online, you want to replicate changes from SERVER_GREEN to SERVER_RED. However, when the first system shut down, it could have failed to replicate some data to the second system. That data, which is now old, should be replaced by the data replicated from SERVER_GREEN. When you replicate the new data, the Q Apply program for SERVER_RED recognizes the conflicts and forces the changes that come from SERVER_GREEN to SERVER_RED.

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

Related concepts:

- "Setting up replication from sources to targets (multidirectional)—Overview" on page 117
- "Q subscriptions" on page 5
- "Replication queue maps" on page 6

- “Options for conflict detection (bidirectional replication)” on page 129

Related tasks:

- “Creating Q subscriptions for bidirectional replication” on page 127

Peer-to-peer replication

Peer-to-peer replication (also known as multimaster replication) is a configuration that has the following characteristics:

- Replication occurs between tables on two or more servers.
- Updates on any one server are replicated to all other associated 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).

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

The following topics describe the number of replication queue maps and Q subscriptions that are needed for peer-to-peer replication and how peer-to-peer replication handles referential integrity:
- “Replication objects for peer-to-peer replication with two servers”
- “Replication objects for peer-to-peer replication with three or more servers” on page 122
- “Conflict resolution and referential integrity” on page 123

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 12 on page 122 shows peer-to-peer replication of one logical table between two servers. For one logical table replicated between two servers, you need two Q subscriptions: one to replicate data from peer table A to peer table B, and one to replicate data from peer table B to peer table A. You also need at least two replication queue maps.

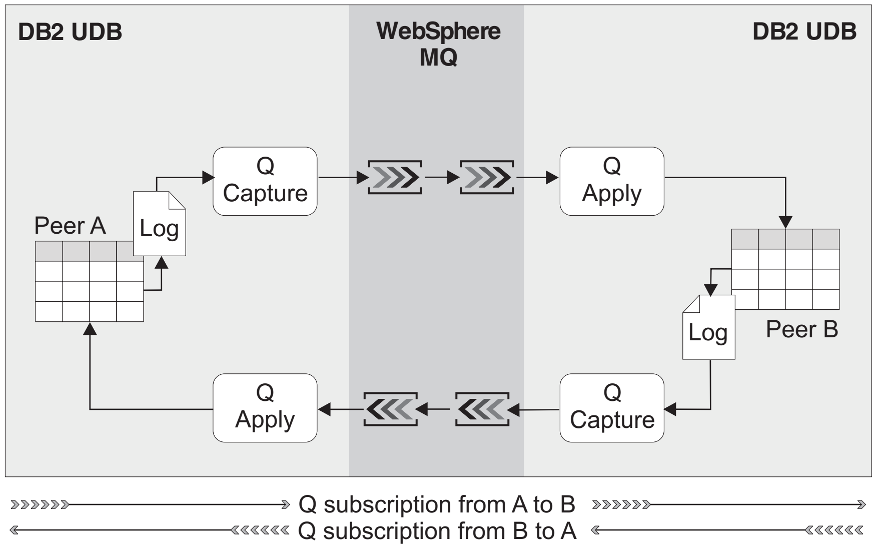


Figure 12. Q subscriptions in peer-to-peer replication with two servers. Changes are replicated from each copy of the table to the other copy of that table over WebSphere MQ queues.

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

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

Number of replication queue maps

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

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

Number of Q subscriptions

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

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

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

Figure 13 shows peer-to-peer replication of one logical table between three servers. In this case, you need six Q subscriptions: two going between each pair of servers. You also need at least six replication queue maps.

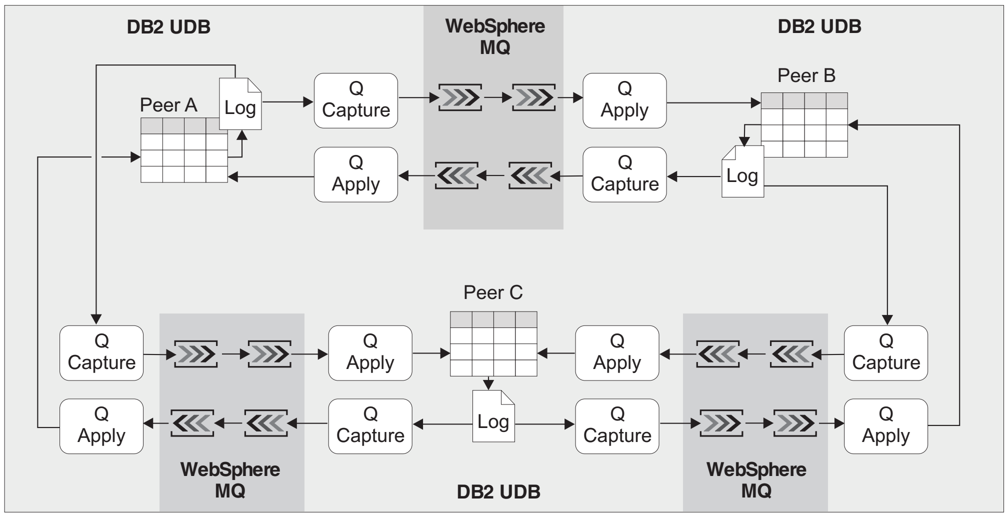


Figure 13. Q subscriptions in peer-to-peer replication with three servers. Changes are replicated from each copy of the table to all other copies of that table over WebSphere MQ queues.

Conflict resolution and referential integrity

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

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

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

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

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

Related concepts:

- "Setting up replication from sources to targets (multidirectional)—Overview" on page 117
- "Creating Q subscriptions for peer-to-peer replication—Overview" on page 135
- "Q replication" on page 3
- "Q subscriptions" on page 5
- "Replication queue maps" on page 6

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.

Bidirectional replication versus peer-to-peer replication for high-availability scenarios

If you are configuring two servers for a high-availability scenario, then you might want to choose either bidirectional or peer-to-peer replication, depending on your business needs. Both types can be used in such a scenario. You should consider the following trade-offs:

- How quickly the secondary server can take over in case the primary server fails, or how quickly you can switch back to the primary server
- The need for manual or automated procedures to control the process of taking over
- The overhead required by each method

Bidirectional replication for high-availability scenarios

You might want to choose bidirectional replication for your high availability scenario if you want the change with the most recent timestamp to win when a conflict occurs.

Recommendation: If you choose bidirectional replication for high availability, set up your configuration in the following way:

- Set up a primary server that is available for read and write applications.
- Set up a secondary server that is available only for read applications.

- Set the conflict rule to force for the primary server so that the primary server is the designated loser. Any conflicts coming from the secondary server are forced onto the primary server.
- Set the conflict rule to ignore for the secondary server so that the secondary server is the designated winner. Any conflicts coming from the primary server are ignored by the secondary server.

With the configuration set up this way, the primary server is updated by applications and replicates those changes to the secondary server, which is not typically being updated by applications. In the event that the primary server becomes unavailable, redirect applications to update the secondary server. When the primary server is available again, the more recent changes from the secondary server are replicated back to the primary server, and those changes from the secondary server overwrite the primary server's older changes.

While the primary server is unavailable, some data might be stuck on the failed server because it had not yet been replicated before the server failed. This data will be applied to the secondary server later, when the primary server is available again. Other replicated changes might be in the receive queue at the secondary server but not yet applied to the copies of the tables at the secondary server. These changes might conflict with new changes that are made when applications are redirected to the secondary server. If there are collisions between the queued data and the more recent changes on the secondary server, then the new redirected application activity wins, because the secondary server is set to ignore conflicts from the primary server.

It is possible to implement a procedural takeover that requires the receive queue on the secondary server to be emptied before traffic is redirected to the secondary server. Therefore, you eliminate the potential for data collisions during failover. When you switch back to the primary server, the old data that was not yet captured from the primary server is then captured and applied to the secondary server, and any collisions lose because the most recent changes are on the secondary server, where applications were redirected. The data that was captured at the secondary server is then applied to the primary server, and again this newer data will win any collisions.

Before you redirect applications back to the primary server, you must take steps to avoid the case where new changes from applications at the primary server start losing to older transactions that occurred at the secondary server. Quiesce the database activity at the secondary server, and ensure that all data changes are applied at the primary server.

Peer-to-peer replication for high-availability scenarios

If you choose peer-to-peer replication for high availability, then all servers are available for read and write at any time. (More than two servers can be configured for peer-to-peer replication.) Therefore, if one server becomes unavailable, then the other servers are immediately available to take over or switch back. This configuration provides for the most robust conflict detection. However, you must evaluate this ease of use, lack of outage time, and robust conflict detection against the additional overhead that the system incurs by the extra versioning columns on each copy of the replicated table and the triggers that are required to maintain those versioning columns.

Considerations for conflict detection in high-availability scenarios

Even the highest level of value-based conflict detection (checking all columns) in bidirectional replication is not as robust as version-based conflict detection in peer-to-peer replication. Some situations might result in a conflict not being detected.

Recommendation: If you expect conflicts by application design, then choose a peer-to-peer configuration.

When application developers design an application that involves distributed copies of tables that can be updated at any server, the developers must fully explore the potential for conflicts, the impact of conflicts, and how conflicts will be resolved. Because conflicts can result in loss of durability of a transaction, applications should be designed to minimize the potential for conflicts.

Important: When the Q Apply program detects a conflict for a given row, it acts only on that row. In either bidirectional or peer-to-peer replication, the conflicting row is either accepted as a change to the target, or it is ignored and not applied. Because the goal of peer-to-peer replication is to provide a convergent set of copies of a database table, the conflicting row is acted upon, not the entire transaction. The practice of rejecting or accepting whole transactions is likely to quickly lead to a set of database copies that do not converge.

The Q Apply program reports all conflicting rows in the IBMQREP_EXCEPTIONS table. In a peer-to-peer configuration, which might include several servers, Q replication attempts to report the conflicting row only once. But in some cases a conflict might show up in the IBMQREP_EXCEPTIONS table on more than one server. These duplications are easy to see because the data values and versioning information are identical. To see the complete conflict activity for a peer-to-peer or bidirectional configuration, look at the IBMQREP_EXCEPTIONS tables of all servers.

Related concepts:
- “Bidirectional replication” on page 117
- “Peer-to-peer replication” on page 120
- “Options for conflict detection (bidirectional replication)” on page 129

Creating Q subscriptions for bidirectional replication

Creating Q subscriptions for 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 from the second copy are replicated back to the first copy. In bidirectional replication, all rows and columns are replicated, and the column names in each copy of the table must match.

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 Replication Center, then the Replication Center creates both Q subscriptions at one time.

Prerequisites:

Before you create Q subscriptions for bidirectional replication, you must perform the following actions:

- Plan how you want to group replication queue maps and Q subscriptions. See“Grouping replication queue maps and Q subscriptions” on page 83 for details.
- On the server that has the first copy of the table, create the control tables for the Q Capture and Q Apply programs. See “Creating control tables for the Q Capture and Q Apply programs” on page 76 for details.

  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 a schema, and the Q Capture and Q Apply programs that are on SERVER_GREEN must both have a schema.
- On the server that has the second copy of the table, create the control tables for the Q Capture and Q Apply programs. See “Creating control tables for the Q Capture and Q Apply programs” on page 76 for details.

  Important: The control tables for the Q Capture and Q Apply programs that are on this server must have the same schema name.
- 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 define Q subscriptions.) See “Creating replication queue maps” on page 84 for details.

Restrictions:

- A table for a Q subscription in bidirectional replication cannot be a view.
- A target for bidirectional replication cannot be a stored procedure.
- For each server, a single table can participate in only one topology for bidirectional replication. For example, if Server 1 replicates a table using bidirectional replication to Server 2, then that table on either Server 1 or Server 2 cannot be replicated using bidirectional replication to Server 3.
- Because before values of LOB columns are not replicated in bidirectional replication, conflicts for LOB columns are not detected.

Procedure:

To create Q subscriptions for bidirectional replication between two copies of a table:

1. Open the wizard and specify that you are creating Q subscriptions for bidirectional replication:
   a. Open the Create Q Subscriptions wizard from the Replication Center.

      To open the wizard, do one of the following:

      - Expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.
      - Expand the Q Apply schema that identifies the Q Apply program that you want to apply changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.

      See the online help for details.

   b. On the Replication page, specify that you want bidirectional replication.
2. On the Servers page:
   - Specify information about the first server and schema (the one that contains the tables that you want to replicate).
   - Specify information about the second server and schema. If copies of the tables that you want to replicate already exist, then the Replication Center uses them. If the tables do not exist, then the Replication Center creates them.
   - Specify the replication queue maps that you want to use. If an appropriate replication queue map does not exist, create a new one. See "Creating replication queue maps" on page 84 for details.
3. On the Source Tables page, select the source tables that you want to replicate from.
4. 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, and, if one does not exist, then the object is created.
5. On the Conflicts page, specify how the Q Apply program handles conflicts. See "Options for conflict detection (bidirectional replication)" for details.
6. On the Errors page, specify how the Q Apply program responds to errors. See "Error options for Q replication" on page 103 for details.
7. On the Loading Target Table page:
   a. Specify the options for loading the target tables. See "Options for loading target tables for Q replication—Overview" on page 143 for details.
   b. Specify whether the Q subscriptions are active as soon as they are created.
8. On the Review Q Subscriptions page, confirm that each Q subscription is valid:
   - If you want to change anything about a Q subscription, modify the properties of that Q subscription.
   - If any of the Q subscriptions are missing information, modify the properties to add the needed information.
9. On the Summary page, click **Finish**.

Related concepts:
- "Bidirectional replication" on page 117
- "Setting up replication from sources to targets (multidirectional)—Overview" on page 117
- "Creating Q subscriptions for peer-to-peer replication—Overview" on page 135
- "Q subscriptions" on page 5
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "Target object profiles for Q replication" on page 87

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.

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

The following sections describe your options for conflict detection in bidirectional replication and the results of different combinations of conflict options:
- "How the Q Apply program checks for conflicts"
- "How conflicts are resolved at each server" on page 131
- "Outcomes of different choices for checking and resolving conflicts" on page 131

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 11 on page 132 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 11. Outcomes of different combinations of options for conflict detection

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

Table 11. Outcomes of different combinations of options for conflict detection (continued)

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

Table 11. Outcomes of different combinations of options for conflict detection (continued)

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

Related concepts:

- "Bidirectional replication" on page 117

Related tasks:

- "Creating Q subscriptions for bidirectional replication" on page 127

Creating Q subscriptions for peer-to-peer replication

Creating Q subscriptions for peer-to-peer replication—Overview

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).

The following topics explain how to create Q subscriptions and the options to pick based on your needs:

- “Creating Q subscriptions for peer-to-peer replication with two servers”
- “Creating Q subscriptions for peer-to-peer replication with three or more servers” on page 137
- “Error options for Q replication” on page 103

Related concepts:

- “Setting up replication from sources to targets (multidirectional)—Overview” on page 117
- “Peer-to-peer replication” on page 120
- “Q subscriptions” on page 5

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.

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 Replication Center, then the Replication Center creates both Q subscriptions at one time.

Prerequisites:

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. See “Creating control tables for the Q Capture and Q Apply programs” on page 76 for details.

  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 a schema, and the Q Capture and Q Apply programs that are on SERVER_GREEN must both have a 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.) See “Creating replication queue maps” on page 84 for details.

Restrictions:

- A table for a Q subscription in peer-to-peer replication cannot be a view.
- A target for peer-to-peer replication cannot be a stored procedure.

Procedure:

To create Q subscriptions for peer-to-peer replication with two servers:

1. Open the wizard and specify that you are creating Q subscriptions for peer-to-peer replication with two servers:
   a. Open the Create Q Subscriptions wizard from the Replication Center.

      To open the wizard, do one of the following:
      - Expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.
      - Expand the Q Apply schema that identifies the Q Apply program that you want to apply changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.

      See the online help for details.
   b. On the Replication page, specify that you want peer-to-peer replication with two servers.
2. On the Servers page:
   - Specify information about the first peer server and schema (the one that contains the tables that you want to replicate).
   - Specify information about the second peer server and schema. If copies of the tables that you want to replicate already exist, then the Replication Center uses them. If the tables do not exist, then the Replication Center creates them.
   - Specify the replication queue maps that you want to use. If an appropriate replication queue map does not exist, create a new one. See “Creating replication queue maps” on page 84 for details.
3. On the Source Tables page, select the source tables that you want to replicate from.
4. 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, and, if one does not exist, then the object is created.
5. On the Errors page, specify how the Q Apply program responds to errors. See “Error options for Q replication” on page 103 for details.
6. On the Loading Target Table page:
   a. Specify the options for loading the target tables. See “Options for loading target tables for Q replication—Overview” on page 143 for details.
   b. Specify whether the Q subscriptions are active as soon as they are created.
7. On the Review Q Subscriptions page, confirm that each Q subscription is valid:

- If you want to change anything about a Q subscription, modify the properties of that Q subscription.
- If any of the Q subscriptions are missing information, modify the properties to add the needed information.

8. On the Summary page, click **Finish**.

Related concepts:

- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "Target object profiles for Q replication" on page 87

Related tasks:

- "Creating Q subscriptions for peer-to-peer replication with three or more servers" on page 137

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.

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 using the Replication Center, then the Replication Center creates all necessary Q subscriptions at one time.

Prerequisites:

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. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.

  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, a server named SERVER_GREEN, and a server named SERVER_BLUE, then the Q Capture and Q Apply programs that are on SERVER_RED must both have a schema, the Q Capture and Q Apply programs that are on SERVER_GREEN must both have a schema, and the Q Capture and Q Apply programs that are on SERVER_BLUE must both have a 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 define Q subscriptions.) See "Creating replication queue maps" on page 84 for details.

Restrictions:

- A table for a Q subscription in peer-to-peer replication cannot be a view.
- A target for peer-to-peer replication cannot be a stored procedure.

Procedure:

To create Q subscriptions for peer-to-peer replication with three or more servers:

1. Open the wizard and specify that you are creating Q subscriptions for peer-to-peer replication with three or more servers:
   a. Open the Create Q Subscriptions wizard from the Replication Center.

      To open the wizard, do one of the following:

      - Expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.
      - Expand the Q Apply schema that identifies the Q Apply program that you want to apply changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.

      See the online help for details.
   b. On the Replication page, specify that you want peer-to-peer replication with three or more servers.
2. On the Servers page:
   - Specify information about the first peer server and schema (the one that contains the tables that you want to replicate)
   - Specify information about each additional peer server and schema. If copies of the tables that you want to replicate already exist on the server, then the Replication Center uses them. If the tables do not exist, then the Replication Center creates them.

     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 have already selected, and from each existing server back to the new server.
   - Specify the replication queue maps that you want to use. If an appropriate replication queue map does not exist, create a new one. See "Creating replication queue maps" on page 84 for details.
3. On the Source Tables page, select the source tables that you want to replicate from.
4. On the Target Tables page, review the target object profile for the each server other than the one that contains the existing (base) tables. Modify the profile if necessary so that the target tables for the Q subscriptions meet your needs.

   For example, if SERVER_RED contains the existing tables and the peer-to-peer configuration includes servers named SERVER_RED, SERVER_BLUE, and SERVER_GREEN, then the Replication Center shows you the target object profiles for SERVER_BLUE and SERVER_GREEN.
5. On the Errors page, specify how the Q Apply program responds to errors. See "Error options for Q replication" on page 103 for details.
6. On the Loading Target Table page, specify the options for loading the target tables. See "Options for loading target tables for Q replication—Overview" on page 143 for details.

   When you create Q subscriptions for peer-to-peer replication with three or more servers, then the Q subscriptions are always created in an inactive state, and you must activate them.

7. On the Review Q Subscriptions page, confirm that each Q subscription is valid:
   - If you want to change anything about a Q subscription, modify the properties of that Q subscription.
   - If any of the Q subscriptions are missing information, modify the properties to add the needed information.
8. On the Summary page, click **Finish**.

Related concepts:
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "Target object profiles for Q replication" on page 87

Related tasks:
- "Creating Q subscriptions for peer-to-peer replication with two servers" on page 135

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

By default, when you create Q subscriptions for bidirectional or peer-to-peer replication with two servers, the new Q subscriptions are automatically activated when you start or reinitialize the Q Capture program. In the following situations, you must manually start replication in bidirectional or peer-to-peer replication with two servers:
- You created the Q subscriptions so that they would not be automatically activated when they are new.
- You stopped replication of a logical table and want to start replication again. (Q subscriptions are only automatically activated when they are new.)

Prerequisites:
- Both Q subscriptions for the logical table must be defined.
- Both Q subscriptions must be in I (inactive) state.

Procedure:

To manually start replication for a logical table in bidirectional or peer-to-peer replication with two servers, activate one of the two Q subscriptions for the logical table. The Q Capture and Q Apply programs automatically activate the other Q subscription.

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

Related concepts:
- "Options for loading target tables for Q replication—Overview" on page 143

Related tasks:
- "Creating Q subscriptions for bidirectional replication" on page 127
- "Activating Q subscriptions or XML publications" on page 221
- "Creating Q subscriptions for peer-to-peer replication with two servers" on page 135

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.

Prerequisites:

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

Restrictions:

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

Procedure:

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

Related tasks:

- "Deactivating Q subscriptions or XML publications" on page 222

Starting replication in a peer-to-peer group 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 activate the group of Q subscriptions to start replication.

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

Activating a peer-to-peer group with three or more servers is a staged process. First you activate 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.

Prerequisites:

The Q Capture and Q Apply programs must be running at all servers in the group.

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. Activate one of the two Q subscriptions for a logical table between the two servers.

   The Q Capture and Q Apply programs will automatically activate the other Q subscription for this logical table between the two servers.

   Note: If the Q subscriptions specify a load phase, the Q subscription that you activate must be the Q subscription whose source table is the 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. Activate 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 will activate 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 will also be activated.

   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.

Important: 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.

Related concepts:
- "Options for loading target tables for Q replication—Overview" on page 143

Related tasks:
- "Activating Q subscriptions or XML publications" on page 221
- "Creating Q subscriptions for peer-to-peer replication with three or more servers" on page 137

Stopping replication in a peer-to-peer group 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, and without stopping replication of other logical tables in the peer-to-peer group.

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

Prerequisites:

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

Procedure:

To stop replication of a logical table at one server:
1. Choose a server in the group that is actively replicating the logical table.
2. Deactivate the Q subscription that specifies this server as its source, and the server where you want replication to stop as its target.

   The Q Capture and Q Apply programs automatically deactivate the other Q subscriptions for this logical table between the remaining servers.

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 deactivated.

Related tasks:
- "Deactivating Q subscriptions or XML publications" on page 222

Chapter 11. Options for loading target tables for Q replication

Options for loading target tables for Q replication—Overview

When you create a Q subscription, you can choose among the following 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.

The following topics provide details about the load options.

- "Recommendations for loading target tables for Q replication"
- "Utilities used for automatic load option for Q replication" on page 145
- "Manual load option for Q replication" on page 148
- "No load option for Q replication" on page 150
- "Load options for different types of Q replication" on page 150
- "Automatic load considerations for z/OS platform" on page 146

Related concepts:

- "Creating Q subscriptions for unidirectional replication—Overview" on page 86

Related tasks:

- "Creating Q subscriptions for bidirectional replication" on page 127
- "Creating Q subscriptions for peer-to-peer replication with two servers" on page 135
- "Creating Q subscriptions for peer-to-peer replication with three or more servers" on page 137
- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Creating a single Q subscription for unidirectional replication" on page 88

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**. See the online help for details.

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.

Related concepts:

- "Options for loading target tables for Q replication—Overview" on page 143

Automatic load option for Q replication

Automatic load option for Q replication—Overview

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.

The following topics provide detail about automatic loads:

- Utilities used for automatic load option for Q replication
- Automatic load considerations for z/OS platform
- Specifying nicknames for the automatic load option for Q replication

Related concepts:

- "Load options for different types of Q replication" on page 150
- "Options for loading target tables for Q replication—Overview" on page 143
- "Manual load option for Q replication" on page 148

- “No load option for Q replication” on page 150

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.

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.

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 file is created in the path specified by the **apply_path** parameter.

Table 12 shows the load utilities that are available depending on your DB2 UDB version and operating system. If you do not specify a load utility, the Q Apply program will use LOAD from CURSOR by default on DB2 UDB Version 8.1 or later for Linux, UNIX, and Windows operating systems, and for DB2 UDB Version 7.1 for z/OS™ or later. LOAD from CURSOR is not supported for DB2 UDB Version 6 for z/OS. If you specify a load utility that is not available, the Q Apply program deactivates the Q subscription.

Table 12. Load options for Q replication

DB2 version	Operating system	Load options
Version 8	Linux, UNIX, Windows	• LOAD from CURSOR • EXPORT and IMPORT utilities • EXPORT and LOAD utilities
	z/OS	LOAD from CURSOR
Version 7	z/OS	LOAD from CURSOR
Version 6	z/OS	User is responsible for loading, so manual load or no load are the only available options.

Related concepts:

- “Automatic load considerations for z/OS platform” on page 146
- “Automatic load option for Q replication—Overview” on page 144

Automatic load considerations for z/OS platform

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 the following issues:

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 UDB 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 Universal Database for OS/390 and 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 activate 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. Activate 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.

Related concepts:

- "Utilities used for automatic load option for Q replication" on page 145
- "Automatic load option for Q replication—Overview" on page 144
- "Operating the Q replication and event publishing programs by using system services (z/OS)—Overview" on page 293

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.

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.

Prerequisites:

- 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.

Procedure:

To specify nicknames for unidirectional Q subscriptions:

If you are creating a single Q subscription:
In the Replication Center, use the Loading Target Tables page of the Create Q Subscriptions wizard 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, do one of the following:

- Expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.
- Expand the Q Apply schema that identifies the Q Apply program that you want to apply changes for the Q subscription. Right-click the **Q Subscriptions** folder and select **Create**.

See the online help for details.

Related concepts:

- “Server definitions and server options” in the *Federated Systems Guide*
- “Utilities used for automatic load option for Q replication” on page 145
- “Automatic load option for Q replication—Overview” on page 144

Related tasks:

- “Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows)” on page 71

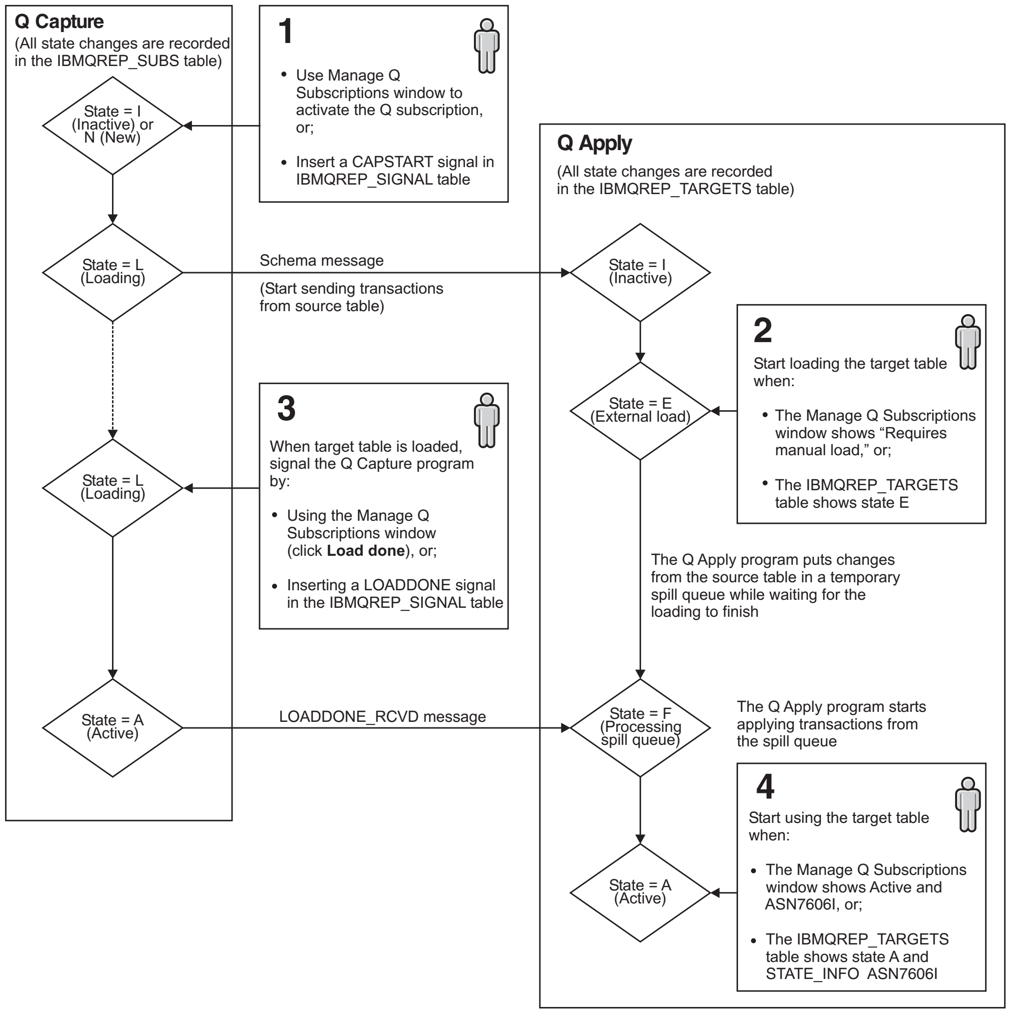
Related reference:

- “Nickname options for federated systems” in the *Federated Systems Guide*
- “User mapping options for federated systems” in the *Federated Systems Guide*
- “Wrapper options for federated systems” in the *Federated Systems Guide*

Manual load option for Q replication

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.

Figure 14 illustrates the stages of the manual loading process.



Q Capture
(All state changes are recorded in the IBMQREP_SUBS table)
State = I (Inactive) or N (New)
State = L (Loading)
State = L (Loading)
State = A (Active)
1
Use Manage Q Subscriptions window to activate the Q subscription, or;
Insert a CAPSTART signal in IBMQREP_SIGNAL table
Schema message
(Start sending transactions from source table)
3
When target table is loaded, signal the Q Capture program by:
Using the Manage Q Subscriptions window (click Load done), or;
Inserting a LOADDONE signal in the IBMQREP_SIGNAL table
LOADDONE_RCVD message
Q Apply
(All state changes are recorded in the IBMQREP_TARGETS table)
State = I (Inactive)
State = E (External load)
2
Start loading the target table when:
The Manage Q Subscriptions window shows "Requires manual load," or;
The IBMQREP_TARGETS table shows state E
The Q Apply program puts changes from the source table in a temporary spill queue while waiting for the loading to finish
State = F (Processing spill queue)
The Q Apply program starts applying transactions from the spill queue
4
Start using the target table when:
The Manage Q Subscriptions window shows Active and ASN7606I, or;
The IBMQREP_TARGETS table shows state A and STATE_INFO ASN7606I
State = A (Active)

Figure 14. Stages of the manual loading process. In a manual load, you must take action at some stages for the process to complete: 1) Activate the Q subscription; 2) Watch for ″Requires manual load″ in the Replication Center (or E in the STATE column of the IBMQREP_TARGETS table) before loading the target table; 3) Notify the Q Capture program when the load is complete; and 4) Avoid using the target table until you see ″Active″ and ″ASN7606I″ in the Replication Center (or A in the STATE column and ASN7606I in the STATE_INFO column of the IBMQREP_TARGETS table).

The manual loading process is as follows:

1. When a Q subscription is activated, the Q Capture program sends a `subscription schema` message that indicates that the target table will be manually loaded. The following changes occur:
   - The Q Capture program changes the Q subscription state from I (inactive) or N (new) to L (loading) in the IBMQREP_SUBS control table.
   - The Q Apply program changes the Q subscription state from I (inactive) to E (external load) in the IBMQREP_TARGETS control table.

- The Manage Q Subscriptions window in the Replication Center shows the state as ″Requires manual load.″ 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**.
- The Q Apply program drops any referential integrity constraints that are defined on the target table.

2. After the Manage Q Subscriptions window shows ″Requires manual load,″ or if a SELECT statement against the IBMQREP_TARGETS table verifies that the value in the STATE column is E, you can start loading the target table with your chosen utility.
3. While the target table is being loaded, the Q Capture program sends transactions from the source table with both before and after values. The Q Apply program puts these transactions in a temporary spill queue.
4. After the target table is loaded, you notify the Q Capture program that the load is complete. You can use one of the following methods:
   - Use the Manage Q Subscriptions window in the Replication Center to indicate that the load is done.
   - Insert a LOADDONE signal into the IBMQREP_SIGNAL table, as follows:

   ```
   insert into schema.IBMQREP_SIGNAL(
       SIGNAL_TIME,
       SIGNAL_TYPE,
       SIGNAL_SUBTYPE,
       SIGNAL_INPUT_IN,
       SIGNAL_STATE
   ) values (
        CURRENT TIMESTAMP,
       'CMD',
       'LOADDONE',
       'subname',
       'P');
   ```

   Where *schema* identifies the Q Capture program that you want to signal, and *subname* is the name of the Q subscription for which you are performing a manual load.
5. After the Q Capture program is notified that a manual load is complete, it changes the state of the Q subscription to A (active) in the IBMQREP_SUBS table, and begins using any column subsetting options that are defined for the Q subscription. The Q Capture program sends a `load done received` message to the Q Apply program.
6. The Q Apply program changes the state of the Q subscription to F (processing spill queue) and starts applying transactions from the spill queue. When the Q Apply program is finished, it deletes the spill queue.
7. 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.
8. The Q Apply program changes the Q subscription state to A (active) and the STATE_INFO column to ASN7606I in the IBMQREP_TARGETS table and begins applying transactions from the receive queue. The Manage Q Subscriptions window shows the state as ″Active.″

Related concepts:

- "Options for loading target tables for Q replication—Overview" on page 143

Related tasks:

- "Creating Q subscriptions for bidirectional replication" on page 127

- “Activating Q subscriptions or XML publications” on page 221
- “Creating Q subscriptions for peer-to-peer replication with two servers” on page 135
- “Creating Q subscriptions for peer-to-peer replication with three or more servers” on page 137
- “Creating multiple Q subscriptions for unidirectional replication” on page 90
- “Creating a single Q subscription for unidirectional replication” on page 88

Related reference:
- “IBMQREP_SIGNAL table” on page 382

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 would 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.

Related concepts:
- “Options for loading target tables for Q replication—Overview” on page 143

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 151
- “Load options for peer-to-peer replication with three or more servers” on page 152

Load options for unidirectional replication

This section explains the load options for unidirectional replication.

Load options
All load options are available.

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 activated automatically when the Q Capture program starts, then you must activate the Q subscription for the load to begin.

Example

Assume that you want to replicate data in one direction from the MANAGERS table at Server A to the MANAGERS 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 MANAGERS 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 MANAGERS table at Server A to the MANAGERS 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 deactivate the Q subscription group and then activate it), you specify which server will be the load source when you decide which of the two Q subscriptions to activate. The source table for the Q subscription that you activate 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 be activated 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 activated automatically, you must take the following actions:

- Start the Q Capture and Q Apply programs at both servers.
- Activate 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. Activate 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 EMPLOYEES 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 EMPLOYEES table at Server A to the EMPLOYEES 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. Activate 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. Activate 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 are activated automatically. You must manually activate 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. Activate one of the two Q subscriptions between the servers. The source table for the Q subscription that you activate 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. Activate 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 activate 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 activate 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 DEPARTMENTS table. You want the Q Apply program to handle the loading and use the best available load utility. You will use Server A as the load source for the tables at both Server B and Server C.

1. You create six Q subscriptions, DEP_A2B, DEP_B2A, DEP_A2C, DEP_C2A, DEP_B2C, and DEP_C2B, all specifying an automatic load using the best available utility.
2. You start the Q Capture and Q Apply programs at Server A and Server B.
3. You activate the Q subscription DEP_A2B.

   The Q Apply program at Server B calls a utility to load the DEPARTMENTS table at Server B with data from the DEPARTMENTS 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 activate the Q subscription DEP_A2C.

The Q Apply program at Server C calls a utility to load the DEPARTMENTS 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.

Related concepts:

- "Options for loading target tables for Q replication—Overview" on page 143

Related tasks:

- "Creating Q subscriptions for bidirectional replication" on page 127
- "Creating Q subscriptions for peer-to-peer replication with two servers" on page 135
- "Creating Q subscriptions for peer-to-peer replication with three or more servers" on page 137
- "Creating multiple Q subscriptions for unidirectional replication" on page 90
- "Creating a single Q subscription for unidirectional replication" on page 88

Chapter 12. Setting up publishing from sources (event publishing)

Setting up publishing from sources (event publishing)—Overview

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 in XML format. You are then responsible for having an application of your choice retrieve those XML messages.

Tip: See the sample program for an example of a Web-based application that retrieves the XML messages that the Q Capture program publishes. The sample demonstrates how to use an XML publication in a business scenario.

The following topics describe event publishing and how to set it up:
- “Grouping publishing queue maps and XML publications”
- “Creating publishing queue maps” on page 156
- “Creating XML publications—Overview” on page 158

Related reference:
- “Samples to set up Q replication and event publishing (Linux, UNIX, Windows)” on page 465

Grouping publishing queue maps and XML publications

Before you define XML publications and publishing queue maps, you should first plan how you want to group them. Each XML publication identifies a single source table from which changes will be published in XML format. When you define an XML 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 XML publications, so you must decide which XML publications use the same publishing queue map to transport data.

When you plan how to group XML 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 XML publications. It stores these changes in memory until it reads the corresponding commit or abort record from the database log. It then sends information on committed transactions to all WebSphere MQ send queues that were defined for the XML publications.

Suggestions for grouping similar XML publications with publishing queue maps

Generally speaking, for tables that are involved in transactions with one or more applications, you should create XML publications for these tables so that they all share a common publishing queue map. Grouping similar XML 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 XML publications that have dependencies share the same publishing queue map. If you define XML 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 XML 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. Still higher throughput and failure isolation might be gained 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.

Related concepts:

- "Creating XML publications—Overview" on page 158
- "Publishing queue maps" on page 12
- "XML publications" on page 11

Related tasks:

- "Creating publishing queue maps" on page 156

Creating publishing queue maps

When you create XML publications, you specify which WebSphere MQ queue to send the data to by associating each XML publication with a publishing queue map. You can create a publishing queue map before you begin creating XML publications or as one of the steps while you are creating XML publications.

Each publishing queue map identifies the following options:

Send queue
The WebSphere MQ queue where the Q Capture program sends source data and informational messages.

Type of 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 XML message from the Q Capture program is called a *row operation message*.)
- Full transactions. (This type of XML 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 XML 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 resulting action if an error occurs at the send queue. You must decide what action the Q Capture program takes:

- Deactivates the XML publications that use the queue
- Stops running

The queue error action is used when a WebSphere MQ error occurs that is specific to this queue. Regardless of which option that you choose, the Q Capture program logs all error message in its diagnostic log and in the IBMQREP_CAPTRACE table.

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.

Prerequisites:

- Plan how you want to group publishing queue maps and XML publications. See "Grouping replication queue maps and Q subscriptions" on page 83 for details.
- On the server that contains the source tables for the XML publications, create the control tables for the Q Capture program. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- Ensure that you have defined the appropriate objects in WebSphere MQ. See "Required settings for WebSphere MQ objects" on page 56 for details.

Restrictions:

The same send queue cannot be used for both Q replication and event publishing because there is a queue-level attribute that specifies whether it transports compact messages (for Q replication) or XML messages (for event publishing).

Procedure:

To define a publishing queue map, use the Create Publishing Queue Map window in the Replication Center. To open the window, expand the Q Capture schema that identifies the Q Capture program that you want to use the queue map. Right-click the **Publishing Queue Maps** folder and select **Create**. See the online help for details.

Related concepts:
- "Publishing queue maps" on page 12

Related tasks:
- "Changing attributes of publishing queue maps" on page 195
- "Deleting publishing queue maps" on page 197
- "Creating replication queue maps" on page 84

Related reference:
- "Transaction message" on page 436
- "Row operation message" on page 445

Creating XML publications

Creating XML publications—Overview

With event publishing, you can publish changed rows or transactions from a source table to a user application by creating XML publications. Each XML 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 XML publications at one time.

Important: XML publications are separate objects from Q subscriptions. XML publications do not publish data to the Q Apply program, but to an application of your choice. XML 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 an XML publication.

The following topics explain how to create one or many XML publications:
- "Creating a single XML publication" on page 159
- "Creating multiple XML publications" on page 161

You can optionally set other properties if you want to further customize the XML publications. The following topics help you decide which options to pick based on your business needs:
- "Source columns for XML publications" on page 163
- "When the Q Capture program publishes a message for XML publications" on page 163
- "Search conditions to filter rows in XML publications" on page 164
- "Key columns for XML publications" on page 167

- "Options for including unchanged columns in messages for XML publications" on page 168
- "Options for including before values in messages for XML publications" on page 169

Related concepts:
- "Setting up publishing from sources (event publishing)—Overview" on page 155
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "XML publications" on page 11

Creating a single XML publication

By creating a single XML publication, you define how data is published in XML format from a source table to a send queue so that a user application of your choice can retrieve and use those XML messages.

Figure 15 shows how a single XML publication connects a source table to a WebSphere MQ send queue.

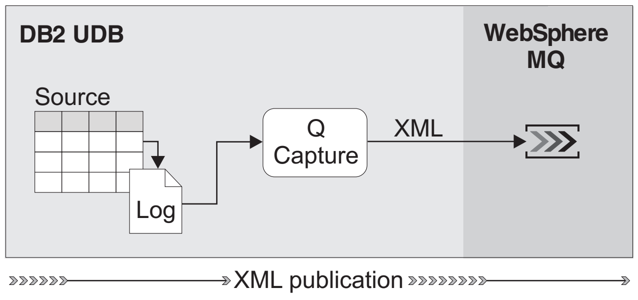


Figure 15. A single XML publication. Changes from a source table are published to a WebSphere MQ send queue in XML format.

Prerequisites:

Before you create an XML publication, you must do the following actions:
- Plan how you want to group publishing queue maps and XML publications. See "Grouping publishing queue maps and XML publications" on page 155 for details.
- On the server that contains the source table for the XML publication, create the control tables for the Q Capture program. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- Create a publishing queue map. (You can do this task before you create an XML publication or while you create an XML publication.) See "Creating publishing queue maps" on page 156 for details.

Restrictions:

A view cannot be a source for an XML publication.

Procedure:

To create an XML publication using one source table:
1. Open the Create XML Publications wizard from the Replication Center.

To open the wizard, expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the XML publication. Right-click the **XML Publications** folder and select **Create**. See the online help for details.

2. On the Server and Queue Map page:
   a. Specify information about the source server.
   b. Specify the publishing queue map that you want to use. If an appropriate publishing queue map does not exist, create a new one. See "Creating publishing queue maps" on page 156 for details.

      Important: Make sure that the publishing queue map that you select is defined with the type of message content that you want the Q Capture program to send for this XML publication (either committed row operations or committed transactions).
3. On the Source Tables page, select the source table that you want to publish changes from.
4. On the Columns and Rows page, specify what data to publish:
   a. Specify the columns that you want to publish:
      1) Use the Select Columns window to select a subset of the columns that you want to publish. See "Source columns for XML publications" on page 163 for details.
      2) If you choose to publish a subset of the columns that are in the source table, specify when the Q Capture program publishes a message. See "When the Q Capture program publishes a message for XML publications" on page 163 for details.
   b. Specify the key columns for XML publications. See "Key columns for XML publications" on page 167 for details.
   c. Specify the rows that you want to publish. See "Search conditions to filter rows in XML publications" on page 164 for details.
5. On the Message Content page, specify how the Q Capture program builds messages:
   a. Specify which (selected) columns to include when a message is sent. This option applies only to update operations. See "Options for including unchanged columns in messages for XML publications" on page 168 for details.
   b. Specify whether the Q Capture program publishes the before and after values for non-key columns when the column is updated. See "Options for including before values in messages for XML publications" on page 169 for details.
6. On the Review XML Publications pages, confirm that the XML publication is valid:
   - If you want to change anything about the XML publication, modify the properties of the XML publication.
   - If the XML publication is missing information, modify its properties so that it is complete.
7. On the Summary page, click **Finish**.

Related concepts:
- "Creating XML publications—Overview" on page 158
- "XML publications" on page 11

Related tasks:

- "Changing attributes of XML publications" on page 191
- "Deleting XML publications" on page 196
- "Creating multiple XML publications" on page 161

Creating multiple XML publications

To save time, you can define multiple XML publications at one time. After you define the attributes for all of the XML publications that you want to create, you can then change the properties of individual XML publications before the Replication Center creates them.

Tip: Create multiple XML publications at once if you have many to define and most will share the same attributes.

Figure 16 shows how multiple XML publications can use the same publishing queue map and Q Capture program.

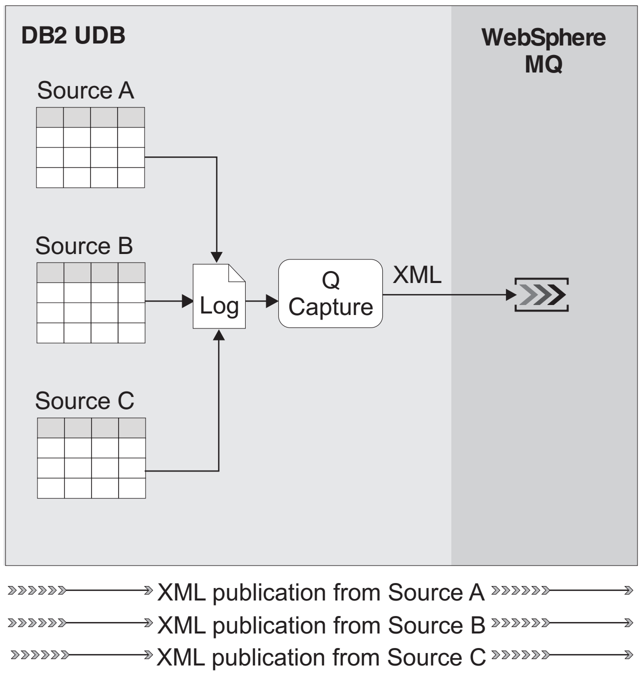

DB2 UDB
WebSphere MQ
Source A
Source B
Source C
Log
Q Capture
XML
XML publication from Source A
XML publication from Source B
XML publication from Source C

Figure 16. Multiple XML publications that use the same publishing queue map and Q Capture program. Changes from each source table are published to a WebSphere MQ send queue so that a user application can retrieve and use those messages.

Prerequisites:

Before you create XML publications, you must do the following actions:

- Plan how you want to group publishing queue maps and XML publications. See "Grouping publishing queue maps and XML publications" on page 155 for details.
- On the server that contains the source tables for the XML publications, create the control tables for the Q Capture program. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- Create a publishing queue map. (You can do this task before you create XML publications or while you define XML publications.) See "Creating publishing queue maps" on page 156 for details.

Restrictions:

A view cannot be a source for an XML publication.

Procedure:

To create XML publications using multiple source tables at one time:

1. Open the Create XML Publications wizard from the Replication Center.

   To open the wizard, expand the Q Capture schema that identifies the Q Capture program that you want to capture changes for the XML publication. Right-click the **XML Publications** folder and select **Create**. See the online help for details.
2. On the Server and Queue Map page:
   a. Specify information about the source server.
   b. Specify the publishing queue map that you want to use. If an appropriate publishing queue map does not exist, create a new one. See "Creating publishing queue maps" on page 156 for details.

      Important: Make sure that the publishing queue map that you select is defined with the type of message content that you want the Q Capture program to send for this XML publication (either committed row operations or committed transactions).
3. On the Source Tables page, select the source tables that you want to publish changes from.

   When you create multiple XML 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 XML publications, you can modify individual XML publications so that only a subset of the source columns and rows are published.
4. On the Message Contents page, specify how the Q Capture program builds messages:
   a. Specify which (selected) columns to include when a message is sent. This option applies only to update operations. See "Options for including unchanged columns in messages for XML publications" on page 168 for details.
   b. Specify whether the Q Capture program publishes the before and after values for non-key columns when the column is updated. See "Options for including before values in messages for XML publications" on page 169 for details.
5. On the Review XML Publications pages, confirm that each XML publication is valid:
   - If you want to change anything about an XML publication, modify the properties of that XML publication. For example, you can choose to publish only a subset of the rows or columns from the source table for a particular XML publication.
   - If any of the XML publications are missing information, modify the properties to add the needed information.
6. On the Summary page, click **Finish**.

Related concepts:

- "Creating XML publications—Overview" on page 158
- "XML publications" on page 11

Related tasks:
- "Changing attributes of XML publications" on page 191
- "Deleting XML publications" on page 196
- "Creating a single XML publication" on page 159

Source columns for XML publications

By default when you create XML 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 an XML 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 XML publication, then the Create XML 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 XML publications at one time, then, on the Review page of the Create XML Publications wizard, select the individual XML publication where you want to subset columns and edit the properties for that XML publication.

Important for LOB columns: If you select columns that contain LOB data types for an XML 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 XML publication.

Related concepts:
- "Creating XML publications—Overview" on page 158
- "Source columns for Q subscriptions (unidirectional replication)" on page 92

Related tasks:
- "Creating multiple XML publications" on page 161
- "Creating a single XML publication" on page 159

When the Q Capture program publishes a message for XML publications

When you create XML 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 an XML 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 XML publications change" on page 164
- "Message is sent every time a change occurs in the source table" on page 164

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

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.

Message is sent only when columns in XML publications change

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

Example: Assume that you have 100 columns in your source table and you select 25 of those columns to be published in an XML publication. If you specify for a message to be sent only when columns in XML publications change, then any time a change is made to any of the 25 columns that are part of the XML publication, the Q Capture program publishes an XML message. Any time a change is made in any of the 75 columns that are *not* part of the XML publication, the Q Capture program does *not* publish an XML message.

Message is sent every time a change occurs in the source table

You can define XML 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 an XML publication.

Example: Assume that you have 100 columns in your source table and you select 25 of those columns to be published in an XML publication. If you specify for a message to be sent every time a change occurs in the source table, any time that a change is made to *any* of the of the 100 columns in your source table, the Q Capture program publishes an XML message.

Related concepts:
- "Creating XML publications—Overview" on page 158

Related tasks:
- "Creating multiple XML publications" on page 161
- "Creating a single XML publication" on page 159

Search conditions to filter rows in XML publications

By default when you create XML publications, all rows from the source table are published. However, when you create an XML 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 XML publication, then the Create XML 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 XML publications at one time, then, on the Review page of the Create XML publications wizard, select the individual XML publication for which you want to subset rows and edit the properties for that XML publication to add the WHERE clause.

When you specify a WHERE clause, you can specify whether the column is evaluated with values from the current log record. If you want a column in the

WHERE clause to be evaluated with values from the current log record, place a single colon directly in front of the column name.

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

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

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

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

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

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

Suppose that the following change occurs at the source table:

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

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

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

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

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

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

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

Complex search conditions:

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

Example of WHERE clause with a subselect:

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

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

When you define an XML publication with a subselect in a WHERE clause, the following problems might occur:

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

Restrictions for search conditions:

- Search conditions cannot contain column functions, unless the column function appears within a subselect statement.

  Invalid WHERE clause with column functions:

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

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

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

  Invalid WHERE clause with GROUP BY:

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

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

  Valid WHERE clause with GROUP BY:

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

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

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

  WHERE :ADMINISTRATOR.SALES > 100000
  ```

In this example of a WHERE clause that has the actual names of the source table and columns, the table that is published is `ADMINISTRATOR` and `SALES` is the column name. This invalid WHERE clause is intended to select only the values of the `SALES` column of the `ADMINISTRATOR` table, for which `SALES` is greater than `100000`.

Valid WHERE clause with column name:

```
WHERE :SALES > 100000
```

In this example of a WHERE clause that has a column name, `SALES` is the column name.

- Search conditions cannot reference values that were in columns before a change occurred; they can reference only after values.
- Search conditions cannot contain `IN` predicates that use a full select.

  Invalid WHERE clause with `IN` predicate:

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

  WHERE :COL1 IN (SELECT .... )
  ```

  Valid WHERE clause with `IN` predicate:

  ```
  WHERE :COL1 IN ('CA', 'IL')
  ```
- 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.

Related concepts:
- "Creating XML publications—Overview" on page 158
- "Search conditions to filter rows (unidirectional replication)" on page 94

Related tasks:
- "Creating multiple XML publications" on page 161
- "Creating a single XML publication" on page 159

Key columns for XML publications

For each XML 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 XML publication, then the Create XML 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 multiple XML publications at one time, then you can use the Review page of the Create XML Publications wizard to customize which key columns to use.

Related concepts:
- "Creating XML publications—Overview" on page 158

Related tasks:

- “Creating multiple XML publications” on page 161
- “Creating a single XML publication” on page 159

Options for including unchanged columns in messages for XML publications

When you create XML 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 XML message that it publishes. The following sections describe the values that the Q Capture program can include in the XML message:

- “Only changed columns are sent”
- “Both changed and unchanged columns are sent”

This option applies only to values in non-key columns; the Q Capture program always publishes values in key columns.

Only changed columns are sent

By default when you create XML publications, the Q Capture program sends the values in the columns that you selected for the XML 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 an XML 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 XML publications so that the Q Capture program *always* sends the values in the columns that you selected for the XML 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 an XML 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.

Related concepts:

- “Creating XML publications—Overview” on page 158

Related tasks:

- “Creating multiple XML publications” on page 161
- “Creating a single XML publication” on page 159

Options for including before values in messages for XML publications

When an update occurs in columns that are not part of the target key, the Q Capture program either sends the value in the column after the change occurred, or it sends both the value in the column before the change occurred and the value in the column after the change occurred. When an update occurs in a key column, the before value and after value are always sent.

Because a delete operation always applies to a row and not to a specific column value, deletes are handled differently. For deletes, only before values are ever sent. Before values of key columns are always sent. If you specify for the message to include column values from before and after the change, then, if values in non-key columns are deleted, the before values of the non-key columns are sent.

The sections below describe your two options for before values and after values:
- "Send new data values only"
- "Send both old and new data values"

Send new data values only

By default when an update occurs at the source table, the Q Capture program publishes the values that are in the non-key columns *after* the change occurs. If you specify for the message to include only new data values (values from after the change), then the message does *not* include the values that were in the non-key columns before the change occurred.

Recommendation: If the application that receives the XML messages for the XML 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 XML messages for the XML 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 XML 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.

Related concepts:
- "Creating XML publications—Overview" on page 158

Related tasks:
- "Creating multiple XML publications" on page 161
- "Creating a single XML publication" on page 159

Chapter 13. Considerations for replicating and publishing data types

Considerations for replicating and publishing data types for Q replication and event publishing—Overview

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. The following topics describe how certain data types are handled in Q replication and event publishing:

- "Considerations for general data types for Q replication and event publishing"
- "Considerations for large object (LOB) data types for Q replication and event publishing" on page 172

Considerations for general data types for Q replication and event publishing

This topic explains the data types that are not supported in Q replication and event publishing and data types that you can use only under certain circumstances. Because the data types, length attributes, and null attributes must be same for the source and target columns, these considerations for data types apply to both the source and the target tables.

Currently, the following data cannot be replicated or published:

- DATALINK data types
- Spatial data types
- Any column in DB2® UDB for z/OS™ on which any of the following procedures is defined:
  - EDITPROC
  - FIELDPROC
  - VALIDPROC

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 UDB for Linux, UNIX®, and Windows® to DB2 UDB for z/OS. Fields in DB2 UDB for z/OS that contain long variable characters have a smaller maximum length than the fields in DB2 UDB for Linux, UNIX, and Windows. Therefore, replication of these types of fields to DB2 UDB for z/OS from DB2 UDB 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 Universal Database™) 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.

Related concepts:

- "Considerations for replicating and publishing data types for Q replication and event publishing—Overview" on page 171

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

DB2® Universal Database supports the following large object (LOB) data types:

- Binary LOB (BLOB)
- Character LOB (CLOB)
- Double-byte character LOB (DBCLOB)

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 UDB 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.

Related concepts:

- "Queue depth considerations for large object (LOB) values" on page 64
- "Considerations for replicating and publishing data types for Q replication and event publishing—Overview" on page 171
- "Memory for LOB data types for Q replication and event publishing" on page 31

Related reference:

- "Large object (LOB) message" on page 446

Chapter 14. Running SQL scripts and operational commands from the Replication Center

Running SQL scripts and operational commands from the Replication Center—Overview

The Replication Center creates and manipulates replication objects by generating and then running operational commands and SQL scripts. The following topics explain tasks that you can perform by using SQL scripts and operational commands that the Replication Center generates:

- "SQL scripts and operational commands that the Replication Center generates"
- "Running and saving SQL scripts from the Replication Center" on page 174
- "Running and saving commands from the Replication Center" on page 175

SQL scripts and operational commands that the Replication Center generates

The Replication Center creates and manipulates replication objects by generating and then running operational commands and SQL scripts. For example, when you start a Q Capture program by using the Replication Center, the Replication Center generates and then runs an operational command. As another example, when you delete a Q subscription by using the Replication Center, the Replication Center generates and then runs an SQL script.

The Replication Center can generate operational commands to perform many tasks for the Q Capture program, Q Apply program, and Replication Alert Monitor. For example, the commands can start or stop the program, prune control tables, change parameters, or check program status.

The Replication Center can also generate SQL scripts to create, modify, and delete objects. For example, the scripts can create, change, delete, or activate Q subscriptions or XML publications. The scripts can also create, change, or drop control tables, and they can create and delete alert conditions.

You can also modify commands and scripts to perform customized tasks. For example, you can define more than one Q subscription at the same time by editing the SQL script that the Replication Center generates. As another example you can create the same replication action on different servers and customize the replication for each server by modifying a SQL scripts that the Replication Center generates. Modify commands in the Run Now or Save Command window of the Replication Center. Modify SQL scripts in the Run Now or Save SQL window of the Replication Center.

You can run commands and SQL scripts from the Replication Center, the Task Center (except for the z/OS operating system), or a command line. You can save a command or a script as a task and run it from the Task Center or as a file and run it from the command line. You can also run commands and scripts directly from the Replication Center without saving them.

Related tasks:

- "Running SQL scripts and operational commands from the Replication Center—Overview" on page 173
- "Running and saving SQL scripts from the Replication Center" on page 174
- "Running and saving commands from the Replication Center" on page 175

Running and saving SQL scripts from the Replication Center

There are three ways that you can run SQL scripts from the Replication Center. You can run scripts directly from the Replication Center, save and run SQL scripts as tasks, or you can run SQL scripts from a command line.

You can run SQL scripts directly from the Replication Center by using the Run Now or Save SQL window.

Procedure:

To run a SQL script directly:

1. Optional: Modify the SQL script.
2. Run the script in the Run Now or Save SQL window by selecting the Run now radio button option.

You can also run SQL scripts as tasks from the Task Center (except for z/OS operating systems).

Procedure:

To run a SQL script as a task:

1. Optional: Modify the SQL script.
2. Save the script as a task object in the Run Now or Save Command window by selecting the Save as task radio button option.
3. Run the task object by using the Task Center.

You can save SQL scripts and run them later from a command line.

Prerequisites:

Before you run a SQL script from a command line, you must connect to the server and specify a user ID and password for the server. Use the following statement:

```
CONNECT TO MYDB USER XXXX USING XXXX ;
```

Procedure:

To run a SQL script from the command line:

1. Optional: Modify the script.
2. Save the script as a file using the Run Now or Save SQL window.
3. Run the script from the command line by using one of the following commands:
   - Use the following command if the termination character for the SQL script is a semicolon (;):
     ```
     db2 -tvf filename
     ```
   - If the SQL script uses a character other than a semicolon as the delimiter, use the following command:

```
db2 -td# -vf filename
```

Related concepts:

- "SQL scripts and operational commands that the Replication Center generates" on page 173

Running and saving commands from the Replication Center

There are three ways that you can run operational commands from the replication center. You can run commands directly from the Replication Center, save and run commands as tasks, or you can run commands as batch files from a command line.

You can run commands directly from the Replication Center by using the Run Now or Save Command window.

Procedure:

To run a command directly:

1. Optional: Modify the command.
2. Run the command in the Run Now or Save SQL window by selecting the Run now radio button option.

You can also run commands as tasks from the Task Center (except for z/OS operating systems).

Procedure:

To run a command as a task:

1. Optional: Modify the command.
2. Save the command as a task object in the Run Now or Save Command window by selecting the Save as task radio button option.
3. Run the task object by using the Task Center.

You can save commands as a batch file and run them later from a command line.

Procedure:

To run a command from a command line:

1. Optional: Modify the command.
2. Save the command as a batch file in the Run Now or Save SQL window by selecting the Save to file radio button option.
3. Run the batch file from a command line. To run a command on a system remote to the client machine where Replication Center runs:
   a. Save the file to the system where the command will be run.
   b. Telnet or logon to the remote system to run the command.
   c. To run a command on a remote system after saving it on a client machine, send the command file via FTP to the system where the command will run.

Related concepts:

- "SQL scripts and operational commands that the Replication Center generates" on page 173

Part 4. Administering and monitoring your Q replication and event publishing environment

This part of the book contains the following chapters:

- Chapter 15, "Changing a Q replication environment," on page 179 describes how to alter your Q replication configuration by modifying or stopping Q subscriptions and the programs that process and monitor them.
- Chapter 16, "Changing an event publishing environment," on page 191 describes how to alter your event publishing configuration by modifying or stopping XML publications and the programs that process and monitor them.
- Chapter 17, "Operating a Q Capture program," on page 201 describes how to operate the program that replicates or publishes changes from the source in a Q replication or event publishing environment.
- Chapter 18, "Operating a Q Apply program," on page 227 describes how to operate the program that replicates changes to the target in a Q replication environment.
- Chapter 19, "Viewing reports about the Q replication and event publishing programs," on page 245 describes how to check reports on the Q Capture and Q Apply programs to view their latest statistics.
- Chapter 20, "Monitoring replication with the Replication Alert Monitor," on page 255 describes how set up and operate the program that monitors your Q replication and event publishing environment and automatically notifies you when various conditions are met. This chapter also describes how to monitor an SQL replication environment.
- Chapter 21, "Maintaining a Q replication and publishing environment," on page 277 describes how to maintain your source tables, targets, and control tables, and how to maintain the WebSphere MQ objects that are used for replicating and publishing.
- Chapter 22, "Detecting and repairing differences between source and target tables," on page 289 describes how to synchronize source and target tables with the tdiff and trep utilities.
- Chapter 23, "Using system services to operate the replication programs," on page 293 describes how to schedule and start the programs for Q replication and event publishing by using the commands and services available on various operating systems.

Chapter 15. Changing a Q replication environment

Changing a Q replication environment—Overview

The following topics explain issues and procedures for making day-to-day changes to a Q replication environment:

- "Changing attributes of Q subscriptions"
- "Adding columns to existing Q subscriptions" on page 181
- "Changing acceptable SQL states for Q subscriptions" on page 183
- "Changing attributes of replication queue maps" on page 184
- "Deleting Q subscriptions" on page 186
- "Deleting replication queue maps" on page 188
- "Dropping Q Capture schemas or Q Apply schemas" on page 188

Changing attributes of Q subscriptions

You can change some attributes of unidirectional Q subscriptions. You can change these attributes without stopping replication. The procedure differs depending on whether you are changing the attributes of a single Q subscription, or multiple Q subscriptions.

The following list describes the attributes that you can change:

Search condition
: The WHERE clause that is used to determine which rows from the source table are replicated.

Source table deletes
: The option of whether to replicate delete row operations from the source table.

All changed rows
: The option of whether to replicate a row in the source table when any column changes, even if the changed column is not part of a Q subscription.

You can change attributes for one Q subscription by making your changes and then reinitializing the Q subscription.

You can change attributes for multiple Q subscriptions that are defined within a Q Capture schema by making your changes and then reinitializing the Q Capture program.

Important: If you reinitialize a Q Capture program, this action will also reinitialize any XML 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 attributes of one or two Q subscriptions, it is less costly in terms of resources to reinitialize one Q subscription at a time.

Restrictions:

If you want to change other attributes of a Q subscription, you must delete the Q subscription and recreate it with different attributes. You cannot change any attributes of a Q subscription that is used for bidirectional or peer-to-peer replication without deleting and recreating the Q subscription.

Procedure:

To change attributes for a single Q subscription without stopping replication:

1. Change the attributes of the Q subscription.

   In the Replication Center, use the Q Subscription Properties notebook to change the attributes of one Q subscription at a time. To open the notebook, right-click a Q subscription and select **Properties**. See the online help for details.
2. Reinitialize the Q subscription.

   Replication Center
   : Use the Manage Q Subscriptions window to reinitialize a Q subscription. 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**. See the online help for details.

   SQL
   : Use a command prompt or one of the DB2 command line tools to insert a REINIT_SUB signal into the IBMQREP_SIGNAL table at the Q Capture server:

   ```
   insert into schema.IBMQREP_SIGNAL(
       SIGNAL_TIME,
       SIGNAL_TYPE,
       SIGNAL_SUBTYPE,
       SIGNAL_INPUT_IN,
       SIGNAL_STATE
   ) values (
        CURRENT TIMESTAMP,
       'CMD',
       'REINIT_SUB',
       'subname',
       'P' );
   ```

   Where *schema* identifies the Q Capture program that is processing the Q subscription, and *subname* is the name of the Q subscription that you want to reinitialize.

To change attributes for multiple Q subscriptions without stopping replication:

1. Change the attributes of the Q subscriptions.

   In the Replication Center, use the Q Subscription Properties notebook to change the attributes of one Q subscription at a time. To open the notebook, right-click the Q Capture server where the source table for the Q subscription is located and select **Manage → Q Subscriptions**. See the online help for details.
2. Reinitialize the Q Capture program.

   Replication Center
   : Use the Reinitialize Q Capture window to reinitialize all of the Q subscriptions for one Q Capture program. 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**. See the online help for details.

   asnqccmd system command
   : Use the **asnqccmd reinit** command to reinitialize all of the Q subscriptions for one Q Capture schema:

```
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.

Related concepts:
- "Changing a Q replication environment—Overview" on page 179
- "Q Capture program" on page 15
- "Q subscriptions" on page 5
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86
- "Search conditions to filter rows (unidirectional replication)" on page 94

Related tasks:
- "Changing attributes of XML publications" on page 191

Related reference:
- "asnqccmd: Working with a running Q Capture program" on page 320
- "IBMQREP_SIGNAL table" on page 382

Adding columns to existing Q subscriptions

You can add columns dynamically to an active Q subscription for unidirectional replication without stopping replication of changes from the source table.

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 Q subscription 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. When you insert the signal, 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.

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.

Prerequisites:
- The Q subscription that the columns are being added to must be in A (active) state when the signal is inserted.
- The columns must exist in the source table. If you perform an ALTER TABLE ADD COLUMN operation for the source table in the same transaction as the insert of the signal, the ALTER operation must occur before the signal insert.
- 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:

- You cannot use the ADDCOL signal to add columns to existing Q subscriptions for bidirectional or peer-to-peer replication.
- 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.

Procedure:

To add columns to an existing Q subscription:

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:

```
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' );
```

Where:

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.

After processing the signal, the Q Capture program 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 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.

Tip:

You can add columns to the source table without having to stop and restart the Q Capture program. If you plan to quiesce the source database or instance before adding columns, you can use the Replication Center or a 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 UDB 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.

You can use the Change Parameters – Running Q Capture Program window in the Replication Center or the **asnqccmd chgparms** command to set the TERM parameter while a Q Capture program is running.

Related concepts:

- "Changing a Q replication environment—Overview" on page 179
- "Q subscriptions" on page 5

Related tasks:

- "Changing attributes of Q subscriptions" on page 179

Related reference:

- "Descriptions of Q Capture parameters" on page 206
- "IBMQREP_SIGNAL table" on page 382

Changing acceptable SQL states for Q subscriptions

For unidirectional Q subscriptions, you can change the SQL states that you define as acceptable. If the Q Apply program encounters one of these SQL states while it is applying a transaction, the row that caused the error will not be applied, and the Q Apply program will continue with the next row in a transaction.

You can use the Replication Center to change acceptable SQL states. The acceptable SQL states that you define are stored in the OKSQLSTATES column of the IBMQREP_TARGETS control table.

To change acceptable SQL states, you must stop message processing on the receive queue that is used for the Q subscription, change the attribute, and then start message processing on the receive queue.

Restrictions:

You cannot change acceptable SQL states for a Q subscription that is used for bidirectional or peer-to-peer replication.

Procedure:

To change acceptable SQL states for a Q subscription:

1. Stop message processing on the receive queue that is used for the Q subscription.

   Replication Center

   Use the Manage Receive Queues window to stop message processing on a receive queue. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage → Receive Queues**. See the online help for details.

   asnqacmd command

   Use the **asnqacmd stopq** command to stop message processing on a receive queue:

   ```
   asnqacmd apply_server=server_name apply_schema=schema
      stopq=receive_queue_name
   ```

Where *server_name* is the name of the Q Apply server, *schema* identifies the Q Apply program that is processing messages from the receive queue, and *receive_queue_name* is the name of the receive queue for which you want to stop message processing.

2. Change the OKSQLSTATES attribute of the Q subscription.

   In the Replication Center, use the Q Subscription Properties notebook to change the OKSQLSTATES attribute of a Q subscription. To open the notebook, right-click the Q subscription and select **Properties**. See the online help for details.

3. Start message processing on the receive queue that is used for the Q subscription.

   Replication Center
   Use the Manage Receive Queues window to start message processing on a receive queue. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage → Receive Queues**. See the online help for details.

   asnqacmd command
   Use the **asnqacmd startq** command to start message processing on a receive queue:

```
asnqacmd apply_server=server_name apply_schema=schema
   startq=receive_queue_name
```

   Where *server_name* is the name of the Q Apply server, *schema* identifies a Q Apply program, and *receive_queue_name* is the name of the receive queue for which you want to start message processing.

Related concepts:
- "Changing a Q replication environment—Overview" on page 179
- "Q subscriptions" on page 5
- "Creating Q subscriptions for unidirectional replication—Overview" on page 86

Related tasks:
- "Stopping message processing on a receive queue" on page 241
- "Starting message processing on a receive queue" on page 239

Related reference:
- "asnqacmd: Working with a running Q Apply program" on page 326

Changing attributes of replication queue maps

You can update a replication queue map to change the way a Q Capture program, Q Apply program, or both process transactions that are transmitted over a particular send queue and receive queue.

A replication queue map includes settings 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.

Properties for replication queue maps are saved in the Q Capture and Q Apply 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.

Restrictions:

You cannot update the following attributes of a replication queue map:

- The name of the queue map
- The send queue, receive queue, or administration queue that the queue map points to
- The Q Capture server and Q Capture schema

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.

   In the Replication Center, use the Replication Queue Map Properties window to change any of the following properties:

   Send queue
   - The error action
   - The maximum amount of memory that the Q Capture program allocates as a buffer for messages that it puts on the queue
   - The time between heartbeat messages that are sent by the Q Capture program to indicate that it is still running when there are no messages to put on the queue

   Receive queue
   - The number of agent threads that the Q Apply program uses to process transactions from the queue
   - The maximum amount of memory that the Q Apply program allocates as a buffer for messages that it gets from the queue

   To open the Replication Queue Map Properties window, right-click a replication queue map and select **Properties**. See the online help for details.
2. 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:

   Replication Center
   - Use the Manage Send Queues window to reinitialize a send queue whose properties you changed in the replication queue map. To open the window, right-click the Q Capture server where the send queue is located and select **Manage → Send Queues**.
   - Use the Manage Receive Queues window to reinitialize a receive queue whose properties you changed in the replication queue map. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage → Receive Queues**.

   See the online help for details.

asnqccmd system command (send queue)
Use the **asnqccmd reinitq** command to reinitialize a send queue whose properties you changed in the replication queue map:

```
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, *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.

asnqacmd system command (receive queue)
Use the **asnqacmd reinitq** command to reinitialize a receive queue whose properties you changed in the replication queue map:

```
asnqacmd apply_server=server_name apply_schema=schema
  reinitq=queue_name
```

Where *server_name* is the name of the database or subsystem where the Q Apply program is running, *schema* identifies the Q Apply program that uses the receive queue, and *queue_name* is the name of the receive queue that you want to reinitialize.

Related concepts:
- "Changing a Q replication environment—Overview" on page 179
- "Replication queue maps" on page 6

Related tasks:
- "Changing attributes of publishing queue maps" on page 195
- "Creating replication queue maps" on page 84

Related reference:
- "asnqccmd: Working with a running Q Capture program" on page 320
- "asnqacmd: Working with a running Q Apply program" on page 326
- "IBMQREP_RECVQUEUES table" on page 402
- "IBMQREP_SENDQUEUES table" on page 380

Deleting Q subscriptions

You can delete a Q subscription that is not being actively processed by a Q Capture program. If the Q subscription is active, you must deactivate it before you can delete it.

When you delete a Q subscription with the Replication Center, you have the option of dropping the target table that it refers to. You can also drop the table space for the target table.

Note: Deleting a Q subscription does not delete the replication queue map that it uses.

Prerequisites:
- The Q subscription that you want to delete must be in I (inactive) or N (new) state.

Procedure:

To delete one or more Q subscriptions:

1. If the Q subscriptions are in A (active) state, deactivate them. Use one of the following methods:

   Replication Center
   Use the Manage Q Subscriptions window to deactivate one or more Q subscriptions. 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**. You can also use this window to verify that the Q subscription state has changed from A (active) to I (inactive). The deactivation process must be complete before you delete any Q subscriptions. See the online help for details.

   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 where the Q subscriptions are defined. Insert the signal for each Q subscription that you want to deactivate:

   ```
   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 a Q subscription that you want to deactivate.

2. Delete the Q subscriptions.

   In the Replication Center, use the Delete Q Subscriptions window to delete one or more Q subscriptions. To open the window, right-click a Q subscription and select **Delete**. You can optionally use this window to drop the target tables for Q subscriptions that you delete. If you drop a target table, you can optionally use the window to drop the associated table space, provided that no other tables are using the table space. See the online help for details.

Related concepts:
- "Changing a Q replication environment—Overview" on page 179

Related tasks:
- "Deleting XML publications" on page 196
- "Deactivating Q subscriptions or XML publications" on page 222

Related reference:
- "IBMQREP_SIGNAL table" on page 382

Deleting replication queue maps

You can delete a replication queue map that is no longer needed by any Q subscriptions.

Prerequisites:

Make sure that no Q subscriptions are using the replication queue map or you will receive an error message when you try to generate the script to delete the queue map.

Procedure:

To delete a replication queue map, use the Replication Center:

1. Open the Show Related window to see if any Q subscriptions are using the replication queue map. To open the window, right-click the replication queue map and select **Show Related**.
2. If any Q subscriptions are using the queue map, use the Delete Q Subscriptions window to delete the Q subscriptions. To open the window, right-click the Q subscription and select **Delete**.
3. Use the Delete Replication Queue Maps window to delete the queue map. To open the window, right-click the replication queue map and select **Delete**.

See the online help for details.

Related concepts:

- "Changing a Q replication environment—Overview" on page 179
- "Replication queue maps" on page 6

Related tasks:

- "Deleting publishing queue maps" on page 197

Dropping Q Capture schemas or Q Apply schemas

A schema object represents an instance of the Q Capture or Q Apply program, and its associated control tables. When you drop a Q Capture or Q Apply schema, the schema object is deleted from the Replication Center object tree, and the control tables are dropped.

Dropping a schema also deletes the following objects:

On the Q Capture server
Both Q subscriptions and XML publications are removed because definitions for these two objects are stored in the same control tables. Dropping a schema also removes both replication queue maps and publishing queue maps.

On the Q Apply server
Q subscriptions and replication queue maps are removed.

Important: Q subscriptions and replication queue maps are defined on both the Q Capture server and Q Apply server. If you are dropping only a Q Capture schema or only a Q Apply schema, 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 schema.

Prerequisites:

You must stop the Q Capture program or Q Apply program that is identified by the schema that you want to drop.

Procedure:

To drop a Q Capture or Q Apply schema:

1. Stop the Q Capture program or Q Apply program that is identified by the schema that you want to drop. Use one of the following methods:

   Replication Center

   Use the Stop Q Capture window or Stop Q Apply window to stop a Q Capture program or Q Apply program:
   - To open the Stop Q Capture window, right-click the Q Capture server that contains the Q Capture program that you want to stop and select **Stop Q Capture Program**.
   - To open the Stop Q Apply window, right-click the Q Apply server that contains the Q Apply program that you want to stop and select **Stop Q Apply Program**.

   See the online help for details.

   asnqccmd system command (Q Capture program)

   Use the **asnqccmd stop** command to stop a Q Capture program.
   ```
   asnqccmd capture_server=server_name capture_schema=schema stop
   ```
   Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program that you want to stop.

   SQL (Q Capture program)

   Use a command prompt or one of the DB2 command line tools to insert a STOP signal into the IBMQREP_SIGNAL table at the Q Capture server:
   ```
   insert into schema.IBMQREP_SIGNAL(
       SIGNAL_TIME,
       SIGNAL_TYPE,
       SIGNAL_SUBTYPE,
       SIGNAL_INPUT_IN,
       SIGNAL_STATE )
    values (
        CURRENT TIMESTAMP,
       'CMD',
       'STOP',
       'NULL',
       'P' );
   ```
   Where *schema* identifies the Q Capture program that you want to stop.

   asnqacmd system command (Q Apply program)

   Use the **asnqacmd stop** command to stop a Q Apply program.
   ```
   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.

2. Optional: If you are dropping only a Q Capture schema or only a Q Apply schema, take the following actions:
   - Deactivate any Q subscriptions that are also defined on the opposite server. See "Deactivating Q subscriptions or XML publications" on page 222 for details.
   - Delete the Q subscriptions. See "Deleting Q subscriptions" on page 186 for details.
   - Delete any replication queue maps that are also defined on the opposite server. See "Deleting replication queue maps" on page 188 for details.
3. In the Replication Center, use the Drop Q Capture Schema or Drop Q Apply Schema window to drop a Q Capture or Q Apply schema. To open the windows, right-click the schema and select **Drop**. See the online help for details.

   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.

Related concepts:
- "Changing a Q replication environment—Overview" on page 179
- "Schemas for the Q Apply and Q Capture programs" on page 20

Related reference:
- "asnqccmd: Working with a running Q Capture program" on page 320
- "IBMQREP_SIGNAL table" on page 382

Chapter 16. Changing an event publishing environment

Changing an event publishing environment—Overview

The following topics explain issues and procedures for making day-to-day changes to an event publishing environment:

- "Changing attributes of XML publications"
- "Adding columns to existing XML publications" on page 193
- "Changing attributes of publishing queue maps" on page 195
- "Deleting XML publications" on page 196
- "Deleting publishing queue maps" on page 197
- "Dropping Q Capture schemas" on page 198

Changing attributes of XML publications

You can change some attributes of XML publications. You can change these attributes without stopping publishing. The procedure differs depending on whether you are changing the attributes of a single XML publication, or multiple XML publications.

The following list describes the attributes that you can change:

Search condition
The WHERE clause that is used to determine which rows from the source table are replicated.

Source table deletes
The option of whether to replicate delete row operations from the source table.

All changed rows
The option of whether to replicate a row in the source table when any column changes, even if the changed column is not part of an XML publication.

You can change attributes for one XML publication by making your changes and then reinitializing the XML publication.

You can change attributes for multiple XML publications that are defined within a Q Capture schema by making your changes and then reinitializing the Q Capture program.

Important: 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 XML publications, it is less costly in terms of resources to reinitialize one XML publication at a time.

Restrictions:

If you want to change other attributes of an XML publication, you must delete the XML publication and recreate it with different attributes.

Procedure:

To change attributes for a single XML publication without stopping publishing:

1. Change the attributes of the XML publication.

   In the Replication Center, use the XML Publication Properties notebook to change the attributes of one XML publication at a time. To open the notebook, right-click an XML publication and select **Properties**. See the online help for details.

2. Reinitialize the XML publication.

   Replication Center
   Use the Manage XML Publications window to reinitialize an XML publication. To open the window, right-click the Q Capture server where the source table for the XML publication is located and select **Manage → XML Publications**. See the online help for details.

   SQL Use a command prompt or one of the DB2 command line tools to insert a REINIT_SUB signal into the IBMQREP_SIGNAL table at the Q Capture server:

   ```
   insert into schema.IBMQREP_SIGNAL(
       SIGNAL_TIME,
       SIGNAL_TYPE,
       SIGNAL_SUBTYPE,
       SIGNAL_INPUT_IN,
       SIGNAL_STATE
   ) values (
        CURRENT TIMESTAMP,
       'CMD',
       'REINIT_SUB',
       'subname',
       'P' );
   ```

   Where *schema* identifies the Q Capture program that is processing the XML publication, and *subname* is the name of the XML publication that you want to reinitialize.

To change attributes for multiple XML publications without stopping replication:

1. Change the attributes of the XML publications.

   In the Replication Center, use the XML Publication Properties notebook to change the attributes of one XML publication at a time. To open the notebook, right-click an XML publication and select **Properties**. See the online help for details.

2. Reinitialize the Q Capture program.

   Replication Center
   Use the Reinitialize Q Capture window to reinitialize all of the XML publications for one Q Capture program. 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**. See the online help for details.

   asnqccmd system command
   Use the **asnqccmd reinit** command to reinitialize all of the XML publications for one Q Capture schema:

   ```
   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 XML publications.

Related concepts:
- "Changing an event publishing environment—Overview" on page 191
- "Search conditions to filter rows in XML publications" on page 164
- "When the Q Capture program publishes a message for XML publications" on page 163

Related tasks:
- "Deleting XML publications" on page 196
- "Changing attributes of Q subscriptions" on page 179
- "Creating multiple XML publications" on page 161
- "Creating a single XML publication" on page 159

Related reference:
- "asnqccmd: Working with a running Q Capture program" on page 320

Adding columns to existing XML publications

You can add columns dynamically to an existing XML publication without stopping publishing of changes from the source table.

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 XML 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 an XML publication, you insert one signal for each new column. You can add multiple columns in a single transaction.

Prerequisites:
- The XML publication that the columns are being added to must be in A (active) state when the signal is inserted.
- The columns must exist in the source table. If you perform an ALTER TABLE ADD COLUMN operation for the source table in the same transaction as the insert of the signal, the ALTER operation must occur before the signal insert.
- 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.

Procedure:

To add columns to an existing XML publication:

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:

```
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' );
```

Where:

schema
: Identifies the Q Capture program that is processing the XML publication that you are adding a column to.

pubname;column_name
: The name of the XML 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` XML 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.

Tip:

You can add columns to the source table without having to stop and restart the Q Capture program. If you plan to quiesce the source database or instance before adding columns, you can use the Replication Center or a 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 UDB 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.

You can use the Change Parameters – Running Q Capture Program window in the Replication Center or the **asnqccmd chgparms** command to set the TERM parameter while a Q Capture program is running.

Related concepts:

- "Changing an event publishing environment—Overview" on page 191
- "XML publications" on page 11

Related tasks:

- "Changing attributes of XML publications" on page 191

Related reference:

- "Descriptions of Q Capture parameters" on page 206

- “IBMQREP_SIGNAL table” on page 382

Changing attributes of publishing queue maps

You can update a publishing queue map to change the way a Q Capture program processes transactions or row-level changes that are transmitted using a particular send queue.

A publishing queue map points to one send queue, and includes settings for how a Q Capture program handles the XML 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 needing to stop publishing.

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 XML publications that use the send queue.

Restrictions:

You cannot change the following attributes of a publishing queue map:
- The name of the queue map
- The send queue that the queue map points to
- The content type of the XML messages that use the replication queue map (whether the messages contain a single row or an entire transaction)

Procedure:

To update a publishing queue map without stopping publishing:
1. In the Replication Center, use the Publishing Queue Map Properties window to change any of the following properties:
   - The error action for the send queue.
   - The maximum amount of memory that the Q Capture program allocates as a buffer for messages that it puts on the send queue.
   - The time between heartbeat messages that the Q Capture program sends to indicate that it is still running when there are no transactions to process.

   To open the Publishing Queue Map Properties window, right-click a publishing queue map and select **Properties**. See the online help for details.
2. 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:

   Replication Center
      Use the Manage Send Queues window to reinitialize a send queue whose properties you changed. 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**. See the online help for details.

   asnqccmd system command
      Use the **asnqccmd reinitq** command to reinitialize a send queue whose properties you changed:

```
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, *schema* identifies a Q Capture program, and *queue_name* is the name of the send queue that you want to reinitialize.

Related concepts:

- "Changing an event publishing environment—Overview" on page 191
- "Memory used by the Q Capture program" on page 26
- "Publishing queue maps" on page 12
- "Event publishing" on page 9

Related tasks:

- "Changing attributes of replication queue maps" on page 184
- "Creating publishing queue maps" on page 156

Related reference:

- "asnqccmd: Working with a running Q Capture program" on page 320
- "IBMQREP_SENDQUEUES table" on page 380

Deleting XML publications

You can delete an XML publication that is not being actively processed by a Q Capture program. If the XML publication is active, you must deactivate it before you can delete it.

Note: Deleting an XML publication does not delete the publishing queue map that it uses.

Prerequisites:

The XML publication that you want to delete must be in I (inactive) or N (new) state.

Procedure:

To delete one or more XML publications:

1. If the XML publications are in A (active) state, deactivate them. Use one of the following methods:

   Replication Center
   Use the Manage XML Publications window to deactivate one or more XML publications. To open the window, right-click the Q Capture server where the source table for the XML publication is located and select **Manage → XML Publications**. You can also use this window to verify that the XML publication state changed from A (active) to I (inactive). The deactivation process must be complete before you delete any XML publications. See the online help for details.

   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 where the XML publications are defined. Insert the signal for each XML publication that you want to deactivate:

```
insert into schema.IBMQREP_SIGNAL (
    SIGNAL_TIME,
    SIGNAL_TYPE,
    SIGNAL_SUBTYPE,
    SIGNAL_INPUT_IN,
    SIGNAL_STATE )
 values (
     CURRENT TIMESTAMP,
    'CMD',
    'CAPSTOP',
    'xml_publication_name',
    'P' );
```

Where *schema* identifies a Q Capture program and *xml_publication_name* is the name of an XML publication that you want to deactivate.

2. Delete the XML publications.

   In the Replication Center, use the Delete XML Publications window to delete one or more XML publications. To open the window, right-click an XML publication and select **Delete**. See the online help for details.

Related concepts:

- "Changing an event publishing environment—Overview" on page 191
- "Event publishing" on page 9

Related tasks:

- "Deleting Q subscriptions" on page 186
- "Deactivating Q subscriptions or XML publications" on page 222
- "Creating multiple XML publications" on page 161
- "Creating a single XML publication" on page 159

Related reference:

- "IBMQREP_SIGNAL table" on page 382

Deleting publishing queue maps

You can delete a publishing queue map that is no longer needed by any XML publications.

Prerequisites:

Make sure that no XML publications are using the publishing queue map or you will receive an error message when you try to generate the script to delete the queue map.

Procedure:

To delete a publishing queue map, use the Replication Center:

1. Open the Show Related window to see if any XML publications are using the publishing queue map. To open the window, right-click the publishing queue map and select **Show Related**.
2. If any XML publications are using the queue map, use the Delete XML Publications window to delete the XML publications. To open the window, right-click the XML publication and select **Delete**.

3. Use the Delete Publishing Queue Maps window to delete the queue map. To open the window, right-click the publishing queue map and select **Delete**.

See the online help for details.

Related concepts:

- "Changing an event publishing environment—Overview" on page 191
- "Publishing queue maps" on page 12
- "Event publishing" on page 9

Related tasks:

- "Deleting replication queue maps" on page 188
- "Creating publishing queue maps" on page 156

Dropping Q Capture schemas

A schema object represents an instance of the Q Capture program, and its associated control tables. When you drop a Q Capture schema, the schema object is deleted from the Replication Center object tree, and the control tables are dropped.

Dropping a schema also deletes any XML publications, Q subscriptions, publishing queue maps, or replication queue maps that are defined in the control tables.

Prerequisites:

You must stop the Q Capture program that is identified by the schema that you want to drop.

Procedure:

To drop a Q Capture schema:

1. Stop the Q Capture program that is identified by the schema that you want to drop. Use one of the following methods:

   Replication Center
   Use the Stop Q Capture window to stop a Q Capture program. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to stop and select **Stop Q Capture Program**. See the online help for details.

   asnqccmd system command
   Use the **asnqccmd stop** command to stop a Q Capture program:

   ```
   asnqccmd capture_server=server_name capture_schema=schema stop
   ```

   Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program that you want to stop.

   SQL
   Use a command prompt or one of the DB2 command line tools to insert a STOP signal into the IBMQREP_SIGNAL table at the Q Capture server:

   ```
   insert into ASN.IBMQREP_SIGNAL(
       SIGNAL_TIME,
       SIGNAL_TYPE,
       SIGNAL_SUBTYPE,
       SIGNAL_INPUT_IN,
   ```

```
     SIGNAL_STATE )
  values (
      CURRENT TIMESTAMP,
     'CMD',
     'STOP',
     'NULL',
     'P' );
```

2. In the Replication Center, use the Drop Q Capture Schema window to drop the schema. To open the window, right-click the schema and select **Drop**. See the online help for details.

Related concepts:

- "Changing an event publishing environment—Overview" on page 191
- "Publishing queue maps" on page 12
- "Q subscriptions" on page 5
- "Replication queue maps" on page 6
- "Schemas for the Q Apply and Q Capture programs" on page 20
- "XML publications" on page 11

Related tasks:

- "Dropping Q Capture schemas or Q Apply schemas" on page 188
- "Stopping a Q Capture program" on page 224

Related reference:

- "asnqccmd: Working with a running Q Capture program" on page 320
- "IBMQREP_SIGNAL table" on page 382

Chapter 17. Operating a Q Capture program

Operating a Q Capture program—Overview

A Q Capture program captures transactions or row-level changes from source tables that are part of a Q subscription or XML 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.

The following topics explain how to operate a Q Capture program:
- "Starting a Q Capture program"
- "Considerations for cold starting a Q Capture program" on page 203
- "Parameters of a Q Capture program—Overview" on page 205
- "Changing the Q Capture parameters—Overview" on page 217
- "Activating Q subscriptions or XML publications" on page 221
- "Deactivating Q subscriptions or XML publications" on page 222
- "Stopping a Q Capture program" on page 224

Related concepts:
- "Q Capture program" on page 15
- "Schemas for the Q Apply and Q Capture programs" on page 20

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 XML publications, and sending the transactional data as messages over WebSphere MQ queues.

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 XML publications that are in N (new) or A (active) state. Any Q subscriptions or XML 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 XML publications are in A (active) state. When you activate the Q subscriptions or XML 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:
- You change the parameter values while the program is running.
- You stop and restart the program, which prompts it to read the IBMQREP_CAPPARMS table and use the values saved there.

Prerequisites:

- If you are starting a Q Capture program from a remote workstation, connections must be configured to the Q Capture server. See "Connectivity requirements for Q replication and event publishing" on page 67 for details.
- A WebSphere MQ queue manager, queues, and other necessary objects must be created and configured on the same system as the Q Capture server. See "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43 for details.
- You must have the proper authorization for Q replication and event publishing objects, and WebSphere MQ objects. See "Authorization requirements for the Q Capture program" on page 37 and "Connectivity and authorization requirements for WebSphere MQ objects" on page 62 for details.
- The control tables must be created for the appropriate Q Capture schema. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.
- The source database or subsystem must be configured to work with the Q Capture program. See "Configuring the source database to work with the Q Capture program (Linux, UNIX, Windows)" on page 69 and "Configuring databases for Q replication and event publishing (z/OS)" on page 75 for details.
- If any Q subscriptions that specify an automatic load that uses the EXPORT utility are in N (new) or A (active) state, a password file must be created on the Q Apply server to allow the utility to connect to the Q Capture server. See "Storing user IDs and passwords for remote servers" on page 41 for details.

Procedure:

Use one of the following methods to start a Q Capture program:

Replication Center
: Use the Start Q Capture window to start a Q Capture program. 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. See the online help for details.

asnqcap system command
: Use the **asnqcap** command to start a Q Capture program and specify startup parameters:

```
asnqcap capture_server=server_name capture_schema=schema parameters
```

Where *server_name* is the name of the database or subsystem that contains the Q Capture control tables, *schema* identifies the Q Capture program that you want to start, and *parameters* is one or more parameters that you can specify at startup.

z/OS console or TSO
: On z/OS, you can start the Q Capture program using JCL or as a system-started task. You can specify new invocation parameter values when you start a Q Capture program with JCL. The best method for specifying invocation parameters when using JCL is to store them in the IBMQREP_CAPPARMS table. The PARM parameter of the EXEC statement cannot have subparameters that exceed 100 characters.

Windows services

You can create a DB2 replication service on Windows operating systems to start the Q Capture program automatically when the system is started.

To verify whether a Q Capture program started, use one of the following methods:

- 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**. See the online help for details.
- 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**. See the online help for details.
- Examine the Q Capture diagnostic log file (*db2instance.capture_server.capture_schema*.QCAP.log on Linux, UNIX, and Windows and *capture_server.capture_schema*.QCAP.log on z/OS) for a message that indicates that the program is capturing changes.
- Check the IBMQREP_CAPTRACE table for a message that indicates that the program is capturing changes.
- **z/OS:** If you are running in batch mode, examine the z/OS console or z/OS job log for messages that indicate that the program started.

Related concepts:

- "Operating a Q Capture program—Overview" on page 201
- "Parameters of a Q Capture program—Overview" on page 205
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Q Capture program" on page 15

Related tasks:

- "Activating Q subscriptions or XML publications" on page 221
- "Managing replication services with the Windows Service Control Manager (SCM) —Overview" on page 299
- "Starting the Q Capture program by using JCL" on page 294

Related reference:

- "asnqcap: Starting a Q Capture program" on page 315
- "IBMQREP_CAPPARMS table" on page 374
- "IBMQREP_CAPTRACE table" on page 378

Considerations for cold starting a Q Capture program

Before you use the cold start mode to start a Q Capture program, read this topic so that you understand the issues related to cold starting a Q Capture program.

When to cold start a Q Capture program.

- When you start a Q Capture program for the first time (required).
- When you want to begin capturing changes from the end of the active log instead of from the last restart point.

Possible problems that can result from cold starts of the Q Capture program.
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 over log records for data that it would otherwise 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 tables and the target tables is to load the target table (sometimes called a full refresh).
- **Loading the target can cause you to lose historical data.** Historical data, which is kept when you choose to suppress deletes from the source, is a record of deleted rows in the source. The historical data is lost if the target table is loaded.
- **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.

How to prevent unexpected cold starts of the Q Capture program

- **Make sure that the cold start mode is not specified in the IBMQREP_CAPPARMS table.** The **startmode** parameter should not have the value *cold*.
- **Specify the warmns or warmsi start mode instead of the warmsa start mode to restart a Q Capture program whenever possible.** In this case, 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 is down. If the Q Capture program is down, the program can fall behind in reading log records. If the program is down long enough, the environment might require a cold start to synchronize the source and target tables because the Q Capture program is so far behind in reading log records.
- **Ensure that you retain sufficient DB2 log data on your system and that this data is available to Q replication and event publishing.** 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.

Related concepts:

- "Load options for different types of Q replication" on page 150

Related tasks:

- "Starting a Q Capture program" on page 201

Parameters of a Q Capture program

Parameters of a Q Capture program—Overview

A Q Capture program's operating parameters govern the way that it 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. The following topics list the default operating parameters, and provide details about each parameter and reasons that you might want to change the default settings:

- "Default values for Q Capture operating parameters"
- "Descriptions of Q Capture parameters" on page 206

Related concepts:

- "Operating a Q Capture program—Overview" on page 201

Default values for Q Capture operating parameters

When you create control tables using the Replication Center, the IBMQREP_CAPPARMS table is created with a single row that contains default values for the Q Capture program's operating parameters. Table 13 shows these values.

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.

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

Table 13. Default values for Q Capture operating parameters

Operating parameter	Default value	Column name in IBMQREP_CAPPARMS table
add_partition	N [1]	not applicable
adminq	None [2]	ADMINQ
autostop	N [1]	AUTOSTOP
capture_path	Directory where Q Capture was started [3]	CAPTURE_PATH
capture_schema	ASN [4]	not applicable
capture_server	DB2DBDFT [5]	not applicable
commit_interval	500 [6]	COMMIT_INTERVAL
logreuse	N [1]	LOGREUSE
logstdout	N [1]	LOGSTDOUT
memory_limit	32 [7]; 16 [7] for z/OS	MEMORY_LIMIT
monitor_interval	300 [8]	MONITOR_INTERVAL
monitor_limit	10080 [9]	MONITOR_LIMIT
prune_interval	300 [8]	PRUNE_INTERVAL
qmgr	None [2]	QMGR

Table 13. Default values for Q Capture operating parameters (continued)

Operating parameter	Default value	Column name in IBMQREP_CAPPARMS table
restartq	None [2]	RESTARTQ
signal_limit	10080 [9]	SIGNAL_LIMIT
sleep_interval	5000 [6]	SLEEP_INTERVAL
startmode	warmsi [10]	STARTMODE
term	Y [11]	TERM
trace_limit	10080 [9]	TRACE_LIMIT

Notes:

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 a Q Capture program starts as a Windows service, the **capture_path** is \sqllib\bin. If Q Capture is started using JCL on z/OS, the default is the user ID associated with the started task or job.
4. You cannot change the default schema. To use a Q Capture program with a different schema, you specify the **capture_schema** start parameter when you start a Q Capture program.
5. For Linux, UNIX, and Windows, the Q Capture program server is the value of the DB2DBDFT environment variable, if specified. For z/OS, there is no default Q Capture server.
6. Milliseconds.
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.

Related concepts:

- "Changing the Q Capture parameters—Overview" on page 217
- "Parameters of a Q Capture program—Overview" on page 205
- "Q Capture program" on page 15

Related reference:

- "Descriptions of Q Capture parameters" on page 206
- "IBMQREP_CAPPARMS table" on page 374

Descriptions of Q Capture parameters

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.

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. See each parameter description for details:

- “add_partition (Linux, UNIX, Windows)”
- “adminq”
- “autostop” on page 208
- “capture_path” on page 208
- “capture_schema” on page 208
- “capture_server” on page 209
- “commit_interval” on page 209
- “logreuse” on page 211
- “logstdout” on page 211
- “memory_limit” on page 211
- “monitor_interval” on page 212
- “monitor_limit” on page 212
- “prune_interval” on page 213
- “qmgr” on page 213
- “restartq” on page 214
- “signal_limit” on page 214
- “sleep_interval” on page 215
- “startmode” on page 215
- “term” on page 216
- “trace_limit” on page 217

add_partition (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.

adminq

The **adminq** parameter defines the Q Capture administration queue. This is a queue that a Q Capture program uses 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 and queue manager run. 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 will not run.

The administration queue must be a persistent queue, which means logging is enabled. Circular logging is recommended for this queue. Choose other attributes of this queue such as maximum depth (number of messages allowed) based on your replication or publishing environment.

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.

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.

capture_path

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.

Windows
If you start a Q Capture program as a Windows service, by default the program starts in the \sqllib\bin directory.

z/OS
Because the Q Capture program is a POSIX application, the default path depends on how you start the program:

- If you start a Q Capture program from a USS command line prompt, the path is the directory where you started the program.
- If you start a Q Capture program using a started task or through JCL, the default path is the home directory 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 //QCAPV8. When you use an HLQ, sequential files are created that conform to the file naming conventions for z/OS sequential data set file names.

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

Default (Linux, UNIX, Windows): capture_server=value of DB2DBDFT environment variable, if it is set

Default (z/OS): capture_server=None

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 DB2 log, it must run at the source database or subsystem.

z/OS: For data sharing, do not use the group attach name. Instead, specify a member subsystem name.

commit_interval

Default: commit_interval=500 milliseconds

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 DB2 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 DB2 transactions. If the DB2 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 DB2 transaction is put on the send queue.

When the number of committed DB2 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. To judge whether you can reduce latency by shortening the commit interval, determine if a Q Capture program's idle time after processing all transactions is high (check the CAPTURE_IDLE value in the IBMQREP_CAPMON control table). Also, 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 DB2 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.

Procedure for determining transactions published per commit interval: To calculate the number of transactions published per commit interval, use the following formula.

```
TRANS_PUBLISHED / (MONITOR_INTERVAL / 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. See the online help for details.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 4.
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**. See the online help for details.
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.

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.

logreuse

Default: logreuse=N

Each Q Capture program keeps a 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 DB2 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.

Linux, UNIX, Windows: The log file name is *db2instance.capture_server.capture_schema*.QCAP.log. For example, DB2.SAMPLE.ASN.QCAP.log.

z/OS: The log file name does not contain a DB2 instance name. For example, SAMPLE.ASN.CAP.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.

You can set the **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: logstdout=N

By default, a Q Capture program writes its work history only to the log. You can change the **logstdout** parameter if you want to see the program's history on the standard output (stdout) in addition to the log.

Error messages and some log messages (initialization, stop, subscription activation, and subscription deactivation) go to both the standard output and the log file regardless of the setting for this parameter.

You can set the **logstdout** parameter when you start a Q Capture program with the **asnqcap** command, or while the program is running by using the **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.

memory_limit

Default: memory_limit=32 MB

The **memory_limit** parameter specifies the amount of memory that a Q Capture program can use to build DB2 transactions in memory. By default, a Q Capture program uses a maximum of 32 MB. 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.

z/OS: The **REGION** parameter of the JCL also limits memory usage by a Q Capture program. Because a Q Capture program does not know the region size, the program could reach this operating system limit before reaching the **memory_limit**. To avoid this situation, set **memory_limit** to about half of the region size.

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.

monitor_interval

Default: monitor_interval=300 seconds (5 minutes)

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 300 seconds (5 minutes). Typically, a Q Capture program commits WebSphere MQ transactions at a much shorter interval (the default commit interval is a half second). Thus, if you use shipped defaults for the monitor interval and commit interval, each insert into the monitor tables contains totals for 600 commits. If you want to monitor a Q Capture program's 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.

qmgr

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

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

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 log after the Q Capture program restarts. It is a local queue that is defined on the same system where a Q Capture program and queue manager run. 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.

The restart queue must be a persistent queue, which means logging is enabled. Circular logging is recommended for this queue. Choose other attributes of this queue based on your replication or publishing environment.

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.

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 XML publication, to ignore a DB2 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 XML 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 XML 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.

warmsa (warm start if possible; if not, cold start)
If warm start information is available, the Q Capture program starts reading the log at the point where it left off. If the Q Capture program cannot warm start, it switches to a cold start.

During warm starts, the Q Capture program will only load those Q subscriptions or XML 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 DB2 is quiesced.

By default, a Q Capture program quits running when DB2 is quiesced. You can change the default if you want a Q Capture program to keep running while DB2 is in quiesce mode and has forced all applications to disconnect (including the Q Capture program). In this case, 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.

You cannot change a Q Capture program's reaction to a DB2 shutdown. Regardless of the setting for the **term** parameter, a Q Capture program stops when DB2 shuts down. When DB2 starts again, you will need to start the Q Capture program in warmns mode to begin capturing from the last restart point.

Regardless of whether a Q Capture program is forced to stop capturing changes because of a DB2 quiesce or shutdown, the log point where the program starts after reconnecting to the database does not necessarily match the point in time when the Q Capture program had to disconnect. This is because the Q Capture program must recapture transactions that were in memory when it was forced to disconnect. The last restart message in the restart queue tells the program where to begin again.

Important: If you change the default so that the Q Capture program does not stop running, and DB2 is restarted with restricted access (for example, ACCESS MAINT), a Q Capture program cannot connect and subsequently stops.

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: trace_limit=10080 minutes (7 days)

The **trace_limit** parameter specifies how long rows remain in the IBMQREP_CAPTRACE table before they can be pruned.

The Q Capture program inserts all informational, warning, and error messages into the IBMQREP_CAPTRACE table. By default, rows that are older than 10080 minutes (7 days) are pruned at each pruning interval. Modify the trace limit depending on your need for audit information.

You can set the **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.

Related concepts:

- "Required settings for WebSphere MQ objects" on page 56
- "Parameters of a Q Capture program—Overview" on page 205
- "Q Capture program" on page 15
- "Detailed structure of the Q Capture control tables—Overview" on page 370

Changing the Q Capture parameters

Changing the Q Capture parameters—Overview

You can change the Q Capture program's operating parameters when you start the program, while the program is running, or by updating the IBMQREP_CAPPARMS control table. The following topics provide details:

- "Methods of changing the Q Capture operating parameters"
- "Changing saved Q Capture parameters in the IBMQREP_CAPPARMS table" on page 219
- "Dynamically changing parameters while a Q Capture program is running" on page 220

Related concepts:

- "Operating a Q Capture program—Overview" on page 201

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 Change Parameters – Running Q Capture Program window in the Replication Center, or the **asnqccmd chgparms** command to change values while a Q Capture program is running. Your changes will last only until the program stops running, or until the next change-parameters request.

Example: Three ways to change Q Capture operating parameters

Assume that you want to increase the default setting for the commit interval of 500 milliseconds (a half second) 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:

   ```
   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:

   ```
   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 Change Parameters – Running Q Capture Program window in the Replication Center, or the **asnqccmd chgparms** command:

   ```
   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 using the Replication Center or the **asnqccmd chgparms** command. 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.

Related concepts:

- "Changing the Q Capture parameters—Overview" on page 217
- "Q Capture program" on page 15

Related tasks:

- "Changing saved Q Capture parameters in the IBMQREP_CAPPARMS table" on page 219
- "Dynamically changing parameters while a Q Capture program is running" on page 220
- "Starting a Q Capture program" on page 201

Changing saved Q Capture parameters in the IBMQREP_CAPPARMS table

A Q Capture program stores its operating parameters in the IBMQREP_CAPPARMS 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 Capture program, it uses the values saved in the control table. To change the saved parameter values, you must update 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:

Use one of the following methods to change a Q Capture program's saved operating parameters in the IBMQREP_CAPPARMS table:

Replication Center
: Use the Change Parameters – Saved window to view or change any of the values in the IBMQREP_CAPPARMS table. To open the window, right-click the Q Capture server that contains the Q Capture program whose saved parameters you want to view or change and select **Change Parameters → Saved**. See the online help for details.

SQL
: Use a command prompt or one of the DB2 command line tools to issue an SQL UPDATE statement for the IBMQREP_CAPPARMS table. For example, to change the defaults for **monitor_interval** and **logstdout**:

```
update schema.ibmqrep_capparms set monitor_interval=600, logstdout=Y
```

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

Related concepts:
- "Changing the Q Capture parameters—Overview" on page 217
- "Q Capture program" on page 15

Related reference:
- "IBMQREP_CAPPARMS table" on page 374

Dynamically changing parameters while a Q Capture program is running

You can modify the behavior of a Q Capture program without needing to stop capturing 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 a Q Capture program, it uses the saved values in the control table.

You can change the following Q Capture parameters while the program is running:
- **autostop**
- **commit_interval**
- **logreuse**
- **logstdout**
- **memory_limit**
- **monitor_interval**
- **monitor_limit**
- **prune_interval**
- **term**
- **trace_limit**

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.

Prerequisites:

A Q Capture program whose parameters you want to change must be running.

Procedure:

Use one of the following methods to dynamically change the value of a Q Capture program's operating parameters for the current session:

Replication Center
: 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**. See the online help for details.

asnqccmd system command
: Use the **asnqccmd chgparms** command to change parameters for a running Q Capture program:

```
asnqccmd capture_server=server capture_schema=schema chgparms parameters
```

Where *server* is the name of the Q Capture server, *schema* identifies a running Q Capture program, and *parameters* is one or more parameters that you want to change.

z/OS console or TSO
Use the MODIFY command to change parameter values while a Q Capture program is running.

Related concepts:

- "Changing the Q Capture parameters—Overview" on page 217
- "Q Capture program" on page 15

Related tasks:

- "Modifying Q replication and event publishing programs that are already started by using JCL" on page 296

Related reference:

- "asnqccmd: Working with a running Q Capture program" on page 320

Activating Q subscriptions or XML publications

Before the Q Capture program can start replicating or publishing data from source tables, the Q subscriptions or XML publications that specify those source tables must be in A (active) or N (new) state.

By default, newly created Q subscriptions or XML publications are in N (new) state, and they are automatically activated when the Q Capture program is started or reinitialized. If you change this default, you must activate Q subscriptions or XML publications after you create them.

When Q subscriptions or XML publications are activated, the Q Capture program starts capturing source changes and sending those changes via the WebSphere MQ queues that you defined in a replication queue map or publishing queue map.

Prerequisites:

- 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 activate Q subscriptions or XML 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.
- If any of the Q subscriptions that you want to activate 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. See "Storing user IDs and passwords for remote servers" on page 41 for details.

Procedure:

You can use the Replication Center or SQL to activate Q subscriptions or XML publications.

Replication Center
Use one of the following windows:

- Use the Manage Q Subscriptions window to activate a Q subscription. 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**.
- Use the Manage XML Publications window to activate an XML publication. To open the window, right-click the Q Capture server where the source table for the XML publication is located and select **Manage → XML Publications**.

See the online help for details.

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 or XML publication that you want to activate.

Related concepts:
- "Setting up replication from sources to targets (multidirectional)—Overview" on page 117
- "Operating a Q Capture program—Overview" on page 201
- "Setting up publishing from sources (event publishing)—Overview" on page 155
- "Setting up replication from sources to targets (unidirectional)—Overview" on page 81

Related tasks:
- "Deactivating Q subscriptions or XML publications" on page 222

Related reference:
- "IBMQREP_SIGNAL table" on page 382
- "Activate subscription message" on page 463

Deactivating Q subscriptions or XML publications

You deactivate a Q subscription or XML publication to instruct the Q Capture program to stop capturing changes for the Q subscription or XML publication. Deactivating lets you delete or suspend activity for Q subscriptions or XML publications without stopping the Q Capture program.

You can deactivate a Q subscription or XML publication using the Replication Center, or by inserting a SQL signal into the IBMQREP_SIGNAL table. The Q Capture program stops capturing changes for the Q subscription or XML publication and changes its state to I (inactive) in the IBMQREP_SUBS table.

Prerequisites:

The Q Capture program must be running to read the CAPSTOP signal. If the Q Capture program is stopped when you deactivate a Q subscription or XML publication, it will process the signal only if it is warm started. The signal will be lost if you use cold start.

Procedure:

You can use the Replication Center or SQL to deactivate Q subscriptions or XML publications:

Replication Center
Use one of the following windows:

- Use the Manage Q Subscriptions window to deactivate a Q subscription. 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**.
- Use the Manage XML Publications window to deactivate an XML publication. To open the window, right-click the Q Capture server where the source table for the XML publication is located and select **Manage → XML Publications**.

See the online help for details.

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 or XML publication that you want to deactivate.

Related concepts:
- "Operating a Q Capture program—Overview" on page 201

Related tasks:
- "Activating Q subscriptions or XML publications" on page 221

Related reference:
- "IBMQREP_SIGNAL table" on page 382
- "Deactivate subscription message" on page 463
- "Subscription deactivated message" on page 449

Stopping a Q Capture program

You can stop a Q Capture program, and it will stop reading from the DB2 log and building transactions in memory. Messages that were put on queues will be committed to WebSphere MQ before the Q Capture 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 XML publication:

- If you want to add one or two Q subscriptions or XML publications while the program is running, create the Q subscriptions or XML publications so that they do not start automatically, and then activate them.
- If you want to add a large number of Q subscriptions or XML publications, create them so that they start automatically, and then reinitialize the Q Capture program.
- You can delete a Q subscription or XML publication without stopping the Q Capture program by deactivating the Q subscription or XML publication and then deleting it.

Prerequisites:

The Q Capture program that you want to stop must be running.

Procedure:

Use one of the following methods to stop a Q Capture program:

Replication Center
: Use the Stop Q Capture window to stop a Q Capture program. To open the window, right-click the Q Capture server that contains the Q Capture program that you want to stop and select **Stop Q Capture Program**. See the online help for details.

asnqccmd system command
: Use the **asnqccmd stop** command to stop a Q Capture program:

```
asnqccmd capture_server=server_name capture_schema=schema stop
```

Where *server_name* is the name of the database or subsystem where the Q Capture program is running, and *schema* identifies the Q Capture program that you want to stop.

SQL
: Use a command prompt or one of the DB2 command line tools to insert a STOP signal into the IBMQREP_SIGNAL table at the Q Capture server:

```
insert into schema.IBMQREP_SIGNAL(
    SIGNAL_TIME,
    SIGNAL_TYPE,
    SIGNAL_SUBTYPE,
    SIGNAL_INPUT_IN,
    SIGNAL_STATE )
 values (
     CURRENT TIMESTAMP,
    'CMD',
    'STOP',
    'NULL',
    'P' );
```

Where *schema* identifies the Q Capture program that you want to stop.

Windows 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.

z/OS console or TSO
: Use the MODIFY command to stop a Q Capture program.

Related concepts:

- “Operating a Q Capture program—Overview” on page 201
- “Q Capture program” on page 15

Related tasks:

- “Deleting XML publications” on page 196
- “Deleting Q subscriptions” on page 186
- “Stopping replication in a peer-to-peer group with three or more servers” on page 141
- “Stopping bidirectional or peer-to-peer replication with two servers” on page 140
- “Stopping a Q Apply program” on page 242
- “Activating Q subscriptions or XML publications” on page 221
- “Deactivating Q subscriptions or XML publications” on page 222
- “Starting a Q Capture program” on page 201
- “Stopping a replication service” on page 302
- “Modifying Q replication and event publishing programs that are already started by using JCL” on page 296

Related reference:

- “asnqccmd: Working with a running Q Capture program” on page 320
- “IBMQREP_SIGNAL table” on page 382

Chapter 18. Operating a Q Apply program

Operating a Q Apply program—Overview

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 using the Replication Center, system commands, and system services, and you can change the Q Apply operating parameters in several ways.

The following topics explain how to operate a Q Apply program:
- "Starting a Q Apply program"
- "Parameters of a Q Apply program—Overview" on page 229
- "Changing the Q Apply parameters—Overview" on page 237
- "Starting message processing on a receive queue" on page 239
- "Stopping message processing on a receive queue" on page 241
- "Stopping a Q Apply program" on page 242

Related concepts:
- "Q Apply program" on page 18
- "Schemas for the Q Apply and Q Capture programs" on page 20

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.

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.

Prerequisites:
- If you are starting the Q Apply program from a remote workstation, connections must be configured to the Q Apply server. See "Connectivity requirements for Q replication and event publishing" on page 67 for details.
- A WebSphere MQ queue manager, queues and other necessary objects must be created and configured on the same system as the Q Apply server. See "WebSphere MQ objects required for Q replication and event publishing—Overview" on page 43 for details.
- You must have the proper authorization for Q replication objects and WebSphere MQ objects. See "Authorization requirements for the Q Apply program" on page 38 and "Connectivity and authorization requirements for WebSphere MQ objects" on page 62 for details.
- Control tables must be created for the appropriate Q Apply schema. See "Creating control tables for the Q Capture and Q Apply programs" on page 76 for details.

- The target database or subsystem must be configured to work with the Q Apply program. See "Configuring the target database to work with the Q Apply program (Linux, UNIX, Windows)" on page 71 and "Configuring databases for Q replication and event publishing (z/OS)" on page 75 for details.
- A password file for Q Apply authentication at the Q Capture server must exist if you have Q subscriptions that specify an automatic load for targets that use the EXPORT/IMPORT or EXPORT/LOAD utilities. See "asnpwd: Creating and maintaining password files" on page 344 for details.

Procedure:

Use one of the following methods to start a Q Apply program:

Replication Center
: Use the Start Q Apply window to start a Q Apply program. 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. See the online help for details.

asnqapp system command
: Use the **asnqapp** command to start a Q Apply program and optionally specify startup parameters:

```
asnqapp apply_server=server_name apply_schema=schema parameters
```

Where *server_name* is the name of the database or subsystem where the Q Apply control tables are defined and where the Q Apply program will apply changes to targets, *schema* identifies the Q Apply program that you want to start, and *parameters* is one or more parameters that you can specify at startup.

z/OS console or TSO
: On z/OS, you can start the Q Apply program using JCL or as a system-started task. You can specify new invocation parameter values when you start a Q Apply program with JCL. These values will override the parameter values that are stored in the IBMQREP_APPLYPARMS table. The PARM parameter of the EXEC statement cannot have subparameters that exceed 100 characters.

Windows services
: You can create a DB2 replication service on the Windows operating system to start the Q Apply program automatically when the system starts.

To verify whether a Q Apply program is started, use one of the following methods:

- 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**. See the online help for details.
- 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**. See the online help for details.
- Examine the Q Apply log file (*db2instance.apply_server.apply_schema*.QAPP.log on Linux, UNIX, and Windows and *apply_server.apply_schema*.QAPP.log on z/OS) for a message that indicates that the program is applying changes.

- Check the IBMQREP_APPLYTRACE table for a message that indicates that the program is applying changes.

Related concepts:
- "Operating a Q Apply program—Overview" on page 227
- "Parameters of a Q Apply program—Overview" on page 229
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Q Apply program" on page 18

Related tasks:
- "Activating Q subscriptions or XML publications" on page 221
- "Managing replication services with the Windows Service Control Manager (SCM) —Overview" on page 299
- "Starting the Q Apply program by using JCL" on page 295

Related reference:
- "asnqapp: Starting a Q Apply program" on page 323
- "IBMQREP_APPLYPARMS table" on page 395
- "IBMQREP_APPLYTRACE table" on page 397

Parameters of a Q Apply program

Parameters of a Q Apply program—Overview

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. The following topics list the default operating parameters, and provide details about each parameter and reasons that you might want to change the default settings:
- "Default values for Q Apply operating parameters"
- "Descriptions of Q Apply parameters" on page 231

Related concepts:
- "Operating a Q Apply program—Overview" on page 227

Default values for Q Apply operating parameters

When you create control tables using the Replication Center, the IBMQREP_APPLYPARMS table is created with a single row that contains default values for the Q Apply program's operating parameters. Table 14 on page 230 shows these default values.

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 14. Default values for Q Apply operating parameters

Operating parameter	Default value	Column name in IBMQREP_APPLYPARMS table
apply_path	Directory where Q Apply was started [1]	APPLY_PATH
apply_schema	ASN [2]	not applicable
apply_server	DB2DBDFT [3]	not applicable
autostop	N [4]	AUTOSTOP
deadlock_retries	3	DEADLOCK_RETRIES
logreuse	N [4]	LOGREUSE
logstdout	N [4]	LOGSTDOUT
monitor_interval	300 [5]	MONITOR_INTERVAL
monitor_limit	10080 [6]	MONITOR_LIMIT
prune_interval	300 [5]	PRUNE_INTERVAL
pwdfile	`asnpwd.aut`	PWDFILE
qmgr	None [7]	QMGR
term	Y [8]	TERM
trace_limit	10080 [6]	TRACE_LIMIT

Notes:

1. If a Q Apply program starts as a Windows service, the **apply_path** is \sqllib\bin. If Q Apply is started using JCL on z/OS, the default is the user ID associated with the started task or job.
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 Linux, UNIX, and Windows, the default Q Apply server is the value of the DB2DBDFT environment variable, if specified. For z/OS, there is no default Q Apply server.
4. No.
5. Seconds.
6. Minutes.
7. 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.
8. Yes.

Related concepts:

- "Changing the Q Apply parameters—Overview" on page 237
- "Parameters of a Q Apply program—Overview" on page 229
- "Q Apply program" on page 18

Related reference:

- "Descriptions of Q Apply parameters" on page 231
- "IBMQREP_APPLYPARMS table" on page 395

Descriptions of Q Apply parameters

The following topics describe the Q Apply program's operating parameters and discuss reasons that you might want to change the default values based on your needs:

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. See each parameter description for details:

- "autostop"
- "apply_path" on page 232
- "apply_schema" on page 232
- "apply_server" on page 232
- "deadlock_retries" on page 232
- "logreuse" on page 233
- "logstdout" on page 234
- "monitor_interval" on page 234
- "monitor_limit" on page 234
- "pwdfile" on page 235
- "prune_interval" on page 235
- "qmgr" on page 236
- "term" on page 236
- "trace_limit" on page 236

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 they 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.

apply_path

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 of the user ID 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.

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.

apply_server

Default (Linux, UNIX, Windows): apply_server=value of DB2DBDFT environment variable, if it is set

Default (z/OS): apply_server=None

The **apply_server** parameter identifies the database or subsystem where a Q Apply program runs, and where its control tables are stored. The control tables contain information about targets, Q subscriptions, WebSphere MQ queues, and user preferences. The Q Apply server must be the same database or subsystem that contains the targets.

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. The Q Apply program waits one second between each try. 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.

The Q Apply program's behavior when it encounters deadlocks differs depending on what it was doing:

- If the deadlock occurs at a target table, the Q Apply program keeps trying until it reaches the limit that you set. After the limit is reached and deadlocks persist, the Q Apply program takes the error action that you specified for the Q subscription.
- If the browser thread is accessing a control table when the deadlock occurs, the thread keeps trying until it reaches the limit that you set. After the limit is reached and deadlocks persist, the thread stops gracefully.
- If the Q Apply main thread is accessing a control table, the thread keeps trying until it reaches the limit that you set. After the limit is reached and deadlocks persist, the Q Apply program stops gracefully.
- If the Q Apply program is accessing control tables for monitoring or pruning, it keeps trying and logs the error.

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.

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.

Linux, UNIX, Windows: The log file name is *db2instance.apply_server.apply_schema*.QAPP.log. For example, DB2.SAMPLE.ASN.QAPP.log.

z/OS: The log file name does not contain a DB2 instance name. 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.

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.

monitor_interval

Default: monitor_interval=300 seconds (5 minutes)

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. See the Replication Center online help for details.

By default, rows are inserted into this table every 300 seconds (five minutes). 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 minute rather than five-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 five-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 they 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.

pwdfile

Default: pwdfile=asnpwd.aut

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/IMPORT or EXPORT/LOAD utilities. 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

The **qmgr** parameter specifies the name of a WebSphere MQ queue manager that a Q Apply program uses. 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 and event publishing 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.

term

Default: term=Y

The **term** parameter controls whether a Q Apply program keeps running when DB2 is quiesced.

By default (**term**=Y), the Q Apply program terminates when DB2 is quiesced. You can change the default if you want a Q Apply program to keep running while DB2 is in quiesce mode and has forced all applications to disconnect (including the Q Apply program). In this case, when DB2 is taken out of quiesce mode, the Q Apply program begins applying transactions where it left off without requiring you to restart the program.

You cannot change a Q Apply program's reaction to a DB2 shutdown. Regardless of the setting for **term**, a Q Apply program stops when DB2 shuts down. When DB2 starts again, you will need to start the Q Apply program to begin reading messages and applying transactions.

Important: If you change the default so that the Q Apply program does not stop running, and DB2 is restarted with restricted access (for example, ACCESS MAINT), a Q Apply program cannot connect and subsequently stops.

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 they 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.

Related concepts:
- "Required settings for WebSphere MQ objects" on page 56
- "Parameters of a Q Apply program—Overview" on page 229
- "Q Apply program" on page 18
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Changing the Q Apply parameters

Changing the Q Apply parameters—Overview

You can change the Q Apply program's 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 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 217. The methods are the same for a Q Apply program.

See the following topics for more details about the methods of changing the Q Apply parameters:
- "Changing saved Q Apply parameters in the IBMQREP_APPLYPARMS table"
- "Dynamically changing parameters while a Q Apply program is running" on page 238

Related concepts:
- "Operating a Q Apply program—Overview" on page 227

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.

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:

Use one of the following methods to change saved parameters for a Q Apply program in the IBMQREP_APPLYPARMS table:

Replication Center
Open the Change Parameters – Saved window to view or change any of the values in the IBMQREP_APPLYPARMS table. To open the window, right-click the Q Apply server that contains the Q Apply program whose saved parameters you want to view or change and select **Change Parameters → Saved**. See the online help for details.

SQL
Use a command prompt or one of the DB2 command line tools to issue an SQL UPDATE statement for the IBMQREP_APPLYPARMS table. For example, to change the defaults for **prune_interval** and **deadlock_retries**:

```
update schema.ibmqrep_applyparms set prune_interval=600, deadlock_retries=10
```

Where *schema* identifies the Q Apply program whose saved parameter values you want to change.

Related concepts:
- "Changing the Q Apply parameters—Overview" on page 237
- "Q Apply program" on page 18

Related reference:
- "IBMQREP_APPLYPARMS table" on page 395

Dynamically 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.

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**

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 Apply program changes its operation.

Prerequisites:

The Q Apply program whose parameters you want to change must be running.

Procedure:

Use one of the following methods to dynamically change the value of a Q Apply program's operating parameters for the current session:

Replication Center
: In the Replication Center, 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**. See the online help for details.

asnqacmd system command
: Use the **asnqacmd chgparms** command to change parameters for a running Q Apply program:

```
asnqacmd apply_server=server_name apply_schema=schema chgparms parameters
```

Where *server_name* 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.

z/OS console or TSO
: Use the MODIFY command to change parameter values while a Q Apply program is running.

Related concepts:
- "Changing the Q Apply parameters—Overview" on page 237
- "Q Apply program" on page 18

Related tasks:
- "Modifying Q replication and event publishing programs that are already started by using JCL" on page 296

Related reference:
- "asnqacmd: Working with a running Q Apply program" on page 326
- "IBMQREP_APPLYPARMS table" on page 395

Starting message processing on a receive queue

You might need to instruct a Q Apply program to start processing messages on a receive queue for several reasons:
- A conflict, SQL error, or persistent deadlocks occurred at the target, which prompted the Q Apply program to stop or to stop reading from the receive queue.
- You instructed the Q Apply program to stop processing messages from the receive queue.

Prerequisites:

- The Q Apply program must be running.
- The receive queue must be in I (inactive) state.

Procedure:

You can start message processing on a receive queue by using the Replication Center or the command line:

Replication Center
: Use the Manage Receive Queues window to start message processing on a receive queue. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage → Receive Queues**. See the online help for details.

asnqacmd command
: Use the **asnqacmd startq** command to start message processing on a receive queue:

```
asnqacmd apply_server=server_name apply_schema=schema
   startq=receive_queue_name
```

Where *server_name* is the name of the Q Apply server, *schema* identifies a Q Apply program, and *receive_queue_name* is the name of the receive queue for which you want to start message processing.

To verify that the Q Apply program started reading messages from the queue, do one of the following things:

- 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';
  ```

  Where *schema* identifies a Q Apply program, and *receive_queue_name* is the name of the receive queue that you are checking.

Related concepts:

- "Operating a Q Apply program—Overview" on page 227

Related tasks:

- "Stopping message processing on a receive queue" on page 241

Related reference:

- "asnqacmd: Working with a running Q Apply program" on page 326
- "IBMQREP_RECVQUEUES table" on page 402

Stopping message processing on a receive queue

You can use the Replication Center or a command to instruct a Q Apply program to stop processing messages on a receive queue. 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.

Prerequisites:

- The Q Apply program must be running.
- The receive queue must be in A (active) state.

Procedure:

You can stop message processing on a receive queue by using the Replication Center or the command line:

Replication Center
: Use the Manage Receive Queues window to stop message processing on a receive queue. To open the window, right-click the Q Apply server where the receive queue is located and select **Manage → Receive Queues**. See the online help for details.

asnqacmd command
: Use the **asnqacmd stopq** command to stop message processing on a receive queue:

```
asnqacmd apply_server=server_name apply_schema=schema
   stopq=receive_queue_name
```

Where *server_name* is the name of the Q Apply server, *schema* identifies the Q Apply program that is processing messages from the receive queue, and *receive_queue_name* is the name of the receive queue for which you want to stop message processing.

To verify that the Q Apply program stopped reading messages from the queue, do one of the following things:

- Use the Manage Receive Queues window to see whether the state of the queue changed to I (inactive).

- Check the Q Apply diagnostic log file for a message that indicates that processing stopped for the queue.
- Look at the IBMQREP_RECVQUEUES control table to see whether the state of the queue changed to I (inactive) in the STATE column:

  ```
  SELECT RECVQ, STATE FROM schema.IBMQREP_RECVQUEUES
  WHERE RECVQ = 'receive_queue_name';
  ```

  Where *schema* identifies a Q Apply program, and *receive_queue_name* is the name of the receive queue that you are checking.

Related concepts:
- "Operating a Q Apply program—Overview" on page 227

Related tasks:
- "Starting message processing on a receive queue" on page 239

Related reference:
- "asnqacmd: Working with a running Q Apply program" on page 326
- "IBMQREP_RECVQUEUES table" on page 402

Stopping a Q Apply program

You can stop a Q Apply program using the Replication Center or system commands. 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.

Important:
- 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.

Tip: If you want to stop a Q Apply program from reading messages from a single receive queue, use the **stopq** command.

Prerequisites:

The Q Apply program that you want to stop must be running.

Procedure:

Use one of the following methods to stop a Q Apply program:

Replication Center
Use the Stop Q Apply window to stop a Q Apply program. 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**. See the online help for details.

asnqacmd system command
You can stop the Q Apply program using the **asnqacmd stop** command:

```
asnqacmd apply_server=server_name apply_schema=schema stop
```

Where *server_name* is the name of the Q Apply server and *schema* identifies a running Q Apply program.

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 Apply program.

z/OS console or TSO
Use the MODIFY command to stop the Q Apply program.

Related concepts:
- "Operating a Q Apply program—Overview" on page 227
- "Options for loading target tables for Q replication—Overview" on page 143

Related tasks:
- "Starting a Q Apply program" on page 227
- "Stopping a Q Capture program" on page 224
- "Stopping a replication service" on page 302
- "Modifying Q replication and event publishing programs that are already started by using JCL" on page 296

Related reference:
- "asnqacmd: Working with a running Q Apply program" on page 326

Chapter 19. Viewing reports about the Q replication and event publishing programs

Viewing reports about the Q replication and event publishing programs—Overview

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.

The following topics provide details about the type of reports that you can view, and explain two important concepts in understanding performance issues, latency and exceptions:

- "Checking the status of the Q replication and event publishing programs"
- "Threads of the Q Capture, Q Apply, and Replication Alert Monitor programs" on page 246
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Latency" on page 250
- "Exceptions" on page 252

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.

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.

Procedure:

Use one of the following methods to check the status of the Q replication and event publishing programs:

Replication Center

Use the Check Status window to check the current status of the Q Capture or Q Apply program. (You cannot check the status of the Replication Alert Monitor using the Replication Center.)

- To view Q Capture status, right-click the Q Capture server that contains the Q Capture program that you want to check, and select **Check Status**.
- To view Q Apply status, right-click the Q Apply server that contains the Q Apply program that you want to check, and select **Check Status**.

See the online help for details.

Command line

Use the following commands to check program status:

asnqccmd system command (Q Capture program)
Use the **asnqccmd status** command to view messages that indicate the state of each Q Capture thread:

```
asnqccmd capture_server=server_name capture_schema=schema status
```

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.

asnqacmd system command (Q Apply program)
Use the **asnqacmd status** command to view messages that indicate the state of each Q Apply thread:

```
asnqacmd apply_server=server_name apply_schema=schema status
```

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.

asnmcmd system command (Replication Alert Monitor)
Use the **asnmcmd status** command to view messages that indicate the state of each monitor thread:

```
asnmcmd monitor_server=server_name monitor_qual=qualifier status
```

Where *server_name* is the name of the database or subsystem where the monitor is running, and *qualifier* identifies the monitor that you want to check.

Related concepts:
- "Viewing reports about the Q replication and event publishing programs—Overview" on page 245
- "Threads of the Q Capture, Q Apply, and Replication Alert Monitor programs" on page 246

Related reference:
- "asnmcmd: Working with a running Replication Alert Monitor" on page 333
- "asnqccmd: Working with a running Q Capture program" on page 320
- "asnqacmd: Working with a running Q Apply program" on page 326

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:

Worker
Reads the DB2® recovery log, captures changes for subscribed

tables, rebuilds transactions in memory as transaction 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.

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.

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

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 15.

Table 15. Descriptions of thread states for Q replication and event publishing programs, and suggested actions

Thread state	Description	Suggested action
Running	The thread is actively processing.	No action.
Exists	The thread exists but cannot start.	Contact IBM Software Support.
Started	The thread started but cannot initialize.	Investigate potential system resource problems, such as too many threads or not enough processing power.
Initializing	The thread is initialized but cannot work.	Contact IBM Software Support.
Resting	The thread is sleeping and will wake up when there is work to do.	No action.
Stopped	The thread is not running.	Use the Q Capture Messages window, Q Apply Messages window, or Monitor Messages window in the Replication Center to search for messages that explain why the thread stopped. Or look for similar messages in the IBMQREP_CAPTRACE, IBMQREP_APPLYTRACE, or IBMSNAP_MONTRACE control tables.

Related concepts:
- “The Replication Alert Monitor” on page 255
- “Q Apply program” on page 18
- “Q Capture program” on page 15
- “Viewing reports about the Q replication and event publishing programs—Overview” on page 245

Related tasks:
- “Checking the status of the Q replication and event publishing programs” on page 245

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_APPLYMON, 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 16 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 16. Where to find historical and performance information in the Replication Center

To answer this question	Use this window
What are the recent messages from the Q Capture, Q Apply, and Monitor programs?	Q Capture Messages Q Apply Messages Monitor Messages
• How many rows and transactions were put on send queues? • How many rows and transactions were captured but not put on send queues because of Q subscription or XML publication definitions? • How many transactions were spilled to a file? • What was the largest transaction processed? • How much memory is the Q Capture program using?	Q Capture Throughput
• How many rows and transactions were applied? • How many rows were not applied because of conflicts or errors? • How many times were rows retried because of lock timeouts and deadlocks? • How many transactions were not applied in parallel because of conflicts? • How many transactions violated referential integrity constraints? • How many times were rows retried because of referential integrity constraints? • How many rows were put in temporary spill queues? • How many spilled rows were applied?	Q Apply Throughput
Approximately how current is the Q Capture program in reading the DB2 recovery log?	Q Capture Latency
How much time elapsed between: • Transactions being committed at the source database and being put on send queues? • Transactions being put on send queues and being taken from receive queues? • Transactions being taken from receive queues and being committed at the target? • Transactions being committed at the source and being committed at the target?	Latency

Table 16. Where to find historical and performance information in the Replication Center (continued)

To answer this question	Use this window
What kinds of conflicts and SQL errors caused some rows to not be applied? What were the details?	Exceptions
What alerts were issued by the Replication Alert Monitor?	Show Alerts

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

Q Capture server

Reports →
  Messages
  Q Capture Throughput
  Q Capture Latency

Q Apply server

Reports →
  Messages
  Q Apply Throughput
  Latency
  Exceptions

Monitor qualifier

Show Monitor Messages
Show Alerts

See the online help for details about using each window.

Related concepts:
- "Exceptions" on page 252
- "Latency" on page 250
- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255
- "Q Apply program" on page 18
- "Q Capture program" on page 15
- "Viewing reports about the Q replication and event publishing programs—Overview" on page 245
- "Detailed structure of the Q Apply control tables—Overview" on page 391
- "Detailed structure of the Q Capture control tables—Overview" on page 370

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.

Note: 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 would be one minute. To improve log-reading performance, consider creating an additional Q Capture schema and moving some Q subscriptions or XML 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 XML 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_MSG_SIZE parameter, which is defined when you create 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

Q 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**.

See the online help for details.

Related concepts:
- "Q Apply program" on page 18
- "Q Capture program" on page 15
- "Viewing reports about the Q replication and event publishing programs—Overview" on page 245

Related tasks:
- "Creating replication queue maps" on page 84

Related reference:
- "IBMQREP_APPLYMON table" on page 393
- "IBMQREP_CAPMON table" on page 372
- "IBMQREP_CAPQMON table" on page 377

Exceptions

Exceptions are rows that a Q Apply program did not apply to targets 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.

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 that it could not apply, 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, he 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 XML format, using the same tags and schema that the Q Capture program uses to create an XML data message for event publishing. (A schema is a file that defines what tags are legal in an XML document. For messages from the Q Capture program, the schema document is mqcap.xsd.) The rows are saved in XML so that an application can recover the data, and also because the tagging makes it easier to view the column names and values, and the type of row operation (insert, update, or delete).

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**.

Related concepts:

- "Q Apply program" on page 18
- "Viewing reports about the Q replication and event publishing programs—Overview" on page 245
- "Error options for Q replication" on page 103

Related reference:

- "IBMQREP_EXCEPTIONS table" on page 400

Chapter 20. Monitoring replication with the Replication Alert Monitor

Monitoring replication with the Replication Alert Monitor—Overview

You can use the Replication Alert Monitor to monitor an SQL replication, Q replication, or event publishing environment. The following topics explain how the Replication Alert Monitor works:

- "The Replication Alert Monitor"
- "Alert conditions and notifications for the Replication Alert Monitor—Overview" on page 257
- "Setting up the Replication Alert Monitor" on page 263
- "Operating the Replication Alert Monitor" on page 265

The Replication Alert Monitor

The Replication Alert Monitor is a program that checks the status of your replication environment. When the Replication Alert Monitor is running, it the automatically checks the status of replication and notifies you about certain conditions that occur in you 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.

You can check the status of your replication environment manually by using the following methods:

- You can view Replication Center windows that report statistics about the Capture, Q Capture, Apply, and Q Apply programs.
- You can run select statements on your control tables to view statistics about the operation of these programs.

In both of these cases, the statistics are available at any time, but you must retrieve them manually. Manually checking statistics about your replication environment is not an effective way to monitor replication activity continuously on many servers. The Replication Alert Monitor will *automatically* monitor your replication environment across all of your operating systems. It checks replication on your servers and *automatically* alerts you to conditions that require attention.

The Replication Alert Monitor can monitor the following replication environments:

- SQL replication
- Q replication
- Event publishing

You can use the Replication Alert Monitor to monitor the following replication programs:

- Capture program (SQL replication)
- Apply program (SQL replication)
- Q Capture program (Q replication or event publishing)
- Q Apply program (Q replication)

You can configure the Replication Alert Monitor in one of two ways: you can run one monitor, or you can run multiple monitors. Typically you use one monitor when you have few replication programs to monitor. Use additional monitors to monitor many replication programs, prioritize the monitoring of certain programs, or split up the monitoring workload. Setting up multiple monitors is like cloning the Replication Alert Monitor. You create distinct, independent, but similar Replication Alert Monitors, called monitors, to monitor the servers in your system. These monitors do not communicate with each other, but they each send alerts about the servers that they are monitoring. So collectively, these monitors send you alerts for all of the servers in your system. When you set up multiple monitors, the control information for each monitor is stored on the server that it is assigned 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.

Whether you set up many monitors, or whether you configure one monitor, any server that contains control information is called a Monitor control server.

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 via e-mail or pager.

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 for a person to receive alerts from the Replication Alert Monitor.

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

The Replication Alert Monitor monitors servers on DB2® UDB for Linux, UNIX®, Windows®, or z/OS™ operating systems. You can use the Replication Alert Monitor to monitor DB2 UDB replication programs whose control tables are at DB2 replication Version 8 architecture or later.

Restrictions:
- In the case of iSeries™ servers, the Replication Alert Monitor must run on a Linux, UNIX, or Windows server. In this case the Replication Alert Monitor must monitor the iSeries server remotely.
- You cannot set up Monitor control servers on DB2 for iSeries servers.
- The Replication Alert Monitor does not monitor triggers that are associated with non-DB2 relational databases that are used as sources in a federated database system.

Related concepts:
- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255

Alert conditions and notifications for the Replication Alert Monitor

Alert conditions and notifications for the Replication Alert Monitor—Overview

The following topics contain information about alert conditions for the Replication Alert Monitor:
- "Alert conditions for the Replication Alert Monitor" on page 258
- "E-mail notifications for replication alert conditions" on page 261
- "The ASNMAIL exit routine for sending alerts in replication" on page 262

Related concepts:
- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255

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 tables describe alert conditions that you can use to monitor your replication environment.
- "Alert conditions for the Q Capture program"
- "Alert conditions for the Q Apply program"
- "Alert conditions for the Capture program" on page 259
- "Alert conditions for the Apply program" on page 260

Alert conditions for the Q Capture program

Table 17 describes the alert conditions for the Q Capture program.

Table 17. Alert conditions for the Q Capture program

Alert condition	Description
QCAPTURE_STATUS	The Replication Alert Monitor sends an alert when a Q Capture program is not running.
QCAPTURE_ERRORS	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.
QCAPTURE_WARNINGS	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.
QCAPTURE_LATENCY	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.
QCAPTURE_MEMORY	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.
QCAPTURE_TRANSIZE	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.
QCAPTURE_SUBSINACT	The Replication Alert Monitor sends an alert when a Q Capture program deactivates a Q subscription.

Alert conditions for the Q Apply program

Table 18 describes the alert conditions for the Q Apply program.

Table 18. Alert conditions for the Q Apply program

Alert condition	Description
QAPPLY_STATUS	The Replication Alert Monitor sends an alert when a Q Apply program is not running.

Table 18. Alert conditions for the Q Apply program (continued)

Alert condition	Description
QAPPLY_ERRORS	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.
QAPPLY_WARNINGS	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.
QAPPLY_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 Replication Alert Monitor sends an alert when the Q Apply latency exceeds the threshold that you specify. Q Apply latency is measured in milliseconds.
QAPPLY_EELATENCY	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.
QAPPLY_MEMORY	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.
QAPPLY_EXCEPTIONS	The Replication Alert Monitor sends an alert when it finds an exception in the operation of a Q Apply program.
QAPPLY_SPILLQDEPTH	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.
QAPPLY_QDEPTH	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.

Alert conditions for the Capture program

Table 19 describes the alert conditions for the Capture program.

Table 19. Alert conditions for the Capture program

Alert condition	Description
CAPTURE_STATUS	The Replication Alert Monitor sends an alert when a Capture program is not running.
CAPTURE_ERRORS	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.
CAPTURE_WARNINGS	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.
CAPTURE_LASTCOMMIT	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.
CAPTURE_CLATENCY	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.

Table 19. Alert conditions for the Capture program (continued)

Alert condition	Description
CAPTURE_HLATENCY	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.
CAPTURE_MEMORY	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.

Alert conditions for the Apply program

Table 20 describes the alert conditions for the Apply program.

Table 20. Alert Conditions for the Apply program

Alert condition	Description
APPLY_STATUS	The Replication Alert Monitor sends an alert when an Apply program is not running.
APPLY_SUBSFAILING	The Replication Alert Monitor sends an alert when a subscription fails.
APPLY_SUBSINACT	The Replication Alert Monitor sends an alert when a subscription is deactivated.
APPLY_ERRORS	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
APPLY_WARNINGS	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
APPLY_FULLREFRESH	The Replication Alert Monitor sends an alert when there is a full refresh.
APPLY_REJTRANS	The Replication Alert Monitor sends an alert when a transaction is rejected in a subscription set.
APPLY_SUBSDELAY	The Replication Alert Monitor sends an alert when the delay in processing a subscription is longer than the threshold that you specify.
APPLY_REWORKED	The Replication Alert Monitor sends an alert when the Apply program reworks more rows in a subscription set than the threshold that you specify.
APPLY_LATENCY	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.

Related concepts:

- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255
- "The Replication Alert Monitor" on page 255

Related tasks:

- "Selecting alert conditions for the Replication Alert Monitor" on page 267

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.

Example e-mail notification to non-pager 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 Replication Alert 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 Replication Alert Monitor groups all alerts for that server together.

In Q replication, the Replication Alert 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 Replication Alert 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 Replication Alert 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 Replication Alert 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. See "The ASNMAIL exit routine for sending alerts in replication" for details.

Related concepts:
- "Alert conditions for the Replication Alert Monitor" on page 258

Related tasks:
- "Operating the Replication Alert Monitor" on page 265
- "Specifying notification criteria for selected alert conditions" on page 274

Related reference:
- "The ASNMAIL exit routine for sending alerts in replication" on page 262

The ASNMAIL exit routine for sending alerts in replication

The ASNMAIL exit routine handles notification. This exit routine takes the following input:

```
asnmail email_server to_address subject alert_message alert_message
```

Table 21 describes the inputs for the ASNMAIL exit routine.

Table 21. Inputs for the ASNMAIL exit routine

Input	Description
email_server	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.
to_address	This is the e-mail address of the contact to be notified.
subject	This is the subject in the notification.
alert_message	This is the string that contains the alert message.

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\repl\ directory contains a sample of the ASNMAIL exit routine. The asnmail.c sample contains the input parameters and directions for using the sample program.

Related concepts:
- "E-mail notifications for replication alert conditions" on page 261
- "Alert conditions for the Replication Alert Monitor" on page 258

Related tasks:
- "Operating the Replication Alert Monitor" on page 265

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.

Procedure:

Use this procedure once when you install the Replication Alert Monitor. To set up the Replication Alert Monitor to monitor your replication environment:

1. Evaluate your monitoring load. The more servers that you have to monitor, the greater the monitoring load.
2. Choose how you want to configure your monitors. You have two configuration options. You can run one monitor or multiple monitors. 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 also 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.
3. Choose which servers to set up as monitor control servers. Choose one monitor control server for each monitor that you would like to set up.
4. Create control tables for each Monitor control server. See "Creating control tables for the Replication Alert Monitor" for details.
5. Define contact information for the Replication Alert Monitor. See "Defining contact information for the Replication Alert Monitor" on page 264 for details.

Related concepts:

- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255

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

The Monitor control server can be a DB2 UDB for Linux, UNIX, Windows, or z/OS server. 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 control tables for the Replication Alert Monitor, use the Create Monitor Control Tables window in the Replication Center. To open the window, right-click the **Monitor Control Servers** folder and select **Create Monitor Control Tables**. See the online help for details.

Related concepts:

- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255

Related reference:

- "List of tables at the Monitor control server" on page 369

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. You can change contact information after monitors are running.

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.

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 `DB2 administrators` that contains the contact information for all your DB2 administrators. You can also copy contact and group information from one server to another.

Procedure:

To define contact information for the Replication Alert Monitor:

1. Create contacts and contact groups for the monitors on a Monitor control server.
   a. In the Replication Center, use the Create Contact window to define contact information.
   b. Optional: Create contact groups by using the Create Contact Group window in the Replication Center.

   To open the windows, expand the Monitor control server for which you want to add a contact or contact group, right-click the **Contacts** folder and select **Create Contact → Person** or **Create Contact → Group**. See the online help for details.
2. Optional: Copy contact information from one Monitor control server to another Monitor control server 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**. See the online help for details.

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

Related concepts:
- "E-mail notifications for replication alert conditions" on page 261

Related reference:
- "The ASNMAIL exit routine for sending alerts in replication" on page 262

Operating the Replication Alert Monitor

Operating the Replication Alert Monitor

You can monitor your replication environment by running the Replication Alert Monitor.

Prerequisites:

Before you can operate the Replication Alert Monitor, you must set up the Replication Alert Monitor. See "Setting up the Replication Alert Monitor" on page 263 for details.

Procedure:

You must perform the following tasks for each monitor. The following topics explain how to operate the Replication Alert Monitor:
1. Create the monitor. See "Creating monitors for replication or publishing" on page 266 for details.
2. Select alert conditions for the monitor. See "Selecting alert conditions for the Replication Alert Monitor" on page 267 for details.
3. Optional: Set parameters for the monitor. See "Setting parameters for the Replication Alert Monitor—Overview" on page 272 for details.
   - "Descriptions of the parameters that are used to operate the Replication Alert Monitor" on page 270
   - "Default values of the parameters that are used to operate the Replication Alert Monitor" on page 269
4. Start the Replication Alert Monitor. See "Starting monitors" on page 268 for details.

You can set parameters before you start a monitor, when you start a monitor, or while a monitor is running. To refresh the parameter settings for a running monitor, you must reinitialize the monitor. See "Reinitializing monitors" on page 268 for details.

You can also stop a running monitor. See "Stopping a monitor" on page 275 for details.

Related concepts:
- "Monitoring replication with the Replication Alert Monitor—Overview" on page 255

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.

Prerequisites:

Before you create monitors, you must set up the Replication Alert Monitor. See "Setting up the Replication Alert Monitor" on page 263 for details.

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:

   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**.

   a. On the Start page, specify a monitor qualifier. Then, specify the programs that you would like 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

   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:
   a. Specify the programs or subscription sets that you want to monitor.
   b. 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.
3. On the Summary page, click **Finish**.

See the online help for details.

Related concepts:
- "The Replication Alert Monitor" on page 255

Selecting alert conditions for the Replication Alert Monitor

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.

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.

You can also use the Replication Center to change alert conditions while the monitor is running. You do this by opening an existing monitor, changing the alert conditions, and then reinitializing the monitor.

Procedure:

To select alert conditions for the Replication Alert Monitor:

1. 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:
   - 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
2. 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.

To change alert conditions for the Replication Alert Monitor:

1. In the Replication Center, open the Alert Conditions window for a Q Capture program, Q Apply program, Capture program, Apply program, or subscription set. To open the windows:
   a. Expand the **Monitors** folder within the correct Monitor control server.
   b. Select a monitor.
   c. In the contents pane, right-click the Q Capture schema, Q Apply schema, Capture schema, Apply schema, or subscription set that you want to change alert conditions for.
   d. Select **Change**.
2. Change the alert conditions.
3. Reinitialize the monitor. See “Reinitializing monitors” on page 268 for details.

See the online help for details.

Related concepts:

- “Alert conditions for the Replication Alert Monitor” on page 258

- "The Replication Alert Monitor" on page 255

Starting monitors

You use the Replication Center to start a monitor. You can 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.

Prerequisites:

- Create a monitor. See "Creating monitors for replication or publishing" on page 266 for details.
- Create a password file. See "asnpwd: Creating and maintaining password files" on page 344 for details.
- 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:

Replication Center
: 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 **Start Monitor**. See the online help for details.

asnmon system command
: You can use the **asnmon** command to start a monitor and optionally specify startup parameters. See "asnmon: Starting a Replication Alert Monitor" on page 329 for details.

Windows Service Control Manager
: You can set up the Windows Service Control Manager to run a monitor.

z/OS console or TSO
: You can set up the Automatic Restart Manager (ARM) recovery system to start a monitor.

Related concepts:

- "The Replication Alert Monitor" on page 255
- "The Automatic Restart Manager (ARM) recovery system" on page 297

Related tasks:

- "Reinitializing monitors" on page 268
- "Stopping a monitor" on page 275
- "Scheduling the replication programs (Windows)" on page 304

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:

Replication Center
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**. See the online help for details.

asnmcmd system command
You can use the **asnmcmd reinit** command to reinitialize a running monitor. See “asnmcmd: Working with a running Replication Alert Monitor” on page 333 for details.

Related tasks:

- “Starting monitors” on page 268
- “Stopping a monitor” on page 275

Default values of the parameters that are used to operate the Replication Alert Monitor

You can change the behavior of the Replication Alert Monitor by setting values for the Replication Alert Monitor parameters. Table 22 shows the default value for each parameter.

Table 22. Default values for Replication Alert Monitor operating parameters

Operational parameter	Default value
alert_prune_limit	10080 minutes
autoprune	Y
email_server	no default value
max_notification_minutes	60 minutes
max_notifications_per_alert	3
monitor_errors	no default value
monitor_interval	300 seconds
monitor_limit	10080 minutes
monitor_path	the directory where the asnmon command was invoked.
runonce	N
trace_limit	10080 minutes

Related concepts:

- “Descriptions of the parameters that are used to operate the Replication Alert Monitor” on page 270
- “The Replication Alert Monitor” on page 255

Related tasks:

- “Operating the Replication Alert Monitor” on page 265

Descriptions of the parameters that are used to operate the Replication Alert Monitor

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

- "alert_prune_limit"
- "autoprune"
- "email_server"
- "max_notification_minutes"
- "max_notifications_per_alert" on page 271
- "monitor_errors" on page 271
- "monitor_interval" on page 271
- "monitor_limit" on page 271
- "monitor_path" on page 271
- "runonce" on page 271
- "trace_limit" on page 272

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 via 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_notifications_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 **max_notifications_per_alert** parameter tells the Monitor how many notifications to send during the period of time specified by the **max_notifications_minutes** 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.

Related concepts:

- "Default values of the parameters that are used to operate the Replication Alert Monitor" on page 269
- "The Replication Alert Monitor" on page 255

Related tasks:

- "Operating the Replication Alert Monitor" on page 265

Setting parameters for the Replication Alert Monitor

Setting parameters for the Replication Alert Monitor—Overview

You can determine the behavior of the Replication Alert Monitor by setting the values for various parameters. You can set parameters every time you create a monitor.

Procedure:

To set parameters for the Replication Alert Monitor:

1. Specify how often the Replication Alert Monitor runs. See "Specifying how often the Replication Alert Monitor runs" on page 273 for details.
2. Specify prune intervals for data from the Replication Alert Monitor. See "Specifying prune intervals for data from the Replication Alert Monitor" on page 273 for details.
3. Specify notification criteria for selected alert conditions. See "Specifying notification criteria for selected alert conditions" on page 274 for details.
4. Specify notification criteria for operational errors. See "Specifying notification criteria for operational errors" on page 274 for details.

"E-mail notifications for replication alert conditions" on page 261 describes e-mail notification, groups, and contacts in detail.

Related tasks:

- “Setting up the Replication Alert Monitor” on page 263

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.

Related concepts:

- “Default values of the parameters that are used to operate the Replication Alert Monitor” on page 269
- “Descriptions of the parameters that are used to operate the Replication Alert Monitor” on page 270
- “The Replication Alert Monitor” on page 255

Specifying prune intervals for data from the Replication Alert Monitor

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

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.

Related concepts:

- “Default values of the parameters that are used to operate the Replication Alert Monitor” on page 269
- “Descriptions of the parameters that are used to operate the Replication Alert Monitor” on page 270
- “The Replication Alert Monitor” on page 255

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 via electronic mail (e-mail).

Procedure:

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

1. 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.
2. 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.
3. 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.

Related concepts:

- "E-mail notifications for replication alert conditions" on page 261
- "Default values of the parameters that are used to operate the Replication Alert Monitor" on page 269
- "Descriptions of the parameters that are used to operate the Replication Alert Monitor" on page 270
- "Alert conditions for the Replication Alert Monitor" on page 258
- "The Replication Alert Monitor" on page 255

Related tasks:

- "Defining contact information for the Replication Alert Monitor" on page 264

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, use the following method:

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.

Related concepts:

- "Default values of the parameters that are used to operate the Replication Alert Monitor" on page 269
- "Descriptions of the parameters that are used to operate the Replication Alert Monitor" on page 270

Stopping a monitor

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.

Procedure:

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

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**. See the online help for details.

asnmcmd system command
: You can use the **asnmcmd stop** command to stop a monitor. See "asnmcmd: Working with a running Replication Alert Monitor" on page 333 for details.

Windows Service Control Manager
: When you stop a DB2 replication service, the monitor will stop automatically when the replication service stops.

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.

Related tasks:
- "Starting monitors" on page 268
- "Reinitializing monitors" on page 268

Chapter 21. Maintaining a Q replication and publishing environment

Maintaining a Q replication and event publishing environment—Overview

Q replication and event publishing work with your database system and require limited changes to your existing database activities. However, to ensure that your entire system continues to run smoothly and to avoid potential problems, you should determine the processing requirements of your replication environment and the potential impact of these requirements on your database system.

The following topics explain how to maintain the source systems, control tables, and target tables in your Q replication and event publishing environment:

- "Considerations for maintaining Q replication and event publishing source systems—Overview"
- "Maintaining control tables in Q replication and event publishing—Overview" on page 283
- "Maintaining target tables in Q replication and event publishing" on page 286
- "Considerations for rebinding packages and plans for Q replication and event publishing" on page 287

Considerations for maintaining Q replication and event publishing source systems

Considerations for maintaining Q replication and event publishing source systems—Overview

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.

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 following topics contain information you should consider when maintaining source systems for a Q replication and event publishing environment.

- "Maintaining source tables in a Q replication and event publishing environment" on page 278
- "Retaining log files for Q replication and event publishing —Overview" on page 278
- "Considerations for managing compression dictionaries in Q replication and event publishing environment (z/OS)" on page 282

Related concepts:

- "Maintaining a Q replication and event publishing environment—Overview" on page 277

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 log manager attempts to read compressed log records (z/OS).
- The Q Capture program captures LOB data.

Procedure:

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.
- Make sure that your z/OS™ utilities connect to the application server by using a share mode connection if your source tables are compressed. If your utilities and maintenance routines connect to the application server by using an exclusive mode connection, your source tables will be unavailable to Q replication and event publishing. Exclusive mode requires DB2® UDB to take your database and your z/OS compressed table space offline.

Related concepts:

- "Maintaining a Q replication and event publishing environment—Overview" on page 277

Related tasks:

- "Maintaining target tables in Q replication and event publishing" on page 286

Retaining log files for Q replication and event publishing

Retaining log files for Q replication and event publishing —Overview

If you need to know which log files are required by the Q Capture program, you must use a combination of data from the Q Capture program and data from DB2 UDB to determine the oldest log file that the Q Capture program needs. The following topics describe why you must retain log data and how to determine the oldest log file that the Q Capture program requires, based on platform:

- "Why you must retain log data for Q replication and event publishing" on page 279
- "Determining the oldest log file that the Q Capture program requires (Linux, UNIX, Windows)" on page 279
- "Determining the oldest log file that the Q Capture program requires (z/OS)" on page 281

Related concepts:

- "Maintaining source tables in a Q replication and event publishing environment" on page 278
- "Q Capture program" on page 15

Why you must retain log data for Q replication and event publishing

Your DB2® recovery logs:

- provide DB2® recovery capabilities
- provide information to your running Q Capture programs

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.

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 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.

Related concepts:

- "Retaining log files for Q replication and event publishing —Overview" on page 278

Related tasks:

- "Determining the oldest log file that the Q Capture program requires (Linux, UNIX, Windows)" on page 279
- "Determining the oldest log file that the Q Capture program requires (z/OS)" on page 281

Determining the oldest log file that the Q Capture program requires (Linux, UNIX, Windows)

The Q Capture program requests log records from DB2 UDB for Linux, UNIX, Windows and maintains information about the log records that it receives. However, DB2 UDB manages the log files that contain those log records and does not provide log file information to the Q Capture program. Therefore, you must read a Q Capture control table to determine the log sequence number for the oldest log record that the Q Capture program needs. 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 record sequence number for the most recent transaction that the Q Capture program has seen, processed, and recorded a history for in its control tables:

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

   The RESTART_SEQ value is a CHAR(10) FOR BIT DATA column, which looks like 20 hexadecimal characters. For example:

   00000000123456123456

   Make note of the *last* 12 characters of the RESTART_SEQ value. This number is a log sequence number. The Q Capture program does not need log records that are older than this one. In the above example, the log sequence number is:

   123456123456
2. From a DB2® UDB command line, obtain the path for the active log files from the database configuration. For example:

   ```
   db2 get db cfg for yourdbname
   ```

   where *yourdbname* is the database name. From the output displayed on the screen, note the path for the active log files. For example:

   ```
   Path to log files   =C:\DB2\NODE0000\SQL00001\SQLLOGDIR\
   ```
3. Change the current working directory to be one above the path to the active log files. For example:

   ```
   C:\>cd c:\DB2\NODE0000\SQL00001
   ```
4. From a DB2 UDB command line, use the **db2flsn** command and the log sequence number from the first step to get the name of the oldest log file that the Q Capture program requires. For example:

   ```
   C:\DB2\NODE0000\SQL00001\>db2flsn 123456123456
   ```

   To run the **db2flsn** command, you must have read access to the `SQLLOGCTL.LFH` file, which is located one directory above (`C:\DB2\NODE0000\SQL00001\`) the path to active log files.

   The system retrieves and displays the name of the file that contains the log record identified by the log sequence number. For example:

   ```
   Given LSN is contained in the log file S000123.LOG
   ```
5. Note the age of this retrieved log file.

   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.

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 towards the end of the log minimizes the number of older log files that the Q Capture program needs.

Related concepts:

- "Retaining log files for Q replication and event publishing —Overview" on page 278

Related tasks:

- "Determining the oldest log file that the Q Capture program requires (z/OS)" on page 281

Determining the oldest log file that the Q Capture program requires (z/OS)

The Q Capture program requests log records from DB2 UDB for z/OS and maintains information about the log records that it receives. However, DB2 UDB manages the log files that contain those log records and does not provide log file information to the Q Capture program. Therefore, you must read a Q Capture control table to determine the log sequence number for the oldest log record that the Q Capture program needs. Use DB2 UDB 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 a history for in its control tables:

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

   This is an example of a log sequence number:

   0000555551F031230000

   Ignore the first four characters of the log sequence number, which are always 0000. The next 12 characters correspond to the active log sequence number. (This 12–character value is the relative byte address (RBA) in non-data sharing environments and is the log record sequence number (LRSN) in data sharing environments.) The last four characters are 0000 in non-data sharing environments; these last four characters correspond to the member ID in data sharing environments.
2. Use the DSNJU004 utility to invoke the Print Log Map utility. This utility displays information about the bootstrap data sets (BSDS).

   For example:

   ```
   #  ACTIVE LOG COPY 1 DATA SETS
   #   START RBA/TIME    END RBA/TIME         DATE   LTIME   DATA SET INFORMATION
   #------------------  --------------      -------- ------ -------------------------
   # 555551F03000         555551F05FFF      1998.321 12:48  DSN=DSNC710.LOGCOPY1.DS02
   #2001.57 15:46:32.2  2001.057 15:47:03.9 PASSWORD=(NULL)STATUS=TRUNCATED,REUSABLE
   # 555551F06000         555551F09FFF      1998.321 12:49  DSN=DSNC710.LOGCOPY1.DS03
   #2001.57 15:47:32.2  2001.057 15:48:12.9 PASSWORD=(NULL)STATUS=TRUNCATED,REUSABLE
   ```

3. Compare your 12–character active log number of the RESTART_SEQ value to the Start RBA and corresponding End RBA range in each displayed row.
4. Find the row for which the 12–character active log number from the IBMQREP_CAPMON table falls within the start RBA and end RBA. In the example:

```
# 555551F03000         555551F05FFF      1998.321 12:48  DSN=DSNC710.LOGCOPY1.DS02
#2001.57 15:46:32.2  2001.057 15:47:03.9 PASSWORD=(NULL)STATUS=TRUNCATED,REUSABLE
```

5. Note the corresponding Data Set Information for that active log number. In the example:

   ```
   DSNC710.LOGCOPY1.DS02
   ```

6. Note the date and time of this data set.

   The Q Capture program needs this data set and more recent data sets to perform a restart at any particular time.

Related concepts:

- "Retaining log files for Q replication and event publishing —Overview" on page 278

Related tasks:

- "Determining the oldest log file that the Q Capture program requires (Linux, UNIX, Windows)" on page 279

Considerations for managing compression dictionaries in Q replication and event publishing environment (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 stored in a compressed table space. DB2® uses the current compression dictionary for decompression. If the compression dictionary is temporarily unavailable, DB2® returns an error to the Q Capture program. The Q Capture program makes several attempts to continue processing. But if the dictionary remains unavailable, the Q Capture program issues an ASN0011E message, deactivates the Q subscription, and terminates. To avoid these situations, let the Q Capture program process all log records for a table before performing any activity that affects the compression dictionary for that table. These activities include:

- 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

  If you prefer to generate a new compression dictionary, synchronize the REORG utility with your current running applications and with the Q Capture program as follows:

  1. Quiesce all application programs that update the table.
  2. Let the Q Capture program capture all logged updates for the table.
  3. Use the REORG utility on the compressed table to create a new compression dictionary.
  4. Restart your application programs.

Recommendation: Use the KEEPDICTIONARY=YES option to retain the current version of the compression dictionary during a reorganization. The KEEPDICTIONARY=YES option ensures that your dictionary remains compatible with your pre-existing log records.

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.

Related concepts:

- "Considerations for maintaining Q replication and event publishing source systems—Overview" on page 277
- "Q Capture program" on page 15

Maintaining control tables in Q replication and event publishing

Maintaining control tables in Q replication and event publishing—Overview

Q replication uses control tables to store source definitions, Q subscription 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.

The following topics discuss maintenance activities for you to perform on your control tables:

- "Pruning control tables in Q replication and event publishing" on page 284
- "Considerations for using the RUNSTATS utility on control tables for Q replication and event publishing" on page 285
- "Reorganizing Q replication and event publishing control tables" on page 285

You should also consider the situation where the replication programs cannot connect to their DB2 server. See "When replication programs cannot connect to their DB2 UDB server" on page 286 for details.

Related concepts:

- "Maintaining a Q replication and event publishing environment—Overview" on page 277
- "Control tables for Q replication and event publishing—Overview" on page 361

Pruning control tables in Q replication and event publishing

The size of most control tables in Q replication and event publishing does not change often. However, some control tables grow regularly. Table 23 lists control tables that the Q Capture program prunes, which can grow regularly. Table 24 lists control tables that the Q Apply program prunes, which can grow regularly. 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. Monitor the growth of and consider the various pruning methods available for the following control tables.

Procedure:

To manually prune a control table, use the **asnqcmd** command with the prune option.

To control how a table is automatically pruned, modify the value of the parameter shown in Table 23 or Table 24.

The Q Capture program prunes the following tables by using the listed parameters to determine which rows from these tables are eligible for pruning. The **prune_interval** parameter specifies how often the Q Capture program checks for rows that are eligible for pruning.

Table 23. Control tables that the Q Capture program prunes

Control table	Parameter that specifies which rows are eligible for pruning
IBMQREP_CAPMON	monitor_limit
IBMQREP_CAPQMON	monitor_limit
IBMQREP_CAPTRACE	trace_limit
IBMQREP_SIGNAL	signal_limit

The Q Apply program prunes the following tables by using the listed parameters to determine which rows from these tables are eligible for pruning. The **prune_interval** parameter specifies how often the Q Apply program checks for rows that are eligible for pruning.

Table 24. Control tables that the Q Apply program prunes

Control table	Parameter that specifies which rows are eligible for pruning
IBMQREP_APPLYMON	monitor_limit
IBMQREP_APPLYTRACE	trace_limit

To control pruning of the IBMSNAP_ALERTS table, modify the value of the **alert_prune_limit** parameter.

Related concepts:

- “Maintaining control tables in Q replication and event publishing—Overview” on page 283

Considerations for using the RUNSTATS utility on control tables for Q replication and event publishing

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.

Related concepts:

- "Maintaining control tables in Q replication and event publishing—Overview" on page 283
- "Control tables for Q replication and event publishing—Overview" on page 361

Reorganizing Q replication and event publishing control tables

Regularly reorganize any control tables that frequently change size to eliminate fragmented data and reclaim space.

Procedure:

To reorganize control tables:

REORG command (UNIX, Windows)

REORG utility with the PREFORMAT option (z/OS)
The PREFORMAT option of this utility speeds up the insert processing of the Q Capture program.

Recommendation: Reorganize the following control tables once a week:

- IBMQREP_APPLYTRACE
- IBMQREP_CAPMON
- IBMQREP_CAPQMON
- IBMQREP_CAPTRACE
- IBMSNAP_MONTRAIL
- IBMSNAP_MONTRACE
- IBMQREP_SIGNAL

The IBMSNAP_ALERTS and IBMQREP_EXCEPTIONS tables might also become large and change size frequently depending on your replication environment.

Related concepts:

- "Maintaining control tables in Q replication and event publishing—Overview" on page 283

Related tasks:

- "Pruning control tables in Q replication and event publishing" on page 284

When replication programs cannot connect to their DB2 UDB server

To run correctly, the Q Capture program, Q Apply program, and Replication Alert Monitor must be able to connect to the DB2 UDB server that contains its control tables. Each of these programs must also be able to read from and write to those 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 Capture program shuts down because the DB2 UDB server that contains its control tables has been shut down or quiesced, simply restart the Q Capture program when the DB2 UDB server is running.

If the program can connect to the DB2 UDB 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.

Related concepts:
- "Connectivity requirements for Q replication and event publishing" on page 67
- "Error options for Q replication" on page 103

Maintaining target tables in Q replication and event publishing

Maintaining target tables is similar to maintaining other tables. However, the Q Apply program does effect how you maintain target tables. The performance of update and delete statements by the Q Apply program is heavily influenced by the currency of the statistics that the optimizer uses.

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.
- Stop the Q Apply programs before taking a target table offline to run any utility.

Related concepts:
- "Maintaining a Q replication and event publishing environment—Overview" on page 277
- "Maintaining source tables in a Q replication and event publishing environment" on page 278

Considerations for rebinding packages and plans for Q replication and event publishing

Many of the packages and plans for Q replication and event publishing are bound using isolation uncommitted reads (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 cursor stability, they will interfere with the Q Capture program and the Q Apply program. Packages for Q replication and event publishing must remain bound with isolation UR to maintain optimal system performance.

For 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.

For z/OS
The Q Capture program, Q Apply program, and Common packages are bound automatically. You can use the z/OS sample ASNQBNDL to bind ASNCOMMON, ASNQCAPTURE, ASNQAPPLY, and ASNMON packages at a DB2 subsystem.

Related concepts:

- "Optional: Binding the program packages (Linux, UNIX, Windows)—Overview" on page 72

Related tasks:

- "Optional: Binding the Q Capture program packages (Linux, UNIX, Windows)" on page 73
- "Optional: Binding the Q Apply program packages (Linux, UNIX, Windows)" on page 73
- "Optional: Binding the Replication Alert Monitor packages (Linux, UNIX, Windows)" on page 74

Chapter 22. Detecting and repairing differences between source and target tables

Detecting and repairing differences between source and target tables

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 tdiff and trepair 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.

Both 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:

1. Run the tdiff utility. See "Tdiff: Table difference utility" for details.

   The utility generates a list of differences between a source and target table.
2. Run the trepair utility. See "Trepair: Table repair utility" on page 292 for details.

   The utility repairs the target table by deleting rows that have no match in the source table and inserting rows that are in the source table but have no match in the target table.

Tdiff: Table difference utility

The tdiff 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® UDB table.

To use the tdiff utility, you run the **asntdiff** command and specify the name of a Q subscription (Q replication) or subscription set member (SQL replication) that contains the source and target tables that you want to compare.

You can run the **asntdiff** command on Linux, UNIX®, Windows®, and z/OS™ operating systems. The command compares tables on Linux, UNIX, Windows, z/OS, or iSeries™ operating systems. The **asntdiff** command can be used with federated sources and targets.

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:

```
where="where subname = 'my_qsub'"
```

SQL replication

The WHERE clause identifies a row in the IBMSNAP_SUBS_MEMBR table at the Apply control server, based on the value of the SET_NAME column. For example:

```
where="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. The difference table is named *schema*.ASNTDIFF, where *schema* is the schema of the Q Capture control tables or Apply control tables that contain information about the source and target tables that you are comparing.

The difference table has two or more columns. The value in one column (named DIFF , with a blank space at the end) 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 source and target tables.

A value of + 2 in the DIFF column indicates that an unmatched row exists in the target table for one of the following reasons:

- The source has no matching key.
- The source has a matching key but unequal data.
- The row has more duplicates at the target than at the source.

A value of - 2 in the DIFF column indicates that an unmatched row exists in the source table for one of the following reasons:

- The target has no matching key.
- The target has a matching key but unequal data.
- The row has more duplicates at the source than at the target.

A value of ? 1 indicates that there is an invalid character in one or more source key columns.

A value of ? 2 indicates that there is an invalid character in one or more target key 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
+ 2   000010
+ 2   000020
- 2   000999
- 2   000010
- 2   000020
- 2   000030
- 2   000050
- 2   000070
- 2   000090
- 2   000100
```

The two dark-colored rows both show the `EMPNO` value of **`000020`**, but the `DIFF` values of **`+ 2`** and **`- 2`** are different. This indicates that the row exists at both the source and target tables, but non-key data is different between the two tables.

The values `? 1` and `? 2` are not shown in the example.

Note: In Q replication, if the source and target tables that you are comparing are part of a Q subscription with the option to suppress deletes from the source table, the extra rows that are not deleted from the target table will appear as differences when you run the **asntdiff** command.

When to use the tdiff utility

The best time to use the tdiff 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.

Related tasks:

- "Detecting and repairing differences between source and target tables" on page 289

Related reference:

- "ASN.IBMSNAP_SUBS_MEMBR" in the *IBM DB2 Information Integrator SQL Replication Guide and Reference*
- "asntdiff: Comparing data in source and target tables" on page 355
- "asntrep: Repairing differences between source and target tables" on page 357
- "IBMQREP_SUBS table" on page 387

Trepair: Table repair utility

The trepair utility repairs differences between source and target tables on all DB2 servers by deleting unmatched rows from the target and then inserting rows that are missing from the target. Trepair runs on Linux, UNIX, or Windows operating systems.

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. If you use trepair 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 trepair 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. For details, see "Tdiff: Table difference utility" on page 289.

Recommendation: Consider target table attributes before using the trepair utility. During the repair process, referential integrity (RI) constraints on the target table are not dropped. This means that an attempt to insert or delete a row from a target table can fail if the insert or delete violates an RI constraint. Also, a duplicate source row might be impossible to repair at the target due to a unique index there.

Related tasks:
- "Detecting and repairing differences between source and target tables" on page 289

Related reference:
- "ASN.IBMSNAP_SUBS_MEMBR" in the *IBM DB2 Information Integrator SQL Replication Guide and Reference*
- "asntdiff: Comparing data in source and target tables" on page 355
- "asntrep: Repairing differences between source and target tables" on page 357
- "IBMQREP_SUBS table" on page 387

Chapter 23. Using system services to operate the replication programs

Using system services to operate the Q replication and event publishing programs—Overview

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.

The following topics discuss system services and how to use them to operate the replication programs:

- "Operating the Q replication and event publishing programs by using system services (z/OS)—Overview"
- "Managing replication services with the Windows Service Control Manager (SCM) —Overview" on page 299
- "Scheduling the replication programs—Overview" on page 303

Operating the replication programs by using system services (z/OS)

Operating the Q replication and event publishing programs by using system services (z/OS)—Overview

You can start the replication programs for Q replication and event publishing in the z/OS™ operating system by using the job control language (JCL) or system started tasks. You can use the Automatic Restart Manager (ARM) to restart failed replication programs.

The following topics explain these tasks and tools:

- "Running the Q replication and event publishing programs by using JCL—Overview" on page 294
- "Running the Q replication and event publishing programs with system-started tasks" on page 296
- "The Automatic Restart Manager (ARM) recovery system" on page 297
- "Setting up the Automatic Restart Manager (ARM) to restart Q replication and event publishing programs" on page 298

Related concepts:

- "Using system services to operate the Q replication and event publishing programs—Overview" on page 293

Running the replication programs by using JCL

Running the Q replication and event publishing programs by using JCL—Overview

The z/OS operating system can use the job control language (JCL) to operate the replication programs.

The following topics describe how to use JCL to operate the replication programs:

- "Starting the Q Capture program by using JCL"
- "Starting the Q Apply program by using JCL" on page 295
- "Starting the Replication Alert Monitor by using JCL" on page 295
- "Running the Q replication and event publishing programs in batch mode by using JCL" on page 296
- "Modifying Q replication and event publishing programs that are already started by using JCL" on page 296

Related concepts:

- "Operating the Q replication and event publishing programs by using system services (z/OS)—Overview" on page 293

Related tasks:

- "Running the Q replication and event publishing programs with system-started tasks" on page 296

Starting the Q Capture program by using JCL

When you run the Q Capture program by using the JCL, the Q Capture program runs as a batch job. For example, a Q Capture job is the batch job that the operating system uses to run the Q Capture program. In other words, the JCL runs the Q Capture program on the z/OS operating system.

Procedure:

To start a Q Capture program by using the JCL:

1. Prepare the JCL for z/OS by specifying the appropriate optional invocation parameters in the PARM field of the Q Capture job.
   - You must set the TZ environment variables and the language environment variables in the JCL if either of the following conditions are true:
     - You did not set the TZ environment variable in the system-wide /etc/profile file.
     - You did not set the TZ environment variable in the profile file in the home directory of the running replication program.

     For more information about setting the TZ variable, see the *z/OS UNIX System Services User's Guide*.

     The following example of this line in 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'
     ```
   - Specify the LANG and NLSPATH environment variables if the DD statement MSGS is not specified.

- Specify the TMPDIR environment variable if you do not want the replication programs to write temporary files to a directory other than the /tmp directory. The TMPDIR environment variable specifies the directory pathname where the replication programs write temporary files.

2. Submit the JCL from a z/OS console or TSO.

Related tasks:
- "Starting the Q Apply program by using JCL" on page 295
- "Starting the Replication Alert Monitor by using JCL" on page 295

Starting the Q Apply program by using JCL

When you run the Q Apply program by using JCL, the Q apply program runs as a batch job. For example, a Q Apply job is the batch job that the operating system uses to run the Q Apply program. In other words, the JCL runs the Q Apply program on the z/OS operating system.

Procedure:

To start a Q Apply program by using JCL, use the following procedure:

Prepare the JCL for z/OS by specifying the appropriate invocation parameters in the PARM field of the Q Apply job. Customize the JCL to meet your site's requirements.

Here is an example of the invocation in JCL for the Q Apply program on z/OS operating systems:

```
//PLS EXEC PGM=ASNQAPP, // PARM='APPLY_SERVER=DQRG APPLY_SCHEMA=JAY'
```

APPLY_SERVER is the DB2 subsystem where the Q Apply target tables are located.

Related tasks:
- "Starting the Q Capture program by using JCL" on page 294
- "Starting the Replication Alert Monitor by using JCL" on page 295

Starting the Replication Alert Monitor by using JCL

When you run the Replication Alert Monitor by using JCL, the Replication Alert Monitor runs as batch job. A Monitor job is the batch job that the operating system uses to run the Replication Alert Monitor. The JCL runs the Replication Alert Monitor on the z/OS operating system.

Procedure:

Prepare the JCL for z/OS by specifying the appropriate invocation parameters in the PARM field of the Replication Alert Monitor job. Customize the JCL to meet your site's requirements. A sample of invocation JCL in library SASNSAMP(ASNMON#) is included with the Replication Alert Monitor for z/OS.

Here is an example of this line in the invocation JCL:

```
//monasn EXEC PGM=ASNMON,PARM='monitor_server=DSN
                         monitor_qual=monqual'
```

Related tasks:
- "Starting the Q Capture program by using JCL" on page 294

- "Starting the Q Apply program by using JCL" on page 295

Running the Q replication and event publishing programs in batch mode by using JCL

Procedure:

You can run replication programs in batch mode by using the JCL

To run replication programs in batch mode:

1. Customize the JCL in library SASNSAMP for the appropriate program. Table 25 shows which sample job to use to start each program:

Table 25.

Sample	Program
ASNQSTRA	Q Apply
ASNQSTRC	Q Capture
ASNSTRM	Replication Alert Monitor
ASNQTON	Trace (for the Q Capture program or the Q Apply program)

2. Prepare the JCL for z/OS by specifying the appropriate optional invocation parameters in the PARM field of the DPROPR jobs (Q Capture, Q Apply, Replication Alert Monitor and Asntrc). Submit the JCL from the z/OS console or TSO.

Modifying Q replication and event publishing programs that are already started by using JCL

You can modify programs that have already started by using the JCL.

Procedure:

To modify programs, use the MODIFY command. For example, use the MODIFY command to stop a Q Apply program that is already running. You must use the MODIFY command from z/OS console. The following syntax example shows how you can use the abbreviation F.

```
►►—F——jobname——,—┤ Parameters ├————————————————————————————————►◄
```

jobname is the job name specified when the program was started. For example, to stop the Capture program you would use the following command:

```
F capjfa,stop
```

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 following replication programs:

- The Q Capture program
- The Q Apply program
- The Replication Alert Monitor

Procedure:

To start a program as a system-started task for the z/OS operating system:

1. Create a procedure (*procname*) in your PROCLIB.
2. Create an entry in the RACF STARTED class for *procname*. This entry associates *procname* with the RACF user ID to be used to start the Q Capture program. Make sure that the necessary DB2 authorization is granted to this user ID before you start the program.
3. From the z/OS console, run the command **start** *procname*.

The following sample procedure is for the Q Capture program:

```
// PARM='CAPTURE_SERVER=DSN7 capture_schema=ASN startmode=cold'
//STEPLIB DD DSN=qrhlqual.SASNLOAD,DISP=SHR
// DD DSN=dsnhlqual.SDSNLOAD,DISP=SHR
//* DD DSN=mqhlqual.SCSQANLE,DISP=SHR
//* DD DSN=mqhlqual.SCSQLOAD,DISP=SHR
//* DD DSN=xmlhlqual.SIXMMOD1,DISP=SHR
//CAPSPILL DD DSN=&&CAPSPILL,DISP=(NEW,DELETE,DELETE),
// UNIT=VIO,SPACE=(CYL,(50,70)),
// DCB=(RECFM=VB,BLKSIZE=6404)
//MSGS DD PATH='/usr/lpp/db2repl_08_02/msg/En_US/db2asn.cat'
//CEEDUMP DD SYSOUT=* //SYSPRINT DD SYSOUT=* //SYSUDUMP DD DUMMY
```

where: ’qrhlqual’ is the Q Replication target library high-level qualifier, ’dsnhlqual’ is the DB2 target library high-level qualifier, ’mqhlqual’ is the MQSeries target library high-level qualifier, and ’xmlhlqual’ is the XML Toolkit library high-level qualifier. JCL that executes the Q Capture program must STEPLIB the MQSeries and XML Toolkit libraries if they are not installed in the LNKLST.

Related concepts:

- “Operating the Q replication and event publishing programs by using system services (z/OS)—Overview” on page 293

Related tasks:

- “Running the Q replication and event publishing programs by using JCL—Overview” on page 294

The Automatic Restart Manager (ARM) recovery system

You can use the ARM to restart the following Q replication and event publishing programs:

- The Q Capture program.
- The Q Apply program.
- The Replication Alert Monitor.

ARM is a recovery system for the MVS™ operating system that can improve the availability of batch jobs or started tasks. 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 MVS 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 26 shows the element names to use for each of the replication programs when you configure ARM.

Table 26. Element names for the replication programs

Replication program	Element name
Q Capture	ASNQC*xxxxyyyy*
Q Apply	ASNQA*xxxxyyyy*
Replication Alert Monitor	ASNAM*xxxxyyyy*

In Table 26, *xxxx* represents the DB2® subsystem name, and *yyyy* represents the data-sharing member name. ARM needs the data-sharing member name only for the data-sharing configurations. 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 due to a STOP command.

Related concepts:

- "Operating the Q replication and event publishing programs by using system services (z/OS)—Overview" on page 293

Related tasks:

- "Setting up the Automatic Restart Manager (ARM) to restart Q replication and event publishing programs" on page 298

Setting up the Automatic Restart Manager (ARM) to restart Q replication and event publishing programs

You can set up the Automatic Restart Manager (ARM) to restart replication programs such as the Q Apply program, Q Capture program, and the Replication Alert Monitor.

Procedure:

To set up the ARM:

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.

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

Related concepts:

- "Operating the Q replication and event publishing programs by using system services (z/OS)—Overview" on page 293
- "The Automatic Restart Manager (ARM) recovery system" on page 297

Operating the replication programs by managing replication services (Windows)

Managing replication services with the Windows Service Control Manager (SCM) —Overview

You can create, run, view a list of, or drop services for each Q Capture server, Q Apply control server, and Monitor control server by using the Windows Service Control Manager. DB2 UDB groups the services with other DB2 services.

First, you create the replication service, then you run the replication service.

You can also view a list of the current replication services, and drop replication services. If you want to change the parameters for a program after the service starts, you must drop the service and create a new one.

The following topics explain replication services and the tasks that you can accomplish with them:

- "Replication services for Q replication and event publishing (Windows)"
- "Creating a replication service" on page 300
- "Starting a replication service" on page 301
- "Stopping a replication service" on page 302
- "Viewing a list of replication services" on page 302
- "Dropping a replication service" on page 303

Related concepts:

- "Using system services to operate the Q replication and event publishing programs—Overview" on page 293

Replication services for Q replication and event publishing (Windows)

A replication service is a program that starts a replication program on the Windows® operating system. You can use replication services to start the following replication programs:

- The Q Capture program.
- The Q Apply program.
- The Replication Alert Monitor.

When you create a replication service, DB2® UDB adds it to the Windows Service Control Manager (SCM) in Automatic mode, and the SCM starts the service. 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 27 describes the inputs for the replication service name.

Table 27. Inputs for the replication service name

Input	Description
instance	The name of the DB2 instance.
alias	The database alias of the Q Capture server, Q Apply server, or Monitor control server.
program	One of the following values: QCAP (for the Q Capture program), QAPP (for the Q Apply program), or MON (for the Replication Alert Monitor).
qualifier_or_schema	One of the following identifiers: Q Apply schema, Q Capture schema, or Monitor qualifier.

Example: The following service name is for a Q Capture program that has the schema ASN and is working with database DB1 under the instance called INST1:

```
DB2.INST1.DB1.QCAP.ASN
```

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 he following display name is for a Q Capture program that has the schema ASN and is working with database DB1 under the instance called INST1:

```
DB2 - INST1 DB1 QCAPTURE ASN
```

If you want to add a description for the service, use the SCM after you create a replication service. You can also use the SCM to specify a user name and a password for a service.

Related tasks:

- "Managing replication services with the Windows Service Control Manager (SCM) —Overview" on page 299
- "Creating a replication service" on page 300
- "Viewing a list of replication services" on page 302
- "Dropping a replication service" on page 303

Creating a replication service

You can create a DB2 replication service to start a Q Capture program, Q Apply program, and the Replication Alert Monitor program on Windows operating systems.

You can add more than one replication service to your system. You can add one service for each schema on every Q Capture server, and for each qualifier on every

Q Apply server and Monitor control server, respectively. For example, if you have five databases and each database is a Q Apply server, a Q Capture server, and a Monitor control server, you can create fifteen replication services. If you have multiple schemas or qualifiers on each server, you could create more services.

Prerequisites:

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.

Procedure:

To create a replication service:

Use the **asnscrt** command. See "asnscrt: Creating a DB2 replication service to start the replication programs" on page 339 for command syntax and parameter descriptions.

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 diagnostic log files for the program that you were trying to start. By default, the diagnostic 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.

Related concepts:
- "Replication services for Q replication and event publishing (Windows)" on page 299

Related tasks:
- "Viewing a list of replication services" on page 302
- "Dropping a replication service" on page 303

Starting a replication service

After you create a replication service, you can stop it and start it again.

Important: After a replication program starts from a service, starting another replication program with the same schema or qualifier causes an error.

Procedure:

To start a replication service, use one of the following methods:
- The Windows Service Control Manager (SCM)
- The **net start** command

Related concepts:

- “Replication services for Q replication and event publishing (Windows)” on page 299

Related tasks:
- “Stopping a replication service” on page 302

Stopping a replication service

After you create a replication service, you can stop it and start it again.

Important: 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**, or **asnmcmd**), the service that started the program continues to run. You must stop it explicitly.

Procedure:

Use one of the following methods to stop a service:
- The Windows Service Control Manager (SCM)
- The **net stop** command

Related concepts:
- “Replication services for Q replication and event publishing (Windows)” on page 299

Related tasks:
- “Starting a replication service” on page 301

Viewing a list of replication services

You can view a list of all your replication services and their properties.

Procedure:

To view a list of replication services, use the **asnlist** command.

To view a list of replication services and descriptions of each service, use the **asnlist** command with the *details* parameter.

Related concepts:
- “Replication services for Q replication and event publishing (Windows)” on page 299

Related tasks:
- “Creating a replication service” on page 300
- “Dropping a replication service” on page 303

Related reference:
- “asnslist: Listing DB2 replication services” on page 342

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). If you want to change the startup parameters for a program that is started by a service, drop the service. Then you can create a new service by using new startup parameters.

Procedure:

To drop a DB2 replication service, use the **asnsdrop** command.

Related concepts:

- "Replication services for Q replication and event publishing (Windows)" on page 299

Related tasks:

- "Creating a replication service" on page 300
- "Viewing a list of replication services" on page 302

Related reference:

- "asnsdrop: Dropping DB2 replication services" on page 343

Scheduling replication programs

Scheduling the replication programs—Overview

You can schedule the Q Capture program, the Q Apply program or the Replication Alert Monitor program to start at prescribed times. The following topics discuss scheduling programs on various operating systems:

- "Scheduling the replication programs (Linux, UNIX)"
- "Scheduling the replication programs (Windows)" on page 304
- "Scheduling the replication programs (z/OS)" on page 305

Related concepts:

- "Using system services to operate the Q replication and event publishing programs—Overview" on page 293

Scheduling the replication programs (Linux, UNIX)

You can schedule when to start the replication programs.

Procedure:

To start a replication program at a specific time on a Linux or UNIX operating system, use the **at** command.

Table 28 on page 304 shows commands that are used to start the replication programs at 3:00 p.m. on Friday:

Table 28. Scheduling commands for the replication programs (Linux, UNIX).

Replication program	Linux or UNIX command
Q Capture	`at 3pm Friday asnqcap autoprune=n`
Q Apply	`at 3pm Friday asnqapply applyqual=myqual`

Replication Alert Monitor	`at 3pm Friday asnmon monitor_server=db2srv1`
	`  monitor_qualifier=mymon`

Related concepts:
- "Scheduling the replication programs—Overview" on page 303

Related tasks:
- "Scheduling the replication programs (Windows)" on page 304
- "Scheduling the replication programs (z/OS)" on page 305

Scheduling the replication programs (Windows)

You can schedule when to start the replication programs.

Prerequisites:
1. Start the Windows Schedule Service before you use the AT command to schedule the replication programs.
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. Send the output to a file to check for errors. Note the "^" character when redirecting to a file.

Procedure:

To start a replication program at a specific time on a Windows operating system, use one of the following methods:
- Use the Windows Service Control Manager
- Use the AT command to start a program at a specific time.

Table 29 shows commands that are used to start the replication programs at 3:00 p.m. on Friday:

Table 29. Scheduling commands for the replication programs (Windows).

Replication program	Windows command

Q Capture	`c:\>at 15:00 db2cmd asnqcap capture_server=qcapdb`
	`        capture_schema=schema`
	` capture_path=c:\capture ^> c:\capture\asnqcap.out`

Q Apply	`c:\>at 15:00 db2cmd asnqapp apply_server=qappdb`
	`        apply_schema=applyqual`
	` apply_path=c:\apply ^> c:\apply\asnqapp.out`

Table 29. Scheduling commands for the replication programs (Windows). (continued)

Replication program	Windows command
Replication Alert Monitor	`c:\>at 15:00 db2cmd asnmon monitor_server=mondb monitor_qual=monqual monitor_path=c:\monitor ^> c:\monitor\asnmon.out`

Related concepts:

- "Scheduling the replication programs—Overview" on page 303

Related tasks:

- "Scheduling the replication programs (Linux, UNIX)" on page 303
- "Scheduling the replication programs (z/OS)" on page 305

Scheduling the replication programs (z/OS)

You can schedule when to start the replication programs on the z/OS operating system using two different commands.

Procedure:

To schedule programs on the z/OS operating system, use the following methods:

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.

Related concepts:

- "Scheduling the replication programs—Overview" on page 303

Related tasks:

- "Scheduling the replication programs (Linux, UNIX)" on page 303
- "Scheduling the replication programs (Windows)" on page 304

Part 5. Reference information for Q replication and event publishing

This part of the book contains the following chapters:

- Chapter 24, “Naming rules and guidelines,” on page 309 describes the valid names for Q replication and event publishing objects.
- Chapter 25, “System commands for Q replication and event publishing,” on page 313 describes the commands that experienced users can use instead of the Replication Center to start, operate, modify, and monitor the programs for Q replication and event publishing.
- Chapter 26, “Control tables for Q replication and event publishing,” on page 361 describes the structures of the tables that store control information for the Q Capture program, Q Apply program, and Replication Alert Monitor.
- Chapter 27, “Structure of XML messages for event publishing,” on page 431 describes the format of the XML messages that the Q Capture program sends to an application in event publishing.

Chapter 24. Naming rules and guidelines

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. The following topics explain the naming rules and guidelines:

- "Naming rules for Q replication and event publishing objects"
- "How lowercase object names are handled for Q replication and event publishing" on page 312

Naming rules for Q replication and event publishing objects

The following table lists the limits for names of objects in Q replication and event publishing.

Table 30. Name limits for objects in Q replication and event publishing

Object	Name limits	Length limit
Source and target tables	The names of source tables and target tables must follow the naming rules for DB2 UDB table names.	
Source and target columns	The names of the source and target columns must follow the naming rules for DB2 UDB column names.	**Important for z/OS**: Both short and long schema names are supported for table columns for DB2 UDB for z/OS. The names of table columns can include up to: • 18 bytes for subsystems that are running DB2 UDB for z/OS Version 8 compatibility mode or earlier • 128 bytes for subsystems that are running DB2 UDB for z/OS Version 8 new-function mode
Table owner (z/OS)	Both short and long schema names are supported for table owner for DB2 UDB for z/OS.	The names of the table owner can include up to: • 30 bytes for subsystems that are running DB2 UDB for z/OS Version 8 compatibility mode or earlier • 128 bytes for subsystems that are running DB2 UDB for z/OS Version 8 new-function mode

Table 30. Name limits for objects in Q replication and event publishing (continued)

Object	Name limits	Length limit
Send queue	The name of the send queue can include any characters that DB2 UDB 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 UDB and WebSphere MQ allow for VARCHAR data types. Receive queue names cannot contain spaces.	48 or fewer characters
Restart queue	The name of the restart queue can include any characters that DB2 UDB and WebSphere MQ allow for VARCHAR data types. Restart queue names cannot contain spaces.	48 or fewer characters
Q subscription	The name of a Q subscription can include any characters that DB2 UDB allows for VARCHAR data type columns. For Q Capture program, all Q subscription names must be unique. Because the name of the Q subscription is stored at both the source and target server, be sure that the name is compatible with the code pages for both the source and target servers. Q subscription names cannot contain spaces or semicolons (;).	30 or fewer characters
Q subscription group	The name of the Q subscription group can include any characters that DB2 UDB allows for VARCHAR data type columns. **Recommendation**: Use a unique group name for each logical table.	30 or fewer characters
XML publication	The name of an XML publication can include any characters that DB2 UDB allows for VARCHAR data type columns. For each Q Capture program, all XML publication names must be unique. Be sure that the name of the XML publication is compatible with the code page for the subscribing application. XML publication names cannot contain spaces or semicolons (;).	30 or fewer characters

Table 30. Name limits for objects in Q replication and event publishing *(continued)*

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

Related concepts:
- "The Replication Alert Monitor" on page 255
- "Q subscriptions" on page 5
- "Schemas for the Q Apply and Q Capture programs" on page 20
- "XML publications" on page 11

Related reference:
- "How lowercase object names are handled for Q replication and event publishing" on page 312

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 for Windows: 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.

Related concepts:

- "The Replication Alert Monitor" on page 255
- "Q subscriptions" on page 5
- "Schemas for the Q Apply and Q Capture programs" on page 20
- "XML publications" on page 11

Related reference:

- "Naming rules for Q replication and event publishing objects" on page 309

Chapter 25. System commands for Q replication and event publishing

System commands for Q replication and event publishing—Overview

This section describes commands for Linux, UNIX®, Windows®, and UNIX System Services (USS) on z/OS™ that let you start, operate, modify, and monitor Q replication and event publishing programs. Some of the commands (for the Replication Alert Monitor, Windows services, trace facility, and password files) are shared with SQL replication.

The following topics explain the system commands and their syntax:

- "Road map: Q replication and event publishing system commands"
- "asnqcap: Starting a Q Capture program" on page 315
- "asnqccmd: Working with a running Q Capture program" on page 320
- "asnqapp: Starting a Q Apply program" on page 323
- "asnqacmd: Working with a running Q Apply program" on page 326
- "asnmon: Starting a Replication Alert Monitor" on page 329
- "asnmcmd: Working with a running Replication Alert Monitor" on page 333
- "asnscrt: Creating a DB2 replication service to start the replication programs" on page 339
- "asnslist: Listing DB2 replication services" on page 342
- "asnsdrop: Dropping DB2 replication services" on page 343
- "asnpwd: Creating and maintaining password files" on page 344
- "asntrc: Operating the replication trace facility" on page 348
- "asntdiff: Comparing data in source and target tables" on page 355
- "asntrep: Repairing differences between source and target tables" on page 357
- "asnqmfmt: Formatting and viewing Q replication and event publishing messages" on page 358

Road map: Q replication and event publishing system commands

You can use system commands on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS to start, operate, and modify the replication programs.

You can specify parameters in any order as a **name**=*value* pair. Parameters and their arguments are not case sensitive. Use double quotation marks ("") if you want to preserve case.

Specifying yes/no (Boolean) parameters without an argument is supported, but not recommended. For example, specifying **logreuse** is the same as **logreuse**=**y**. But to specify no logreuse, you must use **logreuse**=**n**.

Invoking commands with a question mark (for example, **asnqcap** "?") , displays a help message that shows the command syntax.

Table 31 helps you match common tasks with the system commands.

Table 31. Q replication and event publishing tasks and their corresponding system commands

If you want to ...	Use this command	See page
Start a Q Capture program and specify startup parameters (Linux, UNIX, Windows, z/OS)	**asnqcap**	315
Work with a running Q Capture program (Linux, UNIX, Windows, z/OS) • Check parameter values • Change parameters • Prune the control tables • Check Q Capture status • Stop Q Capture • Reinitialize one or all Q subscriptions or XML publications • Reinitialize one send queue	**asnqccmd**	320
Start a Q Apply program and specify startup parameters (Linux, UNIX, Windows, z/OS)	**asnqapp**	323
Work with a running Q Apply program (Linux, UNIX, Windows, z/OS) • Check parameter values • Change parameters • Check Q Apply status • Prune the control tables • Stop Q Apply • Stop Q Apply reading from a queue • Start Q Apply reading from a queue • Reinitialize one receive queue	**asnqacmd**	326
Start a Replication Alert Monitor and specify startup parameters (Linux, UNIX, Windows, z/OS)	**asnmon**	329
Work with a running Replication Alert Monitor (Linux, UNIX, Windows, z/OS) • Check parameter values • Change parameters • Check monitor status • Stop a monitor • Reinitialize a monitor	**asnmcmd**	333
Create a DB2 replication service to start Q replication or SQL replication programs (Windows only)	**asnscrt**	339
List DB2 replication services (Windows only)	**asnslist**	342
Drop a DB2 replication service (Windows only)	**asnsdrop**	343
Operate the Q Replication Analyzer (Linux, UNIX, Windows)	**asnqanalyze**	336
Create and maintain password files (Linux, UNIX, Windows)	**asnpwd**	344
Operate the replication trace facility (Linux, UNIX, Windows, z/OS)	**asntrc**	348

Table 31. Q replication and event publishing tasks and their corresponding system commands (continued)

If you want to ...	Use this command	See page
Compare data in source and target tables (Linux, UNIX, Windows, z/OS)	**asntdiff**	355
Repair differences between source and target tables (Linux, UNIX, Windows)	**asntrep**	357
Format and view WebSphere MQ messages that are used in Q replication (Linux, UNIX, Windows, z/OS)	**asnqmfmt**	358

Related concepts:

- "System commands for Q replication and event publishing—Overview" on page 313

asnqcap: Starting a Q Capture program

Use the **asnqcap** command to start a Q Capture program on Linux, UNIX, Windows, and UNIX System Services (USS) on z/OS. Run this command at an operating system prompt or in a shell script. Any startup parameters that you specify will apply to this session only.

After you start the Q Capture program, it runs continuously until you stop it or it detects an unrecoverable error.

Syntax

```
►►─asnqcap─capture_server=db_name─┬─────────────────────────┬──────────────►
                                   └─capture_schema=schema─┘

►─┬────────────────────┬─┬─────────────────────┬─┬────────────────┬────────►
  └─capture_path=path─┘ │               ┌─n─┐ │ │          ┌─n─┐ │
                        └─add_partition=┴─y─┴─┘ └─autostop=┴─y─┴─┘

►─┬─────────────────────┬─┬────────────────┬─┬─────────────────┬──────────►
  └─commit_interval=n─┘ │          ┌─n─┐ │ │           ┌─n─┐ │
                        └─logreuse=┴─y─┴─┘ └─logstdout=┴─y─┴─┘

►─┬──────────────────┬─┬──────────────────────┬─┬───────────────────┬─────►
  └─memory_limit=n─┘ └─monitor_interval=n─┘ └─monitor_limit=n─┘

►─┬──────────────────────────┬─┬──────────────────┬─┬──────────────────┬──►
  │         ┌─asnpwd.aut─┐ │ └─prune_interval=n─┘ └─signal_limit=n─┘
  └─pwdfile=┴─filename───┴─┘

►─┬──────────────────┬─┬────────────────────┬─┬────────────┬───────────────►
  └─sleep_interval=n─┘ │          ┌─warmsa─┐ │ │      ┌─y─┐ │
                       └─startmode=┼─warmns─┼─┘ └─term=┴─n─┴─┘
                                   ├─warmsi─┤
                                   └─cold───┘

►─┬───────────────┬────────────────────────────────────────────────────────►◄
  └─trace_limit=n─┘
```

Parameters

Table 32 defines the invocation parameters for the **asnqcap** command.

Table 32. Definitions for asnqcap invocation parameters

Parameter	Definition
capture_server=*db_name*	Specifies the name of the database or subsystem that contains the Q Capture control tables. **Linux, UNIX, Windows:** If you do not specify a Q Capture server, this parameter defaults to the value from the DB2DBDFT environment variable. **z/OS:** Specifies the name of the DB2 subsystem where the Q Capture program will run. For data sharing, do not use the group attach name. Instead, specify a member subsystem name.
capture_schema=*schema*	Specifies a name that identifies the Q Capture program that you want to start.
capture_path=*path*	Specifies the location where you want a Q Capture program to write its log and work files. The default is the directory where you invoked the **asnqcap** command. This is an absolute path name, and double quotes ("") should be used to preserve case.
add_partition=y/n	**Linux, UNIX, Windows:** 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. **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. **n** (default) No new partitions have been added since the last time the Q Capture program was restarted.
autostop=y/n	Specifies whether a Q Capture program stops after reaching the end of the active DB2 log. **y** The Q Capture program stops when it reaches the end of the active DB2 log. **n** (default) The Q Capture program does not stop after reaching the end of the active DB2 log.
commit_interval=*n*	Specifies how often, in milliseconds, a Q Capture program issues an MQCMIT call. This call signals the WebSphere MQ queue manager to make data messages and informational messages that have been placed on queues available to a Q Apply program or subscribing application. The default is 500 milliseconds (0.5 seconds).

Table 32. Definitions for asnqcap invocation parameters (continued)

Parameter	Definition
logreuse=**y/n**	Specifies whether a Q Capture program reuses or appends messages to its diagnostic log file (on Linux, UNIX, and Windows the log file name is *db2instance.capture_server.capture_schema*.QCAP.log and on z/OS the log file name is *capture_server.capture_schema*.QCAP.log). **y** On restart, the Q Capture program reuses its log file by clearing the file and then writing to the blank file. **n** (default) The Q Capture program appends messages to the log file, even after the Q Capture program is restarted.
logstdout=**y/n**	Specifies whether a Q Capture program sends log messages to both its diagnostic log file and the console. **y** The Q Capture program sends log messages to both its log file and the console (stdout). **n** (default) The Q Capture program directs most log messages to the log file only. Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file regardless of the setting for this parameter.
memory_limit=*n*	Specifies the amount of memory, in megabytes, that a Q Capture program can use to build transactions. After this allocation is used, in-memory transactions spill to a file. The default is 32 MB for Linux, UNIX, and Windows and 16 MB for z/OS.
monitor_interval=*n*	Specifies how often, in seconds, a Q Capture program adds a row to the IBMQREP_CAPMON and IBMQREP_CAPQMON tables. The default is 300 seconds (five minutes).
monitor_limit=*n*	Specifies the number of minutes that rows remain in the IBMQREP_CAPMON and the IBMQREP_CAPQMON tables before they can be pruned. At each pruning interval, the Q Capture program prunes rows in these tables if they are older than this limit based on the current timestamp. The default is 10080 minutes (seven days).
pwdfile=*filename*	Specifies the name of the password file that is used to connect to multiple partition databases. If you do not specify a password file, the default is asnpwd.aut. This command searches for the password file in the directory specified by the **capture_path** parameter. If no **capture_path** parameter is specified, this command searches for the password file in the directory where the command was invoked.
prune_interval=*n*	Specifies how often, in seconds, a Q Capture program looks for rows that are old enough to prune in the IBMQREP_SIGNAL, IBMQREP_CAPTRACE, IBMQREP_CAPMON and IBMQREP_CAPQMON tables. The default is 300 seconds (five minutes).

Table 32. Definitions for asnqcap invocation parameters (continued)

Parameter	Definition
signal_limit=*n*	Specifies the number of minutes that rows remain in the IBMQREP_SIGNAL table before they can be pruned. At each pruning interval, the Q Capture program prunes rows in the IBMQREP_SIGNAL table if they are older than the signal limit based on the current timestamp. The default is 10080 minutes (seven days).
sleep_interval=*n*	Specifies the number of seconds 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).
startmode=*mode*	Specifies the actions that a Q Capture program takes when it starts. **warmsi** (default) The Q Capture program starts reading the log at the point where it left off, except if this is the first time that you are starting the program. In that case the Q Capture program switches to a cold start. The warmsi start mode ensures that the Q Capture program cold starts only when it initially starts. **warmns** The Q Capture program starts reading the log at the point where it left off. If it cannot warm start, it does not switch to cold start. The warmns start mode prevents the Q Capture program from cold starting unexpectedly. When the Q Capture program warm starts, it resumes processing where it ended. If errors occur after the Q Capture program starts, the program terminates and leaves all tables intact. **warmsa** If warm start information is available, the Q Capture program starts reading the log at the point where it left off. If the Q Capture program cannot warm start, it switches to a cold start. **cold** The Q Capture program clears the restart queue and administration queue, and starts processing all Q subscriptions or XML 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 only load those Q subscriptions or XML publications that are in not in I (inactive) state.

Table 32. Definitions for asnqcap invocation parameters (continued)

Parameter	Definition
term=**y**/**n**	Specifies whether the Q Capture program terminates if DB2 is quiesced or terminates. **y** (default) The Q Capture program terminates if DB2 is quiesced or terminates. **n** The Q Capture program continues running if DB2 is quiesced. When DB2 is taken out of quiesce mode, the Q Capture program starts in warmns mode, rereads the IBMQREP_CAPPARMS table, and begins capturing at the point where it left off when DB2 was quiesced. If DB2 terminates via FORCE or because of abnormal termination, the Q Capture program terminates even if you set this parameter to **n**. If you set this parameter to **n** and start DB2 with restricted access (ACCESS MAINT), the Q Capture program cannot connect and subsequently terminates.
trace_limit=*n*	The number of minutes that rows remain in the IBMQREP_CAPTRACE table before they can be pruned. At each pruning interval, the Q Capture program prunes rows in this table if they are older than the trace limit based on the current timestamp. The default is 10080 minutes (seven days).

Examples for asnqcap

The following examples illustrate how to use the **asnqcap** command.

Example 1

To start a Q Capture program on a server called sourcedb with a schema of alpha using the warmsi start mode, and to temporarily override the default settings for logreuse and logstdout:

```
asnqcap capture_server=sourcedb capture_schema="alpha" startmode=warmsi
logreuse=y logstdout=y
```

Example 2

To start a Q Capture program and instruct it to commit messages on queues more frequently than the default:

```
asnqcap capture_server=sourcedb capture_schema="alpha" commit_interval=250
```

Example 3

To start a Q Capture program and temporarily increase the default amount of memory that it uses to build transactions:

```
asnqcap capture_server=sourcedb capture_schema="alpha" memory_limit=64
```

Example 4

To start a Q Capture program and direct its work files to the /home/files/qcapture directory:

```
asnqcap capture_server=sourcedb capture_schema="alpha"
capture_path="/home/files/qcapture"
```

Related concepts:

- “Operating a Q Capture program—Overview” on page 201
- “System commands for Q replication and event publishing—Overview” on page 313

Related reference:

- “Descriptions of Q Capture parameters” on page 206
- “Road map: Q replication and event publishing system commands” on page 313

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.

Syntax

```
►►──asnqccmd──capture_server=db_name──┬──────────────────────────┬──────────►
                                      └─capture_schema=schema─┘

►──┬─chgparms──┤ parameters ├─┬──────────────────────────────────────────►◄
   ├─prune───────────────────┤
   ├─qryparms────────────────┤
   ├─reinit──────────────────┤
   ├─reinitq=send_queue──────┤
   ├─status──────────────────┤
   └─stop────────────────────┘
```

Parameters:

```
├──┬─────────────────────┬──┬──────────────────────┬──┬─────────────────────┬──►
   │            ┌─n─┐    │  └─commit_interval=n─┘  │            ┌─n─┐    │
   └─autostop=──┴─y─┴────┘                          └─logreuse=──┴─y─┴────┘

►──┬──────────────────────┬──┬──────────────────┬──┬─────────────────────┬──►
   │             ┌─n─┐    │  └─memory_limit=n─┘  └─monitor_interval=n─┘
   └─logstdout=──┴─y─┴────┘

►──┬───────────────────┬──┬────────────────────┬──┬──────────────────┬──►
   └─monitor_limit=n─┘  └─prune_interval=n─┘  └─signal_limit=n─┘

►──┬────────────────────┬──┬─────────────────┬──┬─────────────────┬──────────┤
   └─sleep_interval=n─┘  │        ┌─y─┐    │  └─trace_limit=n─┘
                          └─term=──┴─n─┴────┘
```

Parameters

Table 33 on page 321 defines the invocation parameters for the **asnqccmd** command.

Table 33. Definitions for asnqccmd invocation parameters

Parameter	Definition
capture_server=*db_name*	Specifies the name of the database or subsystem that contains the Q Capture control tables. **Linux, UNIX, Windows:** If you do not specify a Q Capture server, this parameter defaults to the value from the DB2DBDFT environment variable. **z/OS:** Specifies the name of the DB2 subsystem where the Q Capture program will run. For data sharing, do not use the group attach name. Instead, specify a member subsystem name.
capture_schema=*schema*	Specifies a name that identifies the Q Capture program that you want to work with.
chgparms	Specify to change one or more of the following operational parameters of a Q Capture program while it is running: • **autostop** • **commit_interval** • **logreuse** • **logstdout** • **memory_limit** • **monitor_interval** • **monitor_limit** • **prune_interval** • **signal_limit** • **sleep_interval** • **term** • **trace_limit** You can specify multiple parameters in one **asnqccmd chgparms** command, and you can change these parameter values as often as you want. The changes temporarily override the values in the IBMQREP_CAPPARMS table, but they are not written to the table. When you stop and restart the Q Capture program, it uses the values in IBMQREP_CAPPARMS. asnqcap: Starting a Q Capture program includes descriptions of the parameters that you can override with this command. **Important:** The parameter that you are changing must immediately follow the **chgparms** parameter.
prune	Specify to instruct a Q Capture program to prune the IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_CAPTRACE, and IBMQREP_SIGNAL tables once. This pruning is in addition to any regularly scheduled pruning that is specified by the **prune_interval** parameter.
qryparms	Specify if you want the current operational parameter values for a Q Capture program written to the standard output (stdout).

Table 33. Definitions for asnqccmd invocation parameters (continued)

Parameter	Definition
reinit	Specify to have a Q Capture program deactivate and then activate all Q subscriptions and XML 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 XML publications while a Q Capture program is running. This command will not prompt a new load of targets.
reinitq=*send_queue*	Specify to have the Q Capture program refresh one send queue using the latest attributes from the IBMQREP_SENDQUEUES table. This command affects all Q subscriptions or XML publications that use this send queue. Only the following attributes will be refreshed: ERROR_ACTION, HEARTBEAT_INTERVAL, MAX_MESSAGE_SIZE.
status	Specify to receive messages that indicate the state of each Q Capture thread (main, administration, prune, holdl, and worker).
stop	Specify to stop the Q Capture program in an orderly way and commit the messages that it processed up to that point.

Examples for asnqccmd

The following examples illustrate how to use the **asnqccmd** command.

Example 1

To instruct a Q Capture program to refresh all Q subscriptions and XML publications using the latest values in the Q Capture control tables:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" reinit
```

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 XML publications that use a send queue called Q1:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" reinitq="Q1"
```

Example 3

To temporarily shorten the default pruning interval for a running Q Capture program to one minute, and temporarily lengthen the default amount of time that the Q Capture program sleeps after processing Q subscriptions and XML publications:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" chgparms
 prune_interval=60 sleep_interval=10000
```

Example 4

To receive messages about the status of the Q Capture program's threads:

```
asnqccmd capture_server=sourcedb capture_schema="alpha" status
```

Related concepts:
- "Operating a Q Capture program—Overview" on page 201
- "System commands for Q replication and event publishing—Overview" on page 313

Related reference:
- "Descriptions of Q Capture parameters" on page 206
- "Road map: Q replication and event publishing system commands" on page 313

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─┘ │          ┌─y─┐ │ └─deadlock_retries=n─┘
                      └─autostop=┴─n─┴─┘

►─┬────────────────┬─┬─────────────────┬─┬────────────────────┬─────────────────►
  │          ┌─n─┐ │ │           ┌─n─┐ │ └─monitor_interval=n─┘
  └─logreuse=┴─y─┴─┘ └─logstdout=┴─y─┴─┘

►─┬─────────────────┬─┬─────────────────┬─┬─────────────────────────┬───────────►
  └─monitor_limit=n─┘ └─prune_interval=n─┘ │         ┌─asnpwd.aut─┐ │
                                           └─pwdfile=┴─filename───┴─┘

►─┬────────────┬─┬───────────────┬──────────────────────────────────────────────►◄
  │      ┌─y─┐ │ └─trace_limit=n─┘
  └─term=┴─n─┴─┘
```

Parameters

Table 34 defines the invocation parameters for the **asnqapp** command.

Table 34. Definitions for asnqapp invocation parameters

Parameter	Definition
apply_server=*db_name*	Specifies the name of the database or subsystem that contains the Q Apply control tables. **Linux, UNIX, Windows:** If you do not specify a Q Apply server, this parameter defaults to the value from the DB2DBDFT environment variable. **z/OS:** Specifies the name of the DB2 subsystem where the Q Apply program will run. For data sharing, do not use the group attach name. Instead, specify a member subsystem name.
apply_schema=*schema*	Specifies a name that identifies the Q Apply program that you want to start.
apply_path=*path*	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 ("") to preserve case.
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.
deadlock_retries=*n*	The number of times that a Q Apply program tries to reapply changes to a table after an SQL deadlock or lock timeout. The Q Apply program waits one second between deadlock retries. The default is three tries. You cannot lower this default value.
logreuse=**y/n**	Specifies whether a Q Apply program reuses or appends to its diagnostic log file (on Linux, UNIX, and Windows the log file name is *db2instance.apply_server.apply_schema*.QAPP.log and on z/OS the log file name is *apply_server.apply_schema*.QAPP.log). **y** On restart, the Q Apply program reuses its log file by clearing the file then writing to the blank file. **n** (default) The Q Apply program appends new information to an existing Q Apply log file when it restarts.

Table 34. Definitions for asnqapp invocation parameters (continued)

Parameter	Definition
logstdout=**y**/**n**	Specifies whether a Q Apply program sends log messages to both its log file and the console. **y** The Q Apply program sends messages to both the log file and the console (stdout). **n** (default) The Q Apply program directs most log messages to the log file only. Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file regardless of the setting for this parameter.
monitor_interval=*n*	Specifies how often, in seconds, a Q Apply program adds rows to the IBMQREP_APPLYMON table. The default is 300 seconds (five minutes).
monitor_limit=*n*	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 **monitor_limit** based on the current timestamp. The default is 10080 minutes (seven days).
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/IMPORT or EXPORT/LOAD utilities. When you use the **asnpwd** command to create a password file, the default file name is asnpwd.aut. The Q Apply program searches for the password file in the directory specified by the **apply_path** parameter. If you do not specify an **apply_path** parameter, the Q Apply program searches for the password file in the directory where you invoked the **asnqapp** command.
term=**y**/**n**	Specifies whether the Q Apply program terminates if DB2 is quiesced. **y** (default) The Q Apply program terminates if DB2 is quiesced. **n** The Q Apply program continues running if DB2 is quiesced. When DB2 is taken out of quiesce mode, the Q Apply program begins applying transactions at the point where it left off when DB2 was quiesced. If DB2 terminates via FORCE or because of abnormal termination, the Q Apply program terminates even if you set this parameter to **n**. If you set this parameter to **n** and start DB2 with restricted access (ACCESS MAINT), the Q Apply program cannot connect and subsequently terminates.

Table 34. Definitions for asnqapp invocation parameters (continued)

Parameter	Definition
trace_limit=*n*	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).

Examples for asnqapp

The following examples illustrate how to use the **asnqapp** command.

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
```

Related concepts:
- "Operating a Q Apply program—Overview" on page 227
- "System commands for Q replication and event publishing—Overview" on page 313

Related reference:
- "Descriptions of Q Apply parameters" on page 231
- "Road map: Q replication and event publishing system commands" on page 313

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.

Syntax

```
>>--asnqacmd--apply_server=db_name--+-------------------------+---------->
                                    '-apply_schema=schema-'

>--+-chgparms--| parameters |-+------------------------------------------><
   +-prune--------------------+
   +-qryparms-----------------+
   +-status-------------------+
   +-stop---------------------+
   +-stopq=receive_queue------+
   +-startq=receive_queue-----+
   '-reinitq=receive_queue----'
```

Parameters:

```
|--+----------------------+--+------------------------+--+----------------------+-->
   |            .-y-.     |  '-deadlock_retries=n-'   |            .-n-.     |
   '-autostop=--+-n-+-----'                           '-logreuse=--+-y-+-----'

>--+-----------------------+--+------------------------+--+---------------------+-->
   |             .-n-.     |  '-monitor_interval=n-'    '-monitor_limit=n-'
   '-logstdout=--+-y-+-----'

>--+----------------------+--+------------------+--+-----------------+------------|
   '-prune_interval=n-'      |        .-y-.     |  '-trace_limit=n-'
                             '-term=--+-n-+-----'
```

Parameters

Table 35 defines the invocation parameters for the **asnqacmd** command.

Table 35. Definitions for the asnqacmd invocation parameters

Parameter	Definition
apply_server=*db_name*	Specifies the name of the database or subsystem that contains the Q Apply control tables. **Linux, UNIX, Windows:** If you do not specify a Q Apply server, this parameter defaults to the value from the DB2DBDFT environment variable. **z/OS:** Specifies the name of the DB2 subsystem where the Q Apply program will run. For data sharing, do not use the group attach name. Instead, specify a member subsystem name.
apply_schema=*schema*	Specifies a name that identifies the running Q Apply program that you want to work with.

Table 35. Definitions for the asnqacmd invocation parameters (continued)

Parameter	Definition
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 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.
prune	Specify to instruct the Q Apply program to prune the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables once. This pruning is in addition to any regularly scheduled pruning as specified by the **prune_interval** parameter.
qryparms	Specify if you want the current operational parameter values for the Q Apply program written to the standard output (stdout).
status	Specify to see a message about the state of each Q Apply thread (main, housekeeping, browsers, and agents).
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 from a receive queue, if they aren't already being processed.
reinitq=*receive_queue*	Specify to have the Q Apply program update the attributes for NUM_APPLY_AGENTS and MEMORY_LIMIT from the IBMQREP_RECVQUEUES table for all Q subscriptions that use a particular receive queue. This command works only if the Q Apply program is reading from the receive queue that is named in the IBMQREP_RECVQUEUES table when you issue the command.

Examples for asnqacmd

The following examples illustrate how to use the **asnqacmd** command.

Example 1

To update all Q subscriptions that use a receive queue named Q1 with the latest values for number of Q Apply agents and memory limit from the IBMQREP_RECVQUEUES table:

```
asnqacmd apply_server=targetdb apply_schema="alpha" reinitq=Q1
```

Example 2

To instruct a running Q Apply program to stop after all queues are emptied once, and to lengthen the monitor interval and shorten the trace limit:

```
asnqacmd apply_server=targetdb apply_schema="alpha" chgparms autostop=y
  monitor_interval=500 trace_limit=5000
```

Example 3

To receive messages about the state of each Q Apply thread:

```
asnqacmd apply_server=targetdb apply_schema="alpha" status
```

Example 4

To send the current operational parameters of the Q Apply program to the standard output:

```
asnqacmd apply_server=targetdb apply_schema="alpha" qryparms
```

Example 5

To prune the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables once:

```
asnqacmd apply_server=targetdb apply_schema="alpha" prune
```

Related concepts:
- "Operating a Q Apply program—Overview" on page 227
- "System commands for Q replication and event publishing—Overview" on page 313

Related reference:
- "Descriptions of Q Apply parameters" on page 231
- "Road map: Q replication and event publishing system commands" on page 313

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 and Q Apply programs, and Capture and Apply programs
- Error messages written to the control tables
- Threshold values

Syntax

```
►►─asnmon──┬─────────────────────────┬─monitor_qual=mon_qual─────────────────►
           └─monitor_server=server───┘

►──┬───────────────────┬──┬───────────────────┬──┬──────────────────────┬────►
   └─monitor_interval=n┘  │          ┌─n─┐    │  │            ┌─y─┐     │
                          └─runonce=─┴─y─┴────┘  └─autoprune=─┴─n─┴─────┘

►──┬─────────────────────┬──┬──────────────────────┬──┬──────────────────────┬─►
   │           ┌─n─┐     │  │            ┌─n─┐     │  └─alert_prune_limit=n──┘
   └─logreuse=─┴─y─┴─────┘  └─logstdout=─┴─y─┴─────┘

►──┬───────────────┬──┬──────────────────────────────────┬───────────────────►
   └─trace_limit=n─┘  └─max_notifications_per_alert=n────┘

►──┬────────────────────────────┬──┬──────────────────────────────┬─────────►
   └─max_notifications_minutes=n┘  │          ┌─asnpwd.aut─┐      │
                                   └─pwdfile=─┴─filepath───┴──────┘

►──┬───────────────────┬──┬────────────────────────────────────────────┬────►
   └─monitor_path=path─┘  │                 ┌─,───────────────┐        │
                          │                 ▼                 │        │
                          └─monitor_errors=───"─address─"─────┴────────┘

►──┬─────────────────────────┬─────────────────────────────────────────────►◄
   └─email_server=servername─┘
```

Parameters

Table 36 defines the invocation parameters for the **asnmon** command.

Table 36. asnmon invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems

Parameter	Definition
monitor_server=*server*	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. **Linux, UNIX, Windows:** If you do not specify a Monitor control server, this parameter defaults to the value from the DB2DBDFT environment variable. **z/OS:** The default is DSN.

Table 36. asnmon invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
monitor_qual=*mon_qual*	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.
monitor_interval=*n*	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.
runonce=y/n	Specifies whether the Replication Alert Monitor program runs only one time for this monitor qualifier. **n** (default) The Replication Alert Monitor program runs at the frequency indicated by the **monitor_interval** parameter. **y** The Replication Alert Monitor program runs only one monitor cycle. If you set the runonce parameter to `y`, the **monitor_interval** parameter is ignored by the Replication Alert Monitor.
autoprune=y/n	Specifies whether automatic pruning of the rows in the Replication Alert Monitor alerts (IBMSNAP_ALERTS) table is enabled. **y** (default) The Replication Alert Monitor program automatically prunes the rows in the IBMSNAP_ALERTS table that are older than the value of the **alert_prune_limit** parameter. **n** Automatic pruning is disabled.
logreuse=y/n	Specifies whether the Replication Alert Monitor program reuses or appends messages to its diagnostic log file (*db2instance.monitor_server.mon_qual*.MON.log). **n** (default) The Replication Alert Monitor program appends messages to the log file. **y** The Replication Alert Monitor program reuses the log file by deleting it and then re-creating it when the Replication Alert Monitor program is restarted.

Table 36. asnmon invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
logstdout=**y/n**	Specifies where messages are sent by the Replication Alert Monitor program. **n** (default) The Replication Alert Monitor program sends messages to the log file only. **y** The Replication Alert Monitor program sends messages to both the log file and the standard output (stdout).
alert_prune_limit=*n*	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).
trace_limit=*n*	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 **trace_limit** parameter are pruned at the next pruning cycle. The default is 10 080 minutes (seven days).
max_notifications_per_alert=*n*	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 **max_notifications_minutes** parameter value. Use this parameter to avoid re-sending the same alerts to a user. The default is 3.
max_notifications_minutes=*n*	This parameter works with the **max_notifications_per_alert** parameter to indicate the time period when alert conditions occurred. The default is 60 minutes.
pwdfile=*filepath*	Specifies the fully qualified name of the password file. You define this file using the **asnpwd** command. The default file name is asnpwd.aut.
monitor_path=*path*	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.
monitor_errors=*address*	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.
email_server=*servername*	Specifies the e-mail server address. Enter this parameter *only* if you use the ASNMAIL exit routine with SMTP (Simple Mail Transfer Protocol).

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:

```
asnmon monitor_server=wsdb monitor_qual=monqual runonce=y
```

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
```

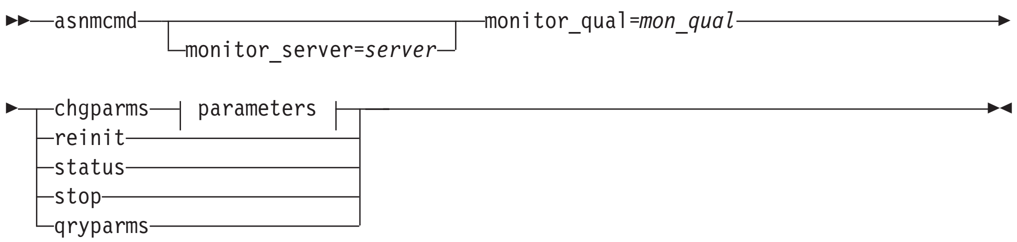
This Replication Alert Monitor program sends a maximum of two alerts when the alerts occurred during the time period specified by the **max_notifications_minutes** parameter value (1440 minutes).

Related reference:
- “asnmcmd: Working with a running Replication Alert Monitor” on page 333

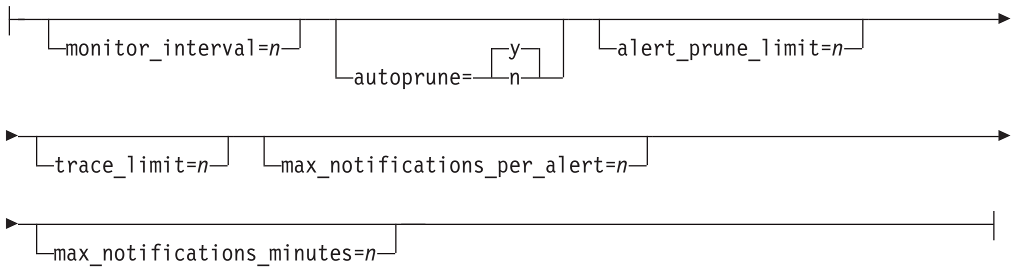
asnmcmd: Working with a running Replication Alert Monitor

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.

Syntax



Parameters:



Parameters

Table 37 defines the invocation parameters for the **asnmcmd** command.

Table 37. asnmcmd invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems

Parameter	Definition
monitor_server=*server*	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. **Linux, UNIX, Windows:** If you do not specify a Monitor control server, this parameter defaults to the value from the DB2DBDFT environment variable. **z/OS:** The default is DSN.
monitor_qual=*mon_qual*	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.

Table 37. asnmcmd invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
chgparms	Specify to change one or more of the following operational parameters of the Replication Alert Monitor while it is running: • **monitor_interval** • **autoprune** • **alert_prune_limit** • **trace_limit** • **max_notifications_per_alert** • **max_notifications_minutes** 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. asnmon: Starting a Replication Alert Monitor includes descriptions of the parameters that you can override with this subcommand. **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.
status	Specify to receive messages that indicate the state of each thread (administration, serialization, and worker) in the Replication Alert Monitor.
qryparms	Specify if you want the current operational parameter values for the Replication Alert Monitor written to the standard output (stdout).
stop	Specify to stop the Replication Alert Monitor in an orderly way.

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
```

Related reference:

- "asnmon: Starting a Replication Alert Monitor" on page 329

asnqanalyze: Operating the Q Replication Analyzer

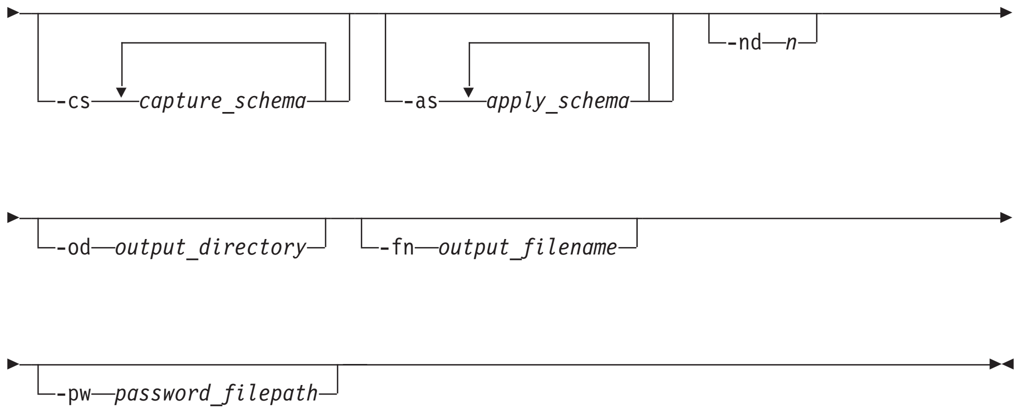
Use the **asnqanalyze** command to generate reports about the state of your Q replication and event publishing environment. The command invokes the Q Replication Analyzer, which generates a formatted HTML report about the state of the Q Capture or Q Apply control tables.

The Analyzer runs on Linux, UNIX, or Windows operating systems only. However, it can connect to a DB2 UDB subsystem on z/OS to analyze the control tables. Run the **asnqanalyze** command at an operating system prompt.

For more information about how to use the Analyzer, see www.ibm.com/software/data/dpropr/library.html.

Syntax

```
                          ┌──────────┐
                          ▼          │
>>──asnqanalyze──-db───db_alias──┴──┬──────────────────────────┬────────────►
                                     │         ┌─standard─┐     │
                                     └─-la───┼─detailed─┼─────┘
                                               └─simple───┘
```



Parameters

Table 38 defines the invocation parameters for the **asnqanalyze** command.

Table 38. asnqanalyze invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
-db *db_alias*	Specifies the Q Capture server or Q Apply server. You must provide at least one database alias. If there is more than one database alias, use blank spaces to separate the values.
-la *level_of_analysis*	Specifies the level of analysis to be reported: **standard** (default) Generates a report that includes the contents of the control tables and status information from the Q Capture and Q Apply programs. **detailed** Generates the information in the standard report and: • Q Capture tuning parameters • Why the Q Capture program is not capturing • Q Apply tuning parameters • Why the Q Apply program is not applying • Incorrect or inefficient table space locksize values • Selected information from the SYSTABLEPART system catalog table for DB2 UDB for z/OS • Information about referential integrity constraints on source and target tables **simple** Generates the information in the standard report, but excludes the detailed information from the IBMQREP_SRC_COLS and IBMQREP_TRG_COLS tables.
-cs *capture_schema*	Specifies the name of the Q Capture schema that you want to analyze. If you specify more than one Q Capture schema, use blank spaces to separate the values. If no **-cs** keyword is specified, then all Q Capture schemas for the specified database aliases are analyzed.

Table 38. asnqanalyze invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
-as *apply_schema*	Specifies the name of the Q Apply schema that you want to analyze. If you specify more than one Q Apply schema, use blank spaces to separate the values. If no **-as** keyword is specified, then all Q Apply schemas for the specified database aliases are analyzed.
-nd *n*	Specifies the date range (0 to 30 days) of information to be retrieved from the following control tables: • IBMQREP_CAPTRACE • IBMQREP_CAPMON • IBMQREP_CAPQMON • IBMQREP_SIGNAL • IBMQREP_APPLYTRACE • IBMQREP_APPLYMON The default is 3 days.
-od *output_directory*	Specifies the directory in which you want to store the Analyzer report. The default is the current directory.
-fn *output_filename*	Specifies the name of the file that will contain the Analyzer report output. Use the file naming conventions of the operating system that you are using to run the Analyzer. If the file name already exists, the file is overwritten. The default file name is asnqanalyze.htm.
-pw *password_filepath*	Specifies the name and path of the password file. If you do not specify this parameter, the Analyzer checks the current directory for the asnpwd.aut file.

Examples for asnqanalyze

The following examples illustrate how to use the **asnqanalyze** command.

Example 1

To analyze the control tables on a database called proddb1:

```
asnqanalyze -db proddb1
```

Example 2

To obtain a detailed level of analysis about the control tables on the proddb1 and proddb2 databases:

```
asnqanalyze -db proddb1 proddb2 -la detailed
```

Example 3

To analyze the last two days of information from the control tables on the proddb1 and proddb2 databases:

```
asnqanalyze -db proddb1 proddb2 -nd 2
```

Example 4

To obtain a simple level of analysis of the control tables on the proddb1 and proddb2 databases and write the analyzer output to a file named anzout under the c:\mydir directory, using the password file stored in the c:\SQLLIB directory:

```
asnqanalyze -db proddb1 proddb2 -la simple -od c:\mydir -fn anzout -pw c:\SQLLIB
```

Example 5

To analyze a specific Q Capture schema on the Q Capture server qcapsvr1:

```
asnqanalyze -db qcapsvr1 -cs ELB
```

Example 6

To analyze a specific Q Apply schema on the Q Apply server qappsvr1:

```
asnqanalyze -db qappsvr1 -as CLS
```

Example 7

To analyze specific Q Capture and Q Apply schemas:

```
asnqanalyze -db qcapsvr1 qappsvr1 -cs qcap1 -as qapp1
```

Example 8

To display command help:

```
asnqanalyze
```

Related concepts:
- "System commands for Q replication and event publishing—Overview" on page 313

Related tasks:
- "Storing user IDs and passwords for remote servers" on page 41

Related reference:
- "Road map: Q replication and event publishing system commands" on page 313

asnscrt: Creating a DB2 replication service to start the replication programs

Use the **asnscrt** command to create a DB2 replication service in the Windows Service Control Manager (SCM) and invoke the **asnqcap**, **asnqapp**, **asnmon**, **asncap**, and **asnapply** commands. Run the **asnscrt** command on the Windows operating system.

Syntax

```
►►──asnscrt──┬─-QC─┬──db2_instance──account──password──┬─asnqcap_command──┬──────►◄
             ├─-QA─┤                                    ├─asnqapp_command──┤
             ├─-M──┤                                    ├─asnmon_command───┤
             ├─-C──┤                                    ├─asncap_command───┤
             └─-A──┘                                    └─asnapply_command─┘
```

Parameters

Table 39 defines the invocation parameters for the **asnscrt** command.

Table 39. asnscrt invocation parameter definitions for Windows operating systems

Parameter	Definition
-QC	Specifies that you are starting a Q capture program.
-QA	Specifies that you are starting a Q apply program.
-M	Specifies that you are starting a Replication Alert Monitor program.
-C	Specifies that you are starting a Capture program.
-A	Specifies that you are starting an Apply program.
db2_instance	Specifies the DB2 instance used to identify a unique DB2 replication service. The DB2 instance can be a maximum of eight characters.
account	Specifies the account name that you use to log on to Windows. The account name must begin with a period and a backslash (.\).
password	Specifies the password used with the account name. If the password contains special characters, type a backslash (\) before each special character.
asnqcap_command	Specifies the complete **asnqcap** command to start a Q capture program. Use the documented **asnqcap** command syntax with the appropriate **asnqcap** parameters. **Important:** If the DB2PATH environment variable is not defined, you must specify a location for the work files by including the **capture_path** parameter with the **asnqcap** command. If the DB2PATH variable is defined and you specify a **capture_path**, the **capture_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnqcap** parameters that you enter.
asnqapp_command	Specifies the complete **asnqapp** command to start a Q apply program. Use the documented **asnqapp** command syntax with the appropriate **asnqapp** parameters. **Important:** If the DB2PATH environment variable is not defined, you must specify the location for the work files by including the **apply_path** parameter with the **asnqapp** command. If the DB2PATH variable is defined and you specify an **apply_path**, the **apply_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnqapp** parameters that you enter.

Table 39. asnscrt invocation parameter definitions for Windows operating systems (continued)

Parameter	Definition
asnmon_command	Specifies the complete **asnmon** command to start a Replication Alert Monitor program. Use the documented **asnmon** command syntax with the appropriate **asnmon** parameters. **Important:** If the DB2PATH environment variable is not defined, you must specify a location for the log files by including the **monitor_path** parameter with the **asnmon** command. If the DB2PATH variable is defined and you specify a **monitor_path**, the **monitor_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnmon** parameters that you enter.
asncap_command	Specifies the complete **asncap** command to start a Capture program. Use the documented **asncap** command syntax with the appropriate **asncap** parameters. **Important:** If the DB2PATH environment variable is not defined, you must specify a location for the work files by including the **capture_path** parameter with the **asncap** command. If the DB2PATH variable is defined and you specify a **capture_path**, the **capture_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asncap** parameters that you enter.
asnapply_command	Specifies the complete **asnapply** command to start an Apply program. Use the documented **asnapply** command syntax with the appropriate **asnapply** parameters. **Important:** If the DB2PATH environment variable is not defined, you must specify the location for the work files by including the **apply_path** parameter with the **asnapply** command. If the DB2PATH variable is defined and you specify an **apply_path**, the **apply_path** parameter overrides the DB2PATH variable. The **asnscrt** command does not validate the syntax of the **asnapply** parameters that you enter.

Examples for asnscrt

The following examples illustrate how to use the **asnscrt** command.

Example 1

To create a DB2 replication service that invokes a Q capture program under a DB2 instance called inst1:

```
asnscrt -QC inst1 .\joesmith password asnqcap capture_server=mydb1
  capture_schema=QC1 capture_path=X:\logfiles
```

Example 2

To create a DB2 replication service that invokes a Q apply program under a DB2 instance called inst2 using 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 3

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 4

To create a DB2 replication service that invokes an Apply program under a DB2 instance called inst2 using 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 5

To create a DB2 replication service that invokes a Replication Alert Monitor program under a DB2 instance called inst3:

```
asnscrt -M inst3 .\joesmith password asnmon monitor_server=db3 monitor_qual=mq3
  monitor_path=X:\logfiles
```

Example 6

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
```

asnslist: Listing DB2 replication services

Use the **asnslist** command to list the DB2 replication services in the Windows Service Control Manager (SCM). You can optionally use the command to list details about each service. Run the **asnslist** command on the Windows operating system.

Syntax

```
►►—asnslist—┬─────────┬——————————————————————————————————————————————————►◄
            └─DETAILS─┘
```

Parameters

Table 40 defines the invocation parameter for the **asnslist** command.

Table 40. asnslist invocation parameter definition for Windows operating systems

Parameter	Definition
details	Specifies that you want to list detailed data about all DB2 replication services on a system.

Examples for asnlist

The following examples illustrate how to use the **asnslist** command.

Example 1

To list the names of DB2 replication services on a system:

```
asnslist
```

Here is an example of the command output:

```
DB2.DB2.SAMPLE.QAPP.ASN
DB2.DB4.SAMPLE.QCAP.ASN
```

Example 2

To list details about all services on a system:

```
asnslist details
```

Here is an example of the command output:

```
DB2.DB2.SAMPLE.QAPP.ASN
Display Name: DB2 DB2 SAMPLE QAPPLY ASN
Image Path:   ASNSERV DB2.DB2.SAMPLE.APP.AQ1 -ASNQAPPLY QAPPLY_SERVER=SAMPLE AP
              PLY_SCHEMA=ASN QAPPLY_PATH=C:\PROGRA~1\SQLLIB
Dependency:   DB2-0

DB2.DB4.SAMPLE.QCAP.ASN
Display Name: DB2 DB4 SAMPLE QAPPLY ASN
Image Path:   ASNSERV DB2.DB4.SAMPLE.APP.AQ1 -ASNQCAP QCAPTURE_SERVER=SAMPLE CA
              PTURE_SCHEMA=ASN QCAPTURE_PATH=C:\PROGRA~1\SQLLIB
Dependency:   DB4-0
```

asnsdrop: Dropping DB2 replication services

Use the **asnsdrop** command to drop DB2 replication services from the Windows Service Control Manager (SCM) on the Windows operating system. (You create a DB2 replication service using the **asnscrt** command.)

Syntax

```
►►—asnsdrop——┬—service_name—┬——————————————————————————————————————►◄
             └—ALL——————————┘
```

Parameters

Table 41 defines the invocation parameters for the **asnsdrop** command.

Table 41. asnsdrop invocation parameter definitions for Windows operating systems

Parameter	Definition
service_name	Specifies the fully qualified name of the DB2 replication service. Enter the Windows SCM to obtain the DB2 replication service name. On Windows operating systems, you can obtain the service name by opening the Properties window of the DB2 replication service. If the DB2 replication service name contains spaces, enclose the entire service name in double quotation marks.
ALL	Specifies that you want to drop all DB2 replication services.

Examples for asnsdrop

The following examples illustrate how to use the **asnsdrop** command.

Example 1

To drop a DB2 replication service:

```
asnsdrop DB2.SAMPLEDB.SAMPLEDB.CAP.ASN
```

Example 2

To drop a DB2 replication service with a schema named A S N:

```
asnsdrop "DB2.SAMPLEDB.SAMPLEDB.CAP.A S N"
```

Example 3

To drop all DB2 replication services:

```
asnsdrop ALL
```

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.

The parameter keywords of this command are *not* case sensitive.

Command help appears if you enter the **asnpwd** command without any parameters, followed by a ?, or followed by incorrect parameters.

Syntax

```
►►─asnpwd──┬─init──┤ Init parameters ├────────┬──────────────────────────────────►◄
           ├─add──┤ Add parameters ├──────────┤
           ├─modify──┤ Modify parameters ├────┤
           ├─delete──┤ Delete parameters ├────┤
           └─list──┤ List parameters ├────────┘
```

Init parameters:

```
├──┬──────────────────────────┬──┬──────────────────────────────┬──────────────────────────┤
   └─encrypt──┬─all──────┬────┘  │         ┌─asnpwd.aut──────┐  │
              └─password─┘       └─using───┴─filepath_name───┴──┘
```

Add parameters:

```
├──alias──db_alias──id──userid──password──password──────────────────────────────────────────►

►──┬──────────────────────────────┬─────────────────────────────────────────────────────────┤
   │         ┌─asnpwd.aut──────┐  │
   └─using───┴─filepath_name───┴──┘
```

Modify parameters:

```
├──alias──db_alias──id──userid──password──password──────────────────────────────────────────►

►──┬──────────────────────────────┬─────────────────────────────────────────────────────────┤
   │         ┌─asnpwd.aut──────┐  │
   └─using───┴─filepath_name───┴──┘
```

Delete parameters:

```
├──alias──db_alias──┬──────────────────────────────┬────────────────────────────────────────┤
                    │         ┌─asnpwd.aut──────┐  │
                    └─using───┴─filepath_name───┴──┘
```

List parameters:

```
├──┬──────────────────────────────┬─────────────────────────────────────────────────────────┤
   │         ┌─asnpwd.aut──────┐  │
   └─using───┴─filepath_name───┴──┘
```

Parameters

Table 42 defines the invocation parameters for the **asnpwd** command.

Table 42. asnpwd invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
init	Specify to create an empty password file. This command will fail if you specify the **init** parameter with a password file that already exists.
add	Specify to add an entry to the password file. This command will fail if you specify the **add** parameter with an entry that already exists in the password file. Use the **modify** parameter to change an existing entry in the password file.
modify	Specify to modify the password or user ID for an entry in the password file.
delete	Specify to delete an entry from the password file.

Table 42. asnpwd invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
list	Specify to list the aliases and user ID entries in a password file. This parameter can be used only if the password file was created using the **encrypt** parameter. Passwords are never displayed by the **list** command.
encrypt	Specifies which entries in a file to encrypt. **all** (default) Encrypt all entries in the specified file such that you cannot list the database aliases, user names, and passwords that are in the file. This option reduces the exposure of information in password files. **password** Encrypt the password entry in the specified file. This option allows users to list the database aliases and user names stored in their password file. Passwords can never be displayed.
using *filepath_name*	Specifies the path and name of the password file. Follow the file naming conventions of your operating system. An example of a valid password file on Windows is C:\sqllib\mypwd.aut. If you specify the path and name of the password file, the path and the password file must already exist. If you are using the **init** parameter and you specify the path and name of the password file, the path must already exist and the command will create the password file for you. If you do not specify this parameter, the default file name is asnpwd.aut and the default file path is the current directory.
alias *db_alias*	Specifies the alias of the database to which the user ID has access. The alias is always folded to uppercase, regardless of how it is entered.
id *userid*	Specifies the user ID that has access to the database.
password *password*	Specifies the password for the specified user ID. This password is case sensitive and is encrypted in the password file.

Return codes

The **asnpwd** command returns a zero return code upon successful completion. A nonzero return code is returned if the command is unsuccessful.

Examples for asnpwd

The following examples illustrate how to use the **asnpwd** command.

Example 1

To create a password file with the default name of asnpwd.aut in the current directory:

```
asnpwd INIT
```

Example 2

To create a password file named pass1.aut in the c:\myfiles directory:

```
asnpwd INIT USING c:\myfiles\pass1.aut
```

Example 3

To create a password file named mypwd.aut using the **encrypt all** parameter:

```
asnpwd INIT ENCRYPT ALL USING mypwd.aut
```

Example 4

To create a password file named mypwd.aut using the **encrypt password** parameter:

```
asnpwd INIT ENCRYPT PASSWORD USING mypwd.aut
```

Example 5

To create a default password file using 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 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 using the **encrypt all** parameter, the following details are listed:

  ```
  asnpwd LIST USING pass1.aut
  Alias: SAMPLE  ID: chglocalid
  Number of Entries: 1
  ```

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.

Syntax

```
►►—asntrc————————————————————————————————————————————————————————►
```

```
>>-+-on----db--db_name--+-qcap-+----------------------------+-+--| On parameters |----+-><
   |                    |      '--schema--qcapture_schema---' |                        |
   |                    +-qapp-+----------------------------+-+                        |
   |                    |      '--schema--qapply_schema-----' |                        |
   |                    +-cap--+----------------------------+-+                        |
   |                    |      '--schema--capture_schema----' |                        |
   |                    +-app--+----------------------------+-+                        |
   |                    |      '--qualifier--apply_qualifier' |                        |
   |                    '-mon--+------------------------------+                        |
   |                           '--qualifier--monitor_qualifier'                        |
   +-+-off-------+--db--db_name--+-qcap-+----------------------------+-+---------------+
   | +-kill------+               |      '--schema--qcapture_schema---' |               |
   | +-clr-------+               +-qapp-+----------------------------+-+               |
   | +-diag------+               |      '--schema--qapply_schema-----' |               |
   | '-resetlock-'               +-cap--+----------------------------+-+               |
   |                             |      '--schema--capture_schema----' |               |
   |                             +-app--+----------------------------+-+               |
   |                             |      '--qualifier--apply_qualifier' |               |
   |                             '-mon--+------------------------------+               |
   |                                    '--qualifier--monitor_qualifier'               |
   +-dmp--filename----db--db_name--+-qcap-+----------------------------+-+-+-----------+-+
   |                               |      '--schema--qcapture_schema---' | '--holdlock-' |
   |                               +-qapp-+----------------------------+-+               |
   |                               |      '--schema--qapply_schema-----' |               |
   |                               +-cap--+----------------------------+-+               |
   |                               |      '--schema--capture_schema----' |               |
   |                               +-app--+----------------------------+-+               |
   |                               |      '--qualifier--apply_qualifier' |               |
   |                               '-mon--+------------------------------+               |
   |                                      '--qualifier--monitor_qualifier'               |
   +-+-flw---+--+--------------------------------------------------------+--| Format parameters |--+
   | +-fmt---+  '-+--------------+--+-qcap-+----------------------------+-+                        |
   | '-v7fmt-'    '--db--db_name-'  |      '--schema--qcapture_schema---' |                        |
   |                                +-qapp-+----------------------------+-+                        |
   |                                |      '--schema--qapply_schema-----' |                        |
   |                                +-cap--+----------------------------+-+                        |
   |                                |      '--schema--capture_schema----' |                        |
   |                                +-app--+----------------------------+-+                        |
   |                                |      '--qualifier--apply_qualifier' |                        |
   |                                '-mon--+------------------------------+                        |
   |                                       '--qualifier--monitor_qualifier'                        |
   +-+-stat-----+--+---------------------------------------------------------------+---------------+
   | '-statlong-'  +-+--------------+--+-qcap-+----------------------------+-+---+                 |
   |               | '--db--db_name-'  |      '--schema--qcapture_schema---' |   |                 |
   |               |                   +-qapp-+----------------------------+-+   |                 |
   |               |                   |      '--schema--qapply_schema-----' |   |                 |
   |               |                   +-cap--+----------------------------+-+   |                 |
   |               |                   |      '--schema--capture_schema----' |   |                 |
   |               |                   +-app--+----------------------------+-+   |                 |
   |               |                   |      '--qualifier--apply_qualifier' |   |                 |
   |               |                   '-mon--+------------------------------+   |                 |
   |               |                          '--qualifier--monitor_qualifier'   |                 |
   |               '--fn--filename-----------------------------------------------'                 |
   +----db--db_name--+-qcap-+----------------------------+-+--| Change settings parameters |--------+
   |                 |      '--schema--qcapture_schema---' |                                       |
   |                 +-qapp-+----------------------------+-+                                       |
   |                 |      '--schema--qapply_schema-----' |                                       |
   |                 +-cap--+----------------------------+-+                                       |
   |                 |      '--schema--capture_schema----' |                                       |
   |                 +-app--+----------------------------+-+                                       |
   |                 |      '--qualifier--apply_qualifier' |                                       |
   |                 '-mon--+------------------------------+                                       |
   |                        '--qualifier--monitor_qualifier'                                       |
   +-+---------+-----------------------------------------------------------------------------------+
   | '--help---'                                                                                   |
   '--listsymbols----------------------------------------------------------------------------------'
```

On parameters:

```
|--+------------------+--+---------------+--+---------------+----------------------->
   '--b--buffer_size--'  '--fn--filename-'  '--fs--filesize-'

>--+----------------+--+--------------------------------------------------+--------|
   '--d--diag_mask--'  '--df--function_name|component_name diag_mask-----'
```

Format parameters:

```
|--+----------------+--+----------------+------------------------------------------->
   '--fn--filename--'  '--d--diag_mask--'
```

```
►─┬──────────────────────────────────────────────────┬──┬─────────────┬──────────────────┤
  └─-df─function_name|component_name diag_mask─┘  └─-holdlock─┘
```

Change settings parameters:

```
├──┬───────────────┬──┬──────────────────────────────────────────────────┬────────────────┤
   └─-d─diag_mask─┘  └─-df─function_name|component_name diag_mask─┘
```

Parameters

Table 43 defines the invocation parameters for the **asntrc** command.

Table 43. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems

Parameter	Definition
on	Specify to turn on the trace facility for a specific Q Capture, Q Apply, Capture, Apply, or Replication Alert Monitor program. The trace facility creates a shared memory segment used during the tracing process.
-db *db_name*	Specifies the name of the database to be traced: • Specifies the name of the Q Capture server for the Q Capture program to be traced. • Specifies the name of the Q Apply server for the Q Apply program to be traced. • Specifies the name of the Capture control server for the Capture program to be traced. • Specifies the name of the Apply control server for the Apply program to be traced. • Specifies the name of the Monitor control server for the Replication Alert Monitor program to be traced.
-qcap	Specifies that a Q Capture program is to be traced. The Q Capture program is identified by the **-schema** parameter.
-schema *qcapture_schema*	Specifies the name of the Q Capture program to be traced. The Q Capture program is identified by the Q Capture schema that you enter. Use this parameter with the **-qcap** parameter.
-qapp	Specifies that a Q Apply program is to be traced. The Q Apply program is identified by the **-schema** parameter.
-schema *qapply_schema*	Specifies the name of the Q Apply program to be traced. The Q Apply program is identified by the Q Apply schema that you enter. Use this parameter with the **-qapp** parameter.
-cap	Specifies that a Capture program is to be traced. The Capture program is identified by the **-schema** parameter.
-schema *capture_schema*	Specifies the name of the Capture program to be traced. The Capture program is identified by the Capture schema that you enter. Use this parameter with the **-cap** parameter.
-app	Specifies that an Apply program is to be traced. The Apply program is identified by the **-qualifier** parameter.

Table 43. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
-qualifier *apply_qualifier*	Specifies the name of Apply program to be traced. This Apply program is identified by the Apply qualifier that you enter. Use this parameter with the **-app** parameter.
-mon	Specifies that a Replication Alert Monitor program is to be traced. The Replication Alert Monitor program is identified by the **-qualifier** parameter.
-qualifier *monitor_qualifier*	Specifies the name of Replication Alert Monitor program to be traced. This Replication Alert Monitor program is identified by the monitor qualifier that you enter. Use this parameter with the **-mon** parameter.
off	Specify to turn off the trace facility for a specific Q Capture, Q Apply, Capture, Apply, or Replication Alert Monitor program and free the shared memory segment in use.
kill	Specify to force an abnormal termination of the trace facility. Use this parameter only if you encounter a problem and are unable to turn the trace facility off with the **off** parameter.
clr	Specify to clear a trace buffer. This parameter erases the contents of the trace buffer but leaves the buffer active.
diag	Specify to view the filter settings while the trace facility is running.
resetlock	Specify to release the buffer latch of a trace facility. This parameter enables the buffer latch to recover from an error condition in which the trace program terminated while holding the buffer latch.
dmp *filename*	Specify to write the current contents of the trace buffer to a file.
-holdlock	Specifies that the trace facility can complete a file dump or output command while holding a lock, even if the trace facility finds insufficient memory to copy the buffer.
flw	Specify to display summary information produced by the trace facility and stored in shared memory or in a file. This information includes the program flow and is displayed with indentations that show the function and call stack structures for each process and thread.
fmt	Specify to display detailed information produced by the trace facility and stored in shared memory or in a file. This parameter displays the entire contents of the traced data structures in chronological order.
v7fmt	Specify to display information produced by the trace facility and stored in shared memory or in a file. This trace information appears in Version 7 format.
stat	Specify to display the status of a trace facility. This status information includes the trace version, application version, number of entries, buffer size, amount of buffer used, status code, and program timestamp.

Table 43. asntrc invocation parameter definitions for Linux, UNIX, Windows, and z/OS operating systems (continued)

Parameter	Definition
statlong	Specify to display the status of a trace facility with additional z/OS version level information. This additional information includes the service levels of each module in the application and appears as long strings of text.
-fn *filename*	Specifies the file name containing the mirrored trace information, which includes all the output from the trace facility.
-help	Displays the valid command parameters with descriptions.
-listsymbols	Displays the valid function and component identifiers to use with the **-df** parameter.
-b *buffer_size*	Specifies the size of the trace buffer (in bytes). You can enter a `K` or an `M` after the number to indicate kilobytes or megabytes, respectively; these letters are not case sensitive.
-fs *filesize*	Specifies the size limit (in bytes) of the mirrored trace information file.
-d *diag_mask*	Specifies the types of trace records to be recorded by the trace facility. Trace records are categorized by a diagnostic mask number: **1** Flow data, which includes the entry and exit points of functions. **2** Basic data, which includes all major events encountered by the trace facility. **3** Detailed data, which includes the major events with descriptions. **4** Performance data. **Important:** The higher diagnostic mask numbers are *not* inclusive of the lower diagnostic mask numbers. You can enter one or more of these numbers to construct a diagnostic mask that includes only the trace records that you need. For example, specify **-d 4** to record only performance data; specify **-d 1,4** to record only flow and performance data; specify **-d 1,2,3,4** (the default) to record all trace records. Separate the numbers with commas. Enter a diagnostic mask number of 0 (zero) to specify that no global trace records are to be recorded by the trace facility. Type **-d 0** to reset the diagnostic level before specifying new diagnostic mask numbers for a tracing facility.
-df *function_name*|*component_name diag_mask*	Specifies that a particular function or component identifier is to be traced. Type the diagnostic mask number (1,2,3,4) after the function or component identifier name. You can enter one or more of these numbers. Separate the numbers with commas.

Examples for asntrc

The following examples illustrate how to use the **asntrc** command. These examples can be run on Linux, UNIX, Windows, or z/OS operating systems.

Example 1

To trace a running Capture program:

1. Start the trace facility, specifying a trace file name with a maximum buffer and file size:

   ```
   asntrc on -db mydb -cap -schema myschema -b 256k -fn myfile.trc -fs 500m
   ```
2. Start the Capture program, and let it run for an appropriate length of time.
3. While the trace facility is on, display the data directly from shared memory.

   To display the summary process and thread information from the trace facility:

   ```
   asntrc flw -db mydb -cap -schema myschema
   ```

   To view the flow, basic, detailed, and performance data records only from the Capture log reader:

   ```
   asntrc fmt -db mydb -cap -schema myschema -d 0
     -df "Capture Log Read" 1,2,3,4
   ```
4. Stop the trace facility:

   ```
   asntrc off -db mydb -cap -schema myschema
   ```

   The trace file contains all of the Capture program trace data that was generated from the start of the Capture program until the trace facility was turned off.
5. After you stop the trace facility, format the data from the generated binary file:

   ```
   asntrc flw -fn myfile.trc
   ```

   and

   ```
   asntrc fmt -fn myfile.trc -d 0 -df "Capture Log Read" 1,2,3,4
   ```

Example 2

To start a trace facility of a Replication Alert Monitor program:

```
asntrc on -db mydb -mon -qualifier monq
```

Example 3

To trace only performance data of an Apply program:

```
asntrc on -db mydb -app -qualifier aq1 -b 256k -fn myfile.trc -d 4
```

Example 4

To trace all flow and performance data of a Capture program:

```
asntrc on dbserv1 -cap -schema myschema -b 256k
  -fn myfile.trc -d 1,4
```

Example 5

To trace all global performance data and the specific Capture log reader flow data of a Capture program:

```
asntrc on -db mydb -cap -schema myschema -b 256k -fn myfile.trc -d 4
  -df "Capture Log Read" 1
```

Example 6

To trace a running Capture program and then display and save a point-in-time image of the trace facility:

1. Start the trace command, specifying a buffer size large enough to hold the latest records:
   ```
   asntrc on -db mydb -cap -schema myschema -b 4m
   ```
2. Start the Capture program, and let it run for an appropriate length of time.
3. View the detailed point-in-time trace information that is stored in shared memory:
   ```
   asntrc fmt -db mydb -cap -schema myschema
   ```
4. Save the point-in-time trace information to a file:
   ```
   asntrc dmp myfile.trc -db mydb -cap -schema myschema
   ```
5. Stop the trace facility:
   ```
   asntrc off -db mydb -cap -schema myschema
   ```

Examples for asntrc using shared segments

The standalone trace facility, **asntrc**, uses a shared segment to communicate with the respective Q Capture, Q Apply, Capture, Apply or Replication Alert Monitor programs to be traced. The shared segment will also be used to hold the trace entries if a file is not specified. Otherwise, matching options must be specified for both the **asntrc** command and for the respective programs to be traced to match the correct shared segment to control traces. The following examples show the options that need to be specified when the trace facility is used in conjunction with Q Capture, Q Apply, Capture, Apply or Alert Monitor programs.

With the Q Capture program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **capture_server** parameter with the **asnqcap** command:

```
asntrc -db ASN6 -schema EMI -qcap
asnqcap capture_server=ASN6 capture_schema=EMI
```

With the Q Apply program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **apply_server** parameter with the **asnqapp** command:

```
asntrc -db TSN3 -schema ELB -qapp
asnqapp apply_server=TSN3 apply_schema=ELB
```

With the Capture program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **capture_server** parameter with the **asncap** command:

```
asntrc -db DSN6 -schema JAY -cap
asncap capture_server=DSN6 capture_schema=JAY
```

With the Apply program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **control_server** parameter with the **asnapply** command:

```
asntrc -db SVL_LAB_DSN6 -qualifier MYQUAL -app
asnapply control_server=SVL_LAB_DSN6 apply_qual=MYQUAL
```

With the Replication Alert Monitor program, the database specified by the **-db** parameter with the **asntrc** command needs to match the database specified by the **monitor_server** parameter with the **asnmon** command:

```
asntrc -db DSN6 -qualifier MONQUAL -mon
asnmon monitor_server=DSN6 monitor_qual=MONQUAL
```

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 UDB tables on Linux, UNIX, Windows, z/OS, and iSeries operating systems.

Syntax

```
►►──asntdiff──db=server──schema=schema──where=WHERE_clause──────────────►

►──┬─────────────────┬──────────────────────────────────────────────────►◄
   └─diff=table_name─┘
```

Parameters

Table 44 defines the invocation parameters for the **asntdiff** command.

Table 44. asntdiff invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
db=*server*	Specifies the DB2 UDB 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: **Q replication** The value is the name of the Q Capture server, which contains the IBMQREP_SUBS table. **SQL replication** The value is the name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table.
schema=*schema*	Specifies the schema of the Q Capture control tables for Q replication, or the Apply control tables for SQL replication.

Table 44. asntdiff invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
where=*WHERE_clause*	Specifies a SQL WHERE clause that uniquely identifies one row of the control table that stores information about the source and target tables that will be compared. The WHERE clause must be in double quotation marks. The value of this parameter differs depending on whether you are using Q replication or SQL replication: **Q replication** The WHERE clause specifies a row in the IBMQREP_SUBS table, using 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, using 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=*table_name*	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, the difference table will be named *schema*.ASNTDIFF, where *schema* is the schema of the Q Capture control tables or Apply control tables that contain information about the source and target tables that you are comparing.

Examples for asntdiff

The following examples illustrate 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="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="where set_name = 'my_set'
 and target_table = 'trg_table'" diff=diff_table
```

Related concepts:

- "System commands for Q replication and event publishing—Overview" on page 313

Related reference:
- "Road map: Q replication and event publishing system commands" on page 313
- "asntrep: Repairing differences between source and target tables" on page 357

asntrep: Repairing differences between source and target tables

Use the **asntrep** command to synchronize a source and target table by repairing differences between the two tables. Run the **asntrep** command on Linux, UNIX, or Windows at an operating system prompt or in a shell script.

Syntax

```
>>—asntrep—db=server—schema=schema—where=WHERE_clause—┬───────────────────┬——><
                                                      └─diff=table_name─┘
```

Parameters

Table 45 defines the invocation parameters for the **asntrep** command.

Table 45. asntrep invocation parameter definitions for Linux, UNIX, and Windows operating systems

Parameter	Definition
db=*server*	Specifies the DB2 UDB alias of the database that stores information about the source and target tables that you want to synchronize. The value differs depending on whether you are using Q replication or SQL replication: **Q replication** The value is the name of the Q Capture server, which contains the IBMQREP_SUBS table. **SQL replication** The value is the name of the Apply control server, which contains the IBMSNAP_SUBS_MEMBR table.
schema=*schema*	Specifies the schema of the Q Capture control tables for Q replication, or the Apply control tables for SQL replication.
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, using 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, using the SET_NAME, APPLY_QUAL, TARGET_SCHEMA, and TARGET_TABLE columns to identify the subscription set member that contains the source and target tables.

Table 45. asntrep invocation parameter definitions for Linux, UNIX, and Windows operating systems (continued)

Parameter	Definition
diff=*table_name*	Specifies the name of the table that was created in the source database using the **asntdiff** command to store differences between the source and target tables. The information that is stored in this table will be 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="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="where set_name = 'my_set'
 and target_table = 'trg_table'" diff=sql_diff_table
```

Related concepts:

- "System commands for Q replication and event publishing—Overview" on page 313

Related reference:

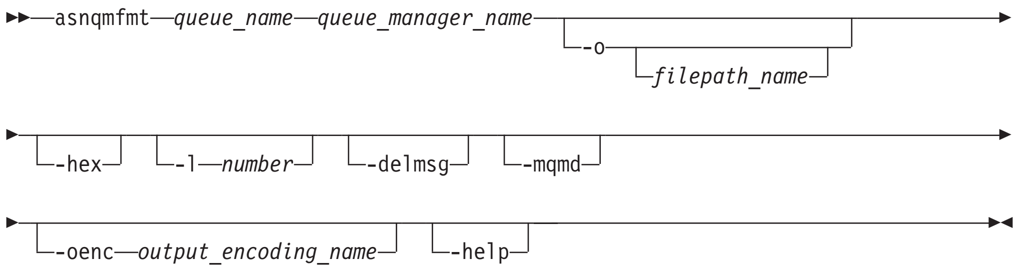
- "Road map: Q replication and event publishing system commands" on page 313
- "asntdiff: Comparing data in source and target tables" on page 355

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.

On z/OS, you can operate the message formatting program with JCL. You can find sample JCL in the sample qrhlqual.SASNSAMP(ASNQMFMT). See "Sample JCL programs for Q replication and event publishing (z/OS)" on page 468.

Syntax



Parameters

Table 46 describes the invocation parameters for **asnqmfmt**:

Table 46. asnqmfmt invocation parameter definitions

Parameter	Definition
queue_name	Specifies the name of a WebSphere MQ queue whose messages you want to format, view, and optionally delete.
queue_manager_name	Specifies the name of a WebSphere MQ queue manager where the queue is defined.
-o *filepath_name*	Specifies the name of the file that contains the formatted output. If the **-o** parameter is not specified, the formatted messages will be written to the standard output (stdout). The output file will be written to a z/OS data set if its name starts with //. By default, the file is created in the directory from which the **asnqmfmt** command was invoked. You can change the directory by specifying a path with the file name.
-hex	Specifies that the messages are formatted in hexadecimal. If you do not specify this parameter, messages will be displayed according to their message format type, either compact or XML.
-1 *number*	Specifies the number of messages that you want to format.
-delmsg	Specifies that the messages will be deleted from the queue after they are formatted.
-mqmd	Specifies that you want to view the WebSphere MQ message descriptor for each message that is formatted.
-oenc *output_encoding_name*	Specifies a code page to be used for formatting the messages. If you do not specify this parameter, messages will be formatted in the default code page for the operating system where the command is invoked.
-help	Displays the valid command parameters with descriptions.

Examples for asnqmfmt

The following examples illustrate how to use the **asnqmfmt** command.

Example 1

To view on the standard output all messages that are on send queue Q1 that is defined in queue manager QMGR1:

```
asnqmfmt Q1 QMGR1
```

Example 2

To view all messages that are on send queue Q1 in a file called Q1_messages that is stored in the C:\qrepl directory (Windows):

```
asnqmfmt Q1 QMGR1 -o C:\qrepl\Q1_messages
```

Example 3

To view on the standard output a hexadecimal version of all messages that are on administration queue ADMNQ1 that is defined in the queue manager QMGR1:

```
asnqmfmt ADMNQ1 QMGR1 -hex
```

Example 4

To view on the standard output the message body and message descriptor of all messages that are on administration queue ADMNQ1 that is defined in the queue manager QMGR1, and then delete the messages from the queue:

```
asnqmfmt ADMNQ1 QMGR1 -delmsg -mqmd
```

Example 5

To view the first 100 messages that are on receive queue Q2 that is defined in the queue manager QMGR2 in a file called Q2_messages that is stored in the C:\qrepl directory (Windows):

```
asnqmfmt Q2 QMGR2 -l 100 -o C:\qrepl\Q2_messages
```

Related concepts:

- "System commands for Q replication and event publishing—Overview" on page 313

Related reference:

- "Road map: Q replication and event publishing system commands" on page 313

Chapter 26. Control tables for Q replication and event publishing

Control tables for Q replication and event publishing—Overview

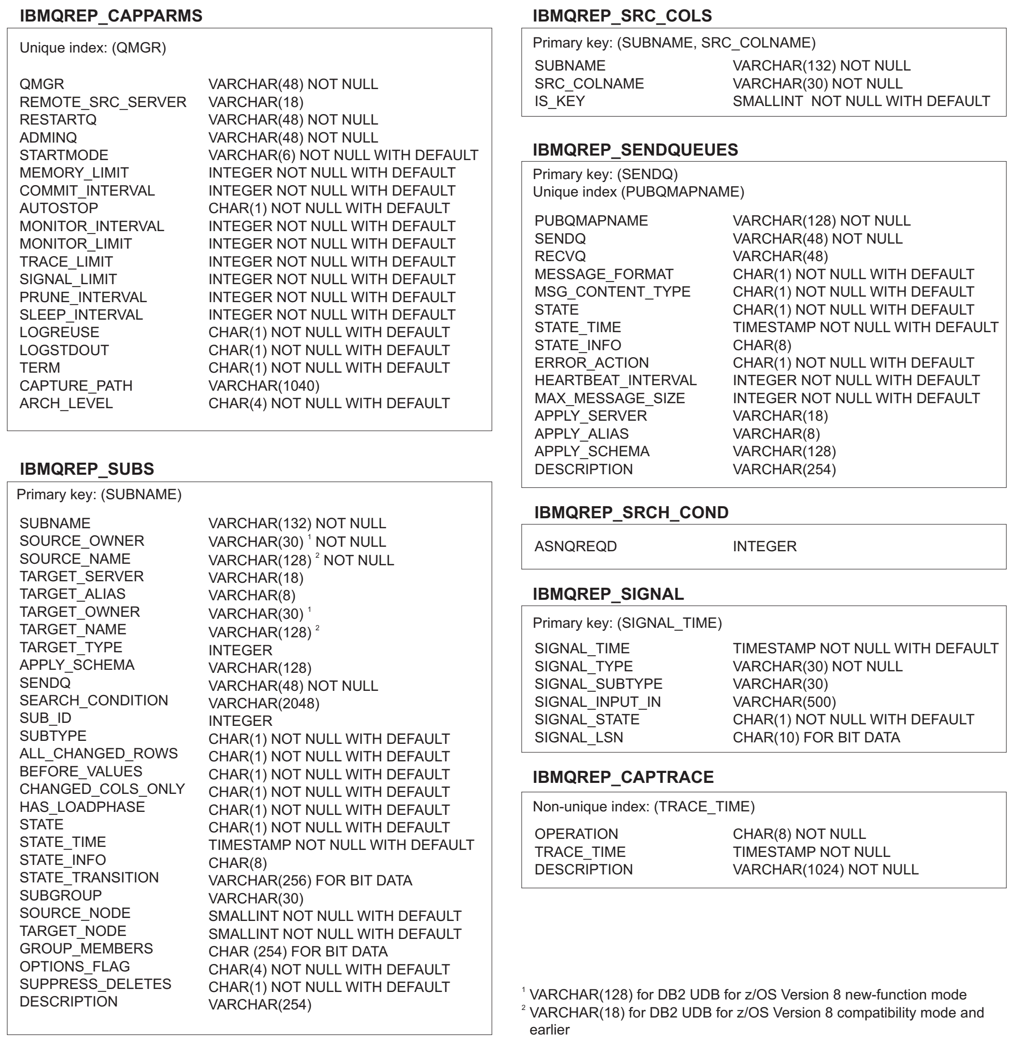
Control tables are relational database tables that are used to store information for the Q replication and event publishing programs. These control tables are stored at the Q Capture server, Q Apply server, and Monitor control server. There are three ways for you to reference the tables:

- A quick reference that includes a list of tables for each server, the columns in each table, and the indexes on each table. See "Control tables at a glance."
- An overview of the tables at each server:
  - "List of control tables at the Q Capture server" on page 367
  - "List of control tables at the Q Apply server" on page 368
  - "List of tables at the Monitor control server" on page 369
- More detailed descriptions about the tables and their columns:
  - "Detailed structure of the Q Capture control tables—Overview" on page 370
  - "Detailed structure of the Q Apply control tables—Overview" on page 391
  - "Detailed structure of the Monitor control tables—Overview" on page 414
  - "Detailed structure of versioning columns for peer-to-peer replication" on page 429

Control tables at a glance

Figure 17 on page 362 and Figure 18 on page 363 show the control tables at the Q Capture server, the columns in each table, and the indexes and keys for each table. Figure 19 on page 364 and Figure 20 on page 365 show the tables at the Q Apply server, the columns in each table, and the indexes and keys for each table. Figure 21 on page 366 and Figure 22 on page 367 show the tables at the Monitor control server, the columns in each table, and the indexes on each table. Figure 23 on page 367 shows the versioning columns that are added to customer tables by the replication administration tools for peer-to-peer replication.

Control tables used at the Q Capture server (image 1 of 2)



IBMQREP_CAPPARMS

Unique index: (QMGR)

QMGR	VARCHAR(48) NOT NULL
REMOTE_SRC_SERVER	VARCHAR(18)
RESTARTQ	VARCHAR(48) NOT NULL
ADMINQ	VARCHAR(48) NOT NULL
STARTMODE	VARCHAR(6) NOT NULL WITH DEFAULT
MEMORY_LIMIT	INTEGER NOT NULL WITH DEFAULT
COMMIT_INTERVAL	INTEGER NOT NULL WITH DEFAULT
AUTOSTOP	CHAR(1) NOT NULL WITH DEFAULT
MONITOR_INTERVAL	INTEGER NOT NULL WITH DEFAULT
MONITOR_LIMIT	INTEGER NOT NULL WITH DEFAULT
TRACE_LIMIT	INTEGER NOT NULL WITH DEFAULT
SIGNAL_LIMIT	INTEGER NOT NULL WITH DEFAULT
PRUNE_INTERVAL	INTEGER NOT NULL WITH DEFAULT
SLEEP_INTERVAL	INTEGER NOT NULL WITH DEFAULT
LOGREUSE	CHAR(1) NOT NULL WITH DEFAULT
LOGSTDOUT	CHAR(1) NOT NULL WITH DEFAULT
TERM	CHAR(1) NOT NULL WITH DEFAULT
CAPTURE_PATH	VARCHAR(1040)
ARCH_LEVEL	CHAR(4) NOT NULL WITH DEFAULT

IBMQREP_SUBS

Primary key: (SUBNAME)

SUBNAME	VARCHAR(132) NOT NULL
SOURCE_OWNER	VARCHAR(30) [1] NOT NULL
SOURCE_NAME	VARCHAR(128) [2] NOT NULL
TARGET_SERVER	VARCHAR(18)
TARGET_ALIAS	VARCHAR(8)
TARGET_OWNER	VARCHAR(30) [1]
TARGET_NAME	VARCHAR(128) [2]
TARGET_TYPE	INTEGER
APPLY_SCHEMA	VARCHAR(128)
SENDQ	VARCHAR(48) NOT NULL
SEARCH_CONDITION	VARCHAR(2048)
SUB_ID	INTEGER
SUBTYPE	CHAR(1) NOT NULL WITH DEFAULT
ALL_CHANGED_ROWS	CHAR(1) NOT NULL WITH DEFAULT
BEFORE_VALUES	CHAR(1) NOT NULL WITH DEFAULT
CHANGED_COLS_ONLY	CHAR(1) NOT NULL WITH DEFAULT
HAS_LOADPHASE	CHAR(1) NOT NULL WITH DEFAULT
STATE	CHAR(1) NOT NULL WITH DEFAULT
STATE_TIME	TIMESTAMP NOT NULL WITH DEFAULT
STATE_INFO	CHAR(8)
STATE_TRANSITION	VARCHAR(256) FOR BIT DATA
SUBGROUP	VARCHAR(30)
SOURCE_NODE	SMALLINT NOT NULL WITH DEFAULT
TARGET_NODE	SMALLINT NOT NULL WITH DEFAULT
GROUP_MEMBERS	CHAR (254) FOR BIT DATA
OPTIONS_FLAG	CHAR(4) NOT NULL WITH DEFAULT
SUPPRESS_DELETES	CHAR(1) NOT NULL WITH DEFAULT
DESCRIPTION	VARCHAR(254)

IBMQREP_SRC_COLS

Primary key: (SUBNAME, SRC_COLNAME)

SUBNAME	VARCHAR(132) NOT NULL
SRC_COLNAME	VARCHAR(30) NOT NULL
IS_KEY	SMALLINT NOT NULL WITH DEFAULT

IBMQREP_SENDQUEUES

Primary key: (SENDQ)
Unique index (PUBQMAPNAME)

PUBQMAPNAME	VARCHAR(128) NOT NULL
SENDQ	VARCHAR(48) NOT NULL
RECVQ	VARCHAR(48)
MESSAGE_FORMAT	CHAR(1) NOT NULL WITH DEFAULT
MSG_CONTENT_TYPE	CHAR(1) NOT NULL WITH DEFAULT
STATE	CHAR(1) NOT NULL WITH DEFAULT
STATE_TIME	TIMESTAMP NOT NULL WITH DEFAULT
STATE_INFO	CHAR(8)
ERROR_ACTION	CHAR(1) NOT NULL WITH DEFAULT
HEARTBEAT_INTERVAL	INTEGER NOT NULL WITH DEFAULT
MAX_MESSAGE_SIZE	INTEGER NOT NULL WITH DEFAULT
APPLY_SERVER	VARCHAR(18)
APPLY_ALIAS	VARCHAR(8)
APPLY_SCHEMA	VARCHAR(128)
DESCRIPTION	VARCHAR(254)

IBMQREP_SRCH_COND

ASNQREQD	INTEGER

IBMQREP_SIGNAL

Primary key: (SIGNAL_TIME)

SIGNAL_TIME	TIMESTAMP NOT NULL WITH DEFAULT
SIGNAL_TYPE	VARCHAR(30) NOT NULL
SIGNAL_SUBTYPE	VARCHAR(30)
SIGNAL_INPUT_IN	VARCHAR(500)
SIGNAL_STATE	CHAR(1) NOT NULL WITH DEFAULT
SIGNAL_LSN	CHAR(10) FOR BIT DATA

IBMQREP_CAPTRACE

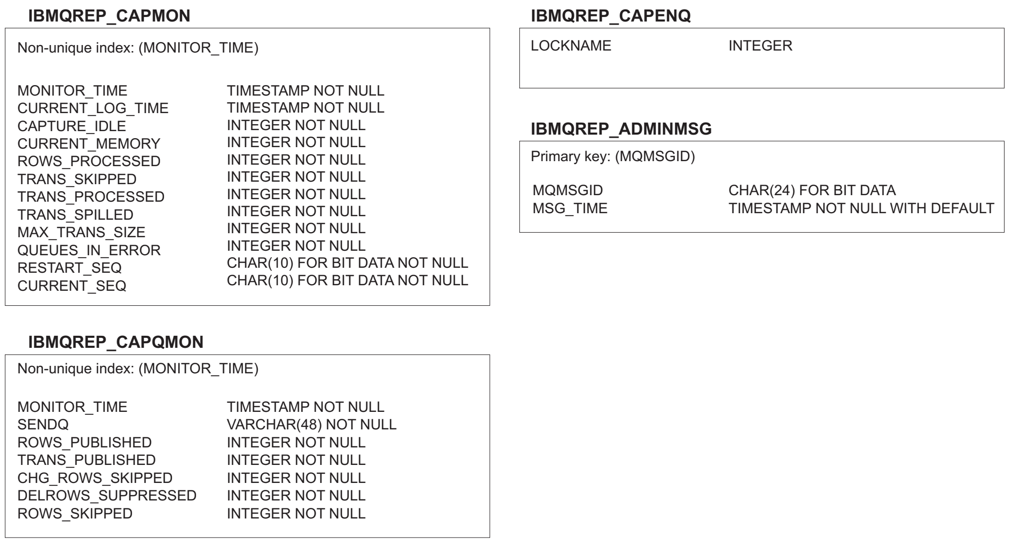
Non-unique index: (TRACE_TIME)

OPERATION	CHAR(8) NOT NULL
TRACE_TIME	TIMESTAMP NOT NULL
DESCRIPTION	VARCHAR(1024) NOT NULL

[1] VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode
[2] VARCHAR(18) for DB2 UDB for z/OS Version 8 compatibility mode and earlier

Figure 17. Tables used at the Q Capture server. These tables are used by the Q Capture program at the Q Capture server. The columns that make up each table's index and key are listed in parentheses under the table name.

Control tables used at the Q Capture server (image 2 of 2)



IBMQREP_CAPMON

Non-unique index: (MONITOR_TIME)

MONITOR_TIME	TIMESTAMP NOT NULL
CURRENT_LOG_TIME	TIMESTAMP NOT NULL
CAPTURE_IDLE	INTEGER NOT NULL
CURRENT_MEMORY	INTEGER NOT NULL
ROWS_PROCESSED	INTEGER NOT NULL
TRANS_SKIPPED	INTEGER NOT NULL
TRANS_PROCESSED	INTEGER NOT NULL
TRANS_SPILLED	INTEGER NOT NULL
MAX_TRANS_SIZE	INTEGER NOT NULL
QUEUES_IN_ERROR	INTEGER NOT NULL
RESTART_SEQ	CHAR(10) FOR BIT DATA NOT NULL
CURRENT_SEQ	CHAR(10) FOR BIT DATA NOT NULL

IBMQREP_CAPENQ

LOCKNAME	INTEGER

IBMQREP_ADMINMSG

Primary key: (MQMSGID)

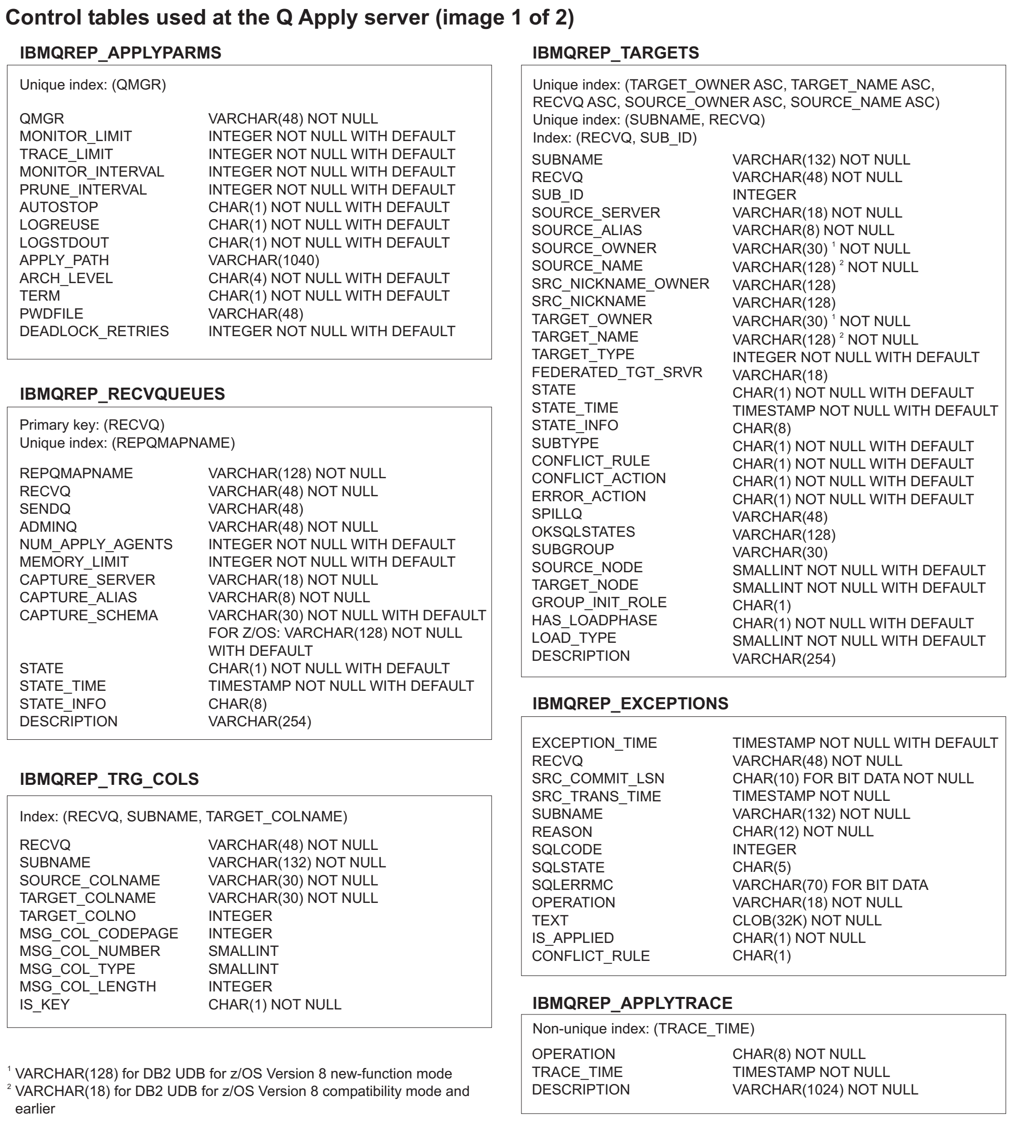
MQMSGID	CHAR(24) FOR BIT DATA
MSG_TIME	TIMESTAMP NOT NULL WITH DEFAULT

IBMQREP_CAPQMON

Non-unique index: (MONITOR_TIME)

MONITOR_TIME	TIMESTAMP NOT NULL
SENDQ	VARCHAR(48) NOT NULL
ROWS_PUBLISHED	INTEGER NOT NULL
TRANS_PUBLISHED	INTEGER NOT NULL
CHG_ROWS_SKIPPED	INTEGER NOT NULL
DELROWS_SUPPRESSED	INTEGER NOT NULL
ROWS_SKIPPED	INTEGER NOT NULL

Figure 18. Tables used at the Q Capture server (continued). These tables are used by the Q Capture program at the Q Capture server. The columns that make up each table's index and key are listed in parentheses under the table name.



Control tables used at the Q Apply server (image 1 of 2)

IBMQREP_APPLYPARMS

Unique index: (QMGR)

QMGR	VARCHAR(48) NOT NULL
MONITOR_LIMIT	INTEGER NOT NULL WITH DEFAULT
TRACE_LIMIT	INTEGER NOT NULL WITH DEFAULT
MONITOR_INTERVAL	INTEGER NOT NULL WITH DEFAULT
PRUNE_INTERVAL	INTEGER NOT NULL WITH DEFAULT
AUTOSTOP	CHAR(1) NOT NULL WITH DEFAULT
LOGREUSE	CHAR(1) NOT NULL WITH DEFAULT
LOGSTDOUT	CHAR(1) NOT NULL WITH DEFAULT
APPLY_PATH	VARCHAR(1040)
ARCH_LEVEL	CHAR(4) NOT NULL WITH DEFAULT
TERM	CHAR(1) NOT NULL WITH DEFAULT
PWDFILE	VARCHAR(48)
DEADLOCK_RETRIES	INTEGER NOT NULL WITH DEFAULT

IBMQREP_RECVQUEUES

Primary key: (RECVQ)
Unique index: (REPQMAPNAME)

REPQMAPNAME	VARCHAR(128) NOT NULL
RECVQ	VARCHAR(48) NOT NULL
SENDQ	VARCHAR(48)
ADMINQ	VARCHAR(48) NOT NULL
NUM_APPLY_AGENTS	INTEGER NOT NULL WITH DEFAULT
MEMORY_LIMIT	INTEGER NOT NULL WITH DEFAULT
CAPTURE_SERVER	VARCHAR(18) NOT NULL
CAPTURE_ALIAS	VARCHAR(8) NOT NULL
CAPTURE_SCHEMA	VARCHAR(30) NOT NULL WITH DEFAULT FOR Z/OS: VARCHAR(128) NOT NULL WITH DEFAULT
STATE	CHAR(1) NOT NULL WITH DEFAULT
STATE_TIME	TIMESTAMP NOT NULL WITH DEFAULT
STATE_INFO	CHAR(8)
DESCRIPTION	VARCHAR(254)

IBMQREP_TRG_COLS

Index: (RECVQ, SUBNAME, TARGET_COLNAME)

RECVQ	VARCHAR(48) NOT NULL
SUBNAME	VARCHAR(132) NOT NULL
SOURCE_COLNAME	VARCHAR(30) NOT NULL
TARGET_COLNAME	VARCHAR(30) NOT NULL
TARGET_COLNO	INTEGER
MSG_COL_CODEPAGE	INTEGER
MSG_COL_NUMBER	SMALLINT
MSG_COL_TYPE	SMALLINT
MSG_COL_LENGTH	INTEGER
IS_KEY	CHAR(1) NOT NULL

[1] VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode
[2] VARCHAR(18) for DB2 UDB for z/OS Version 8 compatibility mode and earlier

IBMQREP_TARGETS

Unique index: (TARGET_OWNER ASC, TARGET_NAME ASC, RECVQ ASC, SOURCE_OWNER ASC, SOURCE_NAME ASC)
Unique index: (SUBNAME, RECVQ)
Index: (RECVQ, SUB_ID)

SUBNAME	VARCHAR(132) NOT NULL
RECVQ	VARCHAR(48) NOT NULL
SUB_ID	INTEGER
SOURCE_SERVER	VARCHAR(18) NOT NULL
SOURCE_ALIAS	VARCHAR(8) NOT NULL
SOURCE_OWNER	VARCHAR(30) [1] NOT NULL
SOURCE_NAME	VARCHAR(128) [2] NOT NULL
SRC_NICKNAME_OWNER	VARCHAR(128)
SRC_NICKNAME	VARCHAR(128)
TARGET_OWNER	VARCHAR(30) [1] NOT NULL
TARGET_NAME	VARCHAR(128) [2] NOT NULL
TARGET_TYPE	INTEGER NOT NULL WITH DEFAULT
FEDERATED_TGT_SRVR	VARCHAR(18)
STATE	CHAR(1) NOT NULL WITH DEFAULT
STATE_TIME	TIMESTAMP NOT NULL WITH DEFAULT
STATE_INFO	CHAR(8)
SUBTYPE	CHAR(1) NOT NULL WITH DEFAULT
CONFLICT_RULE	CHAR(1) NOT NULL WITH DEFAULT
CONFLICT_ACTION	CHAR(1) NOT NULL WITH DEFAULT
ERROR_ACTION	CHAR(1) NOT NULL WITH DEFAULT
SPILLQ	VARCHAR(48)
OKSQLSTATES	VARCHAR(128)
SUBGROUP	VARCHAR(30)
SOURCE_NODE	SMALLINT NOT NULL WITH DEFAULT
TARGET_NODE	SMALLINT NOT NULL WITH DEFAULT
GROUP_INIT_ROLE	CHAR(1)
HAS_LOADPHASE	CHAR(1) NOT NULL WITH DEFAULT
LOAD_TYPE	SMALLINT NOT NULL WITH DEFAULT
DESCRIPTION	VARCHAR(254)

IBMQREP_EXCEPTIONS

EXCEPTION_TIME	TIMESTAMP NOT NULL WITH DEFAULT
RECVQ	VARCHAR(48) NOT NULL
SRC_COMMIT_LSN	CHAR(10) FOR BIT DATA NOT NULL
SRC_TRANS_TIME	TIMESTAMP NOT NULL
SUBNAME	VARCHAR(132) NOT NULL
REASON	CHAR(12) NOT NULL
SQLCODE	INTEGER
SQLSTATE	CHAR(5)
SQLERRMC	VARCHAR(70) FOR BIT DATA
OPERATION	VARCHAR(18) NOT NULL
TEXT	CLOB(32K) NOT NULL
IS_APPLIED	CHAR(1) NOT NULL
CONFLICT_RULE	CHAR(1)

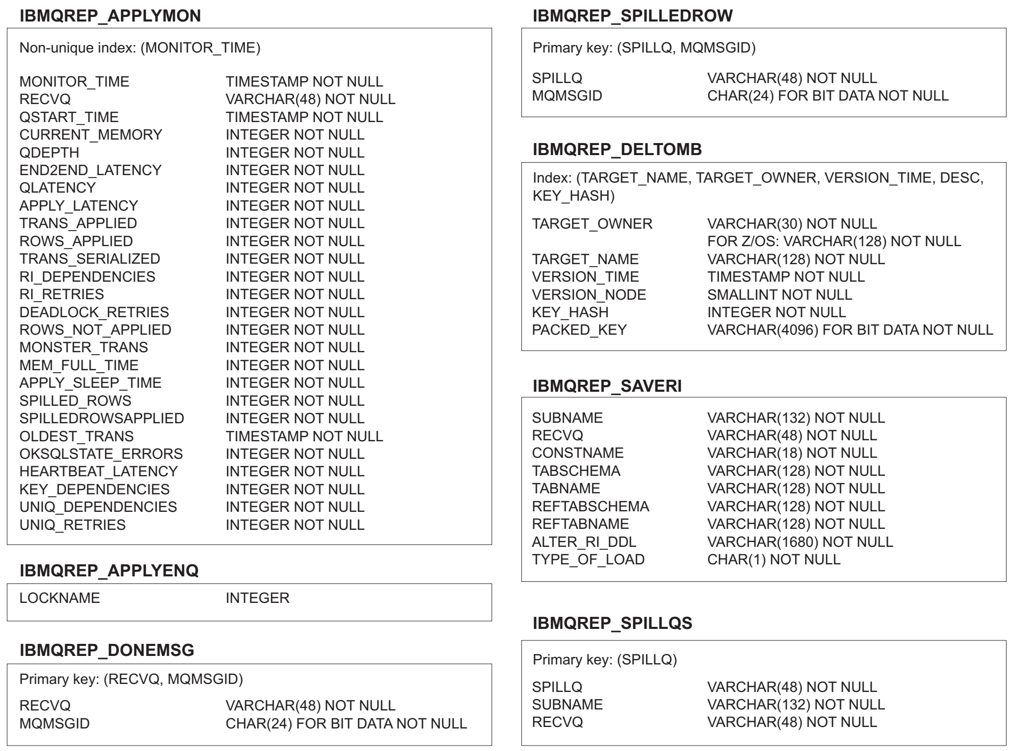
IBMQREP_APPLYTRACE

Non-unique index: (TRACE_TIME)

OPERATION	CHAR(8) NOT NULL
TRACE_TIME	TIMESTAMP NOT NULL
DESCRIPTION	VARCHAR(1024) NOT NULL

Figure 19. Tables used at the Q Apply server. These tables are used by the Q Apply program at the Q Apply server. The columns that make up each table's index and key are listed in parentheses under the table name.

Control tables used at the Q Apply server (image 2 of 2)



IBMQREP_APPLYMON

Non-unique index: (MONITOR_TIME)

MONITOR_TIME	TIMESTAMP NOT NULL
RECVQ	VARCHAR(48) NOT NULL
QSTART_TIME	TIMESTAMP NOT NULL
CURRENT_MEMORY	INTEGER NOT NULL
QDEPTH	INTEGER NOT NULL
END2END_LATENCY	INTEGER NOT NULL
QLATENCY	INTEGER NOT NULL
APPLY_LATENCY	INTEGER NOT NULL
TRANS_APPLIED	INTEGER NOT NULL
ROWS_APPLIED	INTEGER NOT NULL
TRANS_SERIALIZED	INTEGER NOT NULL
RI_DEPENDENCIES	INTEGER NOT NULL
RI_RETRIES	INTEGER NOT NULL
DEADLOCK_RETRIES	INTEGER NOT NULL
ROWS_NOT_APPLIED	INTEGER NOT NULL
MONSTER_TRANS	INTEGER NOT NULL
MEM_FULL_TIME	INTEGER NOT NULL
APPLY_SLEEP_TIME	INTEGER NOT NULL
SPILLED_ROWS	INTEGER NOT NULL
SPILLEDROWSAPPLIED	INTEGER NOT NULL
OLDEST_TRANS	TIMESTAMP NOT NULL
OKSQLSTATE_ERRORS	INTEGER NOT NULL
HEARTBEAT_LATENCY	INTEGER NOT NULL
KEY_DEPENDENCIES	INTEGER NOT NULL
UNIQ_DEPENDENCIES	INTEGER NOT NULL
UNIQ_RETRIES	INTEGER NOT NULL

IBMQREP_APPLYENQ

LOCKNAME	INTEGER

IBMQREP_DONEMSG

Primary key: (RECVQ, MQMSGID)

RECVQ	VARCHAR(48) NOT NULL
MQMSGID	CHAR(24) FOR BIT DATA NOT NULL

IBMQREP_SPILLEDROW

Primary key: (SPILLQ, MQMSGID)

SPILLQ	VARCHAR(48) NOT NULL
MQMSGID	CHAR(24) FOR BIT DATA NOT NULL

IBMQREP_DELTOMB

Index: (TARGET_NAME, TARGET_OWNER, VERSION_TIME, DESC, KEY_HASH)

TARGET_OWNER	VARCHAR(30) NOT NULL FOR Z/OS: VARCHAR(128) NOT NULL
TARGET_NAME	VARCHAR(128) NOT NULL
VERSION_TIME	TIMESTAMP NOT NULL
VERSION_NODE	SMALLINT NOT NULL
KEY_HASH	INTEGER NOT NULL
PACKED_KEY	VARCHAR(4096) FOR BIT DATA NOT NULL

IBMQREP_SAVERI

SUBNAME	VARCHAR(132) NOT NULL
RECVQ	VARCHAR(48) NOT NULL
CONSTNAME	VARCHAR(18) NOT NULL
TABSCHEMA	VARCHAR(128) NOT NULL
TABNAME	VARCHAR(128) NOT NULL
REFTABSCHEMA	VARCHAR(128) NOT NULL
REFTABNAME	VARCHAR(128) NOT NULL
ALTER_RI_DDL	VARCHAR(1680) NOT NULL
TYPE_OF_LOAD	CHAR(1) NOT NULL

IBMQREP_SPILLQS

Primary key: (SPILLQ)

SPILLQ	VARCHAR(48) NOT NULL
SUBNAME	VARCHAR(132) NOT NULL
RECVQ	VARCHAR(48) NOT NULL

Figure 20. Tables used at the Q Apply server (continued). These tables are used by the Q Apply program at the Q Apply server. The columns that make up each table's index and key are listed in parentheses under the table name.

Control tables used at the Monitor control server (image 1 of 2)

ASN.IBMSNAP_ALERTS

(MONITOR_QUAL, COMPONENT, SERVER_NAME, SCHEMA_OR_QUAL, SET_NAME, CONDITION_NAME, ALERT_CODE)

MONITOR_QUAL	CHAR(18) NOT NULL
ALERT_TIME	TIMESTAMP NOT NULL
COMPONENT	CHAR(1) NOT NULL
SERVER_NAME	CHAR(18) NOT NULL
SERVER_ALIAS	CHAR(8)
SCHEMA_OR_QUAL	VARCHAR(30)[1] NOT NULL
SET_NAME	CHAR(18) NOT NULL WITH DEFAULT
CONDITION_NAME	CHAR(18) NOT NULL
OCCURRED_TIME	TIMESTAMP NOT NULL
ALERT_COUNTER	SMALLINT NOT NULL
ALERT_CODE	CHAR(10) NOT NULL
RETURN_CODE	INT NOT NULL
NOTIFICATION_SENT	CHAR(1) NOT NULL
ALERT_MESSAGE	VARCHAR(1024) NOT NULL

ASN.IBMSNAP_CONDITIONS

(MONITOR_QUAL, SERVER_NAME, COMPONENT, SCHEMA_OR_QUAL, SET_NAME, CONDITION_NAME)

MONITOR_QUAL	CHAR(18) NOT NULL
SERVER_NAME	CHAR(18) NOT NULL
COMPONENT	CHAR(1) NOT NULL
SCHEMA_OR_QUAL	VARCHAR(30)[1] NOT NULL
SET_NAME	CHAR(18) NOT NULL WITH DEFAULT
SERVER_ALIAS	CHAR(8)
ENABLED	CHAR(1) NOT NULL
CONDITION_NAME	CHAR(18) NOT NULL
PARM_INT	INT
PARM_CHAR	VARCHAR(128)
CONTACT_TYPE	CHAR(1) NOT NULL
CONTACT	VARCHAR(127) NOT NULL

ASN.IBMSNAP_CONTACTGRP

(GROUP_NAME, CONTACT_NAME)

GROUP_NAME	VARCHAR(127) NOT NULL
CONTACT_NAME	VARCHAR(127) NOT NULL

ASN.IBMSNAP_CONTACTS

(CONTACT_NAME)

CONTACT_NAME	VARCHAR(127) NOT NULL
EMAIL_ADDRESS	VARCHAR(128) NOT NULL
ADDRESS_TYPE	CHAR(1) NOT NULL
DELEGATE	VARCHAR(127)
DELEGATE_START	DATE
DELEGATE_END	DATE
DESCRIPTION	VARCHAR(1024)

ASN.IBMSNAP_GROUPS

(GROUP_NAME)

GROUP_NAME	VARCHAR(127) NOT NULL
DESCRIPTION	VARCHAR(1024)

ASN.IBMSNAP_MONENQ

(no index)

MONITOR_QUAL	CHAR(18) NOT NULL

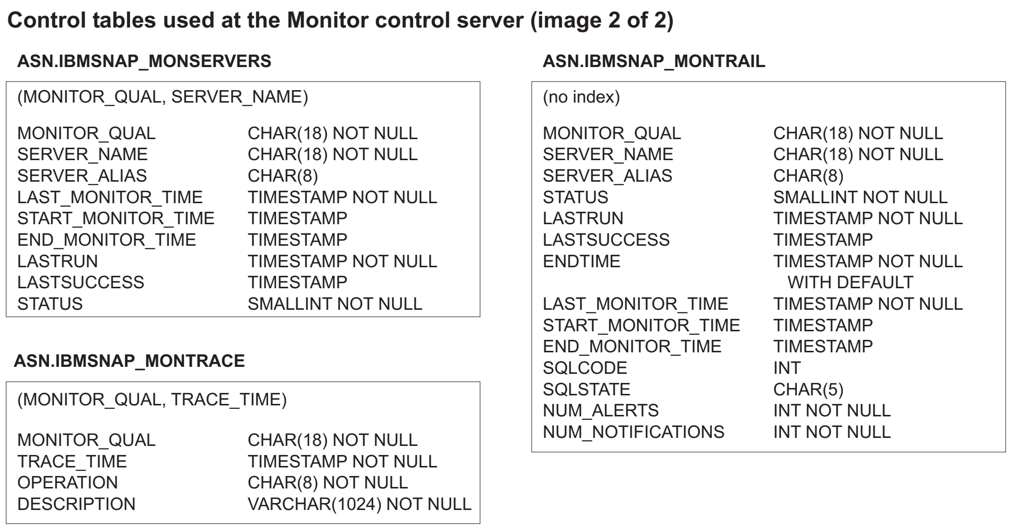
ASN.IBMSNAP_MONPARMS

(MONITOR_QUAL)

MONITOR_QUAL	CHAR(18) NOT NULL
ALERT_PRUNE_LIMIT	INT WITH DEFAULT
AUTOPRUNE	CHAR(1) WITH DEFAULT
EMAIL_SERVER	VARCHAR(128)
LOGREUSE	CHAR(1) WITH DEFAULT
LOGSTDOUT	CHAR(1) WITH DEFAULT
NOTIF_PER_ALERT	INT WITH DEFAULT
NOTIF_MINUTES	INT WITH DEFAULT
MONITOR_ERRORS	VARCHAR(128)
MONITOR_INTERVAL	INT WITH DEFAULT
MONITOR_PATH	VARCHAR(1040)
RUNONCE	CHAR(1) WITH DEFAULT
TERM	CHAR(1)WITH DEFAULT
TRACE_LIMIT	INT WITH DEFAULT

[1] VARCHAR(30) for DB2 for z/OS V8 compatibility mode or earlier; VARCHAR(128) for DB2 for z/OS V8 new-function mode.

Figure 21. Tables used at the Monitor control server. These tables are used by the Replication Alert Monitor program at the Monitor control server. The columns that make up each table's main index are listed in parentheses under the table name.



Control tables used at the Monitor control server (image 2 of 2)

ASN.IBMSNAP_MONSERVERS

(MONITOR_QUAL, SERVER_NAME)

MONITOR_QUAL	CHAR(18) NOT NULL
SERVER_NAME	CHAR(18) NOT NULL
SERVER_ALIAS	CHAR(8)
LAST_MONITOR_TIME	TIMESTAMP NOT NULL
START_MONITOR_TIME	TIMESTAMP
END_MONITOR_TIME	TIMESTAMP
LASTRUN	TIMESTAMP NOT NULL
LASTSUCCESS	TIMESTAMP
STATUS	SMALLINT NOT NULL

ASN.IBMSNAP_MONTRACE

(MONITOR_QUAL, TRACE_TIME)

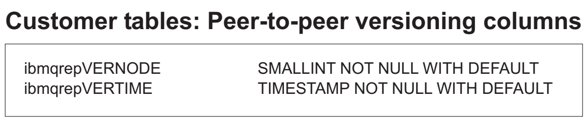
MONITOR_QUAL	CHAR(18) NOT NULL
TRACE_TIME	TIMESTAMP NOT NULL
OPERATION	CHAR(8) NOT NULL
DESCRIPTION	VARCHAR(1024) NOT NULL

ASN.IBMSNAP_MONTRAIL

(no index)

MONITOR_QUAL	CHAR(18) NOT NULL
SERVER_NAME	CHAR(18) NOT NULL
SERVER_ALIAS	CHAR(8)
STATUS	SMALLINT NOT NULL
LASTRUN	TIMESTAMP NOT NULL
LASTSUCCESS	TIMESTAMP
ENDTIME	TIMESTAMP NOT NULL WITH DEFAULT
LAST_MONITOR_TIME	TIMESTAMP NOT NULL
START_MONITOR_TIME	TIMESTAMP
END_MONITOR_TIME	TIMESTAMP
SQLCODE	INT
SQLSTATE	CHAR(5)
NUM_ALERTS	INT NOT NULL
NUM_NOTIFICATIONS	INT NOT NULL

Figure 22. Tables used at the Monitor control server (continued). These tables are used by the Replication Alert Monitor program at the Monitor control server. The columns that make up each table's main index are listed in parentheses under the table name.



Customer tables: Peer-to-peer versioning columns

ibmqrepVERNODE	SMALLINT NOT NULL WITH DEFAULT
ibmqrepVERTIME	TIMESTAMP NOT NULL WITH DEFAULT

Figure 23. Peer-to-peer versioning columns. These columns are added to customer tables by the replication administration tools for peer-to-peer replication.

Related concepts:

- "Control tables for Q replication and event publishing—Overview" on page 361

List of 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 XML 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 Center or SQL scripts.

Table 47 describes the control tables at the Q Capture server.

Table 47. Control tables at the Q Capture server

Table name	Description	See page
IBMQREP_ADMINMSG	An internal table that contains administrative messages received by a Q Capture program.	371
IBMQREP_CAPENQ	Ensures that only one Q Capture program with a given schema is running per Q Capture server.	371
IBMQREP_CAPMON	Contains statistics about the performance of a Q Capture program.	372
IBMQREP_CAPPARMS	Contains parameters that you can specify to control the operations of a Q Capture program.	374
IBMQREP_CAPQMON	Contains statistics about the performance of a Q Capture program for each send queue.	377

Table 47. Control tables at the Q Capture server (continued)

Table name	Description	See page
IBMQREP_CAPTRACE	Contains informational, warning, and error messages from a Q Capture program.	378
IBMQREP_SRCH_COND	An internal table that a Q Capture program uses to evaluate the search condition that you specified for a Q subscription or XML publication.	379
IBMQREP_SENDQUEUES	Contains information about the WebSphere MQ queues that a Q Capture program uses to send transaction, row operation, large object, or informational messages.	380
IBMQREP_SIGNAL	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.	382
IBMQREP_SRC_COLS	Identifies columns in the source table that are replicated or published for a Q subscription or XML publication.	386
IBMQREP_SUBS	Contains information about Q subscriptions and XML publications, including subscription type, source tables, search conditions, data sending options, target loading options, and states.	387

Related concepts:
- "Control tables for Q replication and event publishing—Overview" on page 361

Related tasks:
- "Creating control tables for the Q Capture and Q Apply programs" on page 76

Related reference:
- "List of control tables at the Q Apply server" on page 368
- "List of tables at the Monitor control server" on page 369

List of 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 Center or SQL scripts.

Table 48 describes the control tables at the Q Apply server.

Table 48. Control tables at the Q Apply server

Table name	Description	See page
IBMQREP_APPLYENQ	Ensures that only one Q Apply program with a given schema is running per Q Apply server.	392
IBMQREP_APPLYMON	Contains statistics about the performance of a Q Apply program for each receive queue.	393
IBMQREP_APPLYPARMS	Contains parameters that you can specify to control the operations of a Q Apply program.	395
IBMQREP_APPLYTRACE	Contains informational, warning, and error messages from the Q Apply program.	397

Table 48. Control tables at the Q Apply server (continued)

Table name	Description	See page
IBMQREP_DELTOMB	An internal table used by the Q Apply program to record conflicting deletes in peer-to-peer replication.	398
IBMQREP_DONEMSG	An internal table used by the Q Apply program to record which messages were processed.	399
IBMQREP_EXCEPTIONS	Contains row changes that could not be applied because of conflicts, errors, or rollbacks.	400
IBMQREP_RECVQUEUES	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.	402
IBMQREP_SAVERI	An internal table that the Q Apply program uses to store referential integrity constraints that are dropped while targets are being loaded.	404
IBMQREP_SPILLEDROW	An internal table that the Q Apply program uses to keep track of rows sent to a temporary spill queue.	405
IBMQREP_SPILLQS	Identifies temporary spill queues that will hold changes to source tables before they are applied to targets.	406
IBMQREP_TRG_COLS	Contains information about the mapping between source and target columns.	406
IBMQREP_TARGETS	Contains information about target tables or stored procedures, and options for Q subscriptions.	408

Related concepts:

- "Control tables for Q replication and event publishing—Overview" on page 361

Related tasks:

- "Creating control tables for the Q Capture and Q Apply programs" on page 76

Related reference:

- "List of control tables at the Q Capture server" on page 367
- "List of tables at the Monitor control server" on page 369

List of 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 using the Replication Center. DataPropagator for iSeries does not have Monitor control tables.

Table 49 describes the control tables at the Monitor control server.

Table 49. Control tables at the Monitor control server

Table name	Description	See page
IBMSNAP_ALERTS	Contains a record of all the alerts issued by the Replication Alert Monitor.	414
IBMSNAP_CONDITIONS	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.	416
IBMSNAP_CONTACTGRP	Contains the individual contacts that make up the contact groups.	421

Table 49. Control tables at the Monitor control server (continued)

Table name	Description	See page
IBMSNAP_CONTACTS	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.	427
IBMSNAP_GROUPS	Contains the name and description of each contact group.	422
IBMSNAP_MONENQ	Used to ensure that only one Replication Alert Monitor program is running per Monitor qualifier.	423
IBMSNAP_MONPARMS	Contains parameters that you can modify to control the operations of the Monitor program.	423
IBMSNAP_MONSERVERS	Contains the latest time that a server was monitored by a Replication Alert Monitor program (identified by a Monitor qualifier).	425
IBMSNAP_MONTRACE	Contains important messages from the Monitor program.	426
IBMSNAP_MONTRAIL	Contains important information about each monitor cycle.	423

Related concepts:
- "Control tables for Q replication and event publishing—Overview" on page 361

Related tasks:
- "Creating control tables for the Replication Alert Monitor" on page 263

Related reference:
- "List of control tables at the Q Apply server" on page 368
- "List of control tables at the Q Capture server" on page 367

Detailed structure of the Q Capture control tables

Detailed structure of the Q Capture control tables—Overview

The following topics provide details about each control table at the Q Capture server. The control tables are listed in alphabetical order. The columns within each table are listed in their order of appearance, from left to right.
- "IBMQREP_ADMINMSG table" on page 371
- "IBMQREP_CAPENQ table" on page 371
- "IBMQREP_CAPMON table" on page 372
- "IBMQREP_CAPPARMS table" on page 374
- "IBMQREP_CAPQMON table" on page 377
- "IBMQREP_CAPTRACE table" on page 378
- "IBMQREP_SRCH_COND table" on page 379
- "IBMQREP_SENDQUEUES table" on page 380
- "IBMQREP_SIGNAL table" on page 382
- "IBMQREP_SRC_COLS table" on page 386
- "IBMQREP_SUBS table" on page 387

Related concepts:
- "Control tables for Q replication and event publishing—Overview" on page 361
- "Detailed structure of the Q Apply control tables—Overview" on page 391

- “Detailed structure of the Monitor control tables—Overview” on page 414

IBMQREP_ADMINMSG table

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.

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.

Table 50 provides a brief description of the columns in the IBMQREP_ADMINMSG table.

Table 50. Columns in the IBMQREP_ADMINMSG table

Column name	Description
MQMSGID	**Data type:** CHAR(24) FOR BIT DATA; **Nullable:** No The WebSphere MQ message identifier of the message.
MSG_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Capture server when the message was inserted into this table. Default: Current timestamp.

Related concepts:

- “Detailed structure of the Q Capture control tables—Overview” on page 370

Related reference:

- “List of control tables at the Q Capture server” on page 367

IBMQREP_CAPENQ table

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 UDB for Linux, UNIX, and Windows, only one Q Capture program with a given schema is running per database.
- For non-data-sharing DB2 UDB for z/OS, only one Q Capture program with a given schema is running per subsystem.
- For data-sharing DB2 UDB 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 51 provides a brief description of the column in the IBMQREP_CAPENQ table.

Table 51. Column in the IBMQREP_CAPENQ table

Column name	Description
LOCKNAME	**Data type:** INTEGER; **Nullable:** Yes This column contains no data.

Related concepts:

- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related reference:

- "List of control tables at the Q Capture server" on page 367

IBMQREP_CAPMON 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 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.

Table 52 provides a brief description of the columns in the IBMQREP_CAPMON table.

Table 52. Columns in the IBMQREP_CAPMON table

Column name	Description
MONITOR_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server when the row was inserted into this table.
CURRENT_LOG_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server of the latest DB2 commit seen by the Q Capture log reader. If the Q Capture program has reached the end of the log, this column contains the current timestamp at the Q Capture server.

Table 52. Columns in the IBMQREP_CAPMON table (continued)

Column name	Description
CAPTURE_IDLE	**Data type:** INTEGER; **Nullable:** No The time in seconds that the Q Capture program slept while waiting for new transactions to send after reaching the end of the log.
CURRENT_MEMORY	**Data type:** INTEGER; **Nullable:** No The amount of memory (in megabytes) that the Q Capture program used to reconstruct transactions from the log.
ROWS_PROCESSED	**Data type:** INTEGER; **Nullable:** No The number of rows (individual insert, update, or delete operations) that the Q Capture program read from the log.
TRANS_SKIPPED	**Data type:** INTEGER; **Nullable:** No The number of transactions (containing changed rows) that were not put on queues because the changes were to columns that are not part of a Q subscription or XML publication (the ALL_CHANGED_ROWS parameter in the IBMQREP_SUBS table was set to No, the default).
TRANS_PROCESSED	**Data type:** INTEGER; **Nullable:** No The number of transactions that the Q Capture program processed.
TRANS_SPILLED	**Data type:** INTEGER; **Nullable:** No The number of transactions that the Q Capture program spilled to a file after exceeding the MEMORY_LIMIT threshold.
MAX_TRAN_SIZE	**Data type:** INTEGER; **Nullable:** No The largest transaction, in megabytes, that the Q Capture program processed.
QUEUES_IN_ERROR	**Data type:** INTEGER; **Nullable:** No The number of queues that were not accepting messages.
RESTART_SEQ	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No The logical log sequence number in the DB2 recovery log at which the Q Capture program starts during a warm restart. This value represents the earliest log sequence number that the Q Capture program found for which a commit or abort record has not yet been found.
CURRENT_SEQ	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No The most recent logical log sequence number in the DB2 recovery log that the Q Capture program read.

Related concepts:
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related reference:
- "List of control tables at the Q Capture server" on page 367

IBMQREP_CAPPARMS table

Server: Q Capture server

Default schema: ASN

Unique index: QMGR

This table contains information that you can update by using SQL.

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 to the Q Apply program or a user application. The Q Capture program reads changes to this table only during startup.

Important: If this table has no row, or more than one row, the Q Capture program will not run.

Table 53 provides a brief description of the columns in the IBMQREP_CAPPARMS table.

Table 53. Columns in the IBMQREP_CAPPARMS table

Column name	Description
QMGR	**Data type:** VARCHAR(48); **Nullable:** No The name of the WebSphere MQ queue manager on the source system.
REMOTE_SRC_SERVER	**Data type:** VARCHAR(18); **Nullable:** Yes This column is reserved for future use.
RESTARTQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the queue that stores restart messages for the Q Capture log reader. The name must be unique for each Q Capture program.
ADMINQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the queue that receives control messages from the Q Apply program or a user application. The name must be unique for each Q Capture program.

Table 53. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
STARTMODE	**Data type:** VARCHAR(6); **Nullable:** No, with default The steps that the Q Capture program takes when it starts. **cold** The Q Capture program clears the restart queue and administration queue, and starts processing all Q subscriptions or XML 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. **warmsi** The Q Capture program starts reading the log at the point where it left off, except if this is the first time you are starting it. In that case the Q Capture program switches to a cold start. The warmsi start mode ensures that the Q Capture program cold starts only when it initially starts. **warmns** The Q Capture program starts reading the log at the point where it left off. If it cannot warm start, it does *not* switch to cold start. The warmns start mode prevents the Q Capture program from cold starting unexpectedly. When the Q Capture program warm starts, it resumes processing where it ended. If errors occur after the Q Capture program starts, the program terminates and leaves all tables intact. **warmsa** If warm start information is available, the Q Capture program starts reading the log at the point where it left off. If the Q Capture program cannot warm start, it switches to a cold start. During warm starts, the Q Capture program will process only Q subscriptions or XML publications that are in not in I (inactive) state.
MEMORY_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The amount of memory, in megabytes, that the Q Capture program can use to build transactions. After this allocation is used, in-memory transactions spill to a file.
COMMIT_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in milliseconds, the Q Capture program issues an MQCMIT call. This call signals the WebSphere MQ queue manager to make data messages and informational messages that have been placed on queues available to the Q Apply program or user applications.
AUTOSTOP	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program stops when it reaches the end of the active DB2 log. **Y** The Q Capture program stops as soon as it reaches the end of the active DB2 log. **N** The Q Capture program continues running when it reaches the end of the active DB2 log.
MONITOR_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Capture program adds rows to the IBMQREP_CAPMON and IBMQREP_CAPQMON tables.

Table 53. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
MONITOR_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_CAPMON and the IBMQREP_CAPQMON tables before they are eligible for pruning. At each pruning interval, rows in these tables are pruned if they are older than this limit based on the current timestamp.
TRACE_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_CAPTRACE table before they can be pruned. At each pruning interval, rows in the IBMQREP_CAPTRACE table are pruned if they are older than this limit based on the current timestamp.
SIGNAL_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_SIGNAL table before they can be pruned. At each pruning interval, rows in the IBMQREP_SIGNAL table are pruned if they are older than this limit based on the current timestamp.
PRUNE_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Capture program automatically prunes rows in the IBMQREP_CAPMON, IBMQREP_CAPQMON, IBMQREP_SIGNAL, and IBMQREP_CAPTRACE tables that are no longer needed.
SLEEP_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default The number of seconds that the Q Capture program is idle after processing the active log and any transactions that remain in memory.
LOGREUSE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program reuses the Q Capture log file or appends to it. **Y** On restart, the Q Capture program reuses its log file by clearing the file then writing to the blank file. **N** The Q Capture program appends new information to an existing Q Capture log file when it restarts.
LOGSTDOUT	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program sends log messages to outputs other than its log file. **Y** The Q Capture program sends log messages to both the log file and console (stdout). **N** The Q Capture program directs most log file messages to the log file only. Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file.
TERM	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program stops if DB2 is quiesced. **Y** The Q Capture program stops if DB2 is quiesced. **N** The Q Capture program keeps running if DB2 is quiesced.

Table 53. Columns in the IBMQREP_CAPPARMS table (continued)

Column name	Description
CAPTURE_PATH	**Data type:** VARCHAR(1040); **Nullable:** Yes The path where files created by the Q Capture program are stored.
ARCH_LEVEL	**Data type:** CHAR(4); **Nullable:** No, with default The architectural level of the definition contained in the row. This column identifies the rules under which a row was created. This level is defined by IBM, and for Version 8.2 the level is 0802.

Related concepts:
- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related reference:
- "Default values for Q Capture operating parameters" on page 205
- "Descriptions of Q Capture parameters" on page 206
- "List of control tables at the Q Capture server" on page 367

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 54 provides a brief description of the columns in the IBMQREP_CAPQMON table.

Table 54. Columns in the IBMQREP_CAPQMON table

Column name	Description
MONITOR_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server when the row was inserted into this table.
SENDQ	**Data type:** VARCHAR(97); **Nullable:** No The name of the send queue that this row of monitor statistics tells you about.
ROWS_PUBLISHED	**Data type:** INTEGER; **Nullable:** No The number of rows (individual insert, update, or delete operations) that the Q Capture program sent via this send queue.

Table 54. Columns in the IBMQREP_CAPQMON table (continued)

Column name	Description
TRANS_PUBLISHED	**Data type:** INTEGER; **Nullable:** No The number of transactions that the Q Capture program sent via 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 XML 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 XML 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 XML publication.

Related concepts:
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Control tables for Q replication and event publishing—Overview" on page 361

Related reference:
- "List of control tables at the Q Capture server" on page 367

IBMQREP_CAPTRACE table

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.

The IBMQREP_CAPTRACE table contains informational, warning, and error messages from the Q Capture program.

Table 55 on page 379 provides a brief description of the columns in the IBMQREP_CAPTRACE table.

Table 55. Columns in the IBMQREP_CAPTRACE table

Column name	Description
OPERATION	**Data type:** CHAR(8); **Nullable:** No The type of message from the Q Capture program: **INFO** Describe actions that the Q Capture program takes. **WARNING** Describe conditions that could cause errors for the Q Capture program. **ERROR** Describe errors encountered by the Q Capture program.
TRACE_TIME	**Data type:** TIMESTAMP; **Nullable:** No The time at the Q Capture server that the message was put on a send queue.
DESCRIPTION	**Data type:** VARCHAR(1024); **Nullable:** No The ASN message ID followed by the message text. This column contains English-only text.

Related concepts:

- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related reference:

- "List of control tables at the Q Capture server" on page 367

IBMQREP_SRCH_COND table

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.

IBMQREP_SRCH_COND is an internal table that is used by the Q Capture program to evaluate the search condition clause for a Q subscription or XML publication.

Table 56 provides a brief description of the column in the IBMQREP_SRCH_COND table.

Table 56. Column in the IBMQREP_SRCH_COND table

Column name	Description
ASNQREQD	**Data type:** CHAR(8); **Nullable:** Yes This column contains no data.

Related concepts:

- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related reference:

- "List of control tables at the Q Capture server" on page 367

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.

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.

Table 57 provides a brief description of the columns in the IBMQREP_SENDQUEUES table.

Table 57. Columns in the IBMQREP_SENDQUEUES table

Column name	Description
PUBQMAPNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the publishing queue map that includes this send queue. For Q subscriptions, this name must match the value of REPQMAPNAME in the IBMQREP_RECVQUEUES table.
SENDQ	**Data type:** VARCHAR(48); **Nullable:** No The unique name for this send queue. The name can stand for the local definition of a remote queue, or for a local queue. Queue names cannot contain blanks.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** Yes The name of the receive queue for this Q subscription. This is a local queue on the Q Apply server. Queue names cannot contain blanks.
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the publishing queue map that contains this send queue.
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.

Table 57. Columns in the IBMQREP_SENDQUEUES table (continued)

Column name	Description
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 DB2 transaction, and information about the transaction. **R** Messages contain a single update, insert, or delete operation, and information about the DB2 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 by the worker thread. The most recently committed transaction has been put on the 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.
ERROR_ACTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Capture program what to do when the send queue is no longer accepting messages. For example, the queue might be full, or the queue manager might have reported a severe error for this queue. **I (default)** The Q Capture program invalidates all Q subscriptions and XML publications for this queue, but continues to put messages on other queues. **S** The Q Capture program stops when an error is detected on this queue.
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. **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*.

Table 57. Columns in the IBMQREP_SENDQUEUES table (continued)

Column name	Description
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 XML 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.

Related concepts:
- "Required settings for WebSphere MQ objects" on page 56
- "Publishing queue maps" on page 12
- "Replication queue maps" on page 6
- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related tasks:
- "Creating publishing queue maps" on page 156
- "Creating replication queue maps" on page 84

Related reference:
- "List of control tables at the Q Capture server" on page 367

IBMQREP_SIGNAL table

Server: Q Capture server

Default schema: ASN

Primary key: SIGNAL_TIME

This table contains information that you can update by using SQL.

The IBMQREP_SIGNAL table allows a user, user application, or the Q Apply program to communicate with a Q Capture program. 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 58 provides a brief description of the columns in the IBMQREP_SIGNAL table.

Table 58. Columns in the IBMQREP_SIGNAL table

Column name	Description
SIGNAL_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default A timestamp that is used to uniquely identify the row. The Q Capture program uses this value to find the correct row in the signal table to indicate when it completed processing the Q Capture signal. Default: Current timestamp
SIGNAL_TYPE	**Data type:** VARCHAR(30); **Nullable:** No A flag that indicates the type of signal that was posted: **CMD** A row that is inserted by the administrative commands, **asnqccmd**, the Replication Center, or another application. See the SIGNAL_SUBTYPE column for a list of the available signal subtypes. **USER** A signal posted by a user. The Q Capture program updates the SIGNAL_LSN column with the log sequence number of when the signal was inserted. The Q Capture program also updates the value in the SIGNAL_STATE column from pending (P) to received (R).
SIGNAL_SUBTYPE	**Data type:** VARCHAR(30); **Nullable:** Yes The type of action that a CMD-type signal is requesting that the Q Capture program perform. **CAPSTART** Start capturing changes for a Q subscription or XML publication. **CAPSTOP** Stop capturing changes for a Q subscription or XML 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 XML 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 58. Columns in the IBMQREP_SIGNAL table (continued)

Column name	Description
SIGNAL_SUBTYPE (Continued)	**ADDCOL** Add one column to an active, unidirectional Q subscription or to an XML publication. **P2PNEW2MEMB** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at a new server. **P2PMEMB2NEW** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at active servers. **P2PMEMB2INIT** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at active servers. **P2PSPOOLING** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at the server that initiated a new subscription. **P2PLOADDONE** An internal signal that is used to initialize a peer-to-peer Q subscription. The signal is inserted at a new server. **P2PSUBSTOP** An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the server that is being deactivated. **P2PSUBSTOPPING** An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the remaining active servers. **P2PREADYTOSTOP** An internal signal that is used to deactivate a peer-to-peer Q subscription. The signal is inserted at the server that is being deactivated. **P2PNORECAPTURE** A signal that is inserted by the Q Apply program to prevent the Q Capture program from recapturing changes. Used in bidirectional replication.

Table 58. Columns in the IBMQREP_SIGNAL table (continued)

Column name	Description
SIGNAL_INPUT_IN	**Data type:** VARCHAR(500); **Nullable:** Yes If the SIGNAL_TYPE=USER, then this column contains user-defined input. If the SIGNAL_TYPE=CMD, then this value depends on the SIGNAL_SUBTYPE value: **CMD + CAPSTART** The Q subscription or XML publication name. **CMD + CAPSTOP** The Q subscription or XML publication name. **CMD + LOADDONE** The Q subscription or XML publication name. **CMD + STOP** NULL (no value is required). **CMD + IGNORETRANS** NULL (no value is required). **CMD + QINERROR** For a user application, the name of the send queue that is in error. For the Q Apply program, the name of the send queue that is in error and the ASN message number and space-separated tokens. **CMD + REINIT_SUB** The Q subscription or XML publication name. **CMD + ADDCOL** The Q subscription or XML publication name and the column name, separated by a semicolon. For example, `QSUB1;COL10`.
SIGNAL_STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates the status of the signal. **P (default)** The signal is pending; the Q Capture program did not receive it yet. **R** The Q Capture program received the signal. **C** The Q Capture program completed processing the signal. **F** The signal failed. For example, the Q Capture program cannot perform a CAPSTART because the Q subscription or XML publication is faulty.
SIGNAL_LSN	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** Yes The logical log sequence number of the log record for the insert into the SIGNAL table.

Related concepts:
- "Options for loading target tables for Q replication—Overview" on page 143
- "Detailed structure of the Q Capture control tables—Overview" on page 370

Related tasks:
- "Changing attributes of XML publications" on page 191
- "Adding columns to existing XML publications" on page 193
- "Changing attributes of Q subscriptions" on page 179
- "Adding columns to existing Q subscriptions" on page 181
- "Activating Q subscriptions or XML publications" on page 221

- “Deactivating Q subscriptions or XML publications” on page 222
- “Stopping a Q Capture program” on page 224

Related reference:
- “List of control tables at the Q Capture server” on page 367

IBMQREP_SRC_COLS table

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.

The IBMQREP_SRC_COLS table lists columns at the source table for which changes are to be captured.

Table 59 provides a brief description of the columns in the IBMQREP_SRC_COLS table.

Table 59. Columns in the IBMQREP_SRC_COLS table

Column name	Description
SUBNAME	**Data type:** VARCHAR(30); **Nullable:** No The name of the Q subscription or XML publication for this source table.
SRC_COLNAME	**Data type:** VARCHAR(30); **Nullable:** No The name of the column in the source table for which changes will be captured.
IS_KEY	**Data type:** SMALLINT; **Nullable:** No, with default A flag that indicates whether the column is part of the key to be used for replication or publishing. Any set of columns that are unique at the source can be used. **0 (default)** The column is not part of the unique key. Its order in the transaction message will be the same as its order in the source table. *n* The column is part of the unique key. In a multiple-column key, the column’s order in the transaction message will be encoded based on the number *n* that you specify. At least one of the columns from the source table should have a value greater than 0 in the IBMQREP_SRC_COLS table or the Q subscription or XML publication will be invalid.

Related concepts:
- “Index or key columns for targets (unidirectional replication)” on page 99
- “Detailed structure of the Q Capture control tables—Overview” on page 370

Related reference:
- “List of control tables at the Q Capture server” on page 367

IBMQREP_SUBS table

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.

The IBMQREP_SUBS table contains information about Q subscriptions or XML publications, including the type of subscription, search conditions, data-sending options, load options, and the subscription state.

Table 60 provides a brief description of the columns in the IBMQREP_SUBS table.

Table 60. Columns in the IBMQREP_SUBS table

Column name	Description
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The Q subscription or XML publication name. For each instance of the Q Capture program, all Q subscription or XML publication names must be unique.
SOURCE_OWNER	**Data type**: VARCHAR(30); VARCHAR(128) for DB2 UDB 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 XML publication.
SOURCE_NAME	**Data type:** VARCHAR(128); VARCHAR(18) for DB2 UDB for z/OS Version 7 and Version 8 compatibility mode;**Nullable:** No The name of the source table for this Q subscription or XML 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 UDB 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.
TARGET_NAME	**Data type:** VARCHAR(128); VARCHAR(18) for DB2 UDB for z/OS Version 7 and Version 8 compatibility mode; **Nullable:** Yes The name of the target table for a Q subscription.

Table 60. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
TARGET_TYPE	**Data type:** INTEGER; **Nullable:** Yes A flag that indicates the type of replication target. **1** User table **2** Reserved for future **3** Reserved for future **4** Reserved for future **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 XML 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 XML 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 an XML publication. **U (default)** Unidirectional replication. This is the value used for XML publications. **B** Bidirectional replication. **P** Peer-to-peer replication.
ALL_CHANGED_ROWS	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Capture program publishes 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.

Table 60. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
BEFORE_VALUES	**Data type:** CHAR(1); **Nullable:** No, with default For an update operation, this flag indicates whether the Q Capture program sends the before values of non-key columns in addition to their after values. For a delete, this flag indicates whether the Q Capture program sends the before values of non-key columns in addition to the before values of the key columns. **N (default)** The Q Capture program does not send before values of nonkey columns that change. If a key column changes, the Q Capture program sends both its before and after values. For delete statements involving key columns, only before values are sent. **Y** When there are changes to nonkey columns in the source table that are part of a Q subscription or XML publication, the Q Capture program sends both before and after values.
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 XML 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 XML publication. **N** The Q Capture program sends all columns in a row that are part of a Q subscription or XML 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 XML 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/IMPORT utility, or EXPORT/LOAD utility, 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 60. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that is inserted by the Q Capture program to indicate the current state of the Q subscription or XML publication. The initial state is new, and the STATE_INFO field is initially set to ASN7024I (new Q subscription or XML publication). **N (default)** The Q subscription or XML publication is new. The Q Capture program will automatically activate this Q subscription or XML publication when the program is started or reinitialized. **I** The Q subscription or XML publication is inactive. The Q Capture program saw a CAPSTOP signal in the log, or an error occurred and the Q subscription or XML publication was deactivated. The Q Capture program stopped sending messages for this Q subscription or XML 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 XML 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 XML publication. **T** An internal state that indicates that the Q Capture program read a CAPSTART signal in the log for this peer-to-peer Q subscription, and the Q subscription is being initialized within the peer-to-peer group. **G** An internal state that indicates that the Q Capture program read a CAPSTOP signal in the log for this peer-to-peer Q subscription, and the 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 XML 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 an XML 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 an XML publication. Default: 0

Table 60. Columns in the IBMQREP_SUBS table (continued)

Column name	Description
TARGET_NODE	**Data type:** SMALLINT; **Nullable:** No, with default An identifying number for the target server in a peer-to-peer Q subscription. This column does not apply for an XML publication. Default: 0
GROUP_MEMBERS	**Data type:** CHAR(254) FOR BIT DATA; **Nullable:** Yes This column is updated by the Q Capture program when members join or leave a peer-to-peer group.
OPTIONS_FLAG	**Data type:** CHAR(4) FOR BIT DATA; **Nullable:** No, with default Reserved for future.
SUPPRESS_DELETES	**Data type:** CHAR(1);**Nullable:** No, with default A flag that tells the Q Capture program whether to send rows that were deleted from the source table: **N (default)** Do not send deleted rows. **Y** Send deleted rows.
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the Q subscription or XML publication.

Related concepts:
- "Search conditions to filter rows in XML publications" on page 164
- "When the Q Capture program publishes a message for XML publications" on page 163
- "Options for including before values in messages for XML publications" on page 169
- "Options for including unchanged columns in messages for XML publications" on page 168
- "Q subscriptions" on page 5
- "Search conditions to filter rows (unidirectional replication)" on page 94
- "How often the Q Capture program sends a message (unidirectional replication)" on page 93
- "Options for loading target tables for Q replication—Overview" on page 143
- "Detailed structure of the Q Capture control tables—Overview" on page 370
- "XML publications" on page 11

Related reference:
- "List of control tables at the Q Capture server" on page 367

Detailed structure of the Q Apply control tables

Detailed structure of the Q Apply control tables—Overview

The following topics provide details about each control table at the Q Apply server. The control tables are listed in alphabetical order. The columns within each table are listed in their order of appearance, from left to right.
- "IBMQREP_APPENQ table" on page 392

- "IBMQREP_APPLYMON table" on page 393
- "IBMQREP_APPLYPARMS table" on page 395
- "IBMQREP_APPLYTRACE table" on page 397
- "IBMQREP_DELTOMB table" on page 398
- "IBMQREP_DONEMSG table" on page 399
- "IBMQREP_EXCEPTIONS table" on page 400
- "IBMQREP_RECVQUEUES table" on page 402
- "IBMQREP_SAVERI table" on page 404
- "IBMQREP_SPILLEDROW table" on page 405
- "IBMQREP_SPILLQS table" on page 406
- "IBMQREP_TRG_COLS table" on page 406
- "IBMQREP_TARGETS table" on page 408

Related concepts:
- "Control tables for Q replication and event publishing—Overview" on page 361
- "Detailed structure of the Q Capture control tables—Overview" on page 370
- "Detailed structure of the Monitor control tables—Overview" on page 414

IBMQREP_APPENQ table

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 UDB for Linux, UNIX, and Windows, only one Q Apply program with a given schema is running per database.
- For non-data-sharing DB2 UDB for z/OS, only one Q Apply program with a given schema is running per subsystem.
- For data-sharing DB2 UDB 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 61 provides a brief description of the column in the IBMQREP_APPENQ table.

Table 61. Column in the IBMQREP_APPENQ table

Column name	Description
LOCKNAME	**Data type:** INTEGER; **Nullable:** Yes This column contains no data.

Related concepts:
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_APPLYMON table

Server: Q Apply 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 Apply program periodically inserts rows in the IBMQREP_APPLYMON table to record performance statistics, one row for each receive queue. The value that you specify for MONITOR_INTERVAL in the IBMQREP_APPLYPARMS table determines how often the Q Apply program inserts these rows. The MONITOR_LIMIT value determines how long rows remain in the monitor table before they are eligible for pruning.

Table 62 provides a brief description of the columns in the IBMQREP_APPLYMON table.

Table 62. Columns in the IBMQREP_APPLYMON table

Column name	Description
MONITOR_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Apply server when the row was inserted into the IBMQREP_APPLYMON table.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that this row of Q Apply performance statistics pertains to.
QSTART_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Apply server when the receive queue was started.
CURRENT_MEMORY	**Data type:** INTEGER; **Nullable:** No The amount of memory in bytes that the Q Apply browser thread used for reading transactions from this queue.
QDEPTH	**Data type:** INTEGER; **Nullable:** No The queue depth (number of messages on the queue).
END2END_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that transactions were committed to the source table and the time that they were committed to the target.
QLATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that the Q Capture program put messages on the send queue and the time that the Q Apply program got them from the receive queue.

Table 62. Columns in the IBMQREP_APPLYMON table (continued)

Column name	Description
APPLY_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that the Q Apply program read transactions from the receive queue and the time that they were committed to the target.
TRANS_APPLIED	**Data type:** INTEGER; **Nullable:** No The total number of transactions from this receive queue that the Q Apply committed to the target.
ROWS_APPLIED	**Data type:** INTEGER; **Nullable:** No The total number of insert, update, and delete operations from this receive queue that the Q Apply program applied to the target.
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 total 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 total number of seconds 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.

Table 62. Columns in the IBMQREP_APPLYMON table (continued)

Column name	Description
SPILLEDROWSAPPLIED	**Data type:** INTEGER; **Nullable:** No The number of spilled rows that were applied to the target.
OLDEST_TRANS	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Apply server when the oldest transaction processed during the monitor interval was committed.
OKSQLSTATE_ERRORS	**Data type:** INTEGER; **Nullable:** No The number of row changes that caused an SQL error that is defined as acceptable in the OKSQLSTATES field of the IBMQREP_TARGETS table. The Q Apply program ignores these errors.
HEARTBEAT_LATENCY	**Data type:** INTEGER; **Nullable:** No The average elapsed milliseconds between the time that `heartbeat` messages were sent by the Q Capture program and the time that they were received by the Q Apply program.
KEY_DEPENDENCIES	**Data type:** INTEGER; **Nullable:** No The total number of replication key constraints that were detected, forcing transactions to be serialized.
UNIQ_DEPENDENCIES	**Data type:** INTEGER; **Nullable:** No The total number of unique index constraints that were detected, forcing transactions to be serialized.
UNIQ_RETRIES	**Data type:** INTEGER; **Nullable:** No The number of times that the Q Apply program tried to re-apply rows that were not applied in parallel because of unique index constraints.

Related concepts:
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_APPLYPARMS table

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 parameters that you can modify to control the operations of the Q Apply program. For example, you can set the number of threads that the Q Apply program uses to apply transactions in parallel.

You can also set how long the Q Apply program retains data in the IBMQREP_APPMON table before pruning. The Q Apply program reads changes to this table only during startup.

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 63 provides a brief description of the columns in the IBMQREP_APPLYPARMS table.

Table 63. Columns in the IBMQREP_APPLYPARMS table

Column name	Description
QMGR	**Data type:** VARCHAR(48); **Nullable:** No The name of the WebSphere MQ queue manager on the Q Apply server.
MONITOR_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_APPLYMON table before they are eligible for pruning. At each pruning interval, rows in the IBMQREP_APPLYMON table are pruned if they are older than this limit based on the current timestamp.
TRACE_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The number of minutes that rows remain in the IBMQREP_APPLYTRACE table before they are eligible for pruning. At each pruning interval, rows in the IBMQREP_APPLYTRACE table are pruned if they are older than this limit based on the current timestamp.
MONITOR_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Apply program adds a row to the IBMQREP_APPLYMON table.
PRUNE_INTERVAL	**Data type:** INTEGER; **Nullable:** No, with default How often, in seconds, the Q Apply program automatically prunes rows in the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables.
AUTOSTOP	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program whether to stop when all receive queues have been emptied once. **Y** The Q Apply program stops when all receive queues have been emptied once. **N** The Q Apply program continues running after all receive queues have been emptied once.
LOGREUSE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Apply program reuses the Q Apply log file or appends to it. **Y** On restart, the Q Apply program reuses its log file by clearing the file then writing to the blank file. **N** The Q Apply program appends new information to an existing Q Apply log file when it restarts.

Table 63. Columns in the IBMQREP_APPLYPARMS table (continued)

Column name	Description
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. **Y** The Q Apply program sends log messages to both the log file and the console (stdout). **N** The Q Apply program directs most log messages to the log file only. Initialization, stop, and subscription activation and deactivation messages go to both the console (stdout) and the log file regardless of the setting for this parameter.
APPLY_PATH	**Data type:** VARCHAR(1040); **Nullable:** Yes 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 architectural level of the definition contained in the row. This column identifies the rules under which a row was created. This level is defined by IBM, and for Version 8.2 is 0802.
TERM	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the Q Apply program stops if DB2 is quiesced. **Y** The Q Apply program stops if DB2 is quiesced. **N** The Q Apply program continues running if DB2 is quiesced.
PWDFILE	**Data type:** CHAR(48); **Nullable:** Yes 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 stored 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. The Q Apply program waits one second between deadlock retries.

Related concepts:
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "Default values for Q Apply operating parameters" on page 229
- "Descriptions of Q Apply parameters" on page 231
- "List of control tables at the Q Apply server" on page 368

IBMQREP_APPLYTRACE 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.

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.

Table 64 provides a brief description of the columns in the IBMQREP_APPLYTRACE table.

Table 64. Columns in the IBMQREP_APPLYTRACE table

Column name	Description
OPERATION	**Data type:** CHAR(8); **Nullable:** No The type of message from the Q Apply program: **INFO** Describe actions that the Q Apply program takes. **WARNING** Describe conditions that could cause errors for the Q Apply program. **ERROR** Describe errors encountered by the Q Apply program.
TRACE_TIME	**Data type:** TIMESTAMP; **Nullable:** No The time at the Q Apply server when the row was inserted into this table.
DESCRIPTION	**Data type:** VARCHAR(1024); **Nullable:** No The ASN message ID followed by the message text. This column contains English-only text.

Related concepts:
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_DELTOMB table

Server: Q Apply server

Default schema: ASN

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.

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.

Table 65 provides a brief description of the columns in the IBMQREP_DELTOMB table.

Table 65. Columns in the IBMQREP_DELTOMB table

Column name	Description
TARGET_OWNER	**Data type**: VARCHAR(30), VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode; **Nullable**: No The owner name of the target table for which the conflicting delete was recorded
TARGET_NAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the table for which the conflicting delete was recorded
VERSION_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp of the conflicting delete at the originating server.
VERSION_NODE	**Data type:** SMALLINT; **Nullable:** No Identifies the server in a peer-to-peer group where the conflicting delete originated.
KEY_HASH	**Data type:** INTEGER; **Nullable:** No The hash value of the key for the conflicting delete.
PACKED_KEY	**Data type:** VARCHAR (4096); **Nullable:** No The packed key value of the conflicting delete.

Related concepts:
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_DONEMSG table

Server: Q Apply server

Default schema: ASN

Primary key: RECVQ, MGMSGID

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

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. 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 66 on page 400 provides a brief description of the columns in the IBMQREP_DONEMSG table.

Table 66. Columns in the IBMQREP_DONEMSG table

Column name	Description
RECVQ	**Data type:** VARCHAR(97); **Nullable:** No The name of the receive queue where the message arrived.
MQMSGID	**Data type:** CHAR(24) FOR BIT DATA; **Nullable:** No The WebSphere MQ message identifier of the message.

Related concepts:

- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:

- "List of control tables at the Q Apply server" on page 368

IBMQREP_EXCEPTIONS table

Server: Q Apply server

Default schema: ASN

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. 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 67 provides a brief description of the columns in the IBMQREP_EXCEPTIONS table.

Table 67. Columns in the IBMQREP_EXCEPTIONS table

Column name	Description
EXCEPTION_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Apply server when the error or conflict occurred. Default: Current timestamp
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue where the transaction message arrived.
SRC_COMMIT_LSN	**Data type:** CHAR(10) FOR BIT DATA; **Nullable:** No The logical log sequence number at the Q Capture server for the transaction.
SRC_TRANS_TIME	**Data type:** TIMESTAMP; **Nullable:** No The timestamp at the Q Capture server for the transaction.
SUBNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the Q subscription that the transaction belonged to.

Table 67. Columns in the IBMQREP_EXCEPTIONS table (continued)

Column name	Description
REASON	**Data type:** CHAR(12); **Nullable:** No A description of the error or conflict that caused the transaction to be logged into the IBMQREP_EXCEPTIONS table. **NOTFOUND** An attempt to delete or update a row that did not exist. **DUPLICATE** An attempt to insert a row that was already present. **CHECKFAILED** The conflict detection rule was to check all values or check changed values, and a nonkey value was not as expected. **SQLERROR** An SQL error occurred, and it was not on the list of acceptable errors in the OKSQLSTATES column of the IBMQREP_TARGETS table. **OKSQLSTATE** An SQL error occurred, and it was on the list of acceptable errors in the OKSQLSTATES column of the IBMQREP_TARGETS table. **P2PDUPKEY** In peer-to-peer replication, a key update failed because a target row with the same key already existed, but was newer. **P2PNOTFOUND** In peer-to-peer replication, a delete or update failed because the target row didn't exist. **P2PVERLOSER** In peer-to-peer replication, a delete or update failed because the target row was newer than the row in the change message.
SQLCODE	**Data type:** INTEGER; **Nullable:** Yes The SQL code returned by DB2 for the transaction.
SQLSTATE	**Data type:** CHAR(5); **Nullable:** Yes The SQL state number returned by DB2 for the transaction.
SQLERRMC	**Data type:** CHAR(70) FOR BIT DATA; **Nullable:** Yes The error message tokens from the SQLCA structure used for executing the transaction.
OPERATION	**Data type:** VARCHAR(18); **Nullable:** No The type of SQL operation that failed. Possible values are INSERT, INSERT(LOAD), DELETE, DELETE(LOAD), UPDATE, UPDATE(LOAD), KEY UPDATE, KEY UPDATE(LOAD).
TEXT	**Data type:** LONG VARCHAR(7800); **Nullable:** No An XML description of the data from the row that caused an error. The message is encoded using the same schema used for an XML publication. The root element is either insertRow, deleteRow, or updateRow. Both before and after values are included when available. The XML document is encoded in the target database codepage. For client applications that are using the IBMQREP_EXCEPTIONS table, the encoding XML header attribute of the document will indicate the target database codepage, but the document will be in the client application's codepage.

Table 67. Columns in the IBMQREP_EXCEPTIONS table (continued)

Column name	Description
IS_APPLIED	**Data type:** CHAR(1); **Nullable:** No A flag that indicates whether the row was applied to the target table even though it was entered into the IBMQREP_EXCEPTIONS table. **Y** The row was applied because the CONFLICT_ACTION specified for the Q subscription was F (force). **N** The transaction was not applied.
CONFLICT_RULE	**Data type:** CHAR(1); **Nullable:** Yes The type of conflict detection that resulted in the row being entered in the IBMQREP_EXCEPTIONS table. **K** Only key values were checked. **C** Changed nonkey values as well as key values were checked. **A** All values were checked.

Related concepts:
- "Exceptions" on page 252
- "Historical and performance data for Q replication and event publishing programs" on page 248
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_RECVQUEUES table

Server: Q Apply server

Default schema: ASN

Primary key: RECVQ

Unique index: REPQMAPNAME

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

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.

Table 68 provides a brief description of the columns in the IBMQREP_RECVQUEUES table.

Table 68. Columns in the IBMQREP_RECVQUEUES table

Column name	Description
REPQMAPNAME	**Data type:** VARCHAR(128); **Nullable:** No The name of the replication queue map that includes this receive queue.

Table 68. Columns in the IBMQREP_RECVQUEUES table (continued)

Column name	Description
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue for this Q subscription.
SENDQ	**Data type:** VARCHAR(48); **Nullable:** Yes The name of the send queue used by the Q Capture program for this Q subscription.
ADMINQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the administration queue that is used by the Q Apply program to send control and error messages to the Q Capture program.
NUM_APPLY_AGENTS	**Data type:** INTEGER; **Nullable:** No, with default The number of agent threads that the Q Apply program uses to concurrently apply transactions from this receive queue. A value of 1 requests that transactions be executed in the order they were received from the source table. Default: 16
MEMORY_LIMIT	**Data type:** INTEGER; **Nullable:** No, with default The maximum amount of memory in megabytes that the Q Apply program can use as a buffer for messages that it gets from this receive queue. Default: 32 MB
CAPTURE_SERVER	**Data type:** VARCHAR(18); **Nullable:** No The name of the database where the Q Capture program that uses this receive queue runs. For z/OS, this is a location name.
CAPTURE_ALIAS	**Data type:** VARCHAR(8); **Nullable:** No The DB2 database alias that corresponds to the Q Capture server that is named in the CAPTURE_SERVER column.
CAPTURE_SCHEMA	**Data type**: VARCHAR(30), VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode; **Nullable**: No, with default The schema of the Q Capture program that uses this receive queue. Default: ASN
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that shows the receive queue's current status. **A (default)** Active: The Q Apply program is processing and applying the transactions from this queue. **I** Inactive: A severe error was encountered on the queue.
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Apply server of the last state change for this receive queue. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:**Yes The number for the ASN message about the queue state. For details, see the IBMQREP_APPLYTRACE table, or the Q Apply diagnostic log.
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the replication queue map that contains this receive queue.

Related concepts:
- "Required settings for WebSphere MQ objects" on page 56
- "Replication queue maps" on page 6
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related tasks:
- "Creating replication queue maps" on page 84

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_SAVERI table

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_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.

Table 69 provides a brief description of the columns in the IBMQREP_SAVERI table.

Table 69. Columns in the IBMQREP_SAVERI table

Column name	Description
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription to which the target tables belong.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that is specified for the Q subscription.
CONSTNAME	**Data type:** VARCHAR (18); **Nullable:** No The unique name of the constraint.
TABSCHEMA	**Data type:** VARCHAR(30), VARCHAR(128) for DB2 UDB 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.

Table 69. Columns in the IBMQREP_SAVERI table (continued)

Column name	Description
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.

Related concepts:
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_SPILLEDROW table

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 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. 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 70 provides a brief description of the columns in the IBMQREP_SPILLEDROW table.

Table 70. Columns in the IBMQREP_SPILLEDROW table

Column name	Description
SPILLQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the spill queue where the message was temporarily stored.
MQMSGID	**Data type:** CHAR(24) FOR BIT DATA; **Nullable:** No The WebSphere MQ message identifier of the message.

Related concepts:
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_SPILLQS table

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.

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.

Table 71 provides a brief description of the columns in the IBMQREP_SPILLQS table.

Table 71. Columns in the IBMQREP_SPILLQS table

Column name	Description
SPILLQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the temporary spill queue that is used for this Q subscription.
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that is used for this Q subscription.

Related concepts:
- “Detailed structure of the Q Apply control tables—Overview” on page 391

Related reference:
- “List of control tables at the Q Apply server” on page 368

IBMQREP_TRG_COLS table

Server: Q Apply server

Default schema: ASN

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 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. 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 72 provides a brief description of the columns in the IBMQREP_TRG_COLS table.

Table 72. Columns in the IBMQREP_TRG_COLS table

Column name	Description
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue that is used for this Q subscription.
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription.
SRC_COLNAME	**Data type:** VARCHAR(30); **Nullable:** No The name of the source column.
TARG_COLNAME	**Data type:** VARCHAR(30); **Nullable:** No The name of the target column. If the target is a stored procedure, this column contains the name of the parameter to which the Q Apply program passes the source column value.
TARG_COLNO	**Data type:** INTEGER; **Nullable:** Yes A number assigned to a target column. If the target is a stored procedure, this column contains a number assigned to the parameter to which the Q Apply program passes the source column value.
MSG_COL_CODEPAGE	**Data type:** INTEGER; **Nullable:** Yes An identifier for the code page that is used to encode the value of the source column in the change message.
MSG_COL_NUMBER	**Data type:** SMALLINT; **Nullable:** Yes The source column's order of appearance in a change message, starting from 0.
MSG_COL_TYPE	**Data type:** SMALLINT; **Nullable:** Yes The DB2 data type of the source column.
MSG_COL_LENGTH	**Data type:** SMALLINT; **Nullable:** Yes The maximum data length defined on the source column.
IS_KEY	**Data type:** CHAR(1); **Nullable:** No A flag that indicates whether the source column is part of the key for the source table. If the value of this flag does not match the target table key definition, the Q Apply program rejects the schema message and invalidates the Q subscription: **Y** The column is part of the source table key. **N** The column is not part of the source table key.

Related concepts:
- "Index or key columns for targets (unidirectional replication)" on page 99
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related reference:
- "List of control tables at the Q Apply server" on page 368

IBMQREP_TARGETS table

Server: Q Apply server

Default schema: ASN

Unique index: (TARGET_OWNER ASC, TARGET_NAME ASC, RECVQ ASC, SOURCE_OWNER ASC, SOURCE_NAME ASC)

Unique index: SUBNAME, RECVQ

Index: RECVQ, SUB_ID

Important: Do not alter this table using SQL. Altering this table inappropriately can cause unexpected results and loss of data.

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.

Table 73 provides a brief description of the columns in the IBMQREP_TARGETS table.

Table 73. Columns in the IBMQREP_TARGETS table

Column name	Description
SUBNAME	**Data type:** VARCHAR(132); **Nullable:** No The name of the Q subscription. It must be unique for each source-target pair, and cannot contain blanks.
RECVQ	**Data type:** VARCHAR(48); **Nullable:** No The name of the receive queue used for this Q subscription.
SUB_ID	**Data type:** INTEGER; **Nullable:** Yes An integer that is generated by the Q Capture program and used to uniquely identify a Q subscription in the `subscription schema` message to the Q Apply program.
SOURCE_SERVER	**Data type:** VARCHAR(18); **Nullable:** No The name of the database or subsystem that contains the source table for this Q subscription. For z/OS, this is a location name.
SOURCE_ALIAS	**Data type:** VARCHAR(18); **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(30); VARCHAR(128) for DB2 UDB 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.
SOURCE_NAME	**Data type:** VARCHAR(128); VARCHAR(18) for z/OS Version 7 and Version 8 compatibility mode;**Nullable:** No The name of the source table for this Q subscription.

Table 73. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
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.
SRC_NICKNAME	**Data type:** VARCHAR(128); **Nullable:** Yes The nickname that is assigned to the source table for automatic loads that uses the LOAD from CURSOR utility when the Q Apply program is running on a non-z/OS platform.
TARGET_OWNER	**Data type:** VARCHAR(30); VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode; **Nullable:** No The schema name or high-level qualifier of the target table or stored procedure for this Q subscription.
TARGET_NAME	**Data type:** VARCHAR(128); VARCHAR(18) for DB2 UDB for z/OS Version 7 and Version 8 compatibility mode; **Nullable:** No The name of the target table for this Q subscription.
TARGET_TYPE	**Data type:** INTEGER; **Nullable:** No, with default A flag that indicates the type of replication target. **1 (default)** User table **2** Reserved for future **3** Reserved for future **4** Reserved for future **5** Stored procedure
FEDERATED_TGT_SRVR	**Data type:** VARCHAR(18); **Nullable:** Yes The name of a federated remote server that is the Q subscription target in replication to non-DB2 relational targets.

Table 73. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
STATE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that is inserted by the Q Apply program to describe the current state of the Q subscription. **I (default)** The Q Apply is not applying changes to the target because the Q subscription is new or in error. The Q Apply program discards all transactions that it receives for the Q subscription and waits for a new `subscription schema` message. **L** The Q Capture program has begun activating the Q subscription by sending a `subscription schema` message, and is sending changes from the source table. **E** The target table is being loaded by an external application. The Q Apply program is putting change messages in a spill queue while it waits for the table to be loaded. **D** The target table is loaded and the Q Apply program is ready to send a `load done` message to the Q Capture program. This state is used for automatic loads only. **F** The Q Apply program is applying messages from the spill queue. **T** The Q Apply program is terminating because of an error. It deactivates the Q subscription, then empties and deletes the spill queue. **A** The Q Apply program is applying changes to the target.
STATE_TIME	**Data type:** TIMESTAMP; **Nullable:** No, with default The timestamp at the Q Apply server of the last change in state for this Q subscription. Default: Current timestamp
STATE_INFO	**Data type:** CHAR(8); **Nullable:** Yes The number for the ASN message about the Q subscription state. For details, see the IBMQREP_APPLYTRACE table, or the Q Apply diagnostic log.
SUBTYPE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates the type of replication that the Q subscription is involved in. **U (default)** Unidirectional replication. **B** Bidirectional replication. **P** Peer-to-peer replication.

Table 73. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
CONFLICT_RULE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program how to look for conflicting changes to the target table. Inserts are always checked using the K (check only keys) rule because there are no before values and keys must be used to detect conflicts. For bidirectional replication, the A (check all) conflict detection rule is required. **K (default)** Check only keys. The Q Apply program looks for conflicts by comparing the current value of the primary key in the target table with the old key value sent from the source table. This is the only conflict rule allowed for unidirectional replication. **C** Check changed columns. Before updating target columns, the Q Apply program makes sure their current value matches the before values in the source columns. For deletes, the Q Apply program checks all columns. **A** Check all columns. Before updating or deleting a row, the Q Apply program makes sure that the current values in all columns match the old values in the source table. **V** Check version. In peer-to-peer replication, the Q Apply program checks the version column before applying a row.
CONFLICT_ACTION	**Data type:** CHAR(1); **Nullable:** No, with default A flag that tells the Q Apply program what to do when a row change conflicts: **I (default)** The Q Apply program does not apply the conflicting row but applies other rows in the transaction. **F** The Q Apply program tries to force the change. This requires that the Q Capture program send all columns, so the CHANGED_COLS_ONLY value must be set to N (no) in the IBMQREP_SUBS table. This is the default value while a target table is being loaded. **D** The Q Apply program does not apply the conflicting row but applies other rows in the transaction. Then it disables the Q subscription, stops applying transactions to the target , and sends an error report to the Q Capture program on the administration queue. **S** The Q Apply program rolls back the transaction, commits, and then stops. **Q** The Q Apply program stops reading from the queue. All conflicting rows are inserted into the IBMQREP_EXCEPTIONS table.

Table 73. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
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 SQL error (other than a conflict) that prevents it from applying a row change. **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.
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.

Table 73. Columns in the IBMQREP_TARGETS table (continued)

Column name	Description
HAS_LOADPHASE	**Data type:** CHAR(1); **Nullable:** No, with default A flag that indicates whether the target table will be loaded with data from the source. **N (default)** The target will not be loaded. **I** An automatic load. The Q Apply program calls the LOAD from CURSOR, EXPORT/IMPORT, or EXPORT/LOAD utility, depending on the type of automatic load that is specified for the Q subscription and the operating system of the Q Apply server. When the table is loaded, the Q Apply program sends a `load done` message to notify the Q Capture program. **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 receiving the `load done` message.
LOAD_TYPE	**Data type:** SMALLINT; **Nullable:** No, with default A flag to indicate which utility is called to load the target table when the load type is internal. **0 (default)** The Q Apply program selects the load utility from among the three options below. **1** Use the LOAD from CURSOR utility. **2** Use the EXPORT/IMPORT utility. **3** Use the EXPORT/LOAD utility.
DESCRIPTION	**Data type:** VARCHAR(254); **Nullable:** Yes A user-supplied description of the Q subscription.
SEARCH_CONDITION	**Data type:** VARCHAR(2048); **Nullable:** Yes The search condition that is used to filter rows for the Q subscription. This must be an annotated select WHERE clause, with a single colon directly in front of the names of the source columns.

Related concepts:
- "Q subscriptions" on page 5
- "Search conditions to filter rows (unidirectional replication)" on page 94
- "Options for unexpected conditions in the target table (unidirectional replication)" on page 100
- "Options for loading target tables for Q replication—Overview" on page 143
- "Error options for Q replication" on page 103
- "Options for conflict detection (bidirectional replication)" on page 129
- "Detailed structure of the Q Apply control tables—Overview" on page 391

Related tasks:
- "Changing acceptable SQL states for Q subscriptions" on page 183

Related reference:

- “List of control tables at the Q Apply server” on page 368

Detailed structure of the Monitor control tables

Detailed structure of the Monitor control tables—Overview

The following topics provide details about each control table at the Monitor control server. The control tables are listed in alphabetical order. The columns within each table are listed in their order of appearance, from left to right.

- “IBMSNAP_ALERTS table”
- “IBMSNAP_CONDITIONS table” on page 416
- “IBMSNAP_CONTACTGRP table” on page 421
- “IBMSNAP_CONTACTS table” on page 422
- “IBMSNAP_GROUPS table” on page 423
- “IBMSNAP_MONENQ table” on page 423
- “IBMSNAP_MONPARMS table” on page 423
- “IBMSNAP_MONSERVERS table” on page 425
- “IBMSNAP_MONTRACE table” on page 426
- “IBMSNAP_MONTRAIL table” on page 427

Related concepts:
- “Control tables for Q replication and event publishing—Overview” on page 361
- “Detailed structure of the Q Apply control tables—Overview” on page 391
- “Detailed structure of the Q Capture control tables—Overview” on page 370

IBMSNAP_ALERTS table

Server: Monitor control server

Index: MONITOR_QUAL, COMPONENT, SERVER_NAME, SCHEMA_OR_QUAL, SET_NAME, CONDITION_NAME, ALERT_CODE

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.

Table 74 provides a brief description of the columns in the IBMSNAP_ALERTS table.

Table 74. Columns in the IBMSNAP_ALERTS table

Column name	Description
MONITOR_QUAL	**Data type:** CHAR(18); **Nullable:** No. The Monitor qualifier that identifies which Replication Alert Monitor program issued the alert.

Table 74. Columns in the IBMSNAP_ALERTS table (continued)

Column name	Description
COMPONENT	**Data type:** CHAR(1); **Nullable:** No. The replication component that is being monitored: **C** Capture program **A** Apply program **S** Q Capture program **R** Q Apply program
SERVER_NAME	**Data type:** CHAR(18); **Nullable:** No. The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server where the alert condition occurred.
SERVER_ALIAS	**Data type:** CHAR(8); **Nullable:** Yes. The DB2 UDB alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server where the alert condition occurred.
SCHEMA_OR_QUAL	**Data type:** VARCHAR(30), VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode subsystems; **Nullable:** No. The Capture schema, Apply schema, Q Capture schema, or Q Apply schema that is being monitored.
SET_NAME	**Data type:** CHAR(18); **Nullable:** No, with default; **Default:** Current subscription set. 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. 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.
CONDITION_NAME	**Data type:** CHAR(18); **Nullable:** No. The condition code that was tested when the alert was triggered.
OCCURRED_TIME	**Data type:** TIMESTAMP; **Nullable:** No. The time that the alert condition occurred at the Capture control server, Apply control server, Q Capture server, or Q Apply server.
ALERT_COUNTER	**Data type:** SMALLINT; **Nullable:** No. The number of times that this alert has been previously detected in consecutive monitor cycles.
ALERT_CODE	**Data type:** CHAR(10); **Nullable:** No. The message code that was issued when the alert occurred.
RETURN_CODE	**Data type:** INT; **Nullable:** No. The integer value returned by a user condition.

Table 74. Columns in the IBMSNAP_ALERTS table (continued)

Column name	Description
NOTIFICATION_SENT	**Data type:** CHAR(1); **Nullable:** No. A flag that indicates whether a notification message was sent: **Y** A notification message was sent. **E** A notification was not sent because the **email_server** parameter was not specified. **N** A notification was not sent because the number of notifications already reached the limit set by the **max_notifications_per_alert** parameter.
ALERT_MESSAGE	**Data type:** VARCHAR(1024); **Nullable:** No. The text of the message that was sent, including the message code.

IBMSNAP_CONDITIONS table

Server: Monitor control server

Index: MONITOR_QUAL, COMPONENT, SERVER_NAME, SCHEMA_OR_QUAL, SET_NAME, CONDITION_NAME

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.

Table 75 provides a brief description of the columns in the IBMSNAP_CONDITIONS table.

Table 75. Columns in the IBMSNAP_CONDITIONS table

Column name	Description
SERVER_NAME	**Data type**: CHAR(18); **Nullable**: No. The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server where this condition is being monitored.
COMPONENT	**Data type**: CHAR(1); **Nullable**: No. The replication component that is being monitored: **C** Capture program **A** Apply program **S** Q Capture program **R** Q Apply program
SCHEMA_OR_QUAL	**Data type**: VARCHAR(30), VARCHAR(128) for DB2 UDB for z/OS Version 8 new-function mode subsystems; **Nullable**: No. The Capture schema, Apply schema, Q Capture schema, or Q Apply schema that is being monitored.

Table 75. Columns in the IBMSNAP_CONDITIONS table (continued)

Column name	Description
SET_NAME	**Data type**: CHAR(18); **Nullable**: No; **Default**: Current subscription set. 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. 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.
MONITOR_QUAL	**Data type**: CHAR(18); **Nullable**: No. 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.
SERVER_ALIAS	**Data type**: CHAR(8); **Nullable**: Yes. The DB2 UDB alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server where this condition is being monitored..
ENABLED	**Data type**: CHAR(1); **Nullable**: No. A flag that indicates whether the Replication Alert Monitor will process this condition during the next monitoring cycle: **Y** The Replication Alert Monitor will process this definition during the next cycle. **N** The Replication Alert Monitor will ignore this definition during the next cycle.
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.

Table 75. Columns in the IBMSNAP_CONDITIONS table (continued)

Column name	Description
CONDITION_NAME (Continued)	**APPLY_STATUS** The status of the Apply program. **APPLY_SUBSFAILING** Whether any subscription sets failed. **APPLY_SUBSINACT** Whether any subscription sets either failed or are inactive. **APPLY_ERRORS** Whether the Apply program posts any error messages. **APPLY_WARNINGS** Whether the Apply program posts any warning messages. **APPLY_FULLREFRESH** Whether a full refresh occurred. **APPLY_REJTRANS (update anywhere)** Whether the Apply program rejects transactions in any subscription set. **APPLY_SUBSDELAY** Whether the Apply program delays more than the time that you specified in the PARM_INT parameter. **APPLY_REWORKED** Whether the Apply program reworked any rows at the target table. **APPLY_LATENCY** Whether the end-to-end latency of the Apply program exceeds a threshold.
CONDITION_NAME (Continued)	**QCAPTURE_STATUS** Whether the Q Capture program is down. **QCAPTURE_ERRORS** Whether the Q Capture program posted any error messages. **QCAPTURE_WARNINGS** Whether the Q Capture program posted any warning messages. **QCAPTURE_LATENCY** 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 threshhold. **QCAPTURE_MEMORY** Whether the memory amount that the Q Capture program used exceeds a threshold. **QCAPTURE_TRANSIZE** Whether a transaction exceeds the MAX_TRANS_SIZE (maximum transaction size) that is set in the IBMQREP_CAPMON table. **QCAPTURE_SUBSINACT** Whether a Q Subscription changed to I (inactive) state.

Table 75. Columns in the IBMSNAP_CONDITIONS table (continued)

Column name	Description
CONDITION_NAME (Continued)	**QAPPLY_STATUS** Whether the Q Apply program is down. **QAPPLY_ERRORS** Whether the Q Apply program posted any error messages. **QAPPLY_WARNINGS** Whether the Q Apply program posted any warning messages. **QAPPLY_LATENCY** Whether the queue latency (the time it takes for a message to go from the send queue to the receive queue) exceeds a threshold. **QAPPLY_EELATENCY** Whether the end-to-end latency (the time it takes for a transaction to replicate from the source to the target) exceeds a threshold. **QAPPLY_EXCEPTIONS** Whether the Q Apply inserted a row in the IBMQREP_EXCEPTIONS table because of a SQL error or conflict. **QAPPLY_MEMORY** Whether the amount of memory that the Q Apply program used to read messages from a particular receive queue exceeds a threshold. **QAPPLY_SPILLQDEPTH** Whether the number of messages on a spill queue exceeds a threshold. **QAPPLY_QDEPTH** Whether the number of messages on a receive queue exceeds a threshold.

Table 75. Columns in the IBMSNAP_CONDITIONS table (continued)

Column name	Description
PARM_INT	**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. **CAPTURE_LASTCOMMIT** Threshold in seconds. **CAPTURE_CLATENCY** Threshold in seconds. **CAPTURE_HLATENCY** Threshold in seconds. **CAPTURE_MEMORY** Threshold in megabytes. **APPLY_SUBSDELAY** Threshold in seconds. **APPLY_REWORKED** Threshold in rows reworked. **APPLY_LATENCY** Threshold in seconds. **QCAPTURE_LATENCY** Threshold in seconds **QCAPTURE_MEMORY** Threshold in megabytes **QCAPTURE_TRANSIZE** Threshold in megabytes **QAPPLY_EELATENCY** Threshold in seconds **QAPPLY_LATENCY** Threshold in seconds **QAPPLY_MEMORY** Threshold in megabytes **QAPPLY_SPILLQDEPTH** Threshold in number of messages. **QAPPLY_QDEPTH** Threshold in number of messages.

Table 75. Columns in the IBMSNAP_CONDITIONS table (continued)

Column name	Description
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. • 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 On UNIX servers, this value is the name of the user ID that was used when the server was connected. On Windows servers, the 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
CONTACT	**Data type**: VARCHAR(127); **Nullable**: No. The name of the individual contact or the group of contacts to be notified if this condition occurs.

IBMSNAP_CONTACTGRP table

Server: Monitor control server

Index: GROUP_NAME, CONTACT_NAME

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).

Table 76 provides a brief description of the columns in the IBMSNAP_CONTACTGRP table.

Table 76. Columns in the IBMSNAP_CONTACTGRP table

Column name	Description
GROUP_NAME	**Data type**: VARCHAR(127); **Nullable**: No. The name of the contact group.

Table 76. Columns in the IBMSNAP_CONTACTGRP table (continued)

Column name	Description
CONTACT_NAME	**Data type**: VARCHAR(127); **Nullable**: No. A contact name that is part of the group. These individuals are specified in the Monitor contacts (IBMSNAP_CONTACTS) table.

IBMSNAP_CONTACTS table

Server: Monitor control server

Index: CONTACT_NAME

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.

Table 77 provides a brief description of the columns in the IBMSNAP_CONTACTS table.

Table 77. Columns in the IBMSNAP_CONTACTS table

Column name	Description
CONTACT_NAME	**Data type**: VARCHAR(127); **Nullable**: No. The name of the contact. Only an individual contact is allowed. Group names are not supported.
EMAIL_ADDRESS	**Data type**: VARCHAR(128); **Nullable**: No. The main e-mail or pager address for this contact.
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.
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.
DESCRIPTION	**Data type**: VARCHAR(1024); **Nullable**: Yes. A description of the contact.

IBMSNAP_GROUPS table

Server: Monitor control server

Index: GROUP_NAME

The IBMSNAP_GROUPS table contains the name and description of each contact group. One group per row is specified.

Table 78 provides a brief description of the columns in the IBMSNAP_GROUPS table.

Table 78. Columns in the IBMSNAP_GROUPS table

Column name	Description
GROUP_NAME	**Data type**: VARCHAR(127); **Nullable**: Yes. The name of the contact group.
DESCRIPTION	**Data type**: VARCHAR(1024); **Nullable**: Yes. A description of the contact group.

IBMSNAP_MONENQ table

Server: Monitor control server

Index: MONITOR_QUAL

The IBMSNAP_MONENQ table is reserved for future options of DB2 replication.

Table 79 provides a brief description of the columns in the IBMSNAP_MONENQ table.

Table 79. Columns in the IBMSNAP_MONENQ table

Column name	Description
MONITOR_QUAL	**Data type**: CHAR(18); **Nullable**: No. Reserved for future options of DB2 replication.

IBMSNAP_MONPARMS table

Server: Monitor control server

Index: MONITOR_QUAL

Default schema: ASN

This table contains information that you can update by using SQL.

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 length of time that the Monitor program retains data in the CD and UOW tables before pruning and the number of notification messages that the Monitor program will receive whenever an alert condition is met. If you make changes to the parameters in this table, the Monitor program reads your modifications only during startup.

Table 80 provides a brief description of the columns in the IBMSNAP_MONPARMS table.

Table 80. Columns in the IBMSNAP_MONPARMS table

Column name	Description
MONITOR_QUAL	**Data type**: CHAR(18); **Nullable**: No. The Monitor qualifier matches the parameters to the Replication Alert Monitor program to which these parameters apply.
ALERT_PRUNE_LIMIT	**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.
AUTOPRUNE	**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 CD, UOW, signal, trace, and Monitor tables: **Y** Autopruning is on. **N** Autopruning is off.
EMAIL_SERVER	**Data type**: INT(128); **Nullable**: Yes. The address of the e-mail server using the SMTP protocol.
LOGREUSE	**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 re-creating it when the Monitor program is restarted. **N** The Monitor program appends new information to the Monitor log file.
LOGSTDOUT	**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.
NOTIF_PER_ALERT	**Data type**: INT; **Nullable**: No, with default; **Default:** 3. The number of notification messages that will be sent when an alert condition is met.
NOTIF_MINUTES	**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.
MONITOR_ERRORS	**Data type**: VARCHAR(128); **Nullable**: 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.

Table 80. Columns in the IBMSNAP_MONPARMS table (continued)

Column name	Description
MONITOR_INTERVAL	**Data type**: INT; **Nullable**: No, with default; **Default:** 300 (5 minutes). How often, in seconds, that the Replication Alert Monitor is run to monitor the alert conditions that were selected.
MONITOR_PATH	**Data type**: VARCHAR(1040); **Nullable**: Yes. The path where the output from the Monitor program is sent.
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 terminates: **Y** The Monitor program terminates when DB2 terminates: **N** The Monitor program stays active and waits for DB2 to be restarted.
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.

IBMSNAP_MONSERVERS table

Server: Monitor control server

Index: MONITOR_QUAL, SERVER_NAME

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.

Table 81 provides a brief description of the columns in the IBMSNAP_MONSERVERS table.

Table 81. Columns in the IBMSNAP_MONSERVERS table

Column name	Description
MONITOR_QUAL	**Data type**: CHAR(18); **Nullable**: No. 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.

Table 81. Columns in the IBMSNAP_MONSERVERS table (continued)

Column name	Description
SERVER_NAME	**Data type**: CHAR(18); **Nullable**: No. The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.
SERVER_ALIAS	**Data type**: CHAR(8); **Nullable**: Yes. The DB2 UDB alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.
LAST_MONITOR_TIME	**Data type**: TIMESTAMP; **Nullable**: Yes. 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.
START_MONITOR_TIME	**Data type**: TIMESTAMP; **Nullable**: Yes. 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.
END_MONITOR_TIME	**Data type**: TIMESTAMP; **Nullable**: Yes. 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.
LASTRUN	**Data type**: TIMESTAMP; **Nullable**: No. 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.
LASTSUCCESS	**Data type**: TIMESTAMP; **Nullable**: Yes. 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).
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

Server: Monitor control server

Index: MONITOR_QUAL, TRACE_TIME

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.

Table 82 provides a brief description of the columns in the IBMSNAP_MONTRACE table.

Table 82. Columns in the IBMSNAP_MONTRACE table

Column name	Description
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

Server: Monitor control server

Index: None

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.

Table 83 provides a brief description of the columns in the IBMSNAP_MONTRAIL table.

Table 83. Columns in the IBMSNAP_MONTRAIL table

Column name	Description
MONITOR_QUAL	**Data type**: CHAR(18); **Nullable**: No. 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
SERVER_NAME	**Data type**: CHAR(18); **Nullable**: No. The name of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.

Table 83. Columns in the IBMSNAP_MONTRAIL table (continued)

Column name	Description
SERVER_ALIAS	**Data type**: CHAR(8); **Nullable**: Yes. The DB2 UDB alias of the Capture control server, Apply control server, Q Capture server, or Q Apply server that the Replication Alert Monitor was monitoring.
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.
LASTRUN	**Data type**: TIMESTAMP; **Nullable**: No. 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.
LASTSUCCESS	**Data type**: TIMESTAMP; **Nullable**: Yes. 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.
ENDTIME	**Data type**: TIMESTAMP; **Nullable**: No, with default; **Default**: current timestamp. The time when this row was inserted into this table.
LAST_MONITOR_TIME	**Data type**: TIMESTAMP; **Nullable**: Yes. 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.
START_MONITOR_TIME	**Data type**: TIMESTAMP; **Nullable**: Yes. 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.
END_MONITOR_TIME	**Data type**: TIMESTAMP; **Nullable**: Yes. The last time when the Replication Alert Monitor finished monitoring the Capture control server, Apply control server, Q Capture server, or Q Apply server.
SQLCODE	**Data type**: INT; **Nullable**: Yes. The SQLCODE of any errors that occurred during this monitor cycle.
SQLSTATE	**Data type**: CHAR(5); **Nullable**: Yes. The SQLSTATE of any errors that occurred during this monitor cycle.
NUM_ALERTS	**Data type**: INT; **Nullable**: No. The number of alert conditions that occurred during this monitor cycle.
NUM_NOTIFICATIONS	**Data type**: INT; **Nullable**: No. The number of notifications that were sent during this monitor cycle.

Detailed structure of versioning columns for peer-to-peer replication

User source and target tables that will be part of a peer-to-peer replication scenario require two versioning columns, a timestamp column and a small integer column, which are maintained by triggers. Together, 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 replication columns and triggers are created when you use the 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 84 provides a brief description of the extra columns in user tables that are required for peer-to-peer replication.

Table 84. Extra columns required for peer-to-peer replication

Column name	Description
ibmqrepVERNODE	**Data type:** SMALLINT; **Nullable:** No, with default A number that identifies the database or subsystem that contains the table within a peer-to-peer group. Default: 0
ibmqrepVERTIME	**Data type:** TIMESTAMP; **Nullable:** No, with default A timestamp that records when a change occurs in the table. Default: 0001-01-01-00.00.00

Related concepts:
- "Peer-to-peer replication" on page 120
- "Control tables for Q replication and event publishing—Overview" on page 361

Chapter 27. Structure of XML messages for event publishing

Structure of XML messages for event publishing—Overview

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—Overview"
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433
- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

XML message types and requirements

XML message types and requirements—Overview

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:
- "XML message types"
- "Technical requirements for XML messages" on page 432
- "How XML delimiters are handled in character data" on page 432

Related concepts:
- "Event publishing" on page 9
- "Structure of XML messages for event publishing—Overview" on page 431

XML 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 85 describes the three types of messages.

Table 85. XML messages sent by a Q Capture program and user application

Type of message	Direction	Description
Data	Q Capture to user application	Contains one of the following things from the source table: • All or part of a transaction • A single row operation • All or part of a large object (LOB) value from a row operation within a transaction

Table 85. XML messages sent by a Q Capture program and user application (continued)

Type of message	Direction	Description
Informational	Q Capture to user application	Provides information about the status of the Q Capture program or an XML publication.
Control	User application to Q Capture	Asks the Q Capture program to activate or deactivate an XML publication, invalidate a send queue, or confirm that a target table is loaded.

Related concepts:
- "Q Apply program" on page 18
- "Q Capture program" on page 15
- "Event publishing" on page 9
- "XML message types and requirements—Overview" on page 431

Related reference:
- "List of XML messages from a user application to Q Capture" on page 459
- "List of XML messages from Q Capture to a user application" on page 434

Technical requirements for XML messages

The Q Capture program generates messages in the form of XML document instances according to the following guidelines:
- Messages are encoded in Unicode by using UTF-8 (code page 1208) as specified in the XML 1.0 (2nd edition), W3C Recommendation, 6 October 2000.
- Message structure follows the XML Schema Language (Part 1: Structure and Part 2: Datatypes), W3C Recommendation, 2 May 2001.

Changes from the source database are converted into messages using the version of IBM®'s International Components for Unicode (ICU4C) that is shipped with DB2® UDB Version 8.2.

To interpret control messages from the subscribing application, the Q Capture program uses the IBM XML parser XML4C version 5.3.

Related reference
- Extensible Markup Language (XML) 1.0 (Second Edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- International Components for Unicode

Related concepts:
- "XML message types and requirements—Overview" on page 431

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 would be 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.

Related concepts:

- "XML message types and requirements—Overview" on page 431

Structure of XML messages from Q Capture to a user application

Structure of XML messages from Q Capture to a user application—Overview

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 an XML publication. The informational messages either confirm a user application's request via a control message, report on the status of the Q Capture program, or report XML publication errors.

The following topics describe the XML structure of data messages and informational messages from a Q Capture program to a user application:

- "List of XML messages from Q Capture to a user application" on page 434
- "msg: Root element for XML messages from Q Capture to a user application" on page 435
- "Transaction message" on page 436
- "Row operation message" on page 445
- "Large object (LOB) message" on page 446
- "Subscription deactivated message" on page 449
- "Load done received message" on page 450
- "Error report message" on page 451
- "Heartbeat message" on page 453
- "Subscription schema message (subSchema)" on page 454
- "Add column message" on page 458

Related concepts:

- "Structure of XML messages for event publishing—Overview" on page 431
- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

List of XML messages from Q Capture to a user application

The Q Capture program sends two types of XML messages to a user application:

Data messages
: Contain changes to a source table. Table 86 provides a quick reference to the types of data messages.

Informational messages
: Report on the status of a Q Capture program or XML publication. Table 87 provides a quick reference to the types of informational messages.

Table 86. Data messages from the Q Capture program to a user application

Message type	Description
Transaction	Contains one or more insert, delete, or update operations to a source table. These operations belong to the same database transaction. Also contains commit information for the transaction.
Row operation	Contains a single insert, delete, or update operation to a source table. Also contains commit information about the database transaction that this row is part of.
Large object (LOB)	Contains some or all of the data from a LOB value in the source table. LOB messages are sent separately from the transaction messages and row operation messages that the LOB values belong to.

Table 87. Informational messages from the Q Capture program to a user application

Message type	Description
Subscription deactivated	Tells the user application that the Q Capture program deactivated an XML publication.
Load done received	Acknowledges that the Q Capture program received the message that the target table is loaded.
Error report	Tells the user application that the Q Capture program encountered an XML publication error.
Heartbeat	Tells the user application that the Q Capture program is still running when it has no data messages to send.
Subscription schema	Contains information about the source table and its columns. Also contains data-sending options, send queue name, and information about the Q Capture program and source database.
Add column	Contains information about a column that was added to an existing XML publication.

Related concepts:

- "Q Capture program" on page 15
- "Event publishing" on page 9
- "Structure of XML messages for event publishing—Overview" on page 431

Related reference:

- "List of XML messages from a user application to Q Capture" on page 459

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 88 describes the msg element.

Table 88. Element description for the msg element (Q Capture program to a user application)

Name	Properties
msg	Not empty, complex type, complex content

Structure:

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="XML_schema_instance"
      xsi:noNamespaceSchemaLocation="schema_document"
      version="version" dbName="database_name">

      elements

</msg>
```

Details:

XML_schema_instance
: The URL of the XML schema instance. For event publishing, the URL is www.w3.org/2001/XMLSchema-instance. XML data type: string.

schema_document
: The file name of the XML schema document. XML namespace is not supported in event publishing because messages refer to one XML schema only. Messages from a Q Capture program to a user application refer to the mqcap.xsd schema document. XML data type: string.

version
: The version of the XML message schema. For DB2 UDB Version 8.2, the version is 1.0.0. XML data type: string.

database_name
: The name of the source database or subsystem. XML data type: string.

elements
: One of the elements that the msg element contains. Only one of these elements appears in each message:

- trans
- rowOp
- lob
- subDeactivated
- loadDoneRcvd
- heartbeat
- errorRpt
- subSchema

Example:

The following example shows a message element.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1">

       elements

 </msg>
```

Where *elements* represents one of the following elements: trans, rowOp, lob, subDeactivated, loadDoneRcvd, heartbeat, errorRpt, or subSchema.

Related concepts:

- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related reference:

- "List of XML messages from Q Capture to a user application" on page 434

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 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 438
- "Column element (col)" on page 439
- "Elements for a single-column value" on page 441
- "Elements for a double-column value" on page 442
- "Before-value and after-value elements (beforeVal and afterVal)" on page 444

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 89 describes the trans element.

Table 89. Element description for trans

Name	**Properties**
trans	Not empty, complex type, complex content

Structure:

```
<trans isLast="is_last_indicator" segmentNum="segment_number"
     cmitLSN="commit_logical_sequence_number" cmitTime="commit_time">

             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 using Greenwich mean time (GMT), formatted in microseconds. XML data type: dateTime.

elements
: Each trans element contains one or more of these elements:
  - insertRow
  - updateRow
  - deleteRow

Example:

The following example shows a transaction element that contains one or more of the insert row, update row, or delete row elements.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1">
       <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
             cmitTime="2003-10-31T12:12:12.000122">
```

```
              insertRow, updateRow, or deleteRow

       </trans>
</msg>
```

Where *insertRow, updateRow, or deleteRow* represents the elements that are explained in "Row operation elements (insertRow, updateRow, and deleteRow)."

Row operation elements (insertRow, updateRow, and deleteRow)

Within a transaction element, the row operation elements (insertRow, updateRow, and deleteRow) describe the type of SQL operation that is performed on a row of the source table. Each of these elements contains one or more column elements (col) that describe changes to subscribed columns.

Table 90 describes the insertRow, deleteRow, and updateRow elements.

Table 90. Element description for insertRow, deleteRow, and updateRow

Name	Properties
insertRow	Not empty, complex type, complex content
deleteRow	Not empty, complex type, complex content
updateRow	Not empty, complex type, complex content

Structure:

```
<insertRow subName="XML_publication_name" srcOwner="source_owner"
       srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">

              elements

</insertRow>


<deleteRow subName="XML_publication_name" srcOwner="source_owner"
       srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">

              elements

</deleteRow>


<updateRow subName="XML_publication_name" srcOwner="source_owner"
       srcName="source_name" rowNum="row_number" hasLOBCols="LOB_indicator">

               elements

</updateRow>
```

Details:

XML_publication_name
: The name of the XML publication to which this row operation belongs. XML data type: string.

source_owner
: The schema of the source table where the row operation originated. XML data type: string.

source_name
The name of the source table. XML data type: string.

row_number
If a row operation includes large object (LOB) columns, this attribute will be generated to identify the position number of the row operation in the database transaction. This attribute does not have a default value. XML data type: positiveInteger.

LOB_indicator
A boolean value that indicates whether the row operation contains LOB columns. If it contains LOB columns, the value is 1 (true). The default value is 0 (false). XML data type: boolean.

elements
One or more column elements (col) contained by the insertRow, updateRow, or deleteRow element.

Example:

The following example shows insertRow, updateRow, and deleteRow elements within a `transaction` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
        <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <insertRow subName="S1" srcOwner="USER1" srcName="T1">

                     column_element

              </insertRow>

              <deleteRow subName="S1" srcOwner="USER1" srcName="T1">

                     column_element

              </deleteRow>

              <updateRow subName="S1" srcOwner="USER1" srcName="T1">

                     column_element

              </updateRow>
       </trans>
</msg>
```

Where *column_element* represents the column element that is explained in “Column element (col).”

Column element (col)

The column element (col) describes the name of a subscribed column in the source table, and it also tells whether the column is part of the key to be used for publishing. A col element within an insert or delete operation contains a single value only. Within an update operation, the col element can contain a before value and an after value, depending on the options for sending data that you specified for the XML publication.

Table 91 describes the col element.

Table 91. Element description for col

Name	Properties
col	Not empty, complex type, complex content

Structure:

```
<col name="column_name" isKey="key_indicator">

     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.

single_or_double_column_value
If the column element is part of an insert or delete operation at the source table, it will contain one of the single-column- value elements. For update operations, the column element can contain a double-column value, which includes both a before value and an after value.

Example:

The following example shows an insert operation that contains single column values, and an update operation that contains double column values.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
        <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <insertRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">

                            single_column_value

                     </col>
                     <col name="COL2">

                            single_column_value

                     </col>
              </insertRow>
              <updateRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">

                            double_column_value

                     </col>
                     <col name="COL2">
```

```
                              double_column_value

                        </col>
                    </updateRow>
                </trans>
</msg>
```

Where *single_column_value* represents the elements that are explained in “Elements for a single-column value,” and *double_column_value* represents the elements that are explained in “Elements for a double-column value” on page 442.

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 92 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 92. Element descriptions for single column value

Name	XML data type	Value’s data format
smallint	short	
integer	integer	
bigint	long	
float	float (32 bits) double (64 bits)	[-]d.dddddE[-|+]dd [-]d.dddddddddddddddE[-|+]dd
real	float	
double	double	
decimal	decimal	
date	date	YYYY-MM-DD
time	time	HH:MM:SS.SSS
timestamp	dateTime	YYYY-MM-DDTHH:MM:SS.SSS
char	string	
varchar	string	
long varchar	string	
bitchar	hexBinary	
bitvarchar	hexBinary	
bitlongvarchar	hexBinary	
graphic	string	
vargraphic	string	
longvargraphic	string	
rowid	hexBinary	

Table 92. Element descriptions for single column value (continued)

Name	XML data type	Value's data format
blob	hexBinary	
clob	string	
dbclob	string	

Structure:

```
<data_type xsi:nil="null_indicator">value</data_type>
```

Details:

data_type
: The data type of the column in the source table. This data type is used to name the element.

null_indicator
: Optional: An integer that indicates whether the source column contains a NULL value. The default is 0 (false). If the source column contains a NULL value, the value of this attribute is 1 (true). The blob, clob, and dbclob elements do not have this attribute. XML data type: boolean.

value
: The actual value in the source column. If the source value is NULL or a LOB value, the element is empty.

Example:

The following example shows an insert operation with single column values of 222 in a key column with an integer data type and Hello in a nonkey column with a varchar data type. The example also shows a delete operation of the row with a single-column value of 222 in a key column with an integer data type.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
       <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <insertRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>222</integer>
                     </col>
                     <col name="COL2">
                            <varchar>Hello</varchar>
                     </col>
              </insertRow>
              <deleteRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>222</integer>
                     </col>
              </deleteRow>
       </trans>
</msg>
```

Elements for a double-column value

Double-column-value elements are used in update operations when the Q Capture program needs to send both before and after values from source columns. In XML 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 XML 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 Table 92 on page 441.

Structure:

```
<data_type>

  elements

</data_type>
```

Details:

data_type
: The data type of the column in the source table. This data type is used to name the element. If the data type is blob, clob, or dbclob, the elements are empty and not nullable.

elements
: One or both of the beforeValue or afterValue elements, or empty for blob, clob, and dbclob.

Example:

The following example shows double-column-value elements for:
- A key column (integer data type) that has changed.
- A nonkey column (varchar data type) that has changed, but the BEFORE_VALUES data-sending option for the XML publication is set to "No."

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
        <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">
              <updateRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>

                                   beforeValue
                                   afterValue

                            </integer>
                     </col>
                     <col name="COL2">
                            <varchar>

                                  afterValue

                            </varchar>
                     </col>
```

```
            </updateRow>
      </trans>
</msg>
```

Where *beforeValue* and *afterValue* represent the elements that are explained in "Before-value and after-value elements (beforeVal and afterVal)."

Before-value and after-value elements (beforeVal and afterVal)

Before-value and after-value elements (beforeVal and afterVal) contain actual values from the source table. These elements are used in update operations for key columns that have changed, and for changed nonkey columns when the BEFORE_VALUES data-sending-option for the XML publication is set to "Yes." If the XML 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 93 describes the beforeVal and afterVal elements.

Table 93. Element descriptions for beforeVal and afterVal

Name	Properties
beforeVal	Nullable, complex type, simple content
afterVal	Nullable, complex type, simple content, optional

Structure:

```
<beforeVal xsi:nil="null_indicator">value</beforeVal>
<afterVal xsi:nil="null_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.

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 XML publication is set to "Yes."

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
       <trans isLast="1" segmentNum="1" cmitLSN="0000:0000::0000:06d6:87ab"
            cmitTime="2003-10-31T12:12:12.000122">
              <updateRow subName="S1" srcOwner="USER1" srcName="T1">
                     <col name="COL1" isKey="1">
                            <integer>
                                   <afterVal>222</afterVal>
```

```
                        </integer>
                    </col>
                    <col name="COL2">
                        <varchar>
                            <beforeVal>Hello</beforeVal>
                            <afterVal xsi:nil="1"/>
                        </varchar>
                    </col>
                </updateRow>
            </trans>
</msg>
```

Related concepts:

- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related tasks:

- "Creating publishing queue maps" on page 156

Related reference:

- "Row operation message" on page 445
- "Large object (LOB) message" on page 446

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 94 describes the rowOp element.

Table 94. Element description for rowOp

Name	Properties
rowOp	Not empty, complex type, complex content

Structure:

```
<rowOp cmitLSN="commit_logical_sequence_number"
    cmitTime="commit_time" isLast="is_last_indicator">

        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 using 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.

elements
: Each rowOp element contains one of these elements:
  - insertRow
  - updateRow
  - deleteRow

Example:

The following example shows a row operation element that contains an insertRow, updateRow, or deleteRow element.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0" dbName="DB1">
       <rowOp cmitLSN="0000:0000::0000:06d6:87ab"
              cmitTime="2003-10-31T12:12:12.000122">

              insertRow, deleteRow, or updateRow

       </rowOp>
</msg>
```

Where *insertRow, updateRow, or deleteRow* represents the elements that are explained in Transaction message.

Related concepts:
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related tasks:
- "Creating publishing queue maps" on page 156

Related reference:
- "Transaction message" on page 436
- "Large object (LOB) message" on page 446

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 XML 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 95 describes the lob element.

Table 95. Element description for lob

Name	Properties
lob	Not empty, complex type, complex content

Structure:

```
<lob isLast="is_last_indicator" subName="XML_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.

XML_publication_name
: The name of the XML publication that includes the LOB value. XML data type: string.

source_owner
: The schema of the source table where the LOB originated. XML data type: string.

source_name
: The name of the source table. XML data type: string.

row_number
: Within the database transaction, the position number of the row operation that contains the LOB value. XML data type: positiveInteger.

column_name
: The name of the column in the source table that contains the LOB value. XML data type: string.

LOB_data_length
The total length of the LOB value contained in the source table, in bytes. XML data type: nonNegativeInteger.

segment_data_length
The length of the LOB data contained in a single message segment, in bytes. XML data type: nonNegativeInteger.

LOB_column_value
One of the three LOB column value elements that describe the data type of the LOB value. The three elements are blob, clob, and dbclob.

Example:

The following example shows a LOB message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1">
       <lob isLast="0" subName="S1" srcOwner="USER1" srcName="T1" rowNum="3"
          colName="LOBCOL" totalDataLen="92675" dataLen="100">

            LOB_column_value

       </lob>
</msg>
```

Where *LOB_column_value* describes one of the three elements that are explained in "LOB column value."

LOB column value

The three LOB column value elements each contain actual LOB data from the source table. The elements are named for their data type, either blob, clob, or dbclob. If the value from the source table is NULL, the elements are empty and the xsi:nil attribute is set to 1 (true).

Table 96 describes the LOB column value elements.

Table 96. Element description for LOB column values

Name	Properties
blob	Nullable, complex type, simple content
clob	Nullable, complex type, simple content
dbclob	Nullable, complex type, simple content

Structure:

```
<data_type xsi:nil="null_indicator">

  LOB_value

</data_type>
```

Details:

data_type
The data type of the column in the source table. This data type is used to name the element.

null_indicator
Optional: A boolean value that indicates whether the value in the source column is NULL. The default is 0 (false). If the source column contains a NULL value, the value of this attribute is 1 (true). XML data type: boolean.

LOB_value
Actual data from the large object in the source table.

Example:

The following example shows a LOB message that includes 100 bytes of the 92,675 total bytes of data from a CLOB (character large object) value.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd"
      version="1.0.0" dbName="DB1>
       <lob isLast="0" subName="S1" srcOwner="USER1" srcName="T1" rowNum="3"
          colName="LOBCOL" totalDataLen="92675" dataLen="100">

            <clob>LOB data</clob>

       </lob>
</msg>
```

Related concepts:
- “Queue depth considerations for large object (LOB) values” on page 64
- “Considerations for large object (LOB) data types for Q replication and event publishing” on page 172
- “Memory for LOB data types for Q replication and event publishing” on page 31
- “Structure of XML messages from Q Capture to a user application—Overview” on page 433

Related reference:
- “Transaction message” on page 436
- “Row operation message” on page 445

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 97 describes the subDeactivated element.

Table 97. Element description for subDeactivated

Name	Properties
subDeactivated	Empty, complex type

Structure:

```
<subDeactivated subName="XML_publication_name" srcOwner="source_owner"
       srcName="source_name" stateInfo="state_information"/>
```

Details:

XML_publication_name
The name of the XML publication that was deactivated. XML data type: string.

source_owner
The schema of the source table for the XML 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 XML 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>
```

Related concepts:
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related tasks:
- "Deactivating Q subscriptions or XML publications" on page 222

Related reference:
- "Deactivate subscription message" on page 463

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 98 describes the loadDoneRcvd element.

Table 98. Element description for loadDoneRcvd

Name	Properties
loadDoneRcvd	Empty, complex type

Structure:

```
<loadDoneRcvd subName="XML_publication_name" srcOwner="source_owner"
      srcName="source_name" stateInfo="state_information"/>
```

Details:

XML_publication_name
: The name of the XML publication for which the target table was loaded. XML data type: string.

source_owner
: The schema of the source table for the XML 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 XML publication. This attribute contains an ASN message number. XML data type: string.

Example:

The following example shows a `load done received` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
     dbName="DB1">

      <loadDoneRcvd subName="S1" srcOwner="USER1" srcName="T1"
            stateInfo="ASN7019I"/>

</msg>
```

Related concepts:
- "Options for loading target tables for Q replication—Overview" on page 143
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related reference:
- "Load done message" on page 462

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 an XML 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 99 describes the errorRpt element.

Table 99. Element description for errorRpt

Name	Properties
errorRpt	Empty, complex type

Structure:

```
<errorRpt subName="XML_publication_name" srcOwner="source_owner"
       srcName="source_name" errorMsg="message_text"/>
```

Details:

XML_publication_name
The name of the XML publication that generated an error. XML data type: string.

source_owner
The schema of the source table for the XML publication. XML data type: string.

source_name
The name of the source table. XML data type: string.

message_text
The text of the error message. XML data type: string.

Example:

The following example shows an `error report` message generated after the Q Capture program was unable to activate an XML publication

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">

        <errorRpt subName="S1" srcOwner="USER1" srcName="T1"
           errorMsg="message_text"/>

</msg>
```

Where *message_text* is the text of the error message.

Related concepts:
- "Error options for Q replication" on page 103
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

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 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 100 describes the heartbeat element.

Table 100. Element description for heartbeat

Name	**Properties**
heartbeat	Empty, complex type

Structure:

```
<heartbeat sendQName="send_queue_name" lastCmitTime="last_commit_time"/>
```

Details:

send_queue_name
: The name of the send queue where the Q Capture program put the `heartbeat` message. XML data type: string.

last_commit_time
: Optional: The timestamp of the last committed transaction in Greenwich mean time (GMT). This attribute is optional and has no default value. XML data type: dateTime.

Example:

The following example shows a `heartbeat` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">

       <heartbeat sendQName="Q1" lastcmitTime="2003-10-31T12:12:12.000122"/>

</msg>
```

Related concepts:
- "Required settings for WebSphere MQ objects" on page 56
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related reference:
- "IBMQREP_SENDQUEUES table" on page 380

Subscription schema message (subSchema)

The Q Capture program sends a `subscription schema` message to acknowledge that it activated or reinitialized an XML publication. The message conveys details about the XML 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 456

Subscription schema element (subSchema)

Through its attributes, the subSchema element provides details about an XML publication. The subSchema element contains one or more column elements (col).

Table 101 describes the subSchema element.

Table 101. Element description for subSchema

Name	Properties
subSchema	Not empty, complex type, simple content

Structure:

```
<subSchema subname="XML_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:

XML_publication_name
: The name of the XML publication that was activated or reinitialized. XML data type: string.

source_owner
: The schema of the source table for the XML 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 XML 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 XML 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 XML 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 XML 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 XML 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 UDB 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 102 provides additional details about two XML data types used in attributes for the subSchema element.

Table 102. Additional data type descriptions for subSchema attributes

Type name	Base Type	Values
loadPhaseEnumType	string	none, external
dbServerTypeEnumType	string	QDB2, QDB2/6000, QDB2/HPUX, QDB2/NT, QDB2/SUN, QDB2/LINUX, QDB2/Windows **Note:** QDB2 by itself implies DB2 UDB for z/OS.

Example:

The following `subscription schema` message would be sent for an XML publication that specifies the BEFORE_VALUES data-sending option and a load phase.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
       <subSchema subname="S1"
                  srcOwner="USER1"
                  srcName="T1"
                  sendQName="Q1"
                  beforeValues="yes"
                  hasLoadPhase="external"
                  dbServerType="QDB2/6000"
                  dbRelease="8.2.0"
                  dbInstance="DB2INST"
                  capRelease="8.2.0">

             column_element

       </subSchema>
</msg>
```

Where *column_element* represents one or more column elements, which are explained in "Column element (col) in a subscription schema message."

Column element (col) in a subscription schema message

Within the `subscription schema` message, the column element (col) conveys information about each column in the source table.

Table 103 describes the col element within a schema message.

Table 103. Element description for col

Name	Properties
col	Empty, complex type

Structure:

```
<col name="column_name"
     type="data_type"
     len="data_length"
     precision="data_precision"
     scale="decimal_scale"
     codepage="codepage_number"
     isKey="key_indicator"/>
```

Details:

column_name
: The name of the column in the source table. XML data type: string.

data_type
: The data type of the source column. This must be one of the data types defined for the dataTypeEnumType XML data type. For a list, see Table 104 on page 457. XML data type: dataTypeEnumType.

Table 104. Additional data type description for dataTypeEnumType

Type name	Base type	Values
dataTypeEnumType	string	smallint, integer, bigint, float, real, double, decimal, char, varchar, longvarchar, bitchar, bitvarchar, bitlongvarchar, graphic, vargraphic, longvargraphic, time, timestamp, date, rowid, blob, clob, dbclob

data_length
Optional: The maximum length of the data in the source column. XML data type: unsignedInt.

data_precision
Optional: For decimal data types, the precision of the number. XML data type: unsignedShort.

decimal_scale
Optional: For decimal data types, the scale of the number. XML data type: unsignedShort.

codepage_number
Optional: The code page for character data types. The default is 0. XML data type: unsignedShort.

key_indicator
Optional: A boolean value that indicates whether this is a key column. The default is 0 (false). If it is a key column, the value is 1 (true). XML data type: boolean.

Example:

The following example shows two column elements within a subscription schema message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
     dbName="DB1">
       <subSchema subname="S1"
                 srcOwner="USER1"
                 srcName="T1"
                 sendQName="Q1"
                 beforeValues="yes"
                 hasLoadPhase="external"
                 dbServerType="QDB2/6000"
                 dbRelease="8.2.0"
                 dbInstance="DB2INST"
                 capRelease="8.2.0">
             <col name="COL1" type="integer" len="4"/>
             <col name="COL2" type="varchar" len="50" codepage="1208"/>
       </subSchema>
</msg>
```

Related concepts:
- "Setting up publishing from sources (event publishing)—Overview" on page 155
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Add column message

An `add column` message tells the user application that a Q Capture program added a column to an existing XML 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 105 describes the addColumn element.

Table 105. Element description for addColumn

Name	Properties
addColumn	Not empty, complex type, simple content

Structure:

```
<addColumn subName="XML_publication_name" srcOwner="source_owner"
    srcName="source_name">

       column_element

</addColumn>
```

Details:

XML_publication_name
: The name of the XML publication that the column was added to. XML data type: string.

srcOwner
: The schema of the source table for the XML publication. XML data type: string

srcName
: The name of the source table. XML data type: string.

column_element
: A column schema (col) element that contains details about the column that was added, such as name, data type, data length, and whether the column is a key column.

Example:

The following example shows an `add column` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
      dbName="DB1">
    <addColumn subName="S1" srcOwner="USER1" srcName="T1">

       column_element

    </addColumn>
</msg>
```

Where *column_element* represents the column element that is explained in "Subscription schema message (subSchema)" on page 454.

Related concepts:
- "Structure of XML messages from Q Capture to a user application—Overview" on page 433

Related tasks:
- "Adding columns to existing XML publications" on page 193

Related reference:
- "IBMQREP_SIGNAL table" on page 382

Structure of XML messages from a user application to Q Capture

Structure of XML messages from a user application to Q Capture—Overview

A user application communicates with a Q Capture program by sending XML 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 an XML publication, or invalidate a send queue.

The following topics describe the XML structure of control messages from a user application to a Q Capture program.
- "List of XML messages from a user application to Q Capture"
- "msg: Root element for XML messages from a user application to Q Capture" on page 460
- "Invalidate send queue message" on page 461
- "Load done message" on page 462
- "Activate subscription message" on page 463
- "Deactivate subscription message" on page 463

Related concepts:
- "Structure of XML messages for event publishing—Overview" on page 431

List of XML messages from a user application to Q Capture

Table 106 describes the four types of control messages from a user application to a Q Capture program.

Table 106. Control messages from a user application to a Q Capture program

Message type	Description
Invalidate send queue	Requests that a Q Capture program invalidate a send queue by performing the queue error action that you specified.
Load done	Tells a Q Capture program that the target table for an XML publication is loaded.
Activate subscription	Requests that a Q Capture program activate an XML publication.
Deactivate subscription	Requests that a Q Capture program deactivate an XML publication.

Related concepts:

- "Q Capture program" on page 15
- "Event publishing" on page 9
- "Structure of XML messages for event publishing—Overview" on page 431

Related reference:

- "List of XML messages from Q Capture to a user application" on page 434

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 107 describes the msg element.

Table 107. Element description for msg (user application to a Q Capture program)

Name	Properties
msg	Not empty, complex type, complex content

Structure:

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="XML_schema_instance"
      xsi:noNamespaceSchemaLocation="schema_document"
      version="version">

      elements

<msg>
```

Details:

XML_schema_instance
: The URL of the XML schema instance. For event publishing, the URL is www.w3.org/2001/XMLSchema-instance. XML data type: string.

schema_document
: The file name of the XML schema document. XML namespace is not supported in event publishing because messages refer to one XML schema only. Messages from a user application to a Q Capture program refer to the mqsub.xsd schema document. XML data type: string.

version
: The version of the XML message schema. For DB2 UDB Version 8.2, the version is 1.0.0. XML data type: string.

elements
: One of the elements that the msg element contains. Only one of these elements appears in each message:
  - invalidateSendQ
  - loadDone
  - activateSub
  - deactivateSub

Example:

The following example shows a message from a user application to the Q Capture program.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       elements

</msg>
```

Where *elements* represents one of the following elements: invalidateSendQ, loadDone, activateSub, deactivateSub.

Related concepts:

- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

Related reference:

- "List of XML messages from a user application to Q Capture" on page 459

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 108 describes the invalidateSendQ element.

Table 108. Element description for invalidateSendQ

Name	Properties
invalidateSendQ	Empty, complex type

Structure:

```
<invalidateSendQ sendQName="send_queue_name"/>
```

Details:

send_queue_name
The name of the send queue that the Q Capture program is being asked to invalidate. XML data type: string.

Example:

The following example shows an `invalidate send queue` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <invalidateSendQ sendQName="S1"/>
</msg>
```

Related concepts:
- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

Related reference:
- "List of XML messages from a user application to Q Capture" on page 459

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 109 describes the loadDone element.

Table 109. Element description for loadDone

Name	Properties
loadDone	Empty, complex type

Structure:

```
<loadDone subName="XML_publication_name"/>
```

Details:

XML_publication_name
The name of the XML publication that completed its load phase. XML data type: string.

Example:

The following example shows a `load done` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <loadDone subName="S1"/>

</msg>
```

Related concepts:
- "Options for loading target tables for Q replication—Overview" on page 143
- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

Related reference:
- "List of XML messages from a user application to Q Capture" on page 459
- "Load done received message" on page 450

Activate subscription message

An `activate subscription` message tells a Q Capture program to begin capturing changes for an XML publication.

Table 110 describes the activateSub element.

Table 110. Element description for activateSub

Name	Properties
activateSub	Empty, complex type

Structure:

```
<activateSub subName="XML_publication_name"/>
```

Details:

XML_publication_name
The name of the XML publication that the Q Capture program is being asked to activate. XML data type: string.

Example:

The following example shows an `activate subscription` message.

```
<xml version="1.0" encoding="UTF-8" ?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="mqsub.xsd" version="1.0.0">

       <activateSub subName="S1"/>

</msg>
```

Related concepts:
- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

Related tasks:
- "Activating Q subscriptions or XML publications" on page 221

Related reference:
- "List of XML messages from a user application to Q Capture" on page 459
- "Subscription schema message (subSchema)" on page 454

Deactivate subscription message

A `deactivate subscription` message tells the Q Capture program to stop capturing changes for an XML publication.

Table 111 describes the deactivateSub element.

Table 111. Element description for deactivateSub

Name	Properties
deactivateSub	Empty, complex type

Structure:

```
<deactivateSub subName="XML_publication_name"/>
```

Details:

XML_publication_name
: The name of the XML 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>
```

Related concepts:

- "Structure of XML messages from a user application to Q Capture—Overview" on page 459

Related tasks:

- "Deactivating Q subscriptions or XML publications" on page 222

Related reference:

- "List of XML messages from a user application to Q Capture" on page 459
- "Subscription deactivated message" on page 449

Appendix A. Sample programs to use for Q replication and event publishing

Sample programs to use for Q replication and event publishing—Overview

Sample programs contain example code that you can use to write programs to set up and operate a Q replication environment. The following sections describe sample programs for the Linux operating system, the UNIX operating system, the Windows operating system, and the z/OS operating system:

- "Sample programs to use for Q replication and event publishing (Linux, UNIX, Windows)—Overview"
- "Sample programs to use for Q replication and event publishing (z/OS)—Overview" on page 467

Sample programs (Linux, UNIX, Windows)

Sample programs to use for Q replication and event publishing (Linux, UNIX, Windows)—Overview

The following sections describe sample programs to help you set up and operate your Q replication and event publishing environment for the Linux, UNIX, and Windows operating systems:

- "Samples to set up Q replication and event publishing (Linux, UNIX, Windows)"
- "Samples to operate Q replication programs (Linux, UNIX, Windows)" on page 466

The sample programs for the Linux, UNIX, and Windows operating systems are located in the sqllib/samples/repl directory.

Related reference:

- "Sample programs to use for Q replication and event publishing—Overview" on page 465

Samples to set up Q replication and event publishing (Linux, UNIX, Windows)

Table 112 on page 466 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 112. Sample programs for the Linux, UNIX, and Windows operating systems.

Program	Description
asnqdefq	This sample script is an example that shows how Q replication and event publishing interact with WebSphere MQ. It includes commands to create queues and queue definitions for two for two servers in peer-to-peer replication. Use this sample script to quickly create and set up a Q replication environment.
asnqctlw.sql	This sample script creates control tables for replication.
asnqwxml.zip	This file shows an example of a web-based application for event publishing. The sample demonstrates how to use an XML publication in a business scenario.
asnqspC.SQC	This sample program is an example of a stored procedure that is written in C.
asnqspcreate.sql	This sample script can create a stored procedure that is written in SQL.
asnqspSQL.sql	This sample script is an example of a stored procedure that is written in SQL.

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.

Related reference:

- "Sample programs to use for Q replication and event publishing—Overview" on page 465

Samples to operate Q replication programs (Linux, UNIX, Windows)

Table 113 describes sample programs that can help you start the Q Capture program, the Q Apply program, and the Replication Alert Monitor.

Table 113. Samples to operate Q replication programs

Program	Description
qcapture_api.c	This sample program contains code to start the Q Capture program.
qapply_api.c	This sample program contains code to start the Q Apply program.
monitor_api.c	This sample program contains code to start the Replication Alert Monitor.

Table 114 describes the makefiles that you need to build, compile, and link the programs that are listed in Table 113.

Table 114. Makefiles to build sample programs

Makefile	Description
qcapture_api_nt.mak	This makefile builds the code for the qcapture_api.c sample program on Windows operating systems.
qcapture_api_unix.mak	This makefile builds the code for the qcapture_api.c sample program on Linux and UNIX operating systems.
qapply_api_nt.mak	This makefile builds the code for the qapply_api.c sample program on Windows operating systems.

Table 114. Makefiles to build sample programs (continued)

Makefile	Description
qapply_api_unix.mak	This makefile builds the code for the qapply_api.c sample program on Linux and UNIX operating systems.

Related reference:
- "Sample programs to use for Q replication and event publishing—Overview" on page 465

Sample programs (z/OS)

Sample programs to use for Q replication and event publishing (z/OS)—Overview

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)"
- "Sample JCL programs for Q replication and event publishing (z/OS)" on page 468

The SASNSAMP partitioned data set (PDS) contains the sample programs for the z/OS operating system.

Samples to set up Q replication and event publishing (z/OS)

Table 115 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 115. Table 1. Sample programs for the z/OS operating system.

Program	Description
asnqdefq	This sample script that shows how Q replication and event publishing interact with WebSphere MQ. It includes commands to create queues and queue definitions for two servers in peer-to-peer replication. Use this script to quickly create and set up a Q replication environment.
asnqctlz.sql	This sample script contains code to create control tables for replication.
asnqbnds	This sample program contains code to bind the Q Capture program and the Q Apply program at a local server.
asnqbndl	This sample program contains code to bind the Q Apply program at a remote Windows server.
asnqspC.SQC	This sample program contains example code for a stored procedure that is written in C.

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 JCL programs for Q replication and event publishing (z/OS)

You can use job control language (JCL) to perform many functions in Q replication on the z/OS operating system. Table 116 describes sample programs that are written in JCL.

Table 116. Sample programs for using the job control language (JCL).

Program	Description
asnstpa	This sample program contains JCL to stop a Q Apply program by using the **asnacmd** command.
asnstpc	This sample program contains JCL to stop a Q Capture program by using the **asnccmd** command.
asnstra	This sample program contains JCL to start Q Apply program.
asnstrc	This sample program contains JCL to start a Q Capture program.
asnstrm	This sample program contains JCL to start the Replication Alert Monitor.
asnqtrc	This sample program contains JCL to run the asntrc program for the Q Capture program or the Q Apply program.
asntrdmp	This sample program contains JCL to write the buffer that is generated by the asntrc program to a file.
asntrflw	This sample program contains JCL to generate asntrc flw trace report.
asntrfmt.c	This sample program contains JCL to generate asntrc fmt trace report.
asnqtoff.c	This sample program contains JCL to stop the asntrc program.
asnqton.c	This sample program contains JCL to start the asntrc program.
asntdiff.c	This sample program contains JCL to run the **asntdiff** command.
asnqjspc	This sample program contains JCL to precompile, compile, link-edit, and bind the C stored procedure example.

Related tasks:

- "Scheduling the replication programs (z/OS)" on page 305

Related reference:

- "Sample programs to use for Q replication and event publishing (z/OS)—Overview" on page 467
- "Samples to set up Q replication and event publishing (z/OS)" on page 467

Appendix B. ASNCLP: Command-line interface for administering replication and event publishing

The ASNCLP program is a command-line interface that runs on Linux, UNIX®, or Windows® operating systems. The ASNCLP program provides commands specifically for SQL replication, Q replication, and event publishing. You can use the ASNCLP program to perform many of the tasks that you can perform with the Replication Center. You can use the ASNCLP program to issue administration commands interactively. The output of these commands is always an SQL script, which you can run immediately or at a later time. You can also provide multiple commands in an input file to the ASNCLP program. You can also automate various administration tasks for replication or event publishing by using the command-line interface.

For more information about setting up and using the ASNCLP, refer to the product documentation on the Web:
http://www.ibm.com/software/data/dpropr/library.html.

Glossary

Glossary

A

administration queue. In Q replication and event publishing, a WebSphere MQ queue that is used for communication between a Q Capture program and a Q Apply program or a user application. The administration queue for each Q Capture program must be a local, persistent queue.

after-image. In SQL replication, the updated content of a source-table column that is recorded in a change data (CD) table or in a database log or journal. Contrast with *before-image*.

after-value. In Q replication, the updated content of a source-table column.

agent thread. In Q replication, one of the threads of the Q Apply program that receives transactions from a browser thread and applies this data to target tables on the same server. There can be one or more agent threads for each browser thread.

aggregate table. In SQL replication, a read-only replication target table that contains aggregations of data from the source table. This data is based on SQL column functions such as MIN, MAX, SUM, or AVG.

alert. In replication, a notice that describes events and conditions in replication. The Replication Alert Monitor sends alerts via e-mail or pager.

alert condition. In replication, a condition of the replication environment that causes 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. You choose the alert conditions that the Replication Alert Monitor will check for in your replication environment.

apply. In replication, to refresh or update a replication target table.

Apply control server. In SQL replication, a database that contains Apply control tables that store information about subscription sets and subscription-set members. Contrast with *Apply server*.

Apply cycle. In SQL replication, the interval of time during which data is replicated from a source table to a target table.

Apply latency. In SQL replication, an approximate measurement of the time that replication requires to complete one cycle. See also *Capture latency*.

Apply program. In SQL replication, a program that is used to refresh or update a replication target table. Contrast with *Capture program* and *Capture trigger*.

Apply qualifier. In SQL replication, a case-sensitive character string that identifies replication subscription sets that are unique to an instance of the Apply program.

Apply server. In SQL replication, a system where the Apply program is running. Contrast with *Apply control server*.

archive log. (1) The set of log files that is closed and is no longer needed for normal processing. These files are retained for use in roll-forward recovery. (2) The portion of the DB2 Universal Database for z/OS log that contains log records that are copied from the active log. The archive log holds records that no longer fit on the active log.

ASP. See *auxiliary storage pool (ASP)*.

asynchronous replication. In replication, the process of copying data from a source table to a target table outside the scope of the original transaction that updated the source table. Contrast with *synchronous replication*.

audit trail. Data, in the form of a logical path that links a sequence of events. An audit trail traces the transactions that affect the contents of a record.

authorization token. (1) A token associated with a transaction. (2) For DB2 Universal Database for z/OS, the correlation ID. (3) For DB2 Universal Database for iSeries, the job name of the job that caused a transaction.

auxiliary storage pool (ASP). One or more storage units that are defined from the storage devices or storage device subsystems that make up auxiliary storage. An ASP provides a way of organizing data to limit the impact of storage-device failures and to reduce recovery time.

B

base aggregate table. In SQL replication, a type of replication target table that contains data that is aggregated from a replication source table. Contrast with *change aggregate table*.

before-image. In SQL replication, the content of a replication source-table column prior to an update by a transaction. The content is recorded in a change data (CD) table or in a database log or journal. Contrast with *after-image*.

before-value. In Q replication, the content of a replication source-table column prior to an update by a transaction.

bidirectional replication. In Q replication, a replication configuration in which changes that are made to one copy of a table are replicated to a second copy of that table. Changes that are made to the second copy are replicated back to the first copy. You must choose which copy of the table wins if a conflict occurs.

binary large object (BLOB). A data type that contains a sequence of bytes that can range in size from 0 bytes to 2 gigabytes less 1 byte. This string does not have an associated code page and character set. BLOBs can contain image, audio, and video data. See also *character large object* and *double-byte character large object*.

BLOB. See *binary large object (BLOB)*.

block fetch. A function of DB2 that retrieves (or fetches) a large set of rows together. Using block fetch can significantly reduce the number of messages that are sent across the network. Block fetch applies only to cursors that do not update data.

blocking. In SQL replication, an option that is specified when binding an application. It allows caching of multiple rows of information by the communications subsystem so that each FETCH statement does not require the transmission of one row for each request across the network. See also *block fetch*.

browser thread. In Q replication, a Q Apply program thread that gets messages from a receive queue and passes the messages on to one or more agent threads to be applied to targets.

C

capture. In replication, to gather changes from a source database for replication or event publishing.

Capture control server. (1) In SQL replication, a database that contains the Capture control tables, which store information about registered replication source tables. (2) A system where the Capture program is running.

Capture latency. In SQL replication, an approximate measurement of how recently the Capture program committed data to a CD table. See also *Apply latency*.

Capture program. In SQL replication, a program that reads database log or journal records to capture changes made to DB2 source tables. Contrast with *Apply program* and *Capture trigger*.

Capture schema. In SQL replication, a name that identifies the control tables used by a particular instance of the Capture program.

Capture trigger. In SQL replication, a mechanism that captures delete, insert, and update operations performed on non-DB2 relational source tables. Contrast with *Capture program* and *Apply program*.

cascade rejection. In SQL replication, the process of rejecting a replication transaction because it is associated with a transaction that had a conflict detected and was itself rejected.

CCD table. See *consistent-change data (CCD) table*.

CD table. See *change data table*.

change aggregate table. In SQL replication, a type of replication target table that contains data aggregations that are based on the contents of a CD table. Contrast with *base aggregate table*.

change-capture replication. In SQL replication, the process of capturing changes that are made to a replication source table and applying them to a replication target table. Contrast with *full refresh*.

change data (CD) table. In SQL replication, a replication table at the Capture control server that contains changed data for a replication source table.

character large object (CLOB). A data type that contains a sequence of characters (single-byte, multi-byte, or both) that can range in size from 0 bytes to 2 gigabytes less 1 byte. In general, character large object values are used whenever a character string might exceed the limits of the VARCHAR type. Also called character large object string. See also *binary large object* and *double-byte character large object (DBCLOB)*.

client. Any program or server where a program is running that communicates with and accesses a database server.

CLOB. See *character large object (CLOB)*.

cold start. (1) In SQL replication, the process of starting the Capture program without using restart information from prior operation of the program. When you perform a cold start, a full refresh occurs and all active subscriptions are deactivated and then activated. The cold start process deletes all transactions that were not processed before the cold start. Contrast with *warm start*. (2) In Q replication, the process of starting the Q Capture program without using restart information from prior operation of the program. When you perform a cold start a full refresh occurs. All transactions that were not processed before the cold start are processed after the cold start. The user is responsible for cleaning transactions from the queues prior to the cold start. Contrast with *warm start*. (3) The process of starting a system or program by using an initial program load procedure. (4) A process by which DB2 Universal Database for z/OS restarts without processing any log records.

complete CCD table. In SQL replication, a CCD table that initially contains all the rows from the replication source table or view and any predicates from the source table or view. Contrast with *noncomplete CCD table*. See also *consistent-change data (CCD) table*.

condensed. In SQL replication, a table attribute that indicates that the table contains current data rather than a history of changes to the data. A condensed table includes no more than one row for each primary key value in the table. As a result, a condensed table can be used to supply current information for a refresh.

condensed CCD table. In SQL replication, a CCD table that contains only the most current value for a row and has only one row for each key value. Contrast with *noncondensed CCD table*. See also *consistent-change data (CCD) table*.

conflict detection. In bidirectional replication and update-anywhere replication, conflict detection refers to either of the following processes:

- The process of detecting constraint errors such as key constraints and referential constraints.
- The process of detecting whether the same row was updated by users or application programs in both the source and target tables during the same replication cycle.

consistent-change data (CCD) table. In SQL replication, a type of replication target table that is used for storing history, for auditing data, or for staging data. A CCD table can also be a replication source. See also *complete CCD table*, *condensed CCD table*, *external CCD table*, *internal CCD table*, *noncomplete CCD table*, and *noncondensed CCD table*.

Control Center. The DB2 graphical interface that lets you administrate DB2 databases and perform a variety of tasks including creating objects and monitoring performance. The Control Center shows database objects (such as databases and tables) and their relationship to each other.

control message. In Q replication, a message from a Q Apply program or a user application that asks a Q Capture program to activate or deactivate a Q subscription or an XML publication, invalidate a send queue, or confirm that a target table is loaded.

control server. In SQL replication, a database server that contains replication control tables for the Capture program, the Apply program, or the Replication Alert Monitor. See also *Apply control server*, *Capture control server*, *Q Apply server*, *Q Capture server*, and *Monitor control server*.

control table. See *replication control table*.

D

database management system (DBMS). See *database manager*.

database manager. A program that manages data by providing the services of centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, data currency control, privacy, and security.

database recovery log. In replication, a set of primary and secondary log files that record all changes to a database in log records.

data blocking. In SQL replication, the process of replicating a specific number of minutes' worth of change data during an Apply cycle.

data distribution replication. In replication, a replication configuration that contains a single source table, from which changes are replicated to one or more read-only target tables. Before replication to the target tables can occur, the tables must contain a complete set of data from the source table.

data message. In Q replication, a message that 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.

DB2 replication. See *SQL replication*.

DBCLOB. See *double-byte character large object (DBCLOB)*.

DBMS. Database management system.

delimited identifier. A sequence of characters enclosed within double quotation marks ("). The sequence must consist of a letter followed by zero or more characters, each of which is a letter, a digit, or the underscore character. See also *ordinary identifier*.

differential-refresh replication. See *change-capture replication*.

distinct type. A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes. See also *user-defined type (UDT)*.

double-byte character large object (DBCLOB). A data type that contains a sequence of double-byte characters that can range in size from 0 bytes to 2 gigabytes. This data type can be used to store large double-byte text objects. Also called *double-byte character large object string*. Such a string always has an associated code page. See also *binary large object (BLOB)* and *character large object (CLOB)*.

E

end-to-end latency. In replication, an approximate measurement of the time that replication requires to capture changes from a source database and apply those changes to a target database. See also *Apply latency*, *Capture latency*, *Q Apply latency*, and *Q Capture latency*.

event publishing. A data publishing solution that captures transactional data from DB2 recovery logs and publishes that data as XML messages. The XML messages are published to WebSphere MQ queues where one or more user applications can retrieve and use those messages.

event timing. In SQL replication, the most precise method of controlling when to start a replication subscription cycle. Contrast with *interval timing*.

external CCD table. In SQL replication, a CCD table that can be subscribed to directly because it is a registered replication source. It has its own row in the register table, where it is identified by the SOURCE_OWNER and SOURCE_TABLE columns. See also *consistent-change data table*. Contrast with *internal CCD table*.

F

federated database system. A special type of distributed database management system (DBMS). A federated database system allows you to query and manipulate data located on other servers. The data can be in database managers such as Oracle, Sybase, Microsoft SQL Server, Informix, and Teradata or it can be in lists or stores such as a spreadsheet, Web site, or data mart. SQL statements can refer to multiple database managers or individual databases in a single statement. For example, you can join data located in a DB2 Universal Database table, an Oracle table, and a Sybase view.

full refresh. (1) In SQL replication, the process in which all of the data that matches the registration and the subscription-set predicates for a replication source table is copied to the target table. Also known as loading a target table. A full refresh replaces all existing data in the target table. Contrast with *change-capture replication*. (2) In Q replication, the process in which all of the data that matches the search conditions for a Q subscription for a replication source table is copied to the target table. A full refresh replaces all existing data in the target table.

G

gap. In SQL replication, a situation in which the Capture program is not able to read a range of log or journal records, so there is potential loss of change data.

global record. In SQL replication, the row in the register table that defines global replication characteristics for a particular instance of the Capture program.

H

heterogeneous replication. Replication between DB2 and non-DB2 relational databases. See also *federated database system*.

high-availability disaster recovery. A replication configuration where replicated data is available to dependent applications whenever required and protected against loss that is caused by catastrophic failure.

hot-spot update. A series of repeated updates made to the same rows in a short period of time.

I

IASP. See *Independent Auxiliary Storage Pool (IASP)*.

independent auxiliary storage pool (IASP). One or more storage units that are defined from the disk units or disk-unit subsystems that make up addressable disk storage. An independent auxiliary pool contains objects, the directories that contain the objects, and other object attributes such as authorization ownership attributes. An independent auxiliary storage pool can be made available (varied on) and made unavailable (varied off) without restarting the system. An independent auxiliary pool can either be switchable among multiple systems in a clustering environment or privately connected to a single system.

informational message. In Q replication and event publishing, a message that a Q Capture program sends to inform a Q Apply program or a user application about status of the Q Capture program, a Q subscription, or an XML publication.

internal CCD table. In SQL replication, a CCD table that cannot be subscribed to directly because it is not a registered replication source. It does not have its own row in the register table; it is identified by the CCD_OWNER and CCD_TABLE columns for the row of the associated registered replication source. Contrast with *external CCD table*. See also *consistent-change data (CCD) table*.

interval timing. In SQL replication, the simplest method of controlling when to start a replication subscription cycle. When using interval timing, specify a date and a time for a subscription cycle to start, and set a time interval that describes how frequently the subscription cycle runs. Contrast with *event timing*.

J

join. An SQL relational operation that allows retrieval of data from two or more tables based on matching column values.

journal. For iSeries systems, a system object that identifies the objects being journaled, the current journal receiver, and all the journal receivers on the system for the journal. See also *journal receiver*.

journal code. For iSeries systems, a 1-character code in a journal entry that identifies the category of the journal entry. For example, `F` identifies an operation on a file, `R` identifies an operation on a record, and so forth. See also *journal entry*.

journal entry. For iSeries systems, a record in a journal receiver that contains information about a journaled change or other activity that is journaled. See also *journal code* and *journal entry type*.

journal entry type. For iSeries systems, a 2-character field in a journal entry that identifies the type of operation of a system-generated journal entry or the type of journal entry of a user-generated journal entry. For example, `PT` is the entry type for a write operation. See also *journal code*.

journal identifier (JID). For iSeries systems, a unique identifier that is assigned to a particular object when journaling is started for that object. Journal entries are associated with a particular object by this JID value.

journaling. For iSeries systems, the process of recording, in a journal, the changes made to objects, such as physical file members or access paths, or the depositing of journal entries by system or user functions.

journal receiver. For iSeries systems, a system object that contains journal entries added when events occur that are journaled, such as changes to a database file, changes to other journaled objects, or security-relevant events. See also *journal*.

K

key. (1) In replication, a column or an ordered collection of columns that is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key. (2) In Q replication, the matching column or columns at both the source and target tables that are specified in the Q subscription.

L

large object (LOB). A data type that contains a sequence of bytes that can range in size from 0 bytes to 2 gigabytes less 1 byte. There are three types of large objects: binary large objects (binary), character large objects (single-byte character or mixed), and double-byte character large objects (double-byte character). See *binary large object (BLOB)*, *character large object (CLOB)*, and *double-byte character large object (DBCLOB)*.

latency. The time required for updates that were made to a source to replicate to a target.

load phase. In Q replication, the stage where a target table is loaded with data from a source table so that the two tables are synchronized. With an automatic load, the Q Apply program handles the loading process, and you can either specify a load utility or let the Q Apply program choose the best available utility. With a manual load, you load the target table and then notify the replication programs when the table is loaded.

LOB. See *large object (LOB)*.

logical server. In replication, on Linux, UNIX, and Windows, a DB2 database. On z/OS, a subsystem running DB2.

local database. A database that is physically located on the server in use. Contrast with *remote database*.

lock. (1) A means of serializing events or access to data. (2) A means of preventing uncommitted changes made by one application process from being perceived by another application process and for preventing one application process from updating data that is being accessed by another process.

locking. The mechanism that a database manager uses to ensure the integrity of data. Locking prevents concurrent users from accessing inconsistent data.

log. (1) A file used to record changes that are made in a system. (2) A collection of records that describe the events that occur during DB2 Universal Database for z/OS execution and that indicate their sequence. The information that is recorded is used for recovery in the event of a failure during DB2 Universal Database for z/OS execution. (3) See *database recovery log*.

M

master table. In SQL replication, specifically in update-anywhere replication, the original source table for data in the replica table. If replication conflict detection is enabled, changes made to the master table are retained, whereas changes made to the replica table are rejected. See also *update-anywhere replication*, *replica table*, and *conflict detection*.

member. See *subscription-set member*.

Monitor control server. In replication, a database that contains the Monitor control tables, which store information about alert conditions that the Replication Alert Monitor will monitor.

monitor qualifier. In replication, a case-sensitive character string that identifies a specific instance of the Replication Alert Monitor.

multidirectional replication. In Q replication, a replication configuration that includes peer-to-peer or bidirectional replication.

multi-tier replication. In SQL replication, a replication configuration in which changes are replicated from a replication source in one database to a replication target in another database, and changes from this replication target are replicated again to a replication target in another database.

N

nickname. (1) An identifier that a federated server uses to reference a data source object, such as a table or a view. (2) A name that is defined in a DB2 V8 for Informix sources database or DB2 II database to represent a physical database object (such as a table or stored procedure) in a non-DB2 database.

noncomplete CCD table. In SQL replication, a CCD table that is initially empty and has rows appended to it as changes are made to the replication source. Contrast with *complete CCD table*. See also *consistent-change data (CCD) table*.

noncondensed CCD table. In SQL replication, a CCD table that can contain more than one row for each key value. These duplicate rows represent the history of changes for the values in rows of a table. Contrast with *condensed CCD table*. See also *consistent-change data (CCD) table*.

non-DB2 relational database server. An Informix database server or a relational database server from a vendor other than IBM.

nullable. The condition in which a column, function parameter, or result can have an absence of a value.

null value. A parameter position for which no value is specified.

O

object. (1) Anything that can be created or manipulated with SQL—for example, tables, views, indexes, or packages. (2) In object-oriented design or programming, an abstraction that consists of data and operations associated with that data. (3) For NetWare, an entity that is defined on the network and thus given access to the file server. (4) In the Information Catalog Center, an item that represents a unit or distinct grouping of information. Each Information Catalog Center object identifies and describes information but does not contain the actual information. For example, an object can provide the name of a report, list its creation date, and describe its purpose.

occasionally connected. In SQL replication, a replication configuration that contains target servers that are not always connected to the network. This configuration allows users to connect to a primary data source for a short time to synchronize their local database with the data at the source.

ODBC. See *Open Database Connectivity (ODBC)*.

ODBC driver. A driver that implements ODBC function calls and interacts with a data source.

Open Database Connectivity (ODBC). An application program interface (API) that allows access to database management systems using callable SQL, which does not require the use of an SQL preprocessor. The ODBC architecture allows users to add modules, called *database drivers*, that link the application to their choice of database management systems at run time. Application programs do not need to be linked directly to the modules of all the supported database management systems.

ordinary identifier. (1) In SQL, a letter followed by zero or more characters, each of which is a letter (a-z and A-Z), a symbol, a number, or the underscore character, used to form a name. (2) In DB2 Universal Database for z/OS, an uppercase letter followed by zero or more characters, each of which is an uppercase letter, a number, a digit, or the underscore character.

P

package. A control structure produced during program preparation that is used to execute SQL statements.

peer-to-peer replication. In Q replication, a replication configuration between peer tables in which updates at any table are replicated to the other tables, and convergence is maintained. Peer-to-peer replication can have two servers or three or more servers. Contrast with *update-anywhere replication*. See also *multi-tier replication*.

point-in-time table. In SQL replication, a type of replication target table whose content matches all or part of a source table, with an added column that identifies the approximate time when the particular row was inserted or updated at the source system.

predicate. An element of a search condition that expresses or implies a comparison operation.

primary key. A unique key that is part of the definition of a table. A primary key is the default parent key of a referential constraint definition. It is a column or combination of columns that uniquely identifies a row in a table.

promote. In SQL replication, to copy replication definitions for subscription sets or registered sources from one database to another database, without having to register the sources again or create the subscription sets again.

pruning. In replication, the task of removing obsolete data from replication control tables or log files that are used by the Capture, Q Capture, Apply, and Q Apply programs.

publishing queue map. In event publishing, an object that includes a send queue for sending messages and settings for how a Q Capture program processes all transactions that use the send queue. See also *replication queue map* and *queue map*

pull configuration. In SQL replication, a replication configuration in which the Apply program runs at the target server. The Apply program pulls updates from the source server to apply them to the target. Contrast with *push configuration*.

push configuration. In SQL replication, a replication configuration in which the Apply program runs at the source server or a replication server other than the target server. The Apply program pushes updates from the source server to apply them to the target. Contrast with *pull configuration*.

Q

Q Apply latency. In Q replication, 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.

Q Apply program. In Q replication, a program that reads transactions from a receive queue and applies those changes to one or more target tables or passes the changes to a stored procedure.

Q Apply schema. In Q replication, the identifier for a Q Apply program and its control tables.

Q Apply server. In Q replication, a database or subsystem on which the control tables for the Q Apply program are located and where the Q Apply program runs. It contains one or more sets of the control tables that store information about target tables and other replication definitions.

Q Capture latency. In Q replication, an approximate measure of how current a Q Capture program is in reading the DB2 recovery log. Q Capture latency measures the time between a Q Capture program saving performance data and the timestamp of the last committed transaction that the program had read in the log when it saved the data. For example, if the Q Capture program saved performance data at 10 a.m. and the timestamp of the last committed transaction was 9:59 a.m., the Q Capture latency would be one minute.

Q Capture program. In Q replication and event publishing, a program that reads the DB2 recovery log to capture changes made to DB2 source tables and transmits the changes via one or more send queues.

Q Capture schema. In Q replication, the identifier for a Q Capture program and its control tables.

Q Capture server. In Q replication and event publishing, a database or subsystem on which the control tables for the Q Capture program are located and where the Q Capture program runs. It contains one or more sets of the control tables that store information about Q subscriptions and XML publications and other replication or publishing definitions.

Q Capture transaction latency. In Q replication, the time from when a Q Capture program reads the commit statement for a transaction in the DB2 recovery log, to the time when the Q Capture program puts the message containing the transaction on a send queue.

Q replication. A replication solution that uses WebSphere MQ message queues for high-volume, low-latency replication. It provides a peer-to-peer solution with conflict detection, conflict resolution, and convergence.

Q subscription. In Q replication, an object that identifies a mapping between a source table and target table or stored procedure and specifies what changes are replicated. The Q Capture program replicates changes from a source table and puts those changes on a send queue in compact format. Then, the Q Apply program gets the compact messages from a receive queue and applies the changes to a target table or passes them to a stored procedure for data manipulation. Q subscriptions are separate objects from XML publications in that Q subscriptions do not replicate data that is published in an XML publication.

Q subscription group. In Q replication, the group of Q subscriptions that are involved in replicating the same logical tables.

queue. A WebSphere MQ object. Message queuing applications can put messages on, and get messages from, a queue. The Q Capture and Q Apply programs can put messages on, and get messages from a queue. A queue is owned and maintained by a queue manager.

queue latency. In Q replication and event publishing, 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.

queue map. In Q replication and event publishing, an object that links queues and defines how the Q Capture and Q Apply programs process messages that use the queues. There are two kinds of queue maps: publishing queue maps and replication queue maps. See also *publishing queue map* and *replication queue map*.

R

RDBMS. See *relational database management system (RDBMS)*.

real-time replication. See *synchronous replication*.

receive queue. In Q replication, a WebSphere MQ message queue that is used by a Q Apply program to receive transactions that are captured by a Q Capture program.

recapture. In update-anywhere replication, to capture changes at a replica table and forward these changes to the master table or to other replica tables.

referential constraints. The referential integrity rule that the non-null values of the foreign key are valid only if they also appear as values of a parent key.

referential integrity. The state of a database in which all values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of *referential constraints* on all operations that change the data in a table where the referential constraints are defined.

register. In SQL replication, to define a DB2 table, view, or nickname as a replication source.

registration. (1) In SQL replication, the process of registering a DB2 table, view, or nickname as a replication source. Contrast with *subscription*. (2) See *replication source*.

rejected transaction. A transaction containing one or more updates from replica tables that are in conflict with the master table.

relational database management system (RDBMS) . A collection of hardware and software that organizes and provides access to a relational database.

remote database. A database that is physically located on a server other than the one in use. Contrast with *local database*.

replica table. In SQL replication, specifically in update-anywhere replication, a type of target table that can be updated locally and also receives updates from the master table through a subscription-set definition. If replication conflict detection is enabled, changes made to the replica table are rejected, whereas changes made to the master table are retained. See also *update-anywhere replication*, *master table*, and *conflict detection*.

replication. The process of maintaining a defined set of data in more than one location. It involves copying designated changes for one location (a source) to another (a target), and synchronizing the data in both locations.

replication administrator. (1) In Q replication, the user responsible for creating Q subscriptions and XML publications. This user can also run the Q Capture program and the Q Apply program (2) In SQL replication, the user responsible for registering replication sources and creating subscription sets. This user can also run the Capture program and the Apply program.

Replication Alert Monitor. In replication, a program that checks the operation of the Capture, Apply, Q Capture, and Q Apply programs, and sends alerts to one or more users when it detects the specified alert conditions.

Replication Analyzer. In replication, a program that can analyze a replication environment for setup problems, configuration errors, and performance issues.

Replication Center. In replication, a graphical user interface that lets you define, operate, maintain and monitor the replication environment. It is part of the DB2 Administration Client tool suite.

replication control table. In replication, a table in which replication definitions or control information is stored.

replication queue map. In Q replication, an object that links a send queue and a receive queue. The replication queue map includes settings for how a Q Capture program processes all transactions that use the send queue and how a Q Apply program processes all transactions that use the receive queue. See also *publishing queue map* and *queue map*.

replication source. (1) In SQL replication, a table, view, or nickname that is registered as a source for replication. Changes that are made to this table are captured and copied to a target table that is defined in a subscription-set member. See also *subscription set* and *subscription-set member*. (2) In Q replication, a table that is a source for replication. Changes made to this type of table are captured and copied to a target table that is defined in a Q subscription or an XML publication. See also *Q subscription* and *XML publication*.

replication target. (1) In SQL replication, a table, view, or nickname that is a destination for changes that were replicated from a registered replication source. The Apply program applies these changes. See also *target table*. (2) In Q replication, a table or stored procedure that is a destination for changes that were replicated from a source. The Q Apply program applies these changes. See also *target table*.

retention-limit pruning. In SQL replication, the pruning of CD and UOW tables by the Capture program that are older than a limit that the user specifies.

rework. (1) To convert an insert into a replication target table to an update if the insert fails because the row already exists in the target table. (2) To convert an update to a replication target table to an insert if the update fails because the row does not exist in the target table.

row-capture rules. In SQL replication, rules based on changes to registered columns that define when and whether the Capture program writes a row to a CD table, or when and whether the Capture triggers write a row to a CCD table.

S

send queue. In Q replication, a WebSphere MQ message queue that is used by a Q Capture program to publish transactions that it has captured. A send queue can be used either for Q replication or event publishing, but not both at the same time.

serialization. (1) The consecutive ordering of items. (2) The process of controlling access to a resource to protect the integrity of the resource. (3) In Q replication, the process of applying transactions in the same order that they were committed at the source.

server. See *logical server* See also: *Apply control server*, *Apply server*, *Capture control server*, *Control server*, *Monitor control server*, *Q Apply server*, *Q Capture server*, *source server*, and *target server*.

signal. A communication mechanism for replication that allows communication with the Capture program and the Q Capture program. A signal is an SQL statement that is inserted into the signal control table, and received by the Capture program or the Q Capture program when it reads the log entry for the signal insert.

source server. In replication, a database or subsystem that contains the source tables.

source table. In replication, a table that contains data that is to be replicated to a target table. Contrast with *target table*.

spill agent thread. In Q replication, a thread that applies transactions that are waiting in the spill queue and informs the browser thread once the spill queue is empty and has been deleted.

spill file. In SQL replication, a temporary file created by the Apply program that is used to hold data for updating target tables.

spill queue. In Q replication, a dynamic queue that the Q Apply program creates to hold transactions that occur at the source table while a target table is being loaded. The Q Apply program later applies these transactions and then deletes the spill queue.

SQL replication. A type of replication that uses staging tables.

staging table. In SQL replication, a CCD table that is used to save data before that data is replicated to the target database. A CCD table that is used for staging data can function as an intermediate source for updating data to one or more target tables. See also *consistent-change-data table*.

subscription. (1) In SQL replication, an object that creates subscription sets and subscription-set members. Contrast with *registration* in SQL replication, and *Q subscription* in Q replication. (2) See also *subscription set*.

subscription cycle. The process in which SQL replication retrieves changed data for a given subscription set, replicates the changes to the target table, and updates the appropriate replication control tables to reflect its status and current progress.

subscription set. In SQL replication, a replication definition that controls the replication of changed data during a subscription cycle. A subscription set can contain zero or more subscription-set members.

subscription-set member. In SQL replication, a replication definition that maps a registered replication source with a replication target. Each member defines the structure of the target table and the rows and columns that will be replicated from the source table.

subset. To replicate data from part of a source table, rather than from the entire table, to a target table. You can subset by rows or by columns.

synchpoint. In SQL replication, a replication control table value for the DB2 log or journal record sequence number of the last change applied during the most recent Apply cycle. This value is also used to coordinate the pruning of CD tables.

synchronous replication. Also known as real-time replication, a type of replication that delivers updates continuously and within the scope of source transactions.

T

table-mode processing. In SQL replication, a type of replication subscription-set processing in which the Apply program retrieves all the data from the source CD table, then applies the data (one member at a time) to each target table, and finally commits its work. Contrast with *transaction-mode processing*.

target server. (1) In SQL replication, a database or subsystem that contains replication target tables, views, or stored procedures. (2) In Q replication, a database or subsystem that contains replication target tables or stored procedures.

target table. (1) In SQL replication, a table that is the destination for replicated changes from a registered replication source. It can be a user copy table, a point-in-time table, a base aggregate table, a change aggregate table, a CCD table, or a replica table. (2) In Q replication, a table that is the destination for replicated changes from a source that is part of a Q subscription.

timestamp. A data type, that contains a seven-part value that consists of a date and time expressed in years, months, days, hours, minutes, seconds, and microseconds.

trace. (1) For replication, a facility that provides the ability to collect monitoring, auditing, and performance data for the Capture program, the Q Capture program, Apply program, the Q Apply program, or Replication Alert Monitor. (2) A DB2 Universal Database for z/OS facility that provides the ability to monitor and collect monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

transaction. An exchange between a server and a program, two servers, or two programs that accomplishes a particular action or result. An example of a transaction is the entry of a customer's deposit and the subsequent update of the customer's balance. Synonym for *unit of work*.

transaction-based replication. In SQL replication, a type of replication processing in which every transaction is replicated to the target table when it is committed in the source table. Contrast with *transaction-consistent replication*.

transaction-consistent replication. In SQL replication, a type of replication processing in which the net result of all transaction updates is replicated to the target table. Contrast with *transaction-based replication*.

transaction-mode processing. In SQL replication, a type of replication subscription-set processing in which the Apply program retrieves data from the source CD table, then applies the data to the target table in the same commit sequence used at the source. The Apply program processes transactions for all subscription-set members together, rather than sequentially. Contrast with *table-mode processing*.

trigger. (1) An object in a database that is invoked indirectly by the database manager when a particular SQL statement is run. See also *Capture trigger*. (2) A set of SQL statements that is stored in a DB2 database and executed when a certain event occurs in a DB2 table.

U

UDT. See *user-defined type (UDT)*.

uncommitted read (UR). An isolation level that allows an application to access uncommitted changes of other transactions. The application does not lock other applications out of the row that it is reading unless the other application attempts to drop or alter the table.

Unicode. An international character encoding scheme that is a subset of the ISO 10646 standard. Each character supported is defined using a unique 2-byte code.

unidirectional replication. In Q replication, a replication configuration in which changes that occur at a source table are replicated over WebSphere MQ queues to a target table or are passed to a stored procedure to manipulate the data. Changes that occur at the target table are not replicated back to the source table.

unique index. An index that ensures that no identical key values are stored in a table.

unique key. A key that is constrained so that no two of its values are equal.

unit of work. (1) A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a DB2 Universal Database for z/OS multisite update operation, a single unit of work can include several units of recovery. Synonym for *transaction*. (2) In the Information Catalog Center, a recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations.

unit-of-work (UOW) table. In SQL replication, a replication control table stored in the Capture control server that contains commit records read from the database log or journal. The records show that a transaction or UOW committed successfully and include a unit-of-recovery ID that can be used to join the unit-of-work table and the CD table to produce transaction-consistent change data.

update-anywhere replication. In SQL replication, a replication configuration in which all tables are both registered sources and read-write targets. One table is the primary source table for full refresh of all the others. In this configuration, there is an implicit replication hierarchy among the source and target tables. Contrast with *peer-to-peer replication*. See also *multi-tier replication*, *master table*, and *replica table*.

user copy table. In SQL replication, a replication target table whose content matches all or part of a registered source table and contains only user data columns.

user-defined type (UDT). A data type that is not native to the database manager and was created by a user. In DB2 Universal Database, the term *distinct type* is used instead of user-defined type.

V

view. (1) A logical table that consists of data that is generated by a query. A view is based on an underlying set of base tables, and the data in a view is determined by a select type query that is run on the base tables. (2) A way of looking at the information about, or contained in objects. Each view might reveal different information about its objects.

W

warm start. In replication, the process of starting the Capture program or the Q Capture program so that it reads transactions from the point where it left off. Contrast with *cold start*.

work file. In SQL replication, a temporary file used by the Apply program when processing a subscription set.

X

XML publication. In event publishing, an object that identifies what changes are published 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 in XML format. You provide an application other than the Q Apply program to retrieve and use those XML messages. XML publications are separate objects from Q subscriptions in that Q subscriptions do not replicate data that is published in an XML publication.

Accessibility

Users with physical disabilities, such as restricted mobility or limited vision, can use software products successfully by using accessibility features. These are the major accessibility features in DB2 Information Integrator Version 8:

- You can operate all features by using the keyboard instead of the mouse.
- You can customize the size and color of your fonts.
- You can receive either visual or audio alert cues.
- DB2 supports accessibility applications that use the Java™ Accessibility API.
- DB2 documentation is provided in an accessible format.

Keyboard input and navigation

You can operate the DB2 database tools, such as Control Center, Data Warehouse Center, and Replication Center, by using only the keyboard. You can use keys or key combinations instead of a mouse to perform most operations.

In UNIX-based systems, the position of the keyboard focus is highlighted. This highlighting indicates which area of the window is active and where your keystrokes will have an effect.

Accessible display

The DB2 database tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font settings

For the DB2 database tools, you can use the Tools Settings notebook to select the color, size, and font for the text in menus and windows.

Nondependence on color

You do not need to distinguish between colors to use any of the functions in this product.

Alternative alert cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with assistive technologies

The DB2 Information Integrator graphical interface supports the Java Accessibility API, enabling the use of screen readers and other assistive technologies that are used by people with disabilities.

Accessible documentation

Documentation for the DB2 family of products is available in HTML format. You can view documentation according to the display preferences set in your browser. You can use screen readers and other assistive technologies.

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

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

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

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

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

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

Trademarks

Index

A

abstract data types 171
activate subscription XML message 463
activating
  Q subscriptions or XML publications 221
add column message 458
add_partition parameter, Q Capture (Linux, UNIX, Windows) 206, 315
ADDCOL signal
  for Q subscriptions 181
  for XML publications 193
administration queues
  for Q Apply 44
  for Q Capture 44
  in replication queue maps 6, 84
  required settings 56
administration threads 246
adminq parameter, Q Capture 206
after values
  for key columns 107
  included in XML publications 169
  meeting search conditions 94
  parameters for key 113
agent threads 246
  description 18
  in replication queue maps 84
  latency 250
  message size 60
  replication queue maps 184
alert conditions
  ASNMAIL exit routine 262
  e-mail notification 261
  for Apply program 258
  for Capture program 258
  for Q Apply program 258
  for Q Capture program 258
  list 258
  notification criteria 274
  overview 257
  selecting 267
alert_prune_limit parameter, Replication Alert Monitor 270
APPLHEAPSZ database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
Apply program
  alert conditions 258
apply_path parameter, Q Apply 231, 323
apply_schema parameter, Q Apply 231, 323, 326
apply_server parameter, Q Apply 231, 323, 326
archive logging, turning on 69
ARM (Automatic Restart Manager)
  description 297
  restarting replication programs 298
ASNCLP program 469
ASNMAIL exit routine 262
asnqacmd command 326
asnqanalyze command 336
asnqapp command 323
asnqcap command 315
asnqccmd command 320
asnqfmt command 358
asnslist command 342
asntdiff command 355
asntrep command 357
authorization
  for Q Apply program 38
  for Q Capture program 37
  for Replication Alert Monitor 39
  for replication, administration 40
  for replication, Linux, UNIX, Windows 38
  for replication, overview 37
  for replication, z/OS 38
  WebSphere MQ
    Q Apply program 62
    Q Capture program 62
    Replication Alert Monitor 62
automatic load
  considerations for z/OS 146
  definition 143
  overview 144
  specifying nicknames for 146
  utilities used for 145
Automatic Restart Manager (ARM)
  description 297
  restarting replication programs 298
autoprune parameter, Replication Alert Monitor 270
autostop parameter
  Q Apply 231, 323
  Q Capture 206, 315

B

batch mode, replication programs 296
before values
  bidirectional replication, conflict detection 129
  filtering data 15
  IBMQREP_SUBS table 387
  included in XML publications 169
  LOB columns 127
  LOB data types 172
  non-key columns 113
  restrictions for LOB data types 129
  ROWID columns 172
  stored procedures 107
bidirectional replication
  conflict detection 117, 129
  creating Q subscriptions 127
  creating replication queue maps 84
  description 3, 117
  error options 103
  load options 150
  replication objects 117
  specifying queues 84
  starting 139
bidirectional replication *(continued)*
  stopping 140
  WebSphere MQ objects 50
binding packages
  Q Apply (Linux, UNIX, Windows) 73
  Q Capture (Linux, UNIX, Windows) 73
  Replication Alert Monitor (Linux, UNIX, Windows) 74
BLOB (binary large object) data types
  column values 446
  large object (LOB) XML message 446
  memory 31
  message buffer size 31
  replicating 172
browser threads 246
  description 18
  starting 239
  stopping 241

C

Capture program
  alert conditions 258
capture_path parameter, Q Capture 206, 315
capture_schema parameter, Q Capture 206, 315
capture_server parameter, Q Capture 206, 315
capturing changes
  starting 221
  stopping 222
change capture
  starting 221
  stopping 222
changing
  event publishing environment 191
  publishing queue maps 195
  Q Apply parameters 237
  Q Capture parameters 217
  Q replication environment 179
  Q subscription attributes 179
  replication queue maps 184
  SQL states 183
  XML publication attributes 191
changing parameters
  Q Apply program 237, 238
  Q Capture program 219, 220
channel objects, required settings 56
checking status of programs 245
chgparms parameter
  Q Apply 326
  Q Capture 320
CHLTYPE parameter, WebSphere MQ 56
CLOB (character large object) data types
  column values 446
  large object (LOB) XML message 446
  memory 31

CLOB (character large object) data types *(continued)*
message buffer size 31
replicating 172
CLP (command line processor)
See also ASNCLP program 469
code pages
for event publishing
description 36
overview 35
for Q replication
description 35
overview 35
Q Apply program 35
Q Capture program
for event publishing 36
for Q replication 35
XML messages for event publishing 36
col element in subscription schema message 454
cold starts
effect on Q subscriptions 221, 222
effect on XML publications 221, 222
preventing 203
restart message 15
startmode parameter 206
column (vertical) subsetting
event publishing 163
Q replication 92
Q subscriptions 92
XML publications 163
columns
adding to existing XML publications 193
adding to Q subscriptions 181
key columns
event publishing 167
Q replication 99
Q subscriptions 99
XML publications 167
mapping
in Q replication 98
in Q subscriptions 98
published in XML publications 168
publishing
event publishing 163
XML publications 163
replicating
Q replication 92
Q subscriptions 92
subsetting
event publishing 163
Q replication 92
Q subscriptions 92
XML publications 163
command line processor (CLP)
See also ASNCLP program 469
commands
See replication commands 313
commit_interval parameter, Q Capture 206, 315
compact messages through replication queue maps 84
compact messages, Q replication
through replication queue maps 6
compression dictionaries 282
conflict detection
bidirectional replication 117
peer-to-peer replication 120
SQL states
changing 183
conflict detection, bidirectional replication 129
conflicts
in IBMQREP_EXCEPTIONS table 252
value-based and version-based 252
contact groups 255
contacts
defining 264
description 255
control messages 431
control tables
analyzing 336
creating
Q Capture and Apply 76
Replication Alert Monitor 263
dropping 188, 198
event publishing
illustration 361
overview 361
illustration 361
maintaining 283
Monitor control server
IBMSNAP_ALERTS 414
IBMSNAP_CONDITIONS 416
IBMSNAP_CONTACTGRP 421
IBMSNAP_CONTACTS 422
IBMSNAP_GROUPS 423
IBMSNAP_MONENQ 423
IBMSNAP_MONPARMS 423
IBMSNAP_MONSERVERS 425
IBMSNAP_MONTRAIL 427
list 369, 414
pruning 284
Q Apply server
IBMQREP_APPENQ 392
IBMQREP_APPLYMON 393
IBMQREP_APPLYPARMS 395
IBMQREP_APPLYTRACE 397
IBMQREP_DELTOMB 398
IBMQREP_DONEMSG 399
IBMQREP_EXCEPTIONS 400
IBMQREP_RECVQUEUES 402
IBMQREP_SAVERI 404
IBMQREP_SPILLEDROW 405
IBMQREP_SPILLQS 406
IBMQREP_TARGETS 408
IBMQREP_TRG_COLS 406
list 368, 391
Q Capture server
IBMQREP_ADMINMSG 371
IBMQREP_CAPENQ 371
IBMQREP_CAPMON 372
IBMQREP_CAPPARMS 374
IBMQREP_CAPQMON 377
IBMQREP_CAPTRACE 378
IBMQREP_SENDQUEUES 380
IBMQREP_SIGNAL 382
IBMQREP_SRC_COLS 386
IBMQREP_SRCH_COND 379
IBMQREP_SUBS 387
list 367, 370
control tables *(continued)*
Q replication
illustration 361
overview 361
reorganizing 285
RUNSTATS utility 285
converting
data for event publishing
description 36
overview 35
data for Q replication
description 35
overview 35
creating
alert conditions 173
control tables
from Replication Center 173
Q Capture and Q Apply 76
Replication Alert Monitor 263
monitors 266
publishing queue maps 156
Q subscriptions
bidirectional replication 127
peer-to-peer replication, overview 135
peer-to-peer replication, three or more servers 137
peer-to-peer replication, two servers 135
unidirectional replication, multiple 90
unidirectional replication, overview 86
unidirectional replication, single 88
replication queue maps 84
replication services 300
WebSphere MQ objects for replication 43
XML publications
from Replication Center 173
multiple 161
overview 158
single 159

D

data conversion
event publishing, description 36
overview 35
Q replication, description 35
data exceptions, overview 252
data messages 431
data transformation, Q replication
stored procedures, description 107
stored procedures, overview 106
stored procedures, writing 108
data types
BLOB, replicating 172
CLOB, replicating 172
considerations
when publishing 171
when replicating 171
DBCLOB, replicating 172
LOB, replicating 172
restrictions 171

database
  configuring for Linux, UNIX, Windows 68
  configuring for Q Apply target 71
  configuring for Q Capture source 69
  configuring for z/OS 75
DATALINK data types 171
DB2 extenders
  restrictions 172
DB2 UDB log
  storage requirements 32
DB2CODEPAGE
  environment variable
    setting for replication programs 68
DB2INSTANCE
  environment variable
    setting for replication programs 68
DBCLOB (double-byte character large object) data types
  column values 446
  large object (LOB) XML message 446
  memory 31
  message buffer size 31
  replicating 172
DBHEAP database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
deactivate subscription XML message 463
deactivating
  Q subscriptions 222
  XML publications 222
deadlock_retries parameter, Q Apply 231, 323
default values
  Q Apply parameters 229
  Q Capture parameters 205
DEFPSIST parameter, WebSphere MQ 56
DEFSOPT parameter, WebSphere MQ 56
DEFTYPE parameter, WebSphere MQ 56
deleting
  publishing queue maps 197
  Q subscriptions 186
  replication queue maps 188
  XML publications 196
diagnostic files
  storage requirements 32
difference table 289
DISCINT parameter, WebSphere MQ 56
distinct data types 171
dropping
  control tables 188, 198
  schemas
    Q Capture 198
    Q Capture and Q Apply 188
DSNUTILS stored procedure for z/OS 146

E

e-mail notification, replication 261
EDITPROC clauses 171
email_server parameter, Replication Alert Monitor 270
enabling
  change capture 69
end-to-end latency 248, 250
environment variables
  setting
    replication 68
error messages
  from a Q Apply program 248
  from Q Capture program 248
  from the Replication Alert Monitor 248
error report XML message 451
errors
  handling for Q replication 103
  handling in unidirectional replication 100
  handling logic for stored procedures 110
  monitor_errors parameter 270
  monitoring with alert conditions 255
  operational 274
  peer-to-peer replication 120
  queue errors
    publishing queue maps 156
    replication queue maps 84
  recovering 286
  replication
    alert conditions, APPLY_ERRORS 258
    alert conditions, CAPTURE_ERRORS 258
    alert conditions, QAPPLY_ERRORS 258
    alert conditions, QCAPTURE_ERRORS 258
  replication queue maps 184
  replication service 301
  SQL 239, 258
  SQL states 183
  XML message 451
event publishing
  adding columns to XML publications 193
  authorization requirements
    administration 40
    overview 37
    Q Capture 37
    Replication Alert Monitor 39
  changing 191
  code pages 36
  columns included in message 168
  command-line processor 469
  compression dictionaries 282
  configuring servers, overview 67
  connectivity requirements 67
  control tables
    creating 76
    dropping 198
    illustration 361
    maintaining 283
    overview 361
    pruning 284
    reorganizing 285
  creating
    publishing queue maps 156
event publishing *(continued)*
  creating *(continued)*
    single XML publication 159
    XML publications 161
  creating XML publications 158
  data conversion
    description 36
    overview 35
  databases, configuring
    for Linux, UNIX 68
    for Windows 68
    for z/OS 75
  description 9
  environment variables, setting 68
  errors, recovering 286
  grouping publishing queue maps and XML publications 155
  including before values 169
  introduction 9
  key columns for XML publications 167
  maintaining
    cold starts, preventing 203
    log data, retaining 279, 281
    overview 277
    source logs, retaining 278
    source objects 278
    source systems 277
  memory, Q Capture program 26
  message content 9
  object names
    case 312
    naming rules 309
  objects 9
  packages, binding
    overview 72
    Q Capture 72
    Replication Alert Monitor 72
  planning
    memory and storage 25
    memory, overview 25
    storage requirements 31
  programs
    introduction 15
    Q Capture program 15
  Q Capture program 15
  queue manager clustering 64
  rebinding packages and plans 287
  restrictions, data types 171
  RUNSTATS utility 285
  sample programs
    Linux, UNIX, Windows 465
    overview 465
    z/OS 467
  set up 155
  setting up passwords 37
  source columns for XML publications 163
  specifying queues for event publishing 156
  storage requirements
    WebSphere MQ 61
  subsetting source columns 163
  system services 293
  viewing reports about 245
  WebSphere MQ
    objects 43, 48

event publishing *(continued)*
WebSphere MQ *(continued)*
setting up, overview 43
when Q Capture sends a message 163
XML messages
activate subscription message 463
add column message 458
control messages 431
data messages 431
deactivate subscription message 463
delimiters in character data 432
error report message 451
heartbeat message 453
informational messages 431
invalidate send queue message 461
large object (LOB) message 446
list of messages from Q Capture 434
list of types 431
load done message 462
load done received message 450
msg: root element 435, 460
overview of types 431
row operation message 445
structure of, from Q Capture 433
structure of, to Q Capture 459
structure, overview 431
subscription deactivated message 449
subscription schema message (subSchema) 454
technical requirements 432
to Q Capture 459
transaction message 436
XML publications 11
filtering rows 164
search conditions 164
WHERE clauses 164
event publishing commands
asnqanalyze 336
asnqcap 315
asnqccmd 320
asnqfmt 358
asnslist 342
asntdiff 355
asntrep 357
list 313
roadmap 313
exceptions
for replication, overview 252
EXPORT and IMPORT utilities 145
EXPORT and LOAD utilities 145
EXTSHM
environment varable
setting for replication programs 68

F

FIELDPROC clauses 171
filtering
rows
search conditions, event publishing 164
filtering *(continued)*
rows *(continued)*
search conditions, Q replication 94
filtering data with search conditions 15

G

grouping replication objects
publishing queue maps and XML publications 155
replication queue maps and Q subscriptions 83

H

HBINT parameter
in publishing queue maps 156
in replication queue maps 84
WebSphere MQ 56
heartbeat interval
in publishing queue maps 12, 156
in replication queue maps 6, 84
heartbeat XML message 453
holdl threads 246
horizontal (row) subsetting
Q replication 94
Q subscriptions 94
housekeeping threads 246

I

IBMQREP_ADMINMSG control table 371
IBMQREP_APPENQ control table 392
IBMQREP_APPLYMON control table 393
IBMQREP_APPLYPARMS control table
description 395
updating 237
IBMQREP_APPLYTRACE control table 397
IBMQREP_CAPENQ control table 371
IBMQREP_CAPMON control table 372
IBMQREP_CAPPARMS control table
description 374
updating 219
IBMQREP_CAPQMON control server 377
IBMQREP_CAPTRACE control table 378
IBMQREP_DELTOMB control table 398
IBMQREP_DONEMSG control table 399
IBMQREP_EXCEPTIONS control table 400
IBMQREP_RECVQUEUES control table 402
IBMQREP_SAVERI control table 404
IBMQREP_SENDQUEUES control table 380
IBMQREP_SIGNAL control table 382
IBMQREP_SPILLEDROW control table 405
IBMQREP_SPILLQS control table 406
IBMQREP_SRC_COLS control table 386
IBMQREP_SRCH_COND control table 379
IBMQREP_SUBS control table 387
IBMQREP_TARGETS control table 408
IBMQREP_TRG_COLS control table 406
IBMSNAP_ALERTS control table 414
IBMSNAP_CONDITIONS control table 416
IBMSNAP_CONTACTGRP control table 421
IBMSNAP_CONTACTS control table 422
IBMSNAP_GROUPS control table 423
IBMSNAP_MONENQ control table 423
IBMSNAP_MONPARMS control table 423
IBMSNAP_MONSERVERS control table 425
IBMSNAP_MONTRAIL control table 427
informational messages 431
invalidate send queue XML message 461
invocation parameters
Q Apply program 227
Q Capture program 201
Replication Alert Monitor
for UNIX 330
for Windows 330
for z/OS 330

J

Java Runtime Environment (JRE) 76
JCL
modifying replication commands 296
modifying replication programs 296
sample programs for Q replication 468
starting a Q Apply program 295
starting a Q Capture program 294
starting the Replication Alert Monitor 295
JCL batch mode 296
JRE (Java Runtime Environment) 76

K

key columns
for event publishing 167
for Q replication 99
for Q subscriptions 99
for XML publications 167

L

large object (LOB) data types
restrictions for bidirectional replication 129
large object (LOB) XML message 446
latency
overview 250
Q Apply program 248
Q Capture program 248
load done received XML message 450
load done XML message 462
LOAD from CURSOR utility 145

load options, replication
  automatic 145
    considerations for z/OS 146
    overview 144
  bidirectional replication 150
  manual 148
  no 150
  overview 143
  peer-to-peer replication
    three or more servers 150
    two servers 150
  unidirectional replication 150
LOB (large object) data types
  column values 446
  large object (LOB) XML message 446
  memory 31
  message buffer size 31
  queue depth considerations 64
  replicating 172
  restrictions for bidirectional replication 129
LOCKLIST database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
LOCKTIMEOUT database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
log data
  retaining
    for Linux, UNIX 279
    for Windows 279
    z/OS 281
log reading
  checking Q Capture progress 248
LOGBUFSZ database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
LOGFILSIZ database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
logical table
  bidirectional replication 117
  peer-to-peer replication 120
LOGPRIMARY database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
LOGRETAIN database configuration parameter 69
logreuse parameter
  Q Apply 231, 323
  Q Capture 206
LOGSECOND database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
logstdout parameter
  Q Apply 231, 323
  Q Capture 206, 315
LONG VARCHAR data types 171
LONG VARGRAPHIC data types 171

M

maintaining
  event publishing environment
    control tables 283
    log data, retaining 279, 281
    overview 277
    source objects 278
    target tables 286
  Q replication environment
    cold starts, preventing 203
    control tables 283
    log data, retaining 279, 281
    overview 277
    source objects 278
    target tables 286
manipulating data
  Q replication
    stored procedures, description 107
    stored procedures, overview 106
    stored procedures, writing 108
  stored procedures for Q subscriptions
    description 107
    overview 106
    writing 108
manual load
  definition 143
  description 148
mapping
  source columns to target columns
    Q replication 98
max_notification_minutes parameter, Replication Alert Monitor 270
max_notifications_per_alert parameter, Replication Alert Monitor 270
MAXAPPLS database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
MAXDEPTH parameter, WebSphere MQ 56
MAXLOCKS database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
MAXMSGL parameter, WebSphere MQ 56
memory
  alert conditions
    APPLY_MEMORY 258
    CAPTURE_MEMORY 258
    QAPPLY_MEMORY 258
    QCAPTURE_MEMORY 258
  exceeded by Q Capture program 33
  LOB data types 31
  planning
    event publishing, overview 25
    Q replication, overview 25
  Q Apply program 29
  Q Capture program 26
  Replication Alert Monitor 30
memory_limit parameter, Q Capture 206, 315
message buffer
  large object (LOB) values 64
message processing
  starting for Q Apply 239
message processing *(continued)*
  stopping for Q Apply 241
messages
  formatting and viewing 358
  from a Q Apply program 248
  from a Q Capture program
    for Q subscriptions 93
  from Q Capture program
    historical and performance data 248
    when Q Capture sends a message for XML publications 163
  from the Replication Alert Monitor 248
  WebSphere MQ
    size limit 60
MODIFY command 296
Monitor control server
  control tables
    list 414
  IBMSNAP_ALERTS control table 414
  IBMSNAP_CONDITIONS control table 416
  IBMSNAP_CONTACTGRP control table 421
  IBMSNAP_CONTACTS control table 422, 423
  IBMSNAP_MONENQ control table 423
  IBMSNAP_MONPARMS control table 423
  IBMSNAP_MONSERVERS control table 425
  IBMSNAP_MONTRAIL control table 427
  list of control tables 369
monitor qualifier
  naming rules 309
  replication 255
monitor_errors parameter, Replication Alert Monitor 270
monitor_interval parameter
  Q Apply 231, 323
  Q Capture 206, 315
monitor_limit parameter
  Q Apply 231, 323
  Q Capture 206, 315
  Replication Alert Monitor 270
monitor_path parameter, Replication Alert Monitor 270
monitoring
  replication 255, 265
msg root element for XML messages 435, 460
MSGDLVSQ parameter, WebSphere MQ 56
multidirectional replication
  bidirectional replication 117
  choosing between bidirectional and peer-to-peer replication 125
  peer-to-peer replication 120
  setting up 117

N

naming conventions
  target object profiles for Q replication 87
naming rules
  for Q replication and event publishing 309
nicknames
  replication
    specifying for automatic load 146
no load
  definition 143
  description 150
NUMTCB parameter for z/OS loads 146

O

OKSQLSTATES
  changing in Q replication 183
  peer-to-peer replication 120
old data values
  included in XML publications 169
operating
  Q Apply program 227, 326
  Q Capture program 201, 320
  Replication Alert Monitor 265
operation parameter, stored procedures 110

P

packages
  binding
    overview 72
    Q Apply program (Linux, UNIX, Windows) 72, 73
    Q Capture program (Linux, UNIX, Windows) 72, 73
    Replication Alert Monitor (Linux, UNIX, Windows) 72, 74
parallel loads on z/OS 146
parallelism
  using Q replication 18
  using the Q Apply program 18
parameters
  Q Apply program
    apply_path 231
    apply_schema 231
    apply_server 231
    autostop 231
    changing 18, 237
    changing saved 237
    changing while running 238, 326
    deadlock_retries 231
    default values 229
    logreuse 231
    logstdout 231
    monitor_interval 231
    monitor_limit 231
    overview 229
    pwdfile 231
    qmgr 231
    term 231
    trace_limit 231
  Q Capture program
    add_partition 206

parameters *(continued)*
  Q Capture program *(continued)*
    adminq 206
    autostop 206
    capture_path 206
    capture_schema 206
    capture_server 206
    changing 217
    changing saved 219
    changing while running 220, 320
    commit_interval 206
    default values 205
    descriptions 206
    logreuse 206
    logstdout 206
    memory_limit 206
    methods of changing 217
    monitor_interval 206
    monitor_limit 206
    overview 205
    prune_interval 206
    qmgr 206
    restartq 206
    signal_limit 206
    sleep_interval 206
    startmode 206
    term 206
    trace_limit 206
  Replication Alert Monitor
    alert_prune_limit 270
    autoprune 270
    default values 269
    description 270
    email_server 270
    max_notification_minutes 270
    max_notifications_per_alert 270
    monitor_errors 270
    monitor_limit 270
    monitor_path 270
    runonce 270
    trace_limit 270
  stored procedures, replication
    identifying suppressed columns 112
    identifying transaction 112
    identifying type of operation 110
    map to source columns 113
    mapping to key columns, Q subscriptions 99
    Q replication, description 107
    Q replication, overview 106
    writing for Q replication 108
  WebSphere MQ 56
parameters, invocation
  Q Apply program 227, 323
  Q Capture program 201, 315
  Replication Alert Monitor
    for UNIX 330
    for Windows 330
    for z/OS 330
passwords
  setting up, overview 37
  storing for remote servers 41
peer-to-peer replication
  conflict detection 120
  creating replication queue maps 84
  description 3, 120

peer-to-peer replication *(continued)*
  error options 103
  Q subscriptions
    creating for three or more servers 137
    creating for two servers 135
    overview 135
  referential integrity 120
  replication objects
    three or more servers 120
    two servers 120
  specifying queues 84
  starting
    three or more servers 140
    two servers 139
  stopping
    three or more servers 141
    two servers 140
  three or more remote servers
    WebSphere MQ objects 53
  three or more servers
    load options 150
    objects 120
    starting 140
    stopping 141
  two remote servers
    WebSphere MQ objects 50
  two servers
    load options 150
    objects 120
    starting 139
    stopping 140
  versioning columns 429
performance, Q Capture and Q Apply programs 248
planning
  memory
    event publishing, overview 25
    Q replication, overview 25
  storage requirements
    event publishing, overview 25, 31
    Q replication, overview 25, 31
    WebSphere MQ 61
problem rows 252
profiles
  target object profiles for Q replication 87
prune intervals
  Replication Alert Monitor 273
prune parameter
  Q Apply 326
  Q Capture 320
prune threads 246
prune_interval parameter
  Q Apply 323
  Q Capture 206, 315
pruning
  Q Apply program
    asnqacmd command 326
    IBMQREP_APPLTRACE control table 284
    IBMQREP_APPLYMON control table 284
    monitor_limit parameter 284
    trace_limit parameter 284

pruning *(continued)*
Q Capture
IBMQREP_CAPQMON control table 284
Q Capture program
IBMQREP_CAPMON control table 284
IBMQREP_CAPTRACE control table 284
IBMQREP_SIGNAL control table 284
monitor_limit parameter 284
signal_limit parameter 284
trace_limit parameter 284
publishing
before values in XML publications 169
converted data 36
event publishing
columns included in message 168
when Q Capture sends a message 163
grouping publishing queue maps and XML publications 155
setting up event publishing 155
subset of data
filtering rows in XML publications 164
subset source columns for XML publications 163
with the Q Capture program 15
columns included in XML publication 168
when Q Capture sends a message 163
XML publications
before values 169
columns included in message 168
description 11
when Q Capture sends a message 163
publishing queue maps
changing 195
creating 156
deleting 197
description 12
grouping with XML publications 155
in event publishing 9
pwdfile parameter
Q Apply 323
Q Capture 315
pwdfile parameter, Q Apply 231

Q

Q Apply control tables
creating 76
dropping 188
list 368, 391
Q Apply latency 250
Q Apply program
agent threads
defined 246
description 18
in replication queue maps 84
alert conditions 258
applying transactions in parallel 18

Q Apply program *(continued)*
asnqacmd command 326
authorization requirements 38
bidirectional replication 117
browser threads
defined 246
description 18
checking run-time parameter values 326
checking status 245, 326
code pages 35
conflict detection, bidirectional replication 129
control tables, creating 76
data conversion 35
description 18
error options 103
handling unexpected conditions 100
introduction 15
invocation parameters 323
memory 29
operating 227
overview 18
packages, binding 72
parameters
apply_path 231
apply_schema 231
apply_server 231
autostop 231
changing 237
changing saved 237
changing while running 238
deadlock_retries 231
default values 229
description 229, 231
logreuse 231
logstdout 231
monitor_interval 231
monitor_limit 231
pwdfile 231
qmgr 231
term 231
trace_limit 231
peer-to-peer replication 120
prune command 326
pruning control tables 284
running
as system-started task 296
in batch (z/OS) 296
scheduling 303
for Linux, UNIX 303
for Windows 304
for z/OS 305
schema
description 20
rules for bidirectional replication 117
rules for peer-to-peer replication 120
starting
asnqapp command 323
description 227
with JCL 295
starting message processing 239, 326
stopping 242, 326
stopping message processing 241, 326

Q Apply program *(continued)*
storage requirements 62
stored procedures
description 107
overview 106
target database, configuring 71
trace files 34
unidirectional replication 81
working with receive queues 83
Q Apply schema
naming rules 309
rules for bidirectional replication 117
rules for peer-to-peer replication 120
Q Apply server
IBMQREP_APPENQ control table 392
IBMQREP_APPLYMON control table 393
IBMQREP_APPLYPARMS control table 395
IBMQREP_APPLYTRACE control table 397
IBMQREP_DELTOMB control table 398
IBMQREP_DONEMSG control table 399
IBMQREP_EXCEPTIONS control table 400
IBMQREP_RECVQUEUES control table 402
IBMQREP_SAVERI control table 404
IBMQREP_SPILLEDROW control table 405
IBMQREP_SPILLQS control table 406
IBMQREP_TARGETS control table 408
IBMQREP_TRG_COLS control table 406
list of control tables 368, 391
Q Apply throughput 248
Q Capture control tables
creating 76
dropping 188, 198
list 367, 370
Q Capture latency 250
Q Capture messages
before values in XML publications 169
columns included in XML message 168
when Q Capture sends a message
for Q subscriptions 93
for XML publications 163
Q Capture program
alert conditions 258
authorization requirements 37
before values in XML publications 169
bidirectional replication 117
capturing changes 15
changing parameters 15
checking run-time parameter values 320
checking status 245, 320
code pages 35
for event publishing 36

Q Capture program *(continued)*
cold starts
effect on Q Subscriptions 221
effect on XML publications 221
preventing 203
restart message 15
startmode parameter 206
columns sent in XML publications 168
communicating with 15
control tables, creating 76
data conversion
for event publishing 36
for Q replication 35
default parameter values 205
description 15
introduction 15
memory 26, 33
operating 201
overview 15
packages, binding 72
parameters 205
add_partition 206
adminq 206
autostop 206
capture_path 206
capture_schema 206
capture_server 206
changing 217
changing saved 219
changing while running 220
commit_interval 206
description 206
logreuse 206
logstout 206
memory_limit 206
methods of changing 217
monitor_interval 206
prune_interval 206
qmgr 206
restartq 206
signal_limit 206
sleep_interval 206
startmode 206
term 206
trace_limit 206
peer-to-peer replication 120
prune command 320
pruning control tables 284
reinitializing 320
reinitializing a send queue 320
restart messages
description 15
how the Q Capture program uses 15
running
as system-started task 296
in batch (z/OS) 296
scheduling 303
for Linux, UNIX 303
for Windows 304
for z/OS 305
Q Capture messages 93
schema
description 20
rules for bidirectional replication 117

Q Capture program *(continued)*
schema *(continued)*
rules for peer-to-peer replication 120
sending commands to a running 320
source database, configuring for Linux, UNIX, Windows 69
starting
asnqcap command 315
description 201
JCL 294
stopping 224, 320
trace files, storage requirements 34
unidirectional replication 81
viewing historical data 248
when Q Capture sends a message
for Q subscriptions 93
for XML publication 163
working with send queues
in event publishing 155
in Q replication 83
XML messages 434, 459
Q Capture schema
naming rules 309
rules for bidirectional replication 117
rules for peer-to-peer replication 120
Q Capture server
code pages for event publishing 36
IBMQREP_ADMINMSG control table 371
IBMQREP_CAPENQ control table 371
IBMQREP_CAPMON control table 372
IBMQREP_CAPPARMS control table 374
IBMQREP_CAPQMON control table 377
IBMQREP_CAPTRACE control table 378
IBMQREP_SENDQUEUES control table 380
IBMQREP_SIGNAL control table 382
IBMQREP_SRC_COLS control table 386
IBMQREP_SRCH_COND control table 379
IBMQREP_SUBS control table 387
list of control tables 367, 370
Q Capture throughput
checking memory usage 26
monitoring 248
Q Capture transaction latency 248, 250
Q replication
adding columns to existing Q subscriptions 181
ASNCLP program 469
authorization requirements
for administration 40
for Q Apply 38
for Q Capture 37
for Replication Alert Monitor 39
overview 37
bidirectional replication
conflict detection 117, 129
creating Q subscriptions 127
description 3, 117

Q replication *(continued)*
bidirectional replication *(continued)*
starting 139
stopping 140
changing 179
code pages 35
command-line processor 469
compression dictionaries 282
configuring servers, overview 67
conflict detection
bidirectional replication 117
peer-to-peer replication 120
conflict detection, bidirectional replication 129
connectivity requirments 67
control tables
creating 76
dropping 188
illustration 361
list, Q Apply 368
list, Q Capture 367
list, Replication Alert Monitor 369
maintaining 283
overview 361
pruning 284
reorganizing 285
data conversion
description 35
overview 35
databases, configuring
for Linux, UNIX 68
for Windows 68
for z/OS 75
description 3
environment variables, setting 68
error options 103
errors, recovering 286
grouping replication queue maps and Q subscriptions 83
introduction 3
maintaining
cold starts, preventing 203
log data, retaining 279, 281
overview 277
source logs, retaining 278
source objects 278
source systems 277
target tables 286
memory
Q Apply program 29
Q Capture program 26
multidirectional replication
bidirectional replication 117
choosing between bidirectional and peer-to-peer replication 125
peer-to-peer replication 120
setting up 117
object names
case 312
naming rules 309
packages
Replication Alert Monitor (Linux, UNIX, Windows) 72
packages, binding
overview 72
Q Apply (Linux, UNIX, Windows) 72

Q replication *(continued)*
packages, binding *(continued)*
Q Capture (Linux, UNIX, Windows) 72
peer-to-peer replication
conflict detection 120
creating Q subscriptions, overview 135
creating Q subscriptions, three or more servers 137
creating Q subscriptions, two servers 135
description 3, 120
starting, three or more servers 140
starting, two servers 139
stopping, three or more servers 141
stopping, two servers 140
versioning columns 429
planning
memory and storage 25
storage requirements 31
programs
introduction 15
Q Apply program 18
Q Capture 15
schemas 20
Q Apply program 18
bidirectional replication 117
peer-to-peer replication 120
schemas 20
unidirectional replication 81
Q Capture program 15
bidirectional replication 117
peer-to-peer replication 120
schemas 20
unidirectional replication 81
Q subscriptions
bidirectional 127
description 5
error options 103
filtering rows 94
naming rules 309
peer-to-peer, three or more servers 137
peer-to-peer, two servers 135
search conditions 94
unidirectional 90
WHERE clauses 94
queue manager clustering 64
rebinding packages and plans 287
recommendations for loading target tables 143
replication queue maps 6
restrictions, data types 171
RUNSTATS utility 285
sample programs
JCL 468
Linux, UNIX, Windows 465, 466
overview 465
z/OS 467
setting up passwords 37
specifying queues 84
storage requirements
DB2 UDB logs 32
spill queues 62

Q replication *(continued)*
storage requirements *(continued)*
trace files 34
WebSphere MQ 61
stored procedures
overview 106
parameters, identifying transaction 112
parameters, map to source columns 113
parameters, suppressed columns 112
parameters, type of operation 110
writing 108
system services 293
target object profiles 87
unidirectional replication
creating a single Q subscription 88
creating Q subscriptions, multiple 90
creating Q subscriptions, overview 86
description 3, 81
filtering rows 94
index columns for targets 99
key columns for targets 99
mapping source and target columns 98
Q Capture messages 93
search conditions 94
setting up 81
source columns for Q subscriptions 92
subsetting source columns 92
unexpected conditions in targets 100
viewing reports about 245
WebSphere MQ
objects 43
setting up, overview 43
Q Replication Analyzer
description 336
parameters 336
Q replication commands
asnqacmd 326
asnqanalyze 336
asnqapp 323
asnqcap 315
asnqccmd 320
asnqfmt 358
asnslist 342
asntdiff 355
asntrep 357
list 313
road map 313
Q subscriptions
activating 221
adding columns to existing 181
bidirectional replication 117
CAPSTART signal 221
CAPSTOP signal 222
changing attributes 179
conflict detection, bidirectional replication 129
creating
bidirectional replication 127

Q subscriptions *(continued)*
creating *(continued)*
peer-to-peer replication, overview 135
peer-to-peer replication, three or more servers 137
peer-to-peer replication, two servers 135
unidirection replication, single 88
unidirectional replication, multiple 90
unidirectional replication, overview 86
deactivating 222
deleting 186
description 5
error options 103
filtering rows 94
grouping with replication queue maps 83
in bidirectional replication
conflict detection 129
index columns for targets 99
key columns for targets 99
naming rules 309
options
mapping source and target columns 98
Q Capture messages 93
peer-to-peer replication 120
search conditions 94
specifying nicknames for automatic load 146
stored procedures
description 107
overview 106
parameters, identifying transaction 112
parameters, map to source columns 113
parameters, suppressed columns 112
parameters, type of operation 110
subsetting source columns 92
target object profiles 87
unexpected conditions in targets 100
unidirectional replication 81
WHERE clauses 94
qmgr parameter
Q Apply 231
Q Capture 206
qryparms parameter
Q Apply 326
Q Capture 320
queue depth for large object (LOB) values 64
queue latency 248, 250
queue manager
description 44
required settings 56
queue manger
clustering 64
queue maps
publishing
changing 195
changing attributes 195
creating 12

queue maps *(continued)*
  publishing *(continued)*
    deleting 197
    description 12
  replication
    changing 184
    creating 84
    deleting 188
    description 6
queues
  administration
    for Q Apply 44
    for Q Capture 44
  in publishing queue maps 12, 156
  in replication queue maps 6, 84
  receive
    description 44
    starting message processing 239
    stopping message processing 241
  restart 44
  send 44
  spill 44

R

receive queues
  description 44
  in replication queue maps 84
  memory for Q replication 29
  naming rules 309
  replication queue maps 6
  required settings 56
  specifying queues for Q replication 84
  starting message processing 239
  stopping message processing 241
  working with the Q Apply program 83
referential constraints
  peer-to-peer replication 120
referential integrity
  peer-to-peer replication 120
reinit parameter
  Q Capture 320
reinitializing
  publishing queue maps 195
  Q Apply program 173
  Q Capture program 179, 191
  Q subscriptions 179
  queues 173, 184
  Replication Alert Monitor 173, 268
  XML publications 191
reinitq parameter
  Q Apply 326
  Q Capture 320
remote servers
  storing passwords and user IDs 41
replication
  bidirectional replication
    conflict detection 129
    description 117
  BLOB data types 172
  CLOB data types 172
  converted data 35
  DBCLOB data types 172
  mapping source and target columns
    Q replication 98
replication *(continued)*
  multidirectional replication, setting up 117
  peer-to-peer replication 120
  Q Apply program 18
  Q Capture program 15
  Q subscriptions
    creating for bidirectional replication 127
    creating for peer-to-peer replication 135
    creating for peer-to-peer replication, three or more servers 137
    creating for peer-to-peer, two servers 135
    creating for unidirectional replication, multiple 90
    creating for unidirectional replication, overview 86
    creating one for unidirectional replication 88
    description 5
    messages from Q Capture 93
    stored procedures, description 107
    stored procedures, overview 106
  ROWID columns 172
  subset of data
    filtering rows in Q subscriptions 94
    source columns for Q replication 92
  to stored procedure in Q subscription 113
  unidirectional replication
    description 81
    setting up 81
    unexpected conditions 100
    using stored procedures, overview 106
Replication Alert Monitor
  alert conditions
    e-mail notifications 261
    events 255
    list 258
    overview 257
    selecting 267
    status 255
    thresholds 255
  alerts 255
  authorization requirements 39
  checking status 245
  contact groups 255
  contacts 255
  control tables
    IBMSNAP_ALERTS 414
    IBMSNAP_CONDITIONS 416
    IBMSNAP_CONTACTGRP 421
    IBMSNAP_CONTACTS 422
    IBMSNAP_GROUPS 423
    IBMSNAP_MONENQ 423
    IBMSNAP_MONPARMS 423
    IBMSNAP_MONSERVERS 425
    IBMSNAP_MONTRAIL 427
  control tables, creating 263
  defining contact information 264
Replication Alert Monitor *(continued)*
  description 255
  memory 30
  monitoring replication, overview 255
  monitors
    creating 266
    reinitializing 268
  operating 265
  packages, binding 72
  parameters
    alert_prune_limit 270
    autoprune 270
    default values 269
    descriptions 270
    email_server 270
    how often the Replication Alert Monitor runs 273
    max_notification_minutes 270
    max_notifications_per_alert 270
    monitor_errors 270
    monitor_interval 270
    monitor_limit 270
    monitor_path 270
    notification criteria for alert conditions 274
    notification criteria for operational errors 274
    prune intervals for data 273
    runonce 270
    setting 272
    trace_limit 270
  pruning control tables 284
  reinitializing 268
  running
    as system-started task 296
    in batch (z/OS) 296
  scheduling 303
    for Linux, UNIX 303
    for Windows 304
    for z/OS 305
  setting up 263
  starting 268
  starting with JCL 295
  stopping 275
  viewing messages 248
Replication Center
  operational commands 173
  running commands 175
  software prerequisites 76
  SQL scripts 173
  target object profiles 87
replication commands
  asnqacmd 326
  asnqanalyze 336
  asnqapp 323
  asnqcap 315
  asnqccmd 320
  asnqmfmt 358
  asnslist 342
  asntdiff 355
  asntrep 357
  for z/OS
    MODIFY 296
  running and saving from Replication Center 175
replication programs
  checking performance 245

replication programs *(continued)*
  in event publishing 15
  in Q replication 15
  Q Apply program
    description 18
    schemas 20
  Q Capture program
    description 15
    schemas 20
  scheduling 303
  schemas 20
replication queue maps
  bidirectional replication 117
  changing attributes 184
  creating 84
  deleting 188
  description 6
  grouping with Q subscriptions 83
  peer-to-peer replication 120
  unidirectional replication 81
replication services
  creating 300
  description 299
  display name 299
  dropping 303
  listing 342
  managing 299
  name 299
  starting 301
  stopping 302
  viewing list 302
replication source tables
  in Q subscriptions 5
  in XML publications 11
replication target tables
  in Q subscriptions 5
restart messages, Q Capture program 15
restart queues
  description 44
  how Q Capture uses 15
restartq parameter, Q Capture 206
restrictions
  abstract data types 171
  data types 171
  DATALINK values 171
  DB2 Extenders large objects 172
  distinct data types 171
  EDITPROC clauses 171
  FIELDPROC clauses 171
  LONG VARCHAR data types 171
  spatial data types 171
  user-defined data types 171
  VALIDPROC clauses 171
retaining log data
  for Linux, UNIX 279
  for Windows 279
  z/OS 281
row (horizontal) subsetting 94
  event publishing 164
row operation XML message 445
ROWID columns, replicating 172
rows
  filtering
    event publishing 164
    Q replication 94
run-time parameters
  changing for Q Apply 238
run-time parameters *(continued)*
  changing for Q Capture 220
runonce parameter, Replication Alert Monitor 270
RUNSTATS utility 285

S

sample programs
  event publishing
    Linux, UNIX, Windows 465
    overview 465
    z/OS 467
  Q replication
    JCL 468
    Linux, UNIX, Windows 465, 466
    overview 465
    z/OS 467
schemas
  for the Q Apply program
    description 20
    dropping 188
    introduction 15
    rules for bidirectional replication 117
    rules for peer-to-peer replication 120
  for the Q Capture program
    description 20
    dropping 188, 198
    introduction 15
    rules for bidirectional replication 117
    rules for peer-to-peer replication 120
  naming rules
    for bidirectional replication 117
    for peer-to-peer replication 120
SCM (Service Control Manager)
  description 299
send queues
  description 44
  event publishing 156
  in replication queue maps 6
  naming rules 309
  publishing queue maps 12, 156
  replication queue maps 84
  required settings 56
  specifying queues for Q replication 84
  working with the Q Capture program
    in event publishing 155
    in Q replication 83
serialization threads 246
Service Control Manager (SCM)
  description 299
setting up
  multidirectional replication 117
  passwords 37
  Replication Alert Monitor 263
  unidirectional replication 81
  user ID's 37
  WebSphere MQ for Q replication 43
SHARE parameter, WebSphere MQ 56
signal_limit parameter, Q Capture 206, 315
sleep_interval parameter, Q Capture 206, 315
software reqirements
  Replication Center 76
source columns
  mapping to stored procedure 113
  mapping to target columns
    Q replication 98
  parameter identifying suppressed columns 112
  published in XML publication 168
  publishing a subset
    event publishing 163
    XML publications 163
  replicating a subset 92
source database
  configuring 69
source logs, retaining 278
source servers
  code pages
    for event publishing 36
    for Q replication 35
  specifying
    for bidirectional replication 127
    for event publishing 161
    for peer-to-peer replication, three or more servers 137
    for peer-to-peer replication, two servers 135
    for unidirectional replication 88
    for unidirectional replication, multiple 90
    for XML publications 159
  when Q Capture sends a message
    for Q subscriptions 93
    for XML publications 163
source systems
  definition 277
  maintaining 277
source tables
  adding columns 181, 193
  for bidirectional replication 117, 127
  for peer-to-peer replication 120
  for peer-to-peer replication with two servers 135
  for peer-to-peer replication, three or more servers 137
  for unidirectional replication
    description 81
    specifying 90
    specifying one 88
  for XML publications
    multiple tables 161
    single table 159
  in Q subscriptions 5
    filtering rows 94
    mapping source and target columns 98
    Q Capture messages 93
    search conditions 94
  in XML publications 11
    filtering rows 164
    search conditions 164
    when Q Capture sends a message 163
    WHERE clauses 164
  maintaining 278

source tables *(continued)*
  mapping to stored procedures 113
  mapping to targets, Q replication 98
  naming rules 309
source-target pairs
  for bidirectional replication
    creating 127
    description 117
  for peer-to-peer replication
    creating for two servers 135
    description 120
  for peer-to-peer with three or more servers 137
  for unidirectional replication 81
    creating 90
    creating one 88
  mapping columns, Q replication 98
spatial data types
  restrictions 171
spill files
  storage for Q Apply program 62
spill queues
  description 44
  required settings 56
  storage requirements 62
SQL errors
  acceptable 252
  unexpected 252
SQL scripts, running and saving from Replication Center 174
SQL states
  acceptable in Q replication
    changing 183
    setting for peer-to-peer 120
SQLSTATE
  acceptable in Q replication
    changing 183
    setting for peer-to-peer 120
starting
  bidirectional replication 139
  event publishing 221
  peer-to-peer replication
    three or more servers 140
    two servers 139
  Q Apply program 227, 323
  Q Capture program 201
  Replication Alert Monitor
    various methods 268
  unidirectional replication 221
startmode parameter, Q Capture 206, 315
startq parameter, Q Apply 326
statistics
  Q Apply program 248
  Q Capture program 248
status parameter
  Q Apply 326
  Q Capture 320
STMTHEAP database configuration parameter
  setting for Q Apply 71
  setting for Q Capture 69
stop parameter
  Q Apply 326
  Q Capture 320
stopping
  bidirectional replication 140
stopping *(continued)*
  event publishing 222
  peer-to-peer replication
    three or more servers 141
    two servers 140
  Q Apply program 173, 242
  Q Capture program 173, 224
  Q subscriptions 222
  Q subscriptions or XML publications 221
  Replication Alert Monitor 173, 275
  unidirectional replication 222
  XML publications 222
stopq parameter, Q Apply 326
storage
  DB2 UDB logs 32
  planning
    event publishing 25
    Q replication 25
  WebSphere MQ 61
storage requirements
  for diagnostic files 32
  planning
    event publishing, overview 31
    Q replication, overview 31
  Q Capture program 33
  trace files 34
stored procedures
  parameters
    identifying suppressed columns 112
    identifying the type of operation 110
    identifying transaction 112
    map to source columns 113
  Q replication
    description 107
    error options 103
    overview 106
    parameters for key columns 99
    unexpected conditions in targets 100
  Q subscriptions
    description 107
    error options 103
    overview 106
    parameters for key columns 99
    unexpected conditions in targets 100
  writing for Q replication 108
subscription deactivated XML message 449
subscription schema XML message 454
subsetting
  source columns
    Q subscriptions 92
    XML publications 163
  source rows
    Q subscriptions 94
    XML publications 164
suppression parameter
  stored procedure for Q replication 112
system commands
  asnqacmd 326
  asnqanalyze 336
  asnqapp 323
system commands *(continued)*
  asnqcap 315
  asnqccmd 320
  asnqmfmt 358
  asnslist 342
  asntdiff 355
  asntrep 357
  See replication commands 313
system-started tasks
  running replication programs 296

T

table differencing utility 289, 355
table repair utility 292, 357
tables
  IBMQREP_ADMINMSG 371
  IBMQREP_APPENQ 392
  IBMQREP_APPLYMON 393
  IBMQREP_APPLYPARMS 395
  IBMQREP_APPLYTRACE 397
  IBMQREP_CAPENQ 371
  IBMQREP_CAPMON 372
  IBMQREP_CAPPARMS 374
  IBMQREP_CAPQMON 377
  IBMQREP_CAPTRACE 378
  IBMQREP_DELTOMB 398
  IBMQREP_DONEMSG 399
  IBMQREP_EXCEPTIONS 400
  IBMQREP_RECVQUEUES 402
  IBMQREP_SAVERI 404
  IBMQREP_SENDQUEUES 380
  IBMQREP_SIGNAL 382
  IBMQREP_SPILLEDROW 405
  IBMQREP_SPILLQS 406
  IBMQREP_SRC_COLS 386
  IBMQREP_SRCH_COND 379
  IBMQREP_SUBS 387
  IBMQREP_TARGETS 408
  IBMQREP_TRG_COLS 406
  IBMSNAP_ALERTS 414
  IBMSNAP_CONDITIONS 416
  IBMSNAP_CONTACTGRP 421
  IBMSNAP_CONTACTS 422
  IBMSNAP_GROUPS 423
  IBMSNAP_MONENQ 423
  IBMSNAP_MONPARMS 423
  IBMSNAP_MONSERVERS 425
  IBMSNAP_MONTRAIL 427
target databases
  configuring for Q Apply 71
target key
  Q replication 99
  Q subscriptions 99
  XML publications 167
target object profiles 87
target servers
  specifying
    for bidirectional replication 127
    for peer-to-peer replication with two servers 135
    for peer-to-peer replication, three or more servers 137
    for unidirectional replication 88, 90
  target object profiles 87

target tables
  adding columns 181
  applying transactions 18
  authorization requirements 38
  connectivity requirements 67
  for bidirectional replication 117, 127
  for peer-to-peer replication 120
  for peer-to-peer replication with two servers 135
  for peer-to-peer replication, three or more servers 137
  for unidirectional replication
    description 81
    specifying multiple 90
    specifying one 88
    unexpected conditions 100
  IBMQREP_TARGETS control table 368, 408
  in Q subscriptions
    description 5
    unexpected conditions 100
  index columns, Q subscriptions 99
  key columns
    Q subscriptions 99
    XML publications 167
  load options
    automatic 145
    automatic load 143, 146
    bidirectional replication 150
    manual 148
    manual load 143
    no load 143, 150
    overview 143
    peer-to-peer replication 150
    recommendations 143
    unidirectional replication 150
  maintaining 286
  mapping to sources
    Q replication 98
  naming rules 309
  nicknames 146
  repairing 292
  storage space for changes 32
  target key
    Q subscriptions 99
    XML publications 167
  target object profiles 87
term parameter
  Q Apply 231, 323
  Q Capture 206, 315
threads
  checking status 245
  Q replication
    administration 246
    agent 18, 246
    browser 18, 246
    checking status 246
    holdl 246
    housekeeping 246
    prune 246
    serialization 246
    worker 246
throughput
  Q Apply program 248
  Q Capture program 248
trace files, storage requirements 34
trace_limit parameter
  Q Apply 231, 323
  Q Capture 206, 315
  Replication Alert Monitor 270
  use with asnmon command 332
transaction latency, Q Capture 250
transaction XML message 436
transactional parameters, stored procedure for Q replication 112
transforming data
  stored procedures for Q subscriptions
    overview 106
transforming data, Q replication
  stored procedures, description 107
  stored procedures, overview 106
  writing stored procedures 108
troubleshooting
  checking program status 246
troubleshooting commands
  asnqanalyze 336
TSO
  running replication programs in batch 296
  starting a Q Apply program 295
  starting a Q Capture program 294
turning on archive logging 69

U

unexpected conditions in targets
  handling with stored procedures 110
  Q subscriptions 100
  unidirectional replication 100
unidirectional replication
  creating replication queue maps 84
  description 3, 81
  error options 103
  load options 150
  Q subscriptions
    creating multiple 90
    creating one 88
    creating, overview 86
    filtering rows 94
    index columns for targets 99
    key columns for targets 99
    mapping source and target columns 98
    Q Capture messages 93
    search conditions 94
    source columns 92
    unexpected conditions in the targets 100
    WHERE clauses 94
  setting up
    description 81
    grouping replication queue maps and Q subscriptions 83
  specifying queues 84
  stored procedures
    description 107
    overview 106
    writing 108
  WebSphere MQ objects required
    remote 47
    same system 45
updating
  publishing queue maps 195
updating *(continued)*
  Q subscriptions 179
  replication queue maps 184
  XML publications 191
user applications
  list of XML messages from 459
user IDs
  storing for remote servers 41
user-defined types (UDTs)
  Q replication 171
utilities
  EXPORT and IMPORT 145
  EXPORT and LOAD 145
  LOAD from CURSOR 145, 146
  message formatter 358
  Q Replication Analyzer 336
  table differencing 355
  table repair 357

V

VALIDPROC clauses 171
value-based conflicts 252
version-based conflicts 252
versioning columns 429
vertical (column) subsetting
  event publishing 163
  Q replication 92
  Q subscriptions 92
  XML publications 163

W

warm start
  CAPSTOP signal 222
  effect on activate subscription XML message 221
  effect on CAPSTART signal 221
  effect on log data 278
  startmode parameter 205, 206
  startmode parameter, three types 315
WebSphere MQ
  authorization requirements
    Q Apply program 62
    Q Capture program 62
    Replication Alert Monitor 62
  message segmentation 60
  message size limit
    Q Apply 60
    Q Capture 60
  objects
    administration queue 44
    authorization requirements 62
    bidirectional replication 50
    connectivity requirments 62
    event publishing 48
    in publishing queue maps 12
    in replication queue maps 6
    list of 44
    peer-to-peer replication (three or more remote servers) 53
    peer-to-peer replication (two remote servers) 50
    receive queue 44
    required settings 56
    restart queue 44

WebSphere MQ *(continued)*
objects *(continued)*
send queue 44
spill queue 44
unidirectional replication, remote 47
unidirectional replication, same system 45
parameters
CHLTYPE 56
DEFPSIST 56
DEFSOPT 56
DEFTYPE 56
DISCINT 56
HBINT 56
MAXDEPTH 56
MAXMSGL 56
MSGDLVSQ 56
SHARE 56
publishing queue maps
creating 156
description 12
replication queue maps
creating 84
description 6
setting up event publishing 43
setting up Q replication 43
storage requirements 61
WHERE clause
filtering data
rows in Q subscriptions 94
rows in XML publications 164
Windows Service Control Manager (SCM)
asnslist command 342
description 299
listing replication services 342
replication services 299
worker threads 246

X

XML messages
activate subscription message 463
add column message 458
before values in XML publications 169
columns included in XML publications 168
control messages 431
data messages 431
deactivate subscription message 463
error report message 451
event publishing
delimiters in character data 432
technical requirements 432
to Q Capture 459
heartbeat message 453
informational messages 431
invalidate send queue message 461
large object (LOB) message 446
list of messages from Q Capture 434
list of types 431
load done message 462
load done received message 450
msg: root element 435, 460
overview of types 431
row operation message 445
XML messages *(continued)*
structure
from Q Capture 433
overview 431
to Q Capture 459
subscription deactivated message 449
subscription schema message 454
through publishing queue maps 12, 156
transaction message 436
XML publications
activating 221
adding columns to existing 193
changing attributes 191
columns included in message 168
creating
multiple 161
overview 158
single 159
deactivating 222
deleting 196
description 11
filtering rows 164
grouping with publishing queue maps 155
in event publishing 9
including before values 169
key columns 167
naming rules 309
Q Capture messages 163
search conditions 164
subsetting source columns 163
WHERE clauses 164

Z

z/OS
load considerations 146
z/OS console
running replication programs in batch 296
starting a Q Apply program 295
starting a Q Capture program 294

Contacting IBM

To contact IBM in the United States or Canada, call one of the following numbers:

- For customer service: 1-800-IBM-SERV (1-800-426-7378)
- For DB2 marketing and sales: 1-800-IBM-4YOU (1-800-426-4968)

To learn about available service options, call one of the following numbers:

- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Product information

Information about DB2 Information Integrator is available by telephone or on the Web.

If you live in the United States, you can call one of the following numbers:

- To order products or to obtain general information: 1-800-IBM-CALL (1-800-426-2255)
- To order publications: 1-800-879-2755

On the Web, go to http://www.ibm.com/software/data/integration/db2ii/support.html. This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, fix packs, news, and links to Web resources.

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Comments on the documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Information Integrator documentation. You can use any of the following methods to provide comments:

- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by electronic mail (e-mail) to comments@us.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Printed in USA

SC18-7568-00

Spine information:

IBM

IBM DB2 Information Integrator

Q Replication and Event Publishing Guide and Reference

Version 8.2