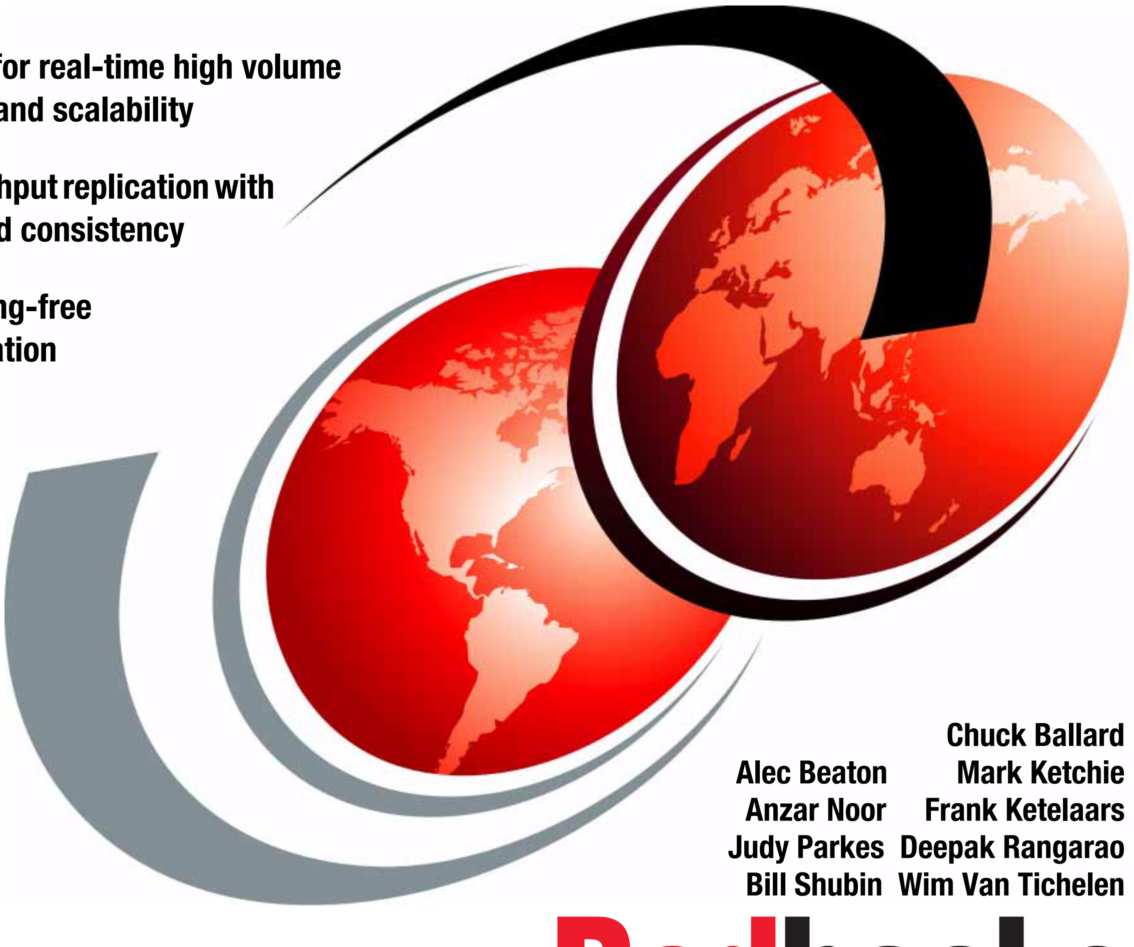
IBM® Information Management Software

Smarter Business Dynamic Information with IBM InfoSphere Data Replication CDC

Log-based for real-time high volume replication and scalability

High throughput replication with integrity and consistency

Programming-free data integration



Chuck Ballard
Alec Beaton
Mark Ketchie
Anzar Noor
Frank Ketelaars
Judy Parkes
Deepak Rangarao
Bill Shubin
Wim Van Tichelen

Redbooks

ibm.com/redbooks

International Technical Support Organization

Smarter Business: Dynamic Information with IBM InfoSphere Data Replication CDC

March 2012

SG24-7941-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (March 2012)

This edition applies to Version 6.5 of IBM InfoSphere Change Data Capture (product number 5724-U70).

Contents

Notices ix
Trademarks x

Preface xi
The team who wrote this book xii
Now you can become a published author, too! xvi
Comments welcome xvii
Stay connected to IBM Redbooks xvii

Chapter 1. Introduction and overview 1
1.1 Optimized data integration 2
1.2 InfoSphere architecture 4

Chapter 2. InfoSphere CDC: Empowering information management 9
2.1 The need for dynamic data 10
2.2 Data delivery methods 11
2.3 Providing dynamic data with InfoSphere CDC 12
2.3.1 InfoSphere CDC architectural overview 14
2.3.2 Reliability and integrity 16

Chapter 3. Business use cases for InfoSphere CDC 19
3.1 InfoSphere CDC techniques for transporting changed data 20
3.1.1 Option 1: Database staging 21
3.1.2 Option 2: Message queue (MQ) based integration 22
3.1.3 Option 3: File-based integration 23
3.1.4 Option 4: InfoSphere DataStage Direct Connect 24
3.2 Data warehousing and business intelligence 24
3.2.1 Active data warehousing 24
3.2.2 Slowly changing dimensions 26
3.3 Consolidation 28
3.3.1 Consolidation: Sample implementation 1 29
3.3.2 Consolidation: Sample implementation 2 30
3.4 Distribution 31
3.4.1 Distribution: Sample implementation 1 32
3.4.2 Distribution: Sample implementation 2 33
3.5 Database migration 35
3.5.1 Database migration: Sample implementation 35
3.6 Application integration 37
3.6.1 Application integration: Sample implementation 1 37

3.6.2 Application integration: Sample implementation 2 38
3.6.3 Application integration: Sample implementation 3 39
3.7 Integration with master data management 40
3.7.1 Integration with master data management: Sample implementation 41
3.8 Integration with IBM Information Server 43
3.8.1 Integration with IBM Information Server: Sample implementation . . 43
3.9 Operational business intelligence 45
3.9.1 Operational business intelligence: Sample implementation 45

Chapter 4. Solution topologies 47
4.1 Unidirectional replication 48
4.2 Cascading replication 49
4.3 Bidirectional replication 52
4.4 Consolidation replication 54
4.5 Data distribution replication 57
4.6 Hub-and-Spoke replication with propagation 58
4.7 Destination 62
4.7.1 JMS Message Queue 63
4.7.2 Flat files 64
4.7.3 DataStage 67
4.7.4 Web services 71

Chapter 5. InfoSphere CDC features and functionality 75
5.1 Transformations 77
5.1.1 Column functions 78
5.1.2 Journal control fields 80
5.1.3 Joining 80
5.1.4 User exits for customizations 82
5.1.5 Considerations for using transformational functionality. 83
5.2 Replication modes 83
5.2.1 Refresh 83
5.2.2 Continuous mirroring 87
5.2.3 Scheduled end (net change) 88
5.3 Filtering 89
5.3.1 Row level 89
5.3.2 Column level 90
5.4 Apply methods 91
5.4.1 Standard 91
5.4.2 LiveAudit 92
5.4.3 Adaptive Apply 93
5.4.4 Summarization 93
5.4.5 Row consolidation 94
5.4.6 Soft deletes 95

5.4.7 Custom apply methods (user exits) 96
5.4.8 Flat files 96
5.4.9 DataStage direct connect 97
5.4.10 JMS message queues 98
5.5 Conflict detection and resolution 99

Chapter 6. Understanding the architecture 103
6.1 Component overview 104
6.1.1 InfoSphere CDC instances 107
6.1.2 Interoperability between the InfoSphere CDC components 110
6.2 Management Console fundamentals 111
6.2.1 Access Manager Interface 112
6.2.2 Configuration Interface 114
6.2.3 Monitoring Interface 117
6.2.4 InfoSphere CDC API 119
6.2.5 Access Server fundamentals 120
6.3 The InfoSphere CDC engine 120
6.3.1 Bookmarks 120
6.3.2 The InfoSphere CDC Linux, UNIX, and Windows engine 122
6.3.3 The InfoSphere CDC for System i engine 131
6.3.4 The InfoSphere CDC for z/OS engine 132
6.4 Communications between source and target 134
6.5 Summary 135

Chapter 7. Environmental considerations 137
7.1 Globalization with InfoSphere CDC 138
7.1.1 Time zone considerations 138
7.1.2 Encoding conversions 147
7.2 Firewall configurations 149
7.2.1 How InfoSphere CDC uses TCP/IP 149
7.2.2 Firewalls 152
7.2.3 InfoSphere CDC in a firewalled network environment 153
7.2.4 Configuring source port restrictions 154
7.2.5 Troubleshooting CDC connection issues 156
7.3 Log retention 158
7.3.1 Log retention general guidelines 158
7.3.2 Log retention platform-specific guidelines 159
7.4 Remote processing capabilities 162
7.4.1 Remote source 163
7.4.2 Remote target 164
7.4.3 Remote source and target 165
7.4.4 Log shipping 166
7.5 Using InfoSphere CDC in resilient environments 168

7.5.1 InfoSphere CDC reachability: Virtual IP 169
7.5.2 InfoSphere CDC binary files and metadata for the Linux, UNIX, and Windows engine 170
7.5.3 InfoSphere CDC on a shared volume 171
7.5.4 InfoSphere CDC on separate nodes with a shared database 173
7.5.5 InfoSphere CDC on separate servers with separate databases . . . 175
7.5.6 System i resilient environments 178
7.5.7 z/OS / Sysplex and InfoSphere CDC in resilient environments . . . 186
7.6 Change management 190
7.6.1 Understanding InfoSphere CDC bookmarks 191
7.6.2 Change Management sample environment 193
7.6.3 DDL changes in a service window 193
7.6.4 DDL changes without a service window 207

Chapter 8. Performance analysis and design considerations 211
8.1 High volume between two systems 212
8.1.1 Latency and throughput 212
8.1.2 InfoSphere CDC architecture 213
8.2 Identification of potential bottlenecks 214
8.3 Performance monitoring in InfoSphere CDC environments 215
8.3.1 Performance monitoring using the Management Console 215
8.3.2 System monitoring tools 215
8.4 Using workflow for performance issues. 216
8.5 Installation considerations 217
8.5.1 Silent installations and instance creation 217
8.6 Design considerations 221
8.6.1 Using multiple parallel subscriptions 221
8.6.2 Using multiple InfoSphere CDC instances 223
8.6.3 Using an n-tiered architecture 224
8.6.4 Using cascading replication to spread the workload 226
8.6.5 Continuous scraping 227

Chapter 9. Customization and automation 231
9.1 Options for managing InfoSphere CDC. 232
9.2 Management Console GUI 232
9.3 Management Console commands. 233
9.3.1 Common uses for the Management Console commands 234
9.3.2 Compiling Management Console command scripts 234
9.4 InfoSphere CDC engine commands (CLI). 237
9.4.1 Running commands for the Linux, UNIX, and Windows engine . . . 237
9.4.2 Running CL commands for System i. 238
9.4.3 Running console commands for IBM System z 238
9.4.4 Sample scripts. 239

9.4.5 Checking an InfoSphere CDC engine and subscriptions activity . . 246
9.4.6 Removing obsolete database logs 257
9.5 InfoSphere CDC API 262
9.5.1 Development environment setup 263
9.5.2 Contents of the api.jar file 263
9.5.3 Connecting to and managing the Access Server 270
9.5.4 Connecting to the data stores 277
9.5.5 Configuring InfoSphere CDC replication 283
9.5.6 Creating a subscription 284
9.5.7 Procedure for mapping tables 287
9.5.8 Table mapping example 289
9.5.9 Procedure for removing mapped tables 303
9.5.10 Table mapping removal example 304
9.5.11 Row and column filtering. 309
9.5.12 Derived columns 313
9.5.13 Encoding conversions (before and after Version 6.5) 315
9.5.14 Operations and user exits 322
9.5.15 Common procedures (updating table definitions) 325
9.5.16 Deploying subscription changes and considerations 338
9.5.17 Starting, stopping, and monitoring subscriptions 339
9.5.18 Monitoring latency 346
9.5.19 Monitoring event logs using the API 359
9.6 Monitoring and integration with external monitoring solutions. 365
9.6.1 Components to monitor. 366
9.6.2 InfoSphere CDC instance activity 366
9.6.3 Subscription activity 367
9.6.4 Events 367
9.6.5 Latency 369
9.7 User exits 369
9.7.1 Common uses for user exits 370
9.7.2 User exit programs 371
9.7.3 Derived expression user exits. 373
9.7.4 Table and row-level user exits 383
9.7.5 Subscription-level (unit of work) 402
9.7.6 Java user exit for flat file custom formatter 412
9.7.7 Notifications. 425

Appendix A. Single scrape events and errors 431
Single scrape error events 433

Appendix B. Additional material 435
Locating the web material 435
Using the web material. 436

How to use the web material 436

Glossary 437

Abbreviations and acronyms 445

Related publications 449
IBM Redbooks 449
Online resources 449
Help from IBM 449

Index 451

Notices

Trademarks

Preface

A new generation of smarter products, embedded with increasingly sophisticated software and instrumentation, able to connect to other devices, and respond intelligently to user needs, is transforming the way the world works. These products are the building blocks of smarter solutions in every industry, from hospitals to manufacturing to the energy grid.

IBM® Smarter Planet™ software solutions are primarily event-driven. The trigger for an action is not typically a direct user query or an administrative command. It is an event triggered by an IT system, an automated business process, or a physical sensor, such as the reading of an RFID tag or an anomaly detected in the flow of electricity in a smart energy grid. Although event-driven systems can be created based on software passing messages (using message-oriented middleware or an enterprise service bus), the simplest and most flexible way to achieve the same result is to monitor changes to data from existing applications. This technique, using IBM InfoSphere® Change Data Capture (InfoSphere CDC) technology, requires no changes to the application generating the event. This technique allows the consumer of events to use a single and unified change tracking interface.

In today's demanding environment, businesses expect real-time access to personalized information and instantaneous updates, setting a new level of expectations within organizations. To make more informed business decisions, better serve customers, and increase operational efficiencies, organizations need to be aware of changes to key data as they occur. These changes must be immediately delivered to the people and processes that need to act upon them. This ability to sense and respond to data changes is fundamental to Dynamic Warehousing, Master Data Management, and other key initiatives. How would you tie all these independent systems together and process the immense data flow requirements while using your existing IT infrastructure?

In this IBM Redbooks® publication, we provide examples of InfoSphere Data Replication change data capture technology being used to achieve a wide variety of business purposes. We also demonstrate how to implement this technology to best use your existing infrastructure and achieve the scalability you require.

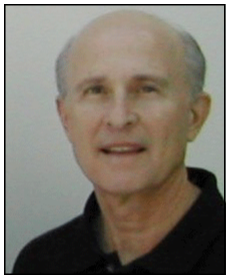
In this book, we introduce you to InfoSphere Data Replication, which is composed of a number of IBM technologies. One of those technologies is IBM InfoSphere Change Data Capture (InfoSphere CDC). We provide an overview of InfoSphere CDC and position it within the InfoSphere architecture.

In addition, we describe the various InfoSphere CDC topologies that can be used to develop business solutions in various environments. We show how InfoSphere CDC can scale to satisfy the requirements of enterprise environments, including high availability. We also describe the flexibility and choices available for managing your InfoSphere CDC environment.

InfoSphere CDC V6.5 and IBM InfoSphere Change Data Delivery V6.5 play roles in capturing and delivering data across enterprise systems in real time. By distributing real-time data across disparate information silos and key systems, they enable businesses to gain immediate awareness of market landscape and operational statistics to streamline business processes, improve customer service, and capture time-sensitive opportunities. InfoSphere Change Data Capture V6.5 and InfoSphere Change Data Delivery V6.5 introduce new capabilities that continue to extend and improve real-time data integration across heterogeneous environments. These advancements include enhanced monitoring, debugging, and improved multi-byte character set (MBCS) configuration. InfoSphere CDC V6.5 also addresses the need for increasing data volumes by providing performance improvements while continuing to achieve and reduce impact on mission-critical production systems.

The team who wrote this book

This book was produced by a team of specialists from around the world working with the International Technical Support Organization, in San Jose, California.



Chuck Ballard is a Project Manager at the International Technical Support organization in San Jose, California. He has over 35 years experience, holding positions in the areas of Product Engineering, Sales, Marketing, Technical Support, and Management. His areas of expertise are database, data management, data warehousing, business intelligence, and process re-engineering. He has written extensively on these subjects, taught classes, and presented at conferences and seminars worldwide. Chuck has both a Bachelor’s degree and a Master’s degree in Industrial Engineering from Purdue University.

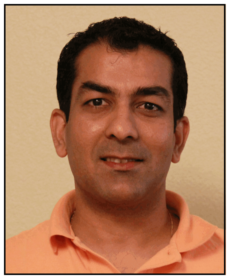
Alec Beaton is a Change Data Capture Specialist and member of the IBM InfoSphere Change Data Capture Center of Excellence. He is focused on making specialized knowledge about this product available to tech sales and IBM Business Partners through the InfoSphere CDC wiki and FAQ systems. Before the acquisition of IBM DataMirror® by IBM in 2007, Alec was involved with the PointBase product line acquired earlier by DataMirror. PointBase remains his secondary focus due to the use of the pure Java database by InfoSphere CDC as its metadata repository. He has an IT background that began in the 1980s, and has worked with various database products, including IBM DB2®, Oracle, and SQL Server, and has programmed in Java and C.

Mark Ketchie is a senior Client Technical Specialist (CTS) focused on InfoSphere CDC for the North American tech team. He was previously a DataMirror employee and joined IBM as part of the DataMirror acquisition. Mark assists regional CTSs in conducting InfoSphere CDC Proof of Technologies (POTs) and Proof of Concepts (POCs). He also assists as a liaison between customers and IBM Support on critical support issues. Mark began working with IBM Transformation Server® (now InfoSphere CDC) in 2000 as a Sales Engineer (SE). His duties included customer presentations and demonstrations, producing POTs and POCs, customer training, and consulting. His primary experience is with DB2 / UDB, DB2/400, Oracle, and Microsoft SQL Server. His prior IT experience includes acting as a Project Lead, IT Manager, and Sr. Programmer / Analyst.

Frank Ketelaars is a senior technical field specialist for a worldwide team focused on IBM Replication solutions. In this capacity, Frank conducts change data capture enablement sessions for IBM tech sales teams and IBM Business Partners across the globe. He assists IBM teams with the design of replication configurations to meet client requirements. Before the acquisition of DataMirror by IBM in 2007, Frank fulfilled an international role in the DataMirror organization for almost 10 years, being involved both in pre-sales and professional services engagements. He has hands-on experience on various databases, programming languages, and platforms, with a high level of expertise with IBM System i®. Frank has a Bachelor of Science degree from Hogeschool Eindhoven, The Netherlands, and has 22 years of experience in the information business.

Anzar Noor is a Senior Consultant with the Information Management Professional Services World Wide Team based in Markham, Canada. His main focus has been IBM InfoSphere Change Data Capture and iReflect (Oracle to Oracle High Availability) product implementations around the world. He has performed many implementations of InfoSphere CDC and iReflect since joining IBM in 2007. He has successfully implemented InfoSphere CDC on various platforms, including IBM AIX®, Sun, HP-UX, Linux, AS/400®, and Windows, and multiple database technologies (Oracle DB2 and SQL Server). Anzar came to IBM as part of the IBM acquisition of the DataMirror corporation. He has been working with the InfoSphere CDC technology for over 11 years.

Judy Parkes is a manager with IBM Platform Integration & Deployment Services, with responsibilities for InfoSphere CDC Product Management, in IBM Software Group, Information Management. Judy is based in Markham, Canada.



Deepak Rangarao is a Senior Certified Technical Sales Specialist with the IBM Netezza® Analytics team. He has over 13 years of cross-industry experience in data warehousing and analytics working for customers and vendors in both a pre-sales and post-sales capacity in retail, banking, telecommunication, and public services. Deepak has taught at universities, on subjects that include electrical engineering, Java network programming, and multimedia and web development. Deepak holds a Master's degree in Information Technology from R.M.I.T in Melbourne, Australia.

Bill Shubin is a technical specialist focusing on the InfoSphere CDC products in a technical pre-sales role. He has over 13 years experience with replication software solutions and also has experience with IBM System i High Availability replication solutions. Bill has hands-on experience with various databases, including DB2 for AS/400, DB2 for Linux, UNIX, and Windows, Oracle, and SQL Server. Before the acquisition of DataMirror by IBM in 2007, he was with the DataMirror organization for almost 10 years, and had a pre-sales role covering the United States and a professional services role covering North America. Bill is based in Southern California and has a total of 25 years of experience in IT.

Wim Van Tichelen is a Principal Consultant within IBM Software Group Benelux. He started his IT career as an IT manager in a hospital over 20 years ago. Over the years, he worked intensively with the InfoSphere CDC products all over Europe, the Middle East, and Africa. As a principal consultant, Wim guides customers at the business and technical levels to implement IT solutions to deliver value to organizations. His main focus is on InfoSphere CDC and InfoSphere Foundation Tools.

Other Contributors:

Thanks to the following people for their contributions of either advice, guidance, or actual written content to this project:

From IBM Locations Worldwide
Alex Lavrov, Information Management Client Technical Professional, IBM Sales & Distribution, Software Sales, IBM Israel. We give a special thanks to Alex for his significant written contributions. He also used his deep technical and implementation skills on InfoSphere CDC to provide detailed technical reviews and feedback, particularly regarding the sample code contained in the book.

We also give a special thanks to the following people for their extra effort and dedication to the development of this book.

Jerome Chailloux, Channel Technical Sales, IT Specialist, Client Technical Professional IBM Sales & Distribution, IBM France

Laura Cuell, Senior Data Replication Specialist, InfoSphere CDC Center of Excellence, IBM Software Group, Information Management, Markham, Ontario

Mike Jory, Senior Software Development Manager, IBM Software Group, Information Management, Markham, Ontario

Matthias Reiss, Senior Technical Sales Professional, IBM Sales & Distribution, Software Sales, IBM Germany

James Spyker, Senior Architect, InfoSphere CDC, IBM Software Group, Information Management, Markham, Ontario

Victor Szabo, Software Developer, InfoSphere CDC, IBM Software Group, Information Management, Markham, Ontario

Thanks also to the following people for their contributions to this project:

Elaine Pang, Change Data Capture Specialist, InfoSphere CDC Center of Excellence, IBM Software Group, Information Management, Markham, Ontario

Glen Sakuth, Sr. Manager, World Wide CDC Center of Excellence, IBM Software Group, Information Management, Markham, Ontario

Dan Snoddy, Development Manager UI Applications, IBM Software Group, Information Management, Markham, Ontario

Igor Siljanovski, Information Management Development Manager - InfoSphere CDC for IBM z/OS®, Markham, Ontario

From the International Technical Support Organization, San Jose, CA
Mary Comianos, Publication Management
Ann Lund, Residency Administration
Emma Jacobs, Graphics

Now you can become a published author, too!

Comments welcome

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction and overview

This book describes the IBM InfoSphere Change Data Capture (InfoSphere CDC) solution and how it fits in an Optimized Data Integration solution. Optimized Data Integration addresses the increasing need of businesses for timely access to changed data before making critical business decisions. It is a solution that allows businesses to access, move, and deliver data in a timely and cost-effective manner from the source systems on which it is located to the target systems and applications where it is required.

In today's demanding business environment, users and consumers of business data want real-time access to personalized information and instantaneous updates, setting a new level of expectation for data within organizations. To make more informed business decisions, better serve customers, and increase operational efficiencies, organizations must ensure that they are aware of changes to key data as they occur. These changes must be immediately delivered to the people and processes that need to act upon them.

From data warehousing to service-oriented architecture (SOA), application consolidation, and master data management, business leaders are aware of the power of information to streamline processes, reduce costs, and make businesses more efficient and effective. To be successful, these projects must have steady and reliable delivery of timely business information from across the enterprise, which can be both expensive and resource-intensive.

Change data capture technology helps businesses overcome these challenges by capturing only changed operational data and transmitting it across the enterprise. This approach provides substantial business value while helping to reduce risk and to deliver cost and speed advantages that enhance traditional data movement processes.

1.1 Optimized data integration

Most businesses have information management environments organized similar to the one shown in Figure 1-1. These environments consist of transactional source databases that contain the basic building blocks of raw data that all companies generate and require to operate their operational systems (lower left). As you move up and right from these source databases through the data usage continuum, you find users, analytics, and the consuming applications that companies use to derive business insight from their data (upper right). In the middle, you find a reporting or data warehousing environment that most companies use to organize and analyze their raw data to enable the delivery of business insights.

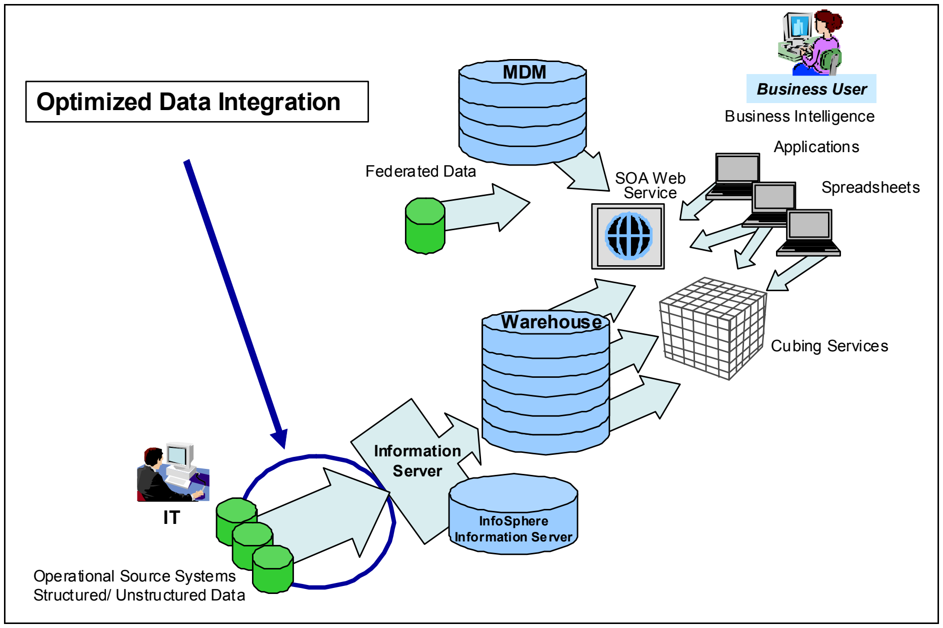


Figure 1-1 Optimized data integration

The main goals of optimized data integration are to move and deliver data in the most timely and cost-effective manner. These goals are achieved by optimizing the consumption and latency period associated with data movement, moving only the data requested by a user or that has been changed or updated on the source systems. Moving only the changed data saves time and money by moving only the subset of data that is required rather than moving the entire source data set. This approach to data movement ensures that any organization that moves data to support business intelligence (BI) and reporting systems, applications consolidations and migrations, continuous availability solutions, or general data distribution and synchronization scenarios, can do so knowing that the raw data is delivered efficiently and effectively.

Figure 1-2 provides a simple schematic of how optimized data integration is most commonly implemented to enable businesses to make better informed business decisions, run smoother operations, win new customers, and increase their bottom line.

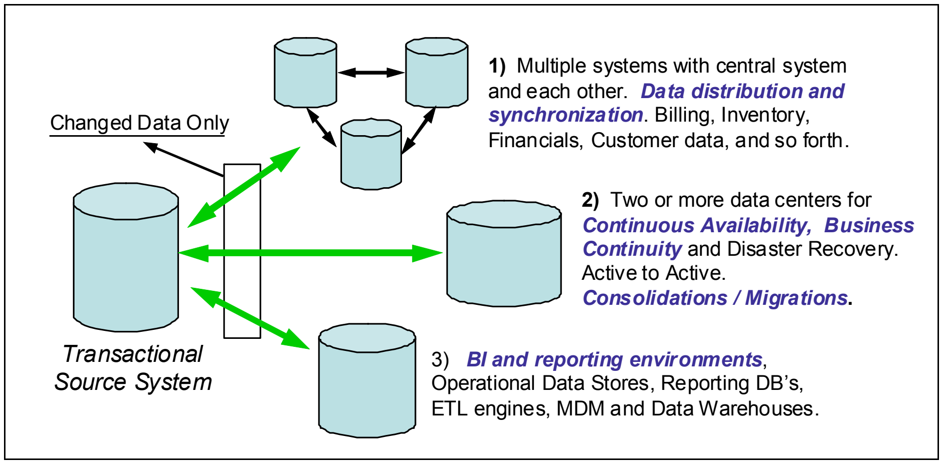

Changed Data Only
Transactional Source System
1) Multiple systems with central system and each other. Data distribution and synchronization. Billing, Inventory, Financials, Customer data, and so forth.
2) Two or more data centers for Continuous Availability, Business Continuity and Disaster Recovery. Active to Active. Consolidations / Migrations.
3) BI and reporting environments, Operational Data Stores, Reporting DB's, ETL engines, MDM and Data Warehouses.

Figure 1-2 How optimized data integration is used

Some examples of how optimized data integration is used include the following:

- Provides feeds of changed data for Data Warehouse or Master Data Management (MDM) projects, enabling users to make operation and tactical business decision making using the latest information.
- Dynamically routes data based on content to message queues to be consumed by one or more applications, ensuring consistent, accurate, and reliable data across the enterprise.

- Populates real-time dashboards for on-demand analytics, continuous business monitoring, and business process management to integrate information between mission-critical applications and web applications, ensuring access to real-time data to customers and employees.
- Consolidates financial data across systems in different regions, departments, and business units.
- Improves the operational performance of systems that are adversely affected by shrinking nightly batch windows or expensive queries and reporting functions.

1.2 InfoSphere architecture

Before beginning an in-depth discussion about change data capture, this section describes how it fits in to the broader IBM InfoSphere Information Server architecture strategy.

IBM InfoSphere Information Server helps business and IT personnel collaborate to understand the meaning, structure, and content of any type of information across any sources. It provides breakthrough productivity and performance for cleansing, transforming, and moving this information consistently and securely throughout the enterprise so it can be accessed and used in new ways to drive innovation, increase operational efficiency, and help lower risk. With a unified metadata foundation, IBM InfoSphere Information Server allows users from various roles in the organization to establish a common vocabulary and understanding of the business from end-to-end to enrich and streamline operations.

InfoSphere Information Server achieves new levels of information integration speed and flexibility by providing the following capabilities:

- A comprehensive and unified foundation for enterprise information architectures, scalable to any volume and processing requirement
- Auditable data quality as a foundation for trusted information across the enterprise
- Metadata-driven integration, providing breakthrough productivity and flexibility for integrating and enriching information
- Consistent and reusable information services, along with application services and process services, essential for enterprises

- Accelerated time to value with proven and industry-aligned solutions and expertise
- Broadest and deepest connectivity to information across diverse sources, such as structured, unstructured, mainframe, and applications

IBM InfoSphere Information Server provides every capability needed to integrate information across heterogeneous systems, including understanding source data, applying data quality, complex transformation, and various ways to deliver information. It has a unique, metadata-driven design that helps align business goals and IT activities, provides a consistent understanding of what things mean, captures business specifications and uses them to automate development tasks, and provides deeper insight into data by tracking its lineage.

InfoSphere Information Server is composed of four main pillars (Figure 1-3):

- Understanding your data
- Cleansing your data
- Transforming your data
- Delivering your data to where it is needed

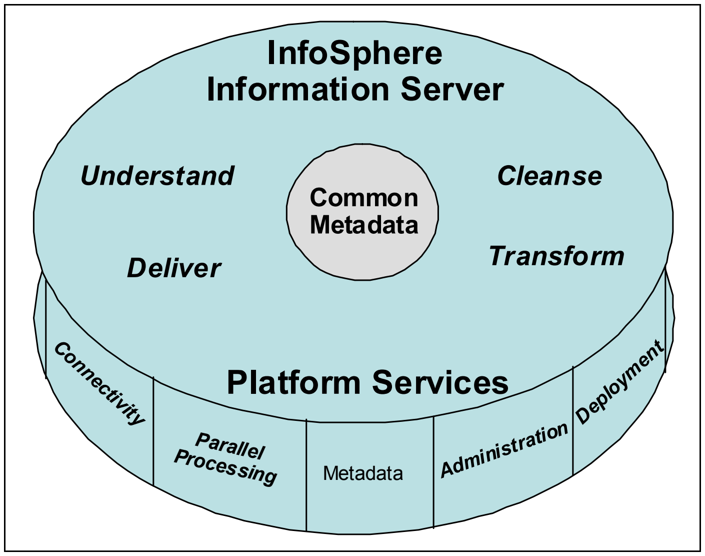


Figure 1-3 InfoSphere Information Server

Here is a brief description of those four pillars:

- Understanding your data: InfoSphere Foundation Tools provides tools that enable you to discover your data across systems by the following means:
  - Through comprehensive data discovery and mapping
  - With trusted information structures for business optimization through a common business vocabulary to define an enterprise model and design specifications
  - Through governing the data over time by managing data quality and monitoring data flows
- Cleansing your data: InfoSphere data quality capabilities ensure that you have reliable and accurate information by identifying the source of data quality problems, defining business rules to monitor and maintain quality, removing duplicate data, and validating, standardizing, and enriching the data.
- Transforming your data: InfoSphere data transformation extracts, transforms, and loads data between multiple sources and targets, supporting massive scalability requirements and delivering data in batch or real time.
- Delivering your data: InfoSphere data delivery provides timely and reliable movement of heterogeneous data.

As shown in Figure 1-4, moving data and getting it to where it needs to be in a timely manner has a key role in data integration projects. Enterprises have many different, disparate, and heterogeneous data sources from which they require updated and fresh data to make timely and trusted business decisions. InfoSphere Information Server supports a broad range of data delivery styles to address business and IT objectives.

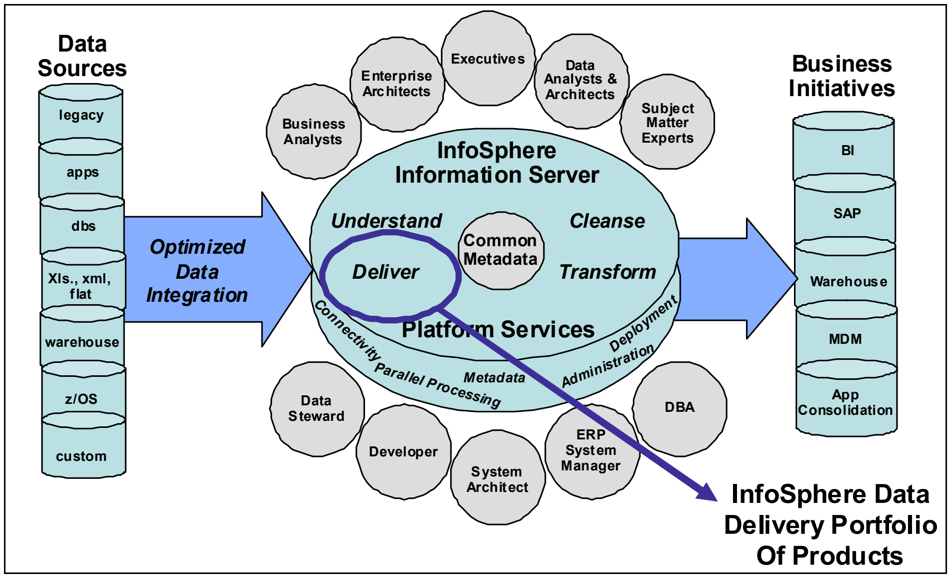


Figure 1-4   InfoSphere Deliver Pillar

The Deliver Pillar addresses the key challenges encountered by businesses that need to have trusted and up-to-date data to make decisions that positively affect business, whether for strategic initiatives or to for improving company efficiencies. A common challenge is heavily or overutilized source systems and applications that contain business critical data that cannot afford additional workload to query or extract data for reporting purposes. Batch windows are the traditional approach for updating data warehouses or reporting databases. As business data volume increases, the length of time required to bulk load these data changes is increased, which can impact the operational systems. Change data capture allows incremental data changes to be captured and delivered throughout the enterprise with minimal processing cost or impact to source systems. Batch windows can be shortened or eliminated by real-time feeds of data changes, and large volumes of data can be moved quickly and efficiently.

The remaining chapters of this book provide details and examples about implementation and operational aspects of InfoSphere Change Data Capture.

2

InfoSphere CDC: Empowering information management

Chapter 1, “Introduction and overview” on page 1 describes how the optimized data integration solution addresses the increasing need for timely access to changing data before making critical business decisions. The optimized data integration solution allows businesses to access, move, and deliver data in a timely and cost-effective manner from source systems on which the data is located to the target system or application where the business requires the data. That chapter also describes how the solution fits into the broader InfoSphere Information Server landscape within the Deliver Pillar by providing timely and reliable movement of heterogeneous data. InfoSphere data delivery capabilities can be used in multiple projects across the enterprise. These projects include keeping back-end inventory and front-end web applications synchronized, distributing global data across regional offices, feeding a data warehouse or MDM system, enabling real-time analytics, and optimizing ETL processes by providing a real-time flow of data changes.

This chapter describes the need for dynamic data, the delivery methods that can be used to move data, and how dynamic data can be provided by the IBM InfoSphere Change Data Capture (InfoSphere CDC) technology.

2.1 The need for dynamic data

There are three basic types of data available that can be used for informational purposes:

- Persistent or static: This type of data is, once created, not changed. As such, it is typically accessed less frequently.
- Streaming: This type of data continuously flows. As such, it must be captured as it passes by, or it is missed. It is dynamic in the sense that it is moving, but there might be no indication that the data elements in the stream have changed, or that they flow in a consistent pattern or period.
- Dynamic: This type of data is changed as changes become available due to the execution of transactional events. Changes are not necessarily made on a consistent periodic basis, and there might be small or large periods when there is no data change activity.

Dynamic data is prevalent and represents constant change in an environment. Consider a manufacturing business that builds parts or assemblies. The time to build parts or assemblies varies, based on a number of parameters. This data needs to be captured because it impacts manufacturing costs, and resource quantities and availability, and is therefore needed in decision making processes. As such, the value of the dynamic data is needed as it changes to support application execution, and enables users to make more informed business decisions by having the most current data.

As competitive and economic pressures increase, up-to-date and trusted dynamic data is needed to make decisions that benefit the business. To be successful, organizations must report and analyze corporate data quickly and easily, regardless of what applications created the data, what platform they are running on, and where or how they are stored. This situation occurs because the data needs to be synchronized between systems and applications that are using that data for informational purposes.

The ability to easily capture and deliver business-critical dynamic data throughout your enterprise provides the following business benefits:

- Increase business agility: Make proactive business decisions based on business relevant events, for example, to notify a customer when their pre-paid phone card is almost empty.
- Make better decisions: Enable customers, employees, and partners to base key decisions on up-to-date information, for example, to purchase additional inventory when there are only two parts left.

- Access near real-time data without impacting operational systems: Gain access to near real-time data without impacting source systems and database resources.

2.2 Data delivery methods

When it comes to moving data, there are essentially three primary approaches:

- Virtual data delivery
- Bulk data delivery
- Incremental data delivery

These approaches can be further segmented into replication and change data capture.

These data delivery methods are shown in Figure 2-1.

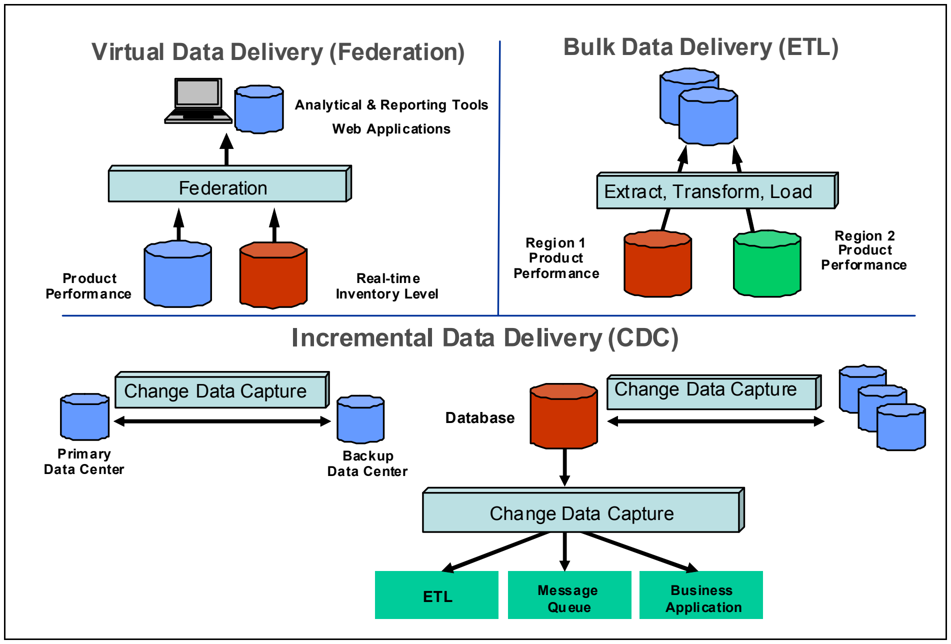


Figure 2-1 Data delivery methods

Brief descriptions of the three approaches are as follows:

- Virtual or Federation: Generates a virtual consolidated view from multiple and disparate sources systems as though they were a single source. This method complements or extends the data warehouse view and is generally used when some data cannot be physically moved due to licensing or security reasons. This approach is frequently used as a first step, before using a fully implemented data warehouse, to query multiple source systems without physically consolidating them.
- Bulk Load or ETL: Data is extracted from the originating source system, transformed, and output to the data warehouse or receiving application. This approach is typically used on a scheduled batch basis when point-in-time data is acceptable to meet the business needs. For example, ETL batch processes are frequently run during end of day jobs, resulting in a data warehouse or reporting database that presents data current to the previous day.
- Incremental Data Delivery: Businesses that opt for this method of data delivery require their data to provide up to the minute or near real-time information. This method includes both replication and change data capture. Replication is typically used for database to database data movement and provides solutions for continuous business availability, live reporting, and database or platform migrations. When using change data capture, the target is not necessarily a database. In addition to the solutions included in replication, this approach can also feed changes to an ETL process or deliver data changes to a downstream application by using a message queue.

2.3 Providing dynamic data with InfoSphere CDC

Change data capture uses a developed technology to integrate data in near real time. InfoSphere CDC detects changes by monitoring or scraping database logs. The capture engine (the log scraper) is a lightweight, small footprint, and low-impact process on the source server running where the database changes are detected. After the log scraper finds new changed data on the source, that data is pushed from the source agent to the target apply engine through a standard Internet Protocol network socket. In a typical continuous mirroring scenario, the change data is applied to the target database through standard SQL statements.

The changed data is scraped from the source log, sent over the network, and applied to the target database without passing through any intermediate tables, files, or queues. A simple and intuitive user interface allows users to determine what data needs to be integrated and what transformations need to be performed on the data before being applied to the target database.

By having the data only interact with the database logs, additional load is not put on the source database and no changes are required to the source application. Change data capture uses both online / active and archive logs with each source engine, optimized for the database and platform on which it is running. For example, when running on the mainframe and monitoring DB2/z logs, the standard DB2 instrumentation facility interface (IFI) is used.

Incremental data delivery is shown in Figure 2-2.

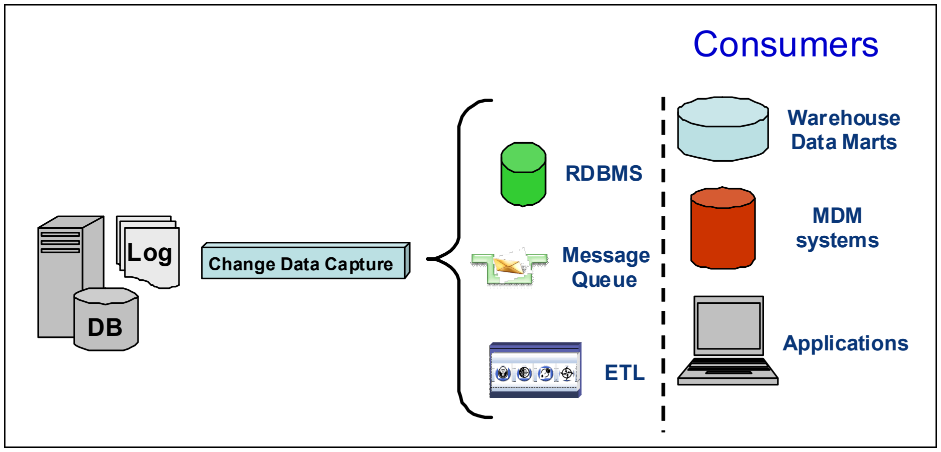


Figure 2-2 Incremental data delivery

2.3.1 InfoSphere CDC architectural overview

Figure 2-3 provides an overview of the InfoSphere CDC architecture.

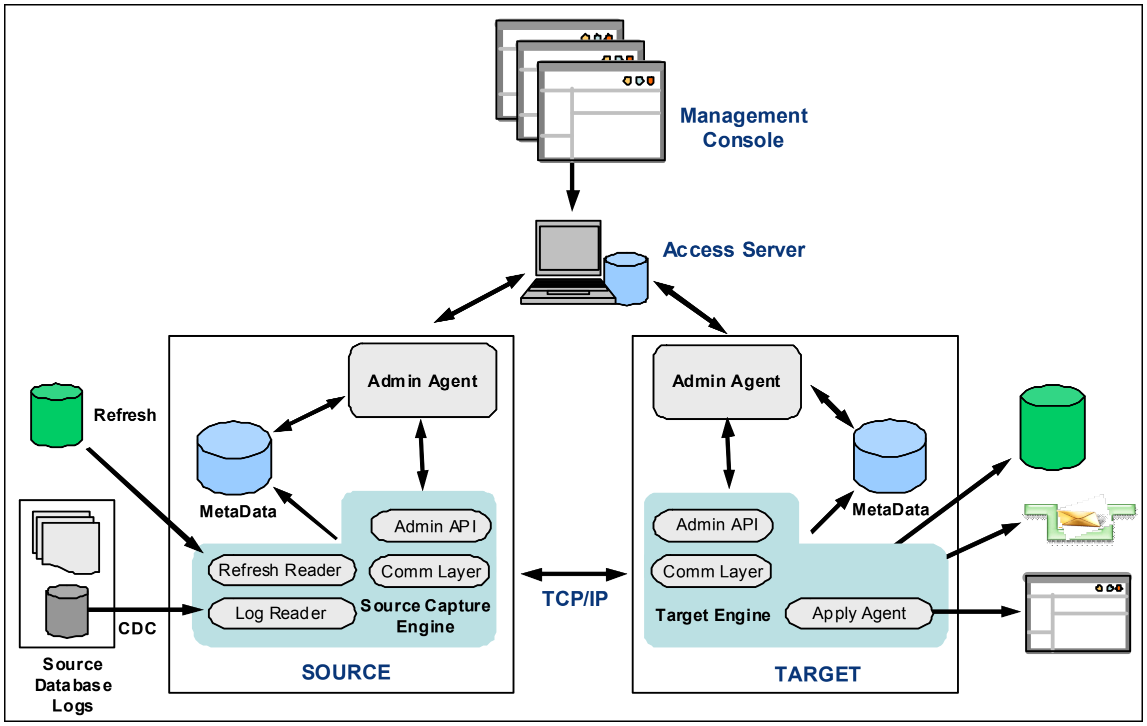


Figure 2-3 Architectural overview

The key components of the InfoSphere CDC architecture are:

- Access Server: Controls all of the non-command-line access to the replication environment. When you log on to Management Console, you connect to the Access Server.
- Admin API: Operates as an optional Java based programming interface that you can use to script operational configurations or interactions. After you have set up replication, Management Console can be closed on the client workstation without affecting active data replication activities between source and target servers.
- Apply Agent: Acts as the agent on the target that processes changes sent by the source.
- Command-line interface: Allows you to administer data stores and user accounts, and to perform administration scripting, independent of Management Console.

- Communication Layer (TCP/IP): Acts as the dedicated network connection between the source and the target.
- Data store: The source and target data stores represent the data files and InfoSphere CDC instances required for data replication. Each data store represents a database to which you want to connect and acts as a container for your tables. Tables made available for replication are contained in a data store.
- InfoSphere CDC Management Console: The interactive application that you use to configure and monitor replication. It allows you to manage replication on various servers, specify replication parameters, and initiate refresh and mirroring operations from a client workstation. Management Console also allows you to monitor replication operations, latency, event messages, and other statistics supported by the source or target data store. The monitor in Management Console is intended for time-critical working environments that require continuous analysis of data movement.
- Metadata: Represents the information about the relevant tables, mappings, subscriptions, notifications, events, and other particulars of a data replication instance that you set up.
- Mirror: Performs the replication of changes to the target table or accumulation of source table changes used to replicate changes to the target table at a later time. If you have implemented bidirectional replication in your environment, mirroring can occur to and from both the source and target tables.
- Refresh: Performs the initial synchronization of the tables from the source database to the target. These tables are read by the Refresh reader.
- Replication Engine: Sends and receives data. The process that sends replicated data is the source capture engine and the process that receives replicated data is the target engine. An InfoSphere CDC instance can operate as a source capture engine and a target engine simultaneously.
- Single Scrape: Acts as a source-only log reader and a log parser component. It checks and analyzes the source database logs for all of the subscriptions on the selected data store.
- Source transformation engine: Used to process row filtering, critical columns, column filtering, encoding conversions, and other data to propagate to the target data store engine.
- Source database logs: Maintained by the source database for its own recovery purposes. The InfoSphere CDC log reader inspects these logs in the mirroring process, but only looks for those tables that are mapped for replication.
- Target transformation engine: Used to process data and value translations, encoding conversions, user exits, conflict detections, and other data on the target data store engine.

The two types of target-only destinations for replication that are not databases are:

- JMS Messages: Acts as a JMS message destination (queue or topic) for row-level operations that are created as XML documents.
- IBM InfoSphere DataStage®: Processes changes delivered from InfoSphere CDC that can be used by InfoSphere DataStage jobs.

2.3.2 Reliability and integrity

The InfoSphere CDC fault-tolerant architecture maintains data consistency and provides recovery from network and database outages. Change data is sent from the source database through the source or log scraper agent to the apply agent through a TCP/IP connection. Only committed transactions are sent to the target. To ensure that the source system transaction order is maintained, the transactions are applied to the target in the same order as the change data found in the source database log (Figure 2-4).

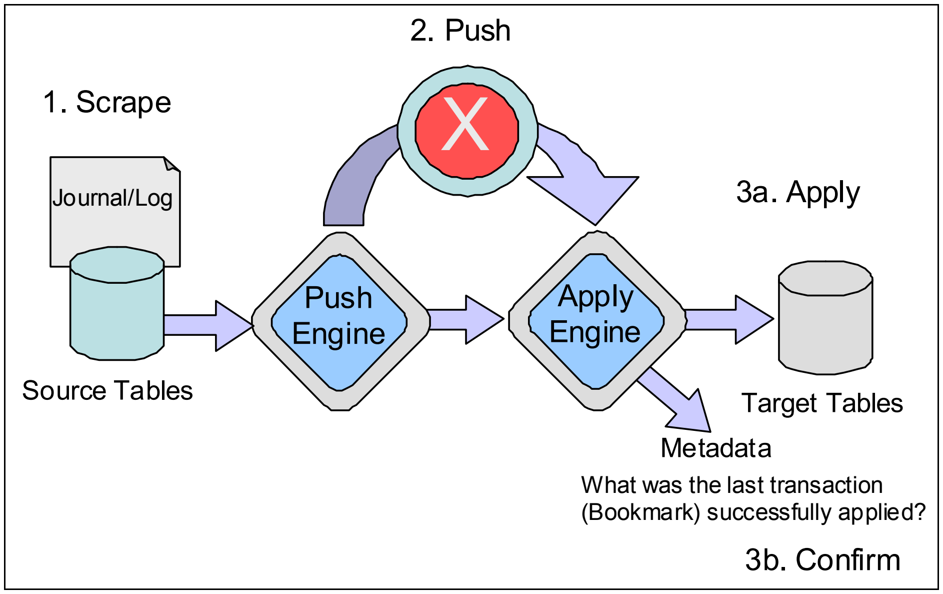


Figure 2-4 Reliability and integrity

As part of each replication thread, there is a mechanism to track which transaction is being processed. That mechanism is called a *bookmark*. The bookmark marks a point in the flow of committed changes. It contains all the information necessary for InfoSphere CDC to be able to restart replication. The InfoSphere CDC agent has a small metadata table stored in the target database that contains the last successfully applied bookmark. Updates to target tables are combined with an update to the bookmark table and applied to the target database as a single unit of work. This setup ensures that the bookmark accurately indicates what changes have been applied to the target database. If there is any failure during the application, neither the change data or the bookmark are updated.

If there is any disruption in the replication stream of source transactions, InfoSphere CDC must reconstruct the source transactions with the data read from the database log. The bookmark is the position in that stream that includes all the information necessary for InfoSphere CDC to recreate that stream and position it appropriately.

Whenever replication is restarted, either after normal or abnormal termination, the InfoSphere CDC target agent notifies the source agent of the bookmark. The source agent then positions the reader in the source database and continues replication.

3

Business use cases for InfoSphere CDC

IBM InfoSphere Change Data Capture (InfoSphere CDC) can be used in any organization where data is changing in one system (or more than one system for bidirectional replication), and those changes must be replicated to other systems within the enterprise. Although this situation is conceptually simple, the additional features and functionality in InfoSphere CDC can be used to resolve many of the typical business problems associated with managing your data environment. This chapter describes the application of InfoSphere CDC and highlight some business use cases from client experiences and internal projects.

3.1 InfoSphere CDC techniques for transporting changed data

There are many methods for integrating data between applications. Some methods work at the application level and others work at the database table or transaction log level. This section focuses on the different techniques for transporting changed data that can be used when using InfoSphere CDC.

In cases where you need to establish a reporting / shadow database that matches the contents and structure of the source database, the topology in Figure 3-1 suffices. Table changes are picked up from the source database through its transaction log, then transported and applied to the target database.

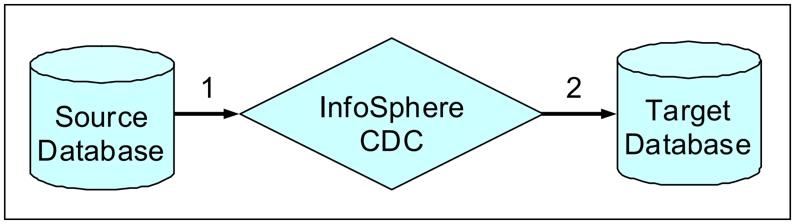


Figure 3-1 Change data capture

For more complex data replication requirements, use InfoSphere CDC with extract, transform, and load (ETL) tools. In those situations, InfoSphere CDC replicates data to the target databases through ETL tools in various ways, such as those described in the following options:

- Option 1: Database staging
- Option 2: Message queue (MQ) based integration
- Option 3: File-based integration
- Option 4: InfoSphere DataStage Direct Connect

3.1.1 Option 1: Database staging

This option (Figure 3-2) typically has the characteristics described in the following list:

1. InfoSphere CDC captures changes made to source systems.
2. Changes are written to a staging area table or tables.
3. The ETL processes extract the changes from the staging table or tables, and transform and clean the data as needed.
4. After the ETL processes are finished, the replication is resumed and the staging tables are populated for the next run.

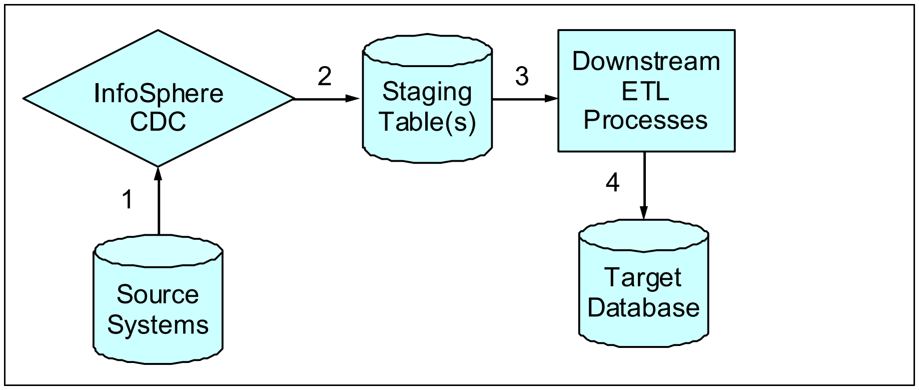


Figure 3-2 Change data capture - database staging

3.1.2 Option 2: Message queue (MQ) based integration

The functionality of this option is delivered with IBM WebSphere® MQ (WebSphere MQ). It is shown in Figure 3-3 and typically has the characteristics shown in the following list:

1. InfoSphere CDC captures changes made to the source systems.
2. The captured changes are extracted and written to a message queue (MQ).
3. The downstream ETL processes (through the WebSphere MQ connector) extract the new messages.
4. Those messages are then passed to other downstream processes to be applied to the target database.

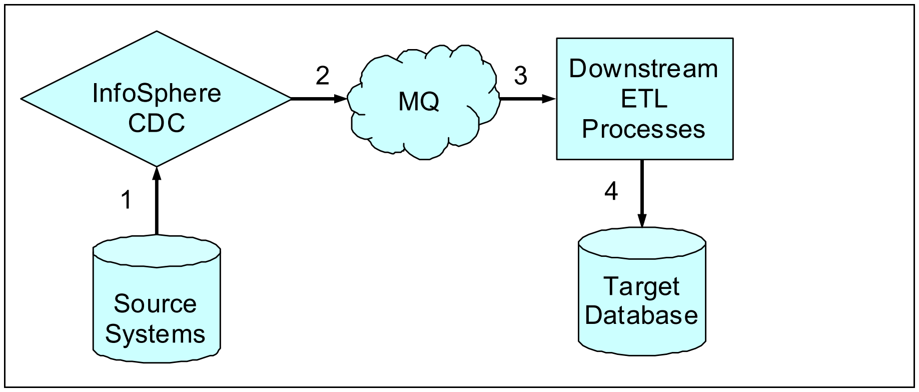


Figure 3-3 Change data capture - MQ-based integration

3.1.3 Option 3: File-based integration

This option (Figure 3-4) typically has the characteristics shown in the following list:

1. InfoSphere CDC captures changes made to source systems.
2. Changes are then written to flat files.
3. The downstream ETL processes extract the changes from the flat files.
4. The data is then passed to other downstream processes to be applied to the target database.

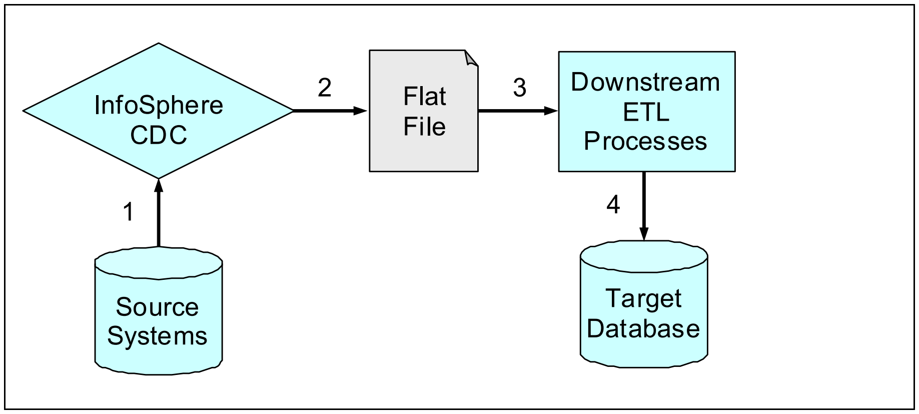


Figure 3-4 Change data capture - file-based integration

3.1.4 Option 4: InfoSphere DataStage Direct Connect

This option (Figure 3-5) typically has the characteristics shown in the following list:

1. InfoSphere CDC captures the changed data from the source systems and passes this data to a continuously active DataStage job.
2. The DataStage job receives transactions through the CDC Transaction Stage operator.
3. In the job, the data can be transformed and passed on to other downstream locations.
4. The CDC bookmark is maintained along with DataStage applying the changes to the target database.

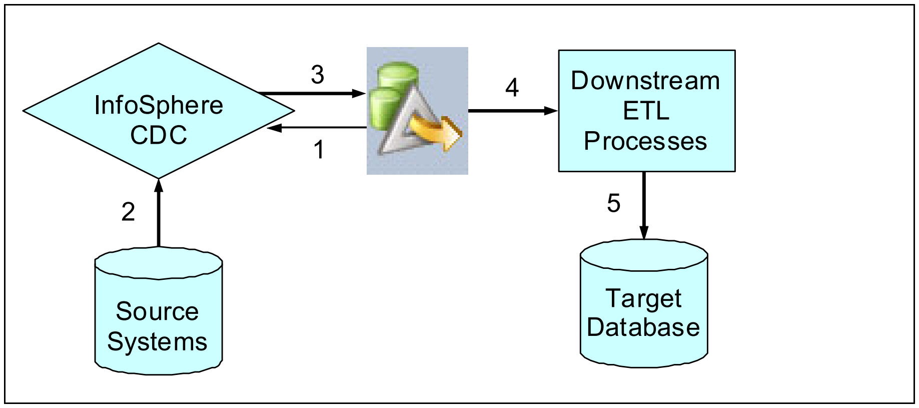


Figure 3-5 Change data capture - direct connection integration

3.2 Data warehousing and business intelligence

A data warehouse is part of a decision support systems environment that is populated by extracting, transforming, and loading operational data from one or more Online Transactional Processing (OLTP) systems. This data is then available for reporting and analysis.

3.2.1 Active data warehousing

At one time, data warehousing and decision support was focused on analysis of historical data. However, this analysis is not sufficient in today's competitive market where decisions need to be based on more current data.

There is an inherent cost to produce up-to-date reports in a traditional fashion using batch extracts, and this cost is typically measured in resource utilization (such as processor usage and processing time) split between data extraction and report generation. This cost continues to increase as corporate data volumes grow and business requirements increase.

Active data warehousing is the process of capturing transactions when they occur, and immediately integrating those changes into the data warehouse. In an active data warehouse, OLTP transactions are automatically captured upon completion, transformed as required, and loaded immediately into the data warehouse. InfoSphere CDC enables active data warehousing.

Active data warehousing: Sample implementation

In this example, clients run multiple operational applications in an IBM DB2/z mainframe environment. Data from these operational applications are extracted at the end of each day through ETL batch processing (with IBM DataStage ETL) and loaded into the data warehouse, which services many lines of business. The long-term objective of the data warehouse is to provide up-to-date business intelligence (BI) and analytics at any point in the business day.

Although the initial batch implementation was successful, some issues and requirements were identified. For example, the data warehouse was only as current as the previous night's batch update. However, the lines of business required more up-to-date information during the business day to make insightful decisions. There are typically two options available to accomplish this task:

1. Allow business users to run reports and queries on the production system during the day, which increases resource utilization and cost.
2. Implement a solution that enables intra-day updates from the production reporting applications to the data warehouse.

Active data warehousing: Proposed solution

The proposed solution is to implement IBM InfoSphere CDC to synchronize changed data from the operational production applications with the DataStage ETL server continuously during the business day. The solution is shown in Figure 3-6, and provides the following benefits:

- Reduces the volume of changed data that needs to be extracted during the nightly batch window (time and resources required), which in turn minimizes the time that production applications are unavailable to users.
- Provides continuous claims data changes (through ETL staging tables), which runs intra-day ETL updates to the data warehouse. This setup improves the overall data visibility for lines of business.

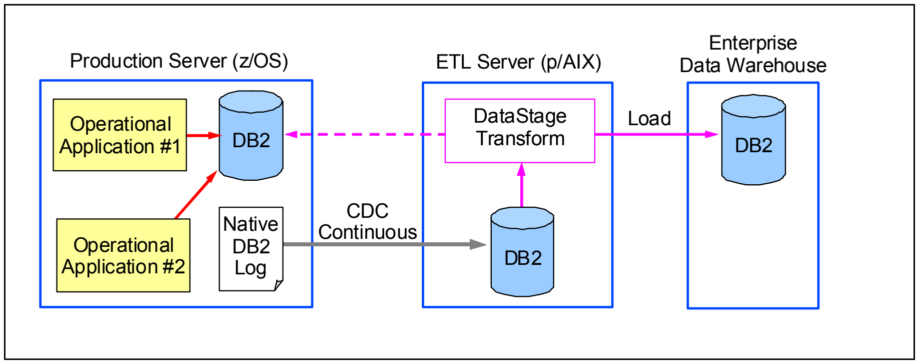


Figure 3-6 Proposed data warehouse solution - ETL-based integration

3.2.2 Slowly changing dimensions

Dimensions are logical groupings of data, such a product, customer, and time. Slowly changing dimensions (SCDs) are dimensions that change slowly over time. The ability to process SCDs is required to track historical changes in dimension tables. There are six different ways of working with SCDs, the most common of which are the following:

- Type 1 SCD: Here the assumption is that the business does not worry about historical changes to attributes in data. Any change in data attributes results in over-writing existing column data. This replication is standard data replication that inserts, updates, and deletes data rows to reflect data operations in the source system.

- Type 2 SCD: With this method, there is a business need to track historical changes to attribute values in data. Updates to data attributes are inserted as new database records using a surrogate key to support the natural key. Two additional columns, Effective Date and Active Indicator, are also included in the database table to track the historical changes to the data. Type 2 SCD allows for unlimited history preservation.
- Type 3 SCD: This method also caters to tracking historical changes to attribute values in data. Although type 2 SCD allows for unlimited history preservation, with type 3 SCD, historical information is stored as an additional column, which limits history preservation.

InfoSphere CDC can be used to update the appropriate data for SCD tables using a combination of the journal control fields and Java or Stored Procedures-based user exits.

Slowly changing dimensions: Sample implementation

In this example, the client runs their operational system on IBM DB2 for Linux, UNIX, and Windows. They have customer data being replicated from their operational systems to the Enterprise Data Warehouse (EDW). Large volumes of customer information are being replicated to the EDW with new customers and updates to customer information, such as location.

The methodology used to update customer information is to use an ETL mechanism with lookups to existing customer information to identify data changes and update customer records as appropriate. Although this approach works, the following problems and new requirements were identified:

- An increasing volume of customer updates results in increased latency.
- Increased workload on the EDW due to increased look-up queries.
- New business requirements dictate the need for near real-time customer data updates.
- Updates to the customer records need to be preserved to be able to reflect historical states.

Slowly changing dimensions: Proposed solution

The proposed solution is to implement IBM InfoSphere CDC to synchronize customer updates from the operational application to the target database tables continuously during the business day (Figure 3-7). The customer requirements dictate unlimited history preservation, so a Type 2 SCD table structure is maintained. Existing customer records are end-dated and the updates to customer information are maintained with the effective date set as the data replication date. The slowly changing dimensional information (customer data) is handled using DB2 stored procedure-based user exits. The user exit stored procedure performs UPDATE and INSERT operations to update SCD columns in the target and end-date current data, and then inserts customer updates as a new column with an effective date set to the data replication date. This solution provided the following benefits:

- Reduces the latency in receiving customer data updates.
- Reduces ongoing workload on the EDW as a result of customer lookup queries.

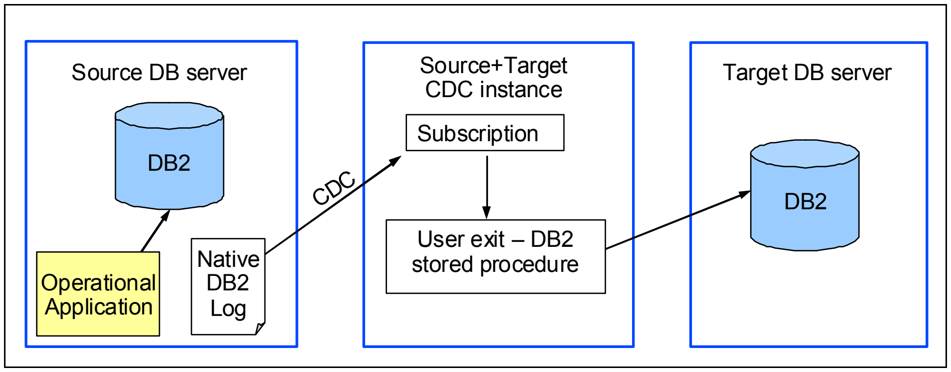


Figure 3-7 Using InfoSphere CDC to synchronize updates

3.3 Consolidation

Organizational business needs can drive the organization in a direction where they need to consolidate their data for various reasons, such as to bring down the cost, increase the quality of data, run decision making reports, or implement data normalization.

IBM InfoSphere Change Data Capture is flexible, and can enable, as examples, the following types of business requirements:

- Consolidation can be achieved at the database level, where clients want to bring together their data from various source application databases and make it available to users in one database.
- A client can consolidate the data from various tables into one table on the target side.
- A client can merge data from multiple tables into one table at the row level.

3.3.1 Consolidation: Sample implementation 1

In this example, the client runs several divisions across the United States and decides to consolidate their customer data from all of their divisions. All divisions are keeping their data on their own source systems. The client wants to consolidate all the data on the target database and then move it to another database. The environment is as follows:

- There are 22 tables on each division, all having the same structure.
- The client wants all source data manipulation language (DML) operations (inserts, updates, and deletes) performed on target database tables.
- The client wants access to all source system information (such as operation type, and date and time) while data is replicating.
- The client wants to stop the data replication process every night and move the data to the data warehouse.
- After processing the data to another database, the client wants to truncate the target tables.
- In addition, the client does not want to perform an initial load from all the source tables to the target tables.

Consolidation: Proposed solution 1

The solution includes using the InfoSphere CDC solution. InfoSphere CDC is installed and configured on each division source and target servers. The client configures all tables using the InfoSphere CDC Live Audit functionality. The client also uses journal control fields for DML operation type and time stamp. The initial load is performed outside of InfoSphere CDC. All table method and status are set to Mirror and Active mode.

The solution is shown in Figure 3-8.

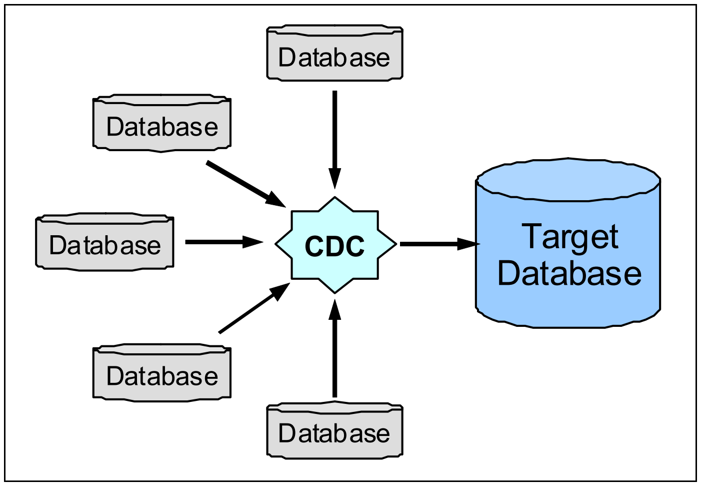


Figure 3-8 Customer consolidation

3.3.2 Consolidation: Sample implementation 2

The client is one of the world's largest manufacturers of adhesives and sealants. They employ over 4000 people and their products are sold in over 120 countries. In this example, the client has merged with another company that uses a different ERP system than they do, so they want to consolidate the data for analysis and reporting. With such large and global operations, they could only gather their data at night to not impact the performance of their systems during the day. However, data volume and reporting demands are growing and, at the same time, the merger caused additional complications in data consolidation, so business operations are not running as well as they should be.

To summarize:

- Corporate data is consolidated on a nightly basis from two ERP systems for analysis and reporting, to avoid impacting daytime performance.
- The volume of data and reporting needs exceeds nightly batch load capacity and hinders business performance.

Consolidation: Proposed solution 2

InfoSphere CDC consolidates data from ERP systems into a single database and transforms it into a single format. The client implements an InfoSphere CDC real-time solution into their infrastructure and realizes significant benefits, such as the following:

- Flexible reporting and analysis schedules. Reporting can now be done at any time.
- Using the most recent data from the consolidated ERP systems provides more accurate reporting, and removes the previously existing execution bottlenecks.
- The data consolidation of two different ERP systems is easily handled by InfoSphere CDC transformation capabilities, and provides a single view of the corporate ERP data.
- The batch time slot at night is free for other activities. As a result, the clients business performance has improved with a 10% increase of on-time delivery of orders and an 80% decline in late orders.

3.4 Distribution

Data and workload distribution is about moving data between different servers that run parts of the same application.

This setup allows the workload to be efficiently distributed across multiple server environments. Hardware vendors, such as IBM, are extending their server ranges from the low end to the high end, which makes this proposition increasingly viable. Software vendors often price their applications based on the processing power band of the host server. It makes economic sense to offload workload, for example, query and reporting, onto other servers. Some servers handle batch work (such as query) better than interactive work (such as user query and data analysis), and they also do this work at a much lower cost. Moving data around to multiple servers in an organization also produces improved availability resilience.

InfoSphere CDC can satisfy these data and workload distribution requirements in near real-time scenarios. An example of data and workload distribution is shown in Figure 3-9.

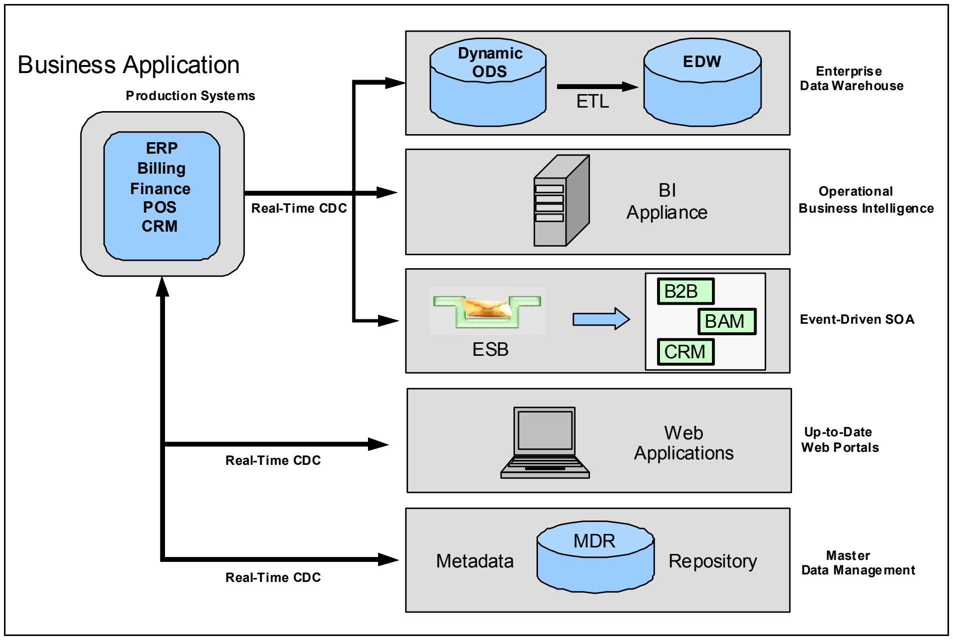


Figure 3-9   Data and workload distribution

3.4.1 Distribution: Sample implementation 1

In this example, a retail department store client has several stores, and a distribution center, which runs a corporate IBM System i server. The client requirement is to upgrade their electronic point of sale (EPOS) equipment in all stores to SQL Server Clients. Within each store, the new EPOS terminals are connected to a consolidation server also running SQL Server. It is important that all SQL Server EPOS terminals have access to up-to-date information for the many thousands of products that the company sells.

The master files are maintained centrally on the System i server at the distribution center. The volume of sales transactions varies significantly throughout the year and is highest in January during the yearly sell-off sale. It is important that the distribution center is kept informed about sales transactions throughout the day for stock level and distribution purposes.

Distribution: Proposed solution 1

The client implements a solution that uses InfoSphere CDC to distribute data changes to the consolidation server in each store. Using the InfoSphere CDC Scheduled End (formerly known as Net Change) functionality, only changes in the master files are replicated, thus reducing the replication workload. Sales transactions are mirrored continuously to the distribution center, which provides a reliable and up-to-date position in each store. The solution is shown in Figure 3-10.

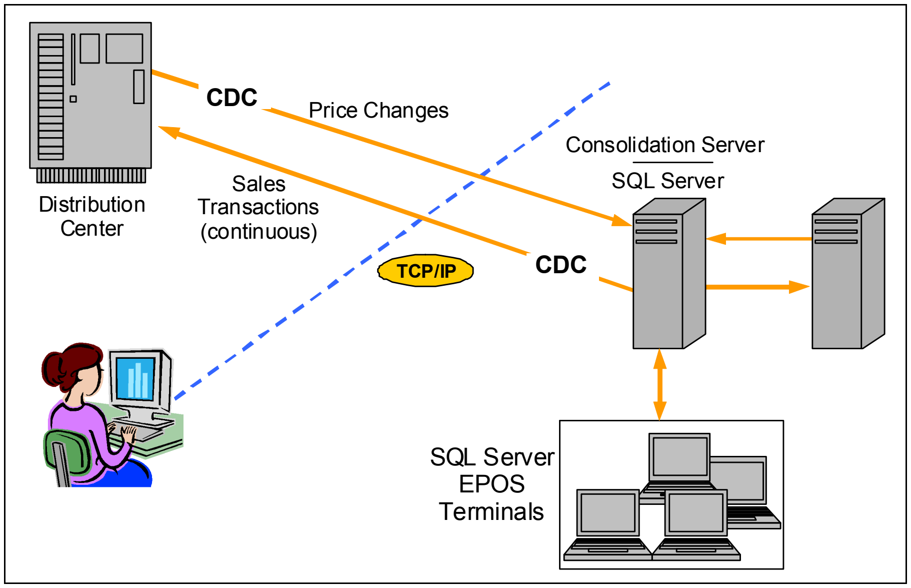


Figure 3-10 InfoSphere CDC Scheduled End

3.4.2 Distribution: Sample implementation 2

A brokerage firm wants to develop an in-house Global Common Database application that could be available at each of their locations worldwide. The application allows entry of new trades from the stock exchange floors to an application using a SQL Server database and running on Microsoft Windows. The client has an existing communications infrastructure with utilities to communicate between all their System i servers worldwide. The first location to go live would be London and then the solution would be rolled out to other locations. The database structures for SQL Server are different from the structures on the System i servers.

Distribution: Proposed solution 2

The client implements a solution that uses InfoSphere CDC to continuously mirror data between SQL Server servers and System i servers using a local LAN and TCP/IP plus an existing worldwide network to distribute data around the world. The ALIAS functionality was used extensively to remap columns from SQL Server tables to fields in System i physical files. SQL Server allows long column names, but System i has a maximum field size of 10.

The implementation time is short, and exceeds the client's performance expectations. The client initially considered developing an in-house replication solution, but rejected it because it would have taken many months to develop, have unknown performance, be expensive, and require a programmer each time changes needed to be made. The solution that was used is shown in Figure 3-11.

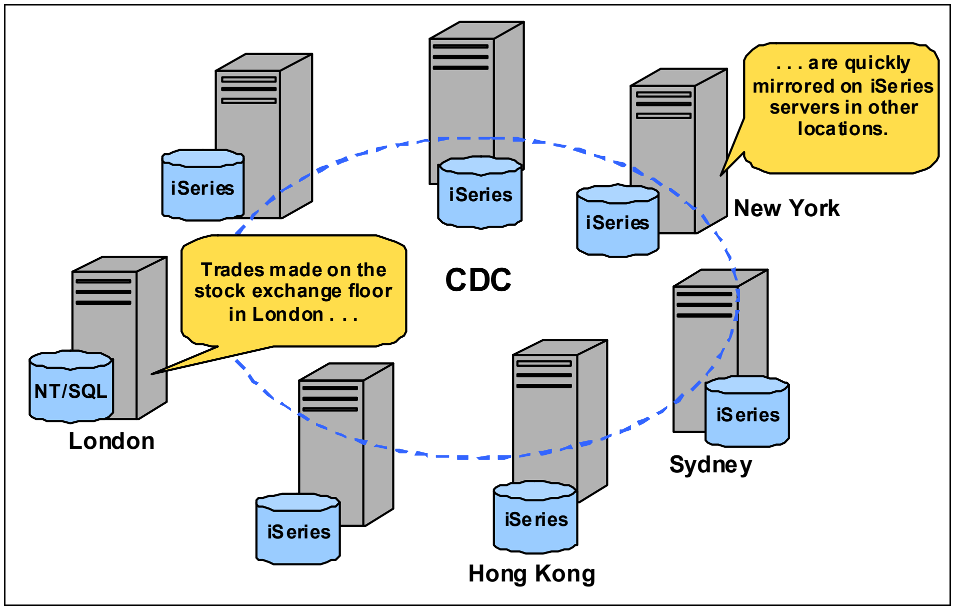

. . . are quickly mirrored on iSeries servers in other locations.
iSeries
iSeries
iSeries
New York
Trades made on the stock exchange floor in London . . .
CDC
NT/SQL
London
iSeries
Sydney
iSeries
iSeries
Hong Kong

Figure 3-11 Global database

3.5 Database migration

Database migrations might be necessary in many organizations. For example, if the company changes their hardware, performs major database upgrades, or switches their business application from one database to another. The basic idea is that whenever a company must migrate their database from one server to another, the process must be smooth, uninterrupted, have minimum downtime, and be less expensive.

InfoSphere CDC can load data from one server to another using the InfoSphere CDC Refresh functionality or the set bookmark capability to set the point from which InfoSphere CDC starts capturing delta changes (after using other vendor tools to load the data on the target database):

- The Refresh capability provides a full copy of the data on the target side while your source applications are performing normal daily business operations without any interruptions. After refreshing the data, InfoSphere CDC continues capturing the delta changes and applying them on target side.
- If the client wants to use another vendor's tool to load the data and then capture the delta changes for their daily business activates. InfoSphere CDC is flexible enough to adjust to this business need by using the Mark Table Capture Point function.

3.5.1 Database migration: Sample implementation

SAP heterogeneous system copy is defined as the change of the hardware or database platform for an SAP instance. Many SAP systems can be migrated with SAP standard tools (such as R3load), which are also available to IBM and other service providers. The requirements can become critical for very large SAP databases, because they typically have small windows of downtime.

The overall approach for porting SAP instances with minimum downtime to another hardware or database platform is based on a combination of using InfoSphere CDC for the 30 - 50 largest tables in the SAP database. You then use standard tools provided by SAP (such as R3load) for the remainder of the tables.

Database migration: Proposed solution

IBM implements an InfoSphere CDC solution SAP utility that fulfills the client business requirements (Figure 3-12).

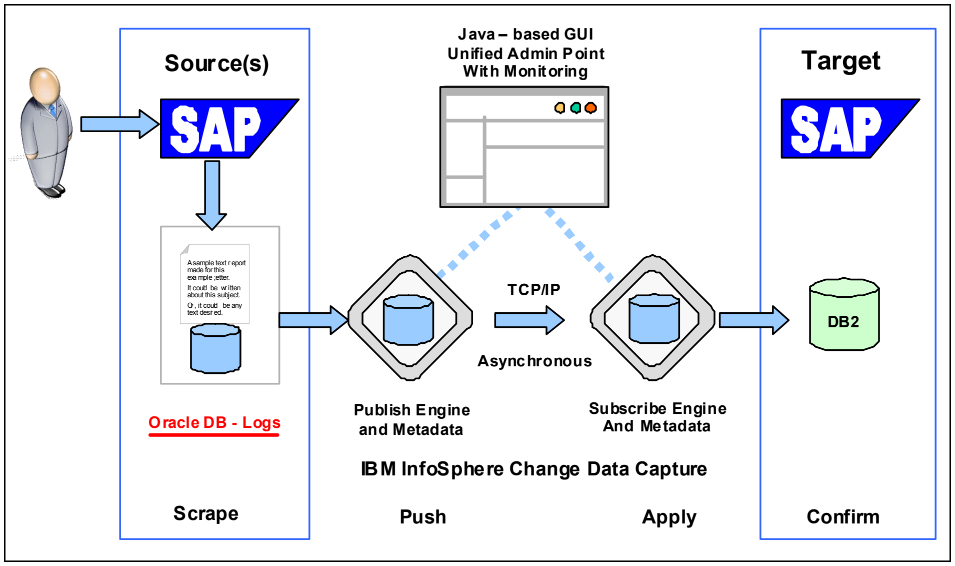

Source(s)
SAP
Oracle DB - Logs
Scrape
Java – based GUI Unified Admin Point With Monitoring
TCP/IP
Asynchronous
Publish Engine and Metadata
Subscribe Engine And Metadata
IBM InfoSphere Change Data Capture
Push
Apply
Target
SAP
DB2
Confirm

Figure 3-12   SAP heterogeneous system copy

The implementation consists essentially of the following actions:

1. Build the target DB2 database and SAP application by running SAPINST until the database load is performed by SAPINST or the migration monitor.
2. Create the large tables in the target system using the SAP R3load utility. This action ensures the consistency of the data definition required for the SAP data dictionary.
3. Start initial data transfer using the Refresh function of InfoSphere CDC. This function reads the data from the source database while the source SAP system is up and running. This action might have a performance impact and should be initiated during times with little workload on the table in question.

   Alternatively, R3load can be used for initial synchronization. InfoSphere CDC is then only used to replicate the delta changes starting from a marked Table Capture Point, which must be consistent with the R3load export.
4. Start replicating the data by scraping the log files and apply the data to the target table.
5. When tables on source and target are synchronized and a downtime window is available, the remaining tables are migrated using the R3load utility.

3.6 Application integration

In this example, an organization wants their different running applications linked together so they can automate their business process.

InfoSphere CDC integrates client applications with each other, which can provide the following benefits:

- InfoSphere CDC unidirectional functionality provides full integration of the data from the applications.
- InfoSphere CDC bidirectional functionality allows the application and user to enter the data from both sides (source and target). InfoSphere CDC makes sure that the data is synchronized between the source and target databases.
- InfoSphere CDC can be used in such a way that data can be extracted from the source database and provided to a client on a handheld device application.

3.6.1 Application integration: Sample implementation 1

A client wants to integrate their billing application data with their CRM process to better serve their customer needs and improve their business process. Here is the list of their objectives and environment:

- Improve customer retention against lower-cost operators.
- Provide better customer service by being able to react in real time.
- No extra bandwidth for this type of processing (the billing system is already overloaded).
- Pre-paid customers should receive short message service (SMS) messages when low on minutes, which prevents customers from losing service, and improves the customer experience. An additional benefit is that revenue is increased.
- When customers activate a new phone, they should be able to start using it immediately because of instantaneous billing.

Application integration: Proposed solution 1

IBM implements an InfoSphere CDC solution, which integrates the client's billing system data with their CRM system by using the InfoSphere CDC event server messages functionality. InfoSphere CDC scrapes the data from source database and generates XML messages that are passed to target queues. Then the Enterprise Service Bus (ESB) application picks up the data from InfoSphere CDC messages queues and makes them available to the CRM system.

The results are:

- Low impact (minimal additional workload on the billing system)
- High volume (typical for billing systems)
- Real-time (Changes are captured and responded to immediately.)
- Easy to implement (Uses the existing infrastructure.)
- Real-time visibility of customer billing information in the CRM system
- Minimal impact on the operational system
- Flexible for changing data requirements
- Minimal latency
- Scalability

The solution is shown in Figure 3-13.

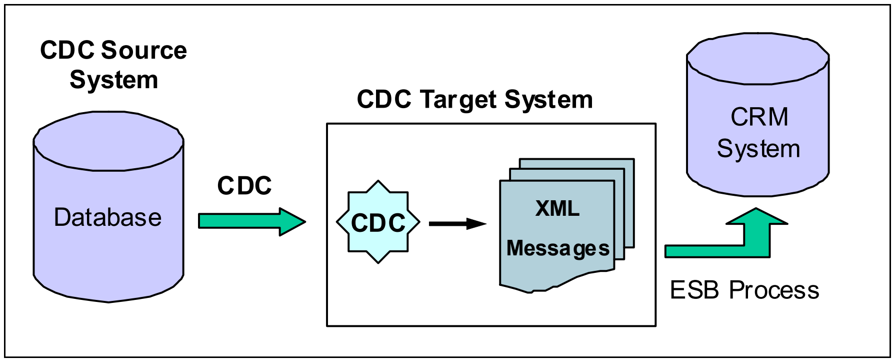


Figure 3-13 Integrating billing and CRM

3.6.2 Application integration: Sample implementation 2

A leasing company is planning to integrate their front-end application with their back-end application. The objective is that the customer wants quick quotations using the front-end application.

The front-end application's strengths are user and web interfaces. The back-end application's strengths are mainly in the calculation area. The quotations are calculated quickly and with maximum flexibility.

To speed up the company's integration process, the company decides to use the strengths of both applications and have them work together. The front-end application enters the information required for a quotation, and the back-end application would do the calculations.

Integration between the applications is a great challenge, primarily because of database differences, which includes code page differences.

Application integration: Proposed solution 2

InfoSphere CDC is installed and configured with a bidirectional solution between these two servers to meet client business objectives. The project has already started and is struggling when InfoSphere CDC is implemented. The result is that development costs are reduced and the application integration goes smoothly, resulting in a quicker time to market. The solution is shown in Figure 3-14.

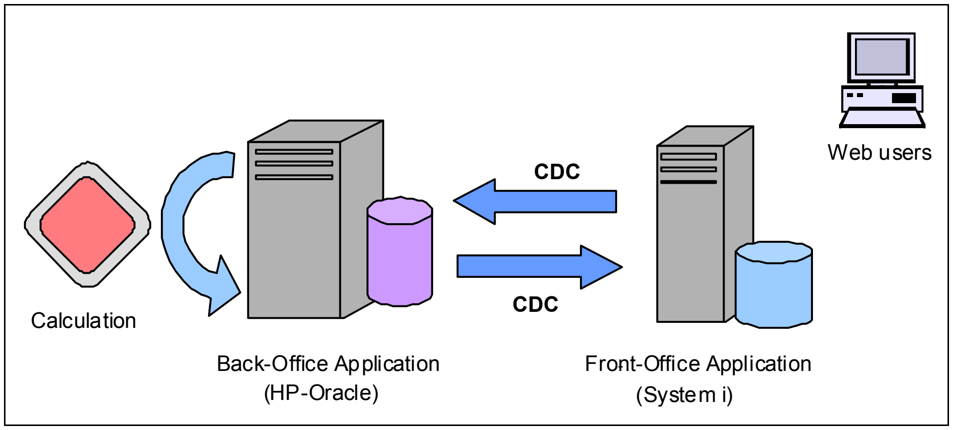


Figure 3-14 Application integration

3.6.3 Application integration: Sample implementation 3

A Fortune 500 retailer needed to integrate their website with their production systems to serve their customers in real time while enhancing their own internal operations.

To provide a superior shopping experience to its customers, the company improves their online gift registry system by implementing real-time data integration. For example, when a customer makes a purchase on the website, the gift registry system is updated immediately, such that no duplicated purchase can occur to cause customer dissatisfaction and lost sales through product returns and refunds. At the same time, the inventory count is adjusted to reflect the new product statistics to all other customers. As a retailer of exquisite service, having real-time data is a mission-critical capability. By integrating the web application to corporate systems, the company is able to deliver accurate information that is updated continuously in real time.

The company's internal operations also benefit greatly from real-time data. Because the company's products are not produced in high volumes, inventory management requires careful planning, especially with 150 stores and product demand that can come from any part of the world. By implementing a real-time infrastructure, the company is able to receive the most updated ordering information, which enables them to make the most informed business decisions in managing its inventory. This setup enables the following capabilities:

- Access to up-to-date inventory and product information for online purchasing
- Purchases made online are accurately reflected in the inventory system

Application integration: Proposed solution 3

The following solution is implemented with the following results:

- Implement InfoSphere CDC. InfoSphere CDC real-time data flows to the e-commerce application for online customer purchasing, then back to the inventory systems after purchases are made.
- Increase customer satisfaction with more accurate online gift registry information, which is a significant portion of the company's online business.
- Improve inventory tracking results due to more effective inventory management.

3.7 Integration with master data management

Leading organizations are working to gain control of their most important and commonly shared enterprise information assets. They are looking for adaptive solutions to help them make more informed business decisions. These initiatives are based on master data management (MDM), a practice designed to achieve and maintain a single version of data across the enterprise. Managing product master data plays an important role in any company's measure of business performance. Establishing a single view of product master data and associated domains, such as suppliers, vendors, and locations, enables companies to address key strategic business initiatives throughout their organization. An implementation for product master data might require features such as a flexible data model, granular data access and security, real-time access, workflow, rich user interfaces, and business logic.

IBM InfoSphere Master Data Management Server is an enterprise customer master data management application that provides a unified customer view and updated environment to multiple channels. It aligns these front-end systems with multiple back-end systems in real time, providing a single source of customer data across the enterprise.

The IBM InfoSphere MDM Server enterprise application uses a relational database management system that contains the operational tables, history tables, reference tables, metadata tables (business objects and attributes, and MDM transactions and actions), management tables, and configuration tables.

InfoSphere CDC provides near real-time data to applications, such as ETL tools. Those applications can then feed the data to an MDM server in near real time.

3.7.1 Integration with master data management: Sample implementation

A high-level solution architecture design provides an enterprise-level system implementation design for collecting, improving, retaining, and distributing information that describes the customer's enterprise master data. It is important to note that the architecture described is a high-level representation of the key components for an MDM Server system based on information available now.

Integration with master data management: Proposed solution

This solution uses InfoSphere CDC to extract changes from the mainframe DB2 logs, and then the MDM services load the MDM database, not the Rapid Deployment Package (RDP) asset. For example, an initial load might consist of 16.5 million customer records, while a delta load would be much lighter, for example, only around 20 K rows per day.

The data volume for the delta load (20 K rows per day) is much lighter than for one initial load (16.5 million customers). A record is added to a flat file whenever a change occurs in a table. This process typically occurs during the day, and at a predefined time the files are closed and the update to MDM is initiated in batch. This situation is shown in Figure 3-15.

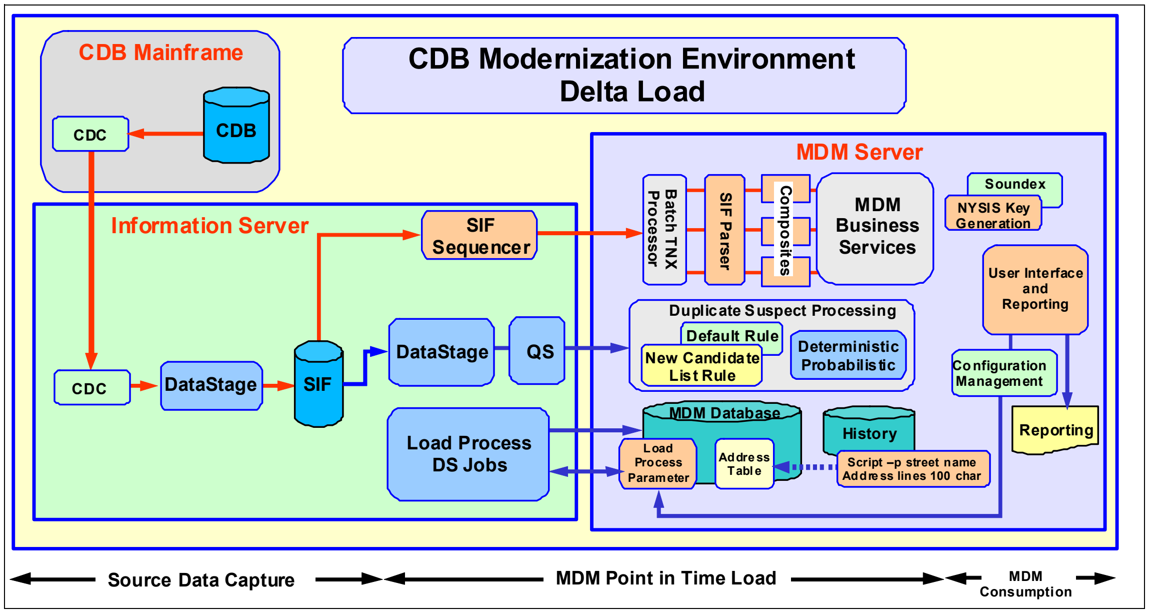


Figure 3-15 Integration with MDM

RDP has a limitation when processing Standard Interface Format (SIF) records with multiple updates of the same type in the same run. For example, if an address for a customer is changed more than once during the delta capture period, only the last row is processed.

Initially, once a day was sufficient, but now it should be run more often. After InfoSphere CDC *hardens* (marks as being complete) the landed files, the delta ETL job can run. If multiple changes are in these files, only the last update is processed. Although this situation should not be a problem because the most up-to-date change is reflected in MDM, the prior change history is lost. The process includes the following actions:

- InfoSphere CDC processes the delta changes from the database log and sends them to an InfoSphere CDC process on the Information Server, which saves them to a local directory in the Flat File format.

- ETL transforms the CDC / CDB data into a Standard Interface Format (SIF) file. The address lines must be calculated (based on codes), while in the initial load this task has been completed by the client. Address data is captured by the InfoSphere CDC functionality (user exit) so that you get the address whether it has been changed.

  The layout of the Standard Interface Format (SIF) is a Flat File format delimited by the pipe symbol between columns and the new line character between records.
- The SIF Sequencer runs. This process consists of ETL jobs designed to convert the regular SIF input into multiple files that contain SIF records for delta maintain transactions.
- The SIF Parser and Maintenance Business Transactions are used to load data through the Batch Framework. A role-based Sample User Interface (UI) is also included. This UI is an umbrella GUI that incorporates a Reporting UI, a registry that serves and renders Business Intelligence and Reporting Tools (BIRT) reports, a Customer Matching Critical Data Rule UI that allows defining matching criteria, a Data Stewardship UI, and a Party Maintenance UI to view and maintain data in the MDM data repository.

3.8 Integration with IBM Information Server

IBM Information Server is a leading solution in the market that integrates data, performs cleansing and transformations, and delivers information to your critical business initiatives.

InfoSphere CDC provides near real-time data for building business initiatives. InfoSphere CDC scrapes the near real-time data from source systems and either writes it into a flat file or uses direct connect by using CDC for DataStage on the target system. The ETL application reads the data from the flat file and processes it into the target.

3.8.1 Integration with IBM Information Server: Sample implementation

In this example, a company creates a Bulk Cumulative Volume of their database once a night for the ETL server. The ETL application uses that data for a comparison between the Bulk Cumulative Volume copy and the existing copy. The delta changes are loaded into the Data Warehouse by using the ETL application. The entire process takes more than 22 hours.

The company has the following requirements.

- The company wants access to more near real-time data for various business initiatives and requirements.
- The company wants to reduce the total processing time to feed the data to a data warehouse.

Integration with IBM Information Server: Proposed solution

The proposed solution is shown in Figure 3-16. It shows that the billing application generates database logs on the production server. To move the data, a script is written on the source side that, using FTP, transfers the archive logs to the InfoSphere CDC and DataStage server upon creation and every 15 minutes. The InfoSphere CDC capture agent has a database connection to the production database, and processes the new logs on the InfoSphere CDC and DataStage server, scraping information from the archived logs. The InfoSphere CDC Apply process also runs on that same server, and generates flat files. Those flat files are then written to the data warehouse.

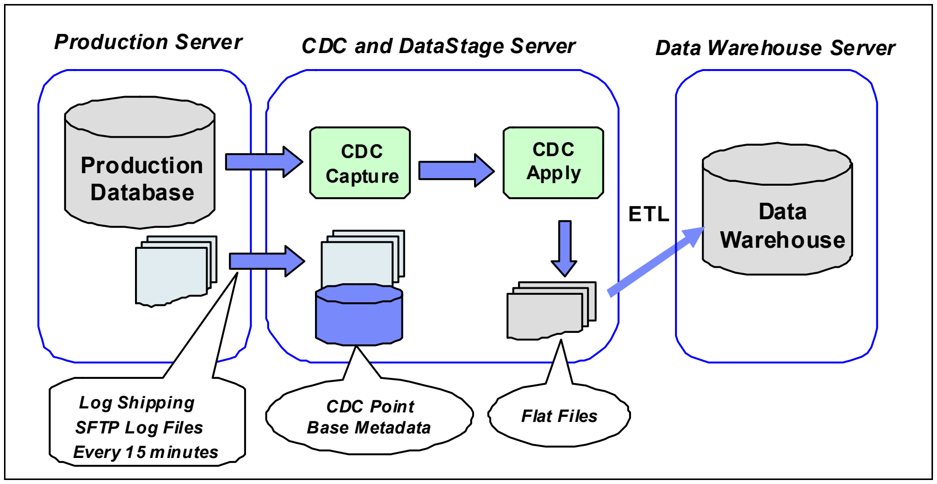


Figure 3-16 Integration with IBM Information Server

3.9 Operational business intelligence

Operational business intelligence, sometimes called real-time business intelligence, can enable companies to achieve their business needs.

Companies have good reasons to develop BI systems to fulfill their operational needs. As an example, BI has become a company's tool that provides comprehensive views of business directions. BI systems provide quantitative data to manipulate data, which enables business analysis and supports decision making. Most of the BI systems are supported by an information management infrastructure that includes data warehouses, data marts, and other integrated data resources, such as an operational data store (ODS). The ODS provides access to data that can help to show the company's current situation.

InfoSphere CDC also plays a vital role in building near real-time operational BI, as a way for companies to fulfill their daily business needs.

3.9.1 Operational business intelligence: Sample implementation

A large company is building their data warehouse to see the daily trends of their customers so they can make quick decisions when fulfilling their customer needs. The company has the following challenges in building their operational BI:

- Effectively loading production data into the data warehouse for business intelligence analytics
- High impact of batch extraction from production system, which places operations at risk

Operational business intelligence: Proposed solution

The company implements InfoSphere CDC and installs and configures it on their source and target systems (Figure 3-17). They use InfoSphere CDC time stamp and soft delete functionality to achieve this business need, as follows:

1. InfoSphere CDC is used to extract and integrate data into the operational data warehouse.
2. Time stamps are added to operations in the ODS and used by the ETL tool.
3. Delete operations are soft deletes in ODS. If the delete operation deletes the record on the target side, InfoSphere CDC flags that record.
4. Granular data provides detailed information for complex analysis to identify customer trends.

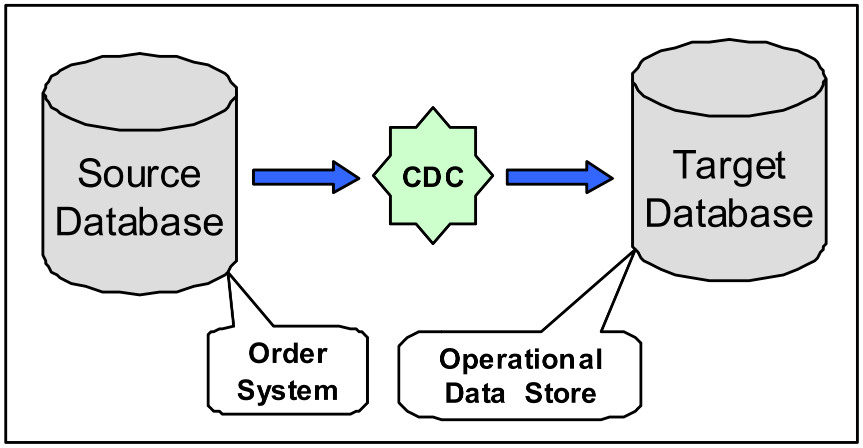


Figure 3-17 Operational BI

4

Solution topologies

This chapter describes the solution topologies and the flexible implementations available when using IBM InfoSphere Change Data Capture (InfoSphere CDC). To position these topologies, this chapter describes the potential benefits of each one and provide examples of their use.

Timing and flexibility are everything in today's competitive business environment, especially when it comes to business information. The increasing expectation that services are available 24x7, combined with the growing demand for real-time reporting and analytics, means that data must be constantly accessible.

However, critical business information is not always available to the people who need it. The data might be out of sync or inaccessible due to overloaded system resources, but the result is often the same: reduced productivity and profits, and diminished customer service.

The ability to use existing underutilized systems to offload reporting or data processing helps companies from needing to increase the size or number of *single use* systems. InfoSphere CDC allows systems to be used in a true master / master (bidirectional) mode for many years. This proven functionality means that data can be selected for reporting from systems, and users can change data that can be replicated back to the primary or other systems to maintain data consistency. Being able to use the processing power of multiple machines, which might be underutilized, as a single system reduces the need for expensive upgrades or replacement of overloaded production systems.

CDC solutions can support any IT infrastructure or environment. They can replicate or mirror data between heterogeneous database versions, and synchronize data between disparate systems. Many replication products require both source and target databases to be the same version *and* be on the same platform. This limitation requires upgrades to the database be done simultaneously, which in many cases requires a database outage during that time.

CDC allows database upgrades to be done independently, which provides maximum uptime of systems, thus reducing the impact of testing and migrating to newer versions. Cross database version replication also allows applications dependent on a particular database version to be kept in sync. CDC allows similar capabilities for cross hardware platforms and operating system versions.

4.1 Unidirectional replication

Unidirectional replication is the movement of data in one direction from the source tables to the target tables, and is used for data redundancy and synchronization (Figure 4-1).

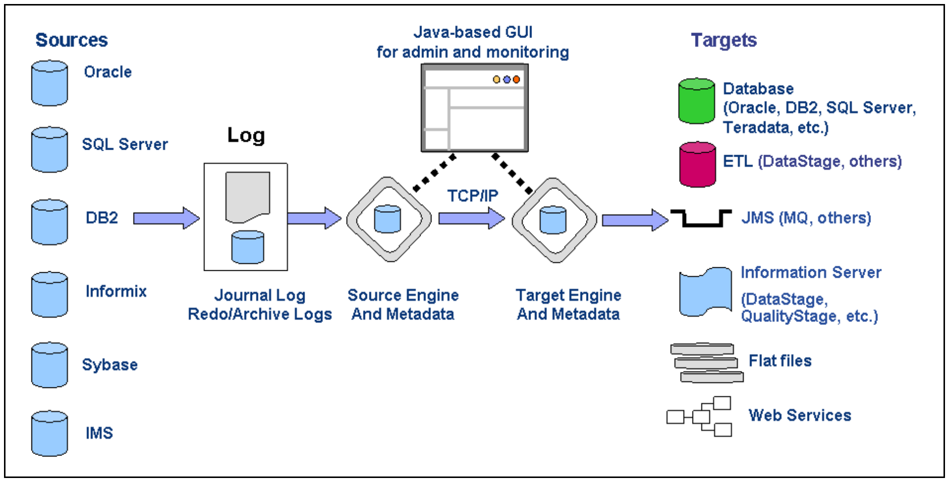


Figure 4-1 InfoSphere CDC unidirectional architecture - source to target

Your source and target tables can be of different types. You can transform the data that you replicate between the source and the target. You can map tables one at a time using standard replication.

The InfoSphere CDC Management Console provides the following two mechanisms for mapping using standard replication:

- One-to-one: Map tables using one-to-one replication when you want to map multiple source tables to multiple target tables at a time. These tables share a table structure and similar table names. The Map Tables wizard automatically maps tables based on an example mapping you define and set up.
- Custom: Map tables individually when you want to map only one source table to one target table at a time. These tables are source tables that might not share a table structure or similar table names as the target tables. This option is the option to map each source table in a one at a time fashion.

4.2 Cascading replication

Cascading replication involves a source system transmitting data to a target system which, in turn, serves as a source for the next system in the integration chain (Figure 4-2).

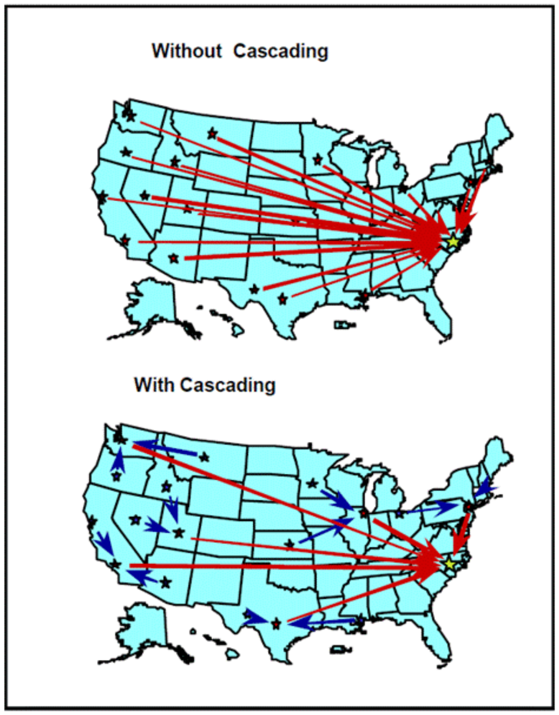


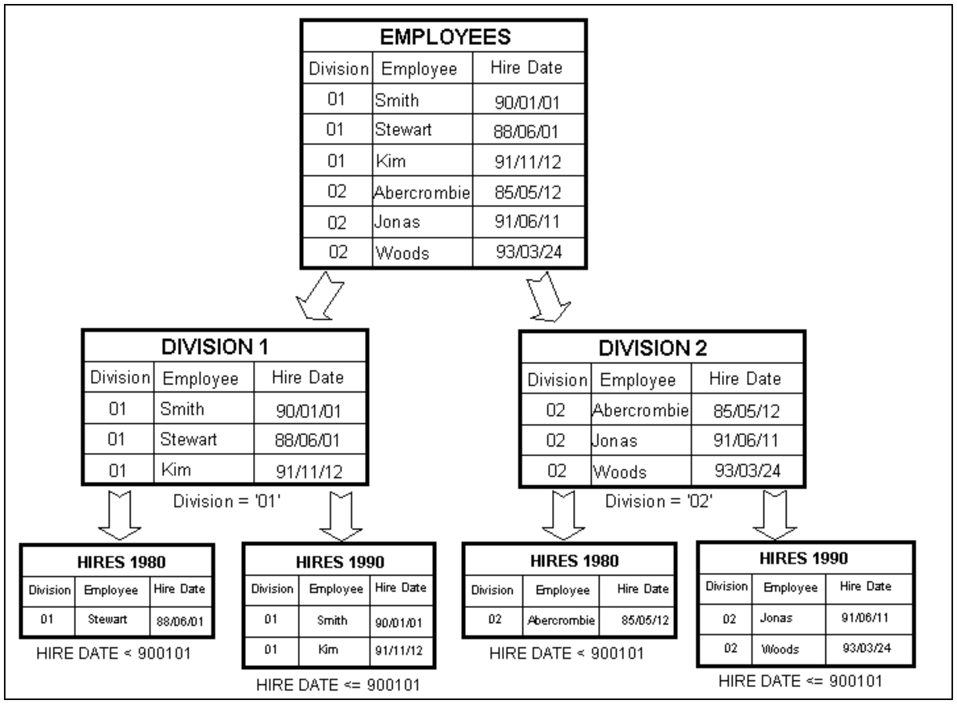
Figure 4-2 Cascading integration

Tools that support cascading integration enable the most efficient movement of data throughout a large organization.

For example, suppose that a company has 12 branch offices and a head office. If it takes an average of 15 minutes for each site to send its nightly data update to head office, the total integration time is three hours. In a cascading integration environment, this time can be cut. If three of the remote sites served as cascade points for the three offices closest to them, the time required to complete the integration process, and the accompanying communication costs incurred, could be reduced dramatically.

Cascading integration streamlines the integration process by enabling organizations to select regional cascade points.

You can use this type of replication to distribute changes across many servers using a multiplier effect. In Figure 4-3, employee data (EMPLOYEES) is replicated to two separate tables (DIVISION1 and DIVISION2), where each table contains data about employees in a specific division. Data from each division table is then replicated to other tables (HIRES1980 and HIRES1990), where the separation is based on hiring dates. Row selection to filter the data into the correct tables is not required if you want to replicate all data in the EMPLOYEE table to all destinations.



EMPLOYEES

Division	Employee	Hire Date
01	Smith	90/01/01
01	Stewart	88/06/01
01	Kim	91/11/12
02	Abercrombie	85/05/12
02	Jonas	91/06/11
02	Woods	93/03/24

DIVISION 1

Division	Employee	Hire Date
01	Smith	90/01/01
01	Stewart	88/06/01
01	Kim	91/11/12

DIVISION 2

Division	Employee	Hire Date
02	Abercrombie	85/05/12
02	Jonas	91/06/11
02	Woods	93/03/24

HIRES 1980

Division	Employee	Hire Date
01	Stewart	88/06/01

HIRES 1990

Division	Employee	Hire Date
01	Smith	90/01/01
01	Kim	91/11/12

HIRES 1980

Division	Employee	Hire Date
02	Abercrombie	85/05/12

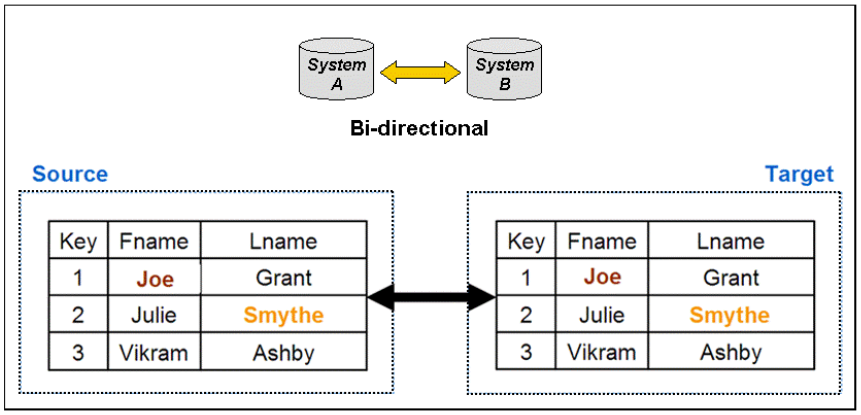
HIRES 1990

Division	Employee	Hire Date
02	Jonas	91/06/11
02	Woods	93/03/24

Figure 4-3 Cascading replication

4.3 Bidirectional replication

Bidirectional replication involves replication from the publication server to the subscription server, and replication in the opposite direction (Figure 4-4). If both systems are used to change the same tables, recursive updates occur. A change on System A is replicated to System B. The change is then replicated back to System A, and then to System B again. This process repeats itself many times. Bidirectional replication requires recursion prevention to prevent repetitive changes.



Key	Fname	Lname
1	Joe	Grant
2	Julie	Smythe
3	Vikram	Ashby

Key	Fname	Lname
1	Joe	Grant
2	Julie	Smythe
3	Vikram	Ashby

Figure 4-4 Bidirectional replication

Figure 4-5 provides another view of bidirectional replication from a systems and processes point of view.

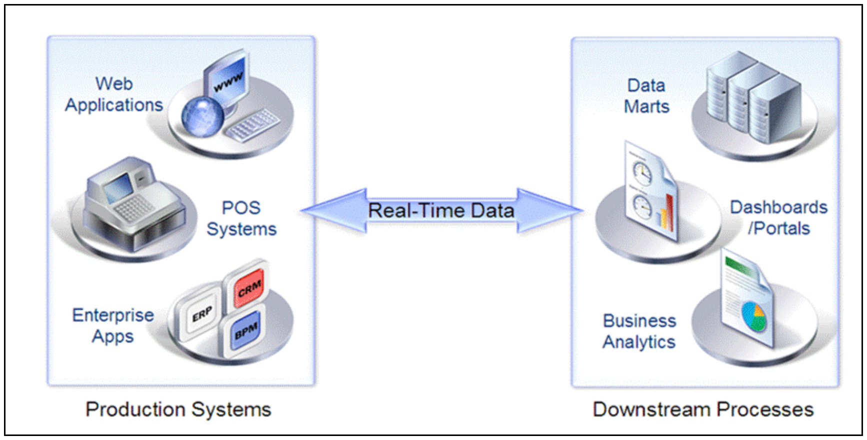


Figure 4-5 Bidirectional replication - a system view

With bidirectional replication, conflict detection and resolution are required, such as in a situation where the same record is updated on both systems at the same time. If there is a larger data latency with bidirectional replication, then there are transaction conflicts. This situation is why it is important to have the fastest replication possible for this type of replication. InfoSphere CDC provides real-time and bidirectional data integration and transformation between diverse relational databases and other data stores on different platforms. Through bidirectional mirroring, workload distribution allows data to be on more than one machine, with the users segmented between them. This setup can reduce the cost of maintaining a fragmented IT environment by enabling incompatible applications to coexist.

To implement bidirectional replication, you must install InfoSphere CDC on both servers, and each server must be able to send and receive replicated data. One of the benefits of the bidirectional capability of InfoSphere CDC is that it provides a rollback strategy for a migration. This strategy can result in, for example, a zero downtime migration capability. This strategy lets you resynchronize changes to the original source system after cutover when, for example, a post migration problem has occurred. Bidirectional replication can also provide e-commerce application synchronization, workload balancing, and application integration.

Another excellent use for bidirectional topology is data synchronization for upgrades, migrations, and workload balancing. This capability keeps data synchronized between the current production server and a deployed server, for example, to test a new application version upgrade or a hardware or OS upgrade. The workload balancing capability (master to master support) allows database instances to remain synchronized where dual or double data entry is required (such as when data entry is occurring on both systems at the same time) (Figure 4-6).

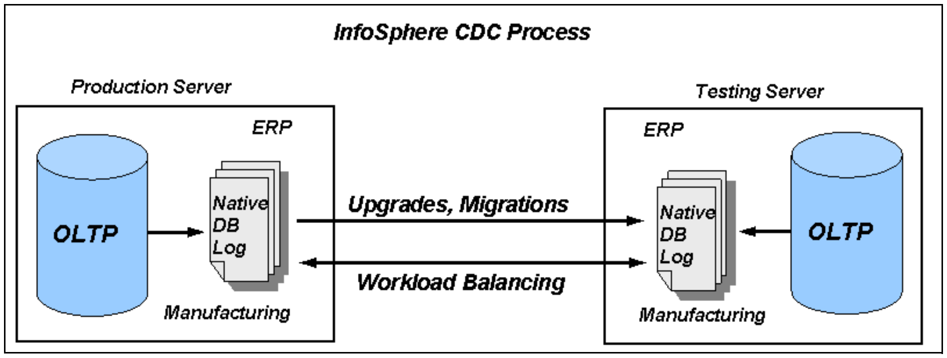


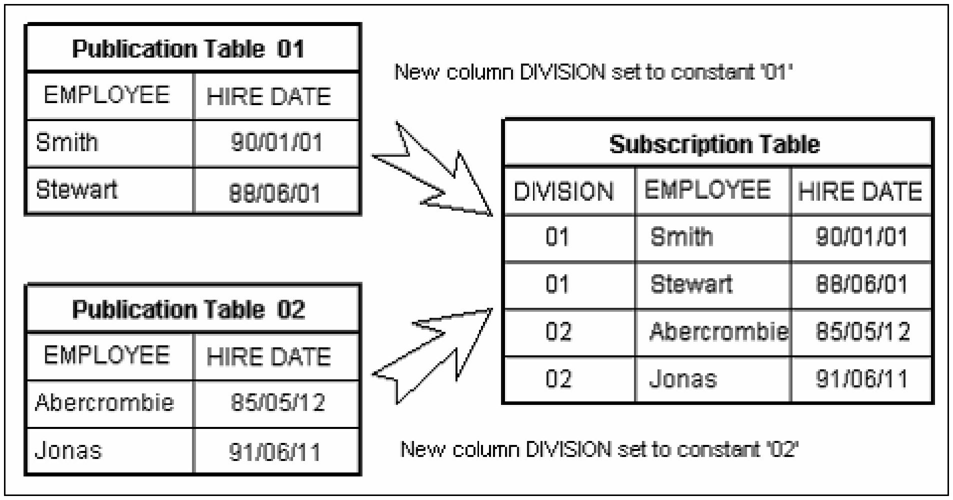
Figure 4-6 Data synchronization for upgrades, migrations, and workload balancing

4.4 Consolidation replication

InfoSphere CDC also supports the implementation of consolidation replication. In this implementation, data from multiple publication servers update a single subscription table. You must define each publication server separately, and then assign the publication tables to the subscription table that serve as the data warehouse. Because multiple publication tables are updating a single subscription table, the publication tables must have the same attributes.

For example, suppose that you need to create a data warehouse to consolidate employee records created and used by two separate divisions. Each of these divisions has a publication table that is maintained separately. A subscription table is used to consolidate the data from these two tables. On each publication server, you define the publication tables to be mirrored, and then transfer the publication table definitions to the subscription. Then you must work in the subscription environment to assign the publication table definitions to the actual subscription table. When defining the subscription table, a new column is added for the division data. After assigning the publication tables to the subscription table, map the new column to a different constant value in each assignment.

Using the example in Figure 4-7, map the DIVISION column to '01' for the first publication table, and map the DIVISION column to '02' for the second publication table. When data is replicated from one of the publication tables, the corresponding value is written in each row replicated to the subscription table. As a result, rows in the subscription table are identifiable by division. A row identifier defined for a table assignment is used to identify the rows in the table originating from the publication table.


New column DIVISION set to constant '01'
New column DIVISION set to constant '02'

Publication Table 01

EMPLOYEE	HIRE DATE
Smith	90/01/01
Stewart	88/06/01

Publication Table 02

EMPLOYEE	HIRE DATE
Abercrombie	85/05/12
Jonas	91/06/11

Subscription Table

DIVISION	EMPLOYEE	HIRE DATE
01	Smith	90/01/01
01	Stewart	88/06/01
02	Abercrombie	85/05/12
02	Jonas	91/06/11

Figure 4-7 Consolidation replication

One reason to use the consolidation topology is to build a low latency operational data store (ODS) for operational reporting and auditing (Figure 4-8).

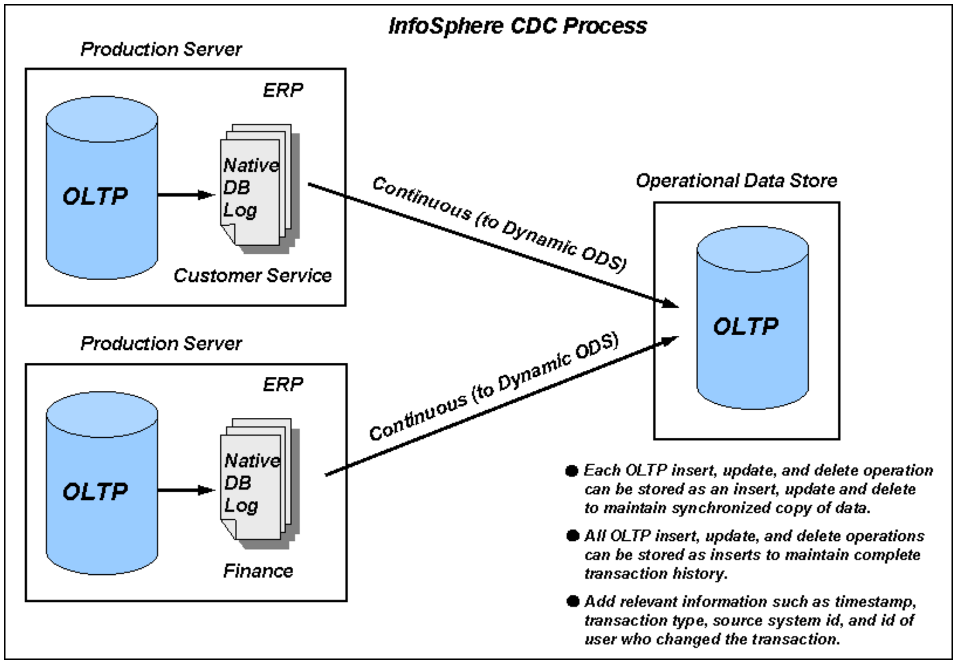

InfoSphere CDC Process
Production Server
ERP
OLTP
Native DB Log
Customer Service
Production Server
ERP
OLTP
Native DB Log
Finance
Continuous (to Dynamic ODS)
Continuous (to Dynamic ODS)
Operational Data Store
OLTP
● Each OLTP insert, update, and delete operation can be stored as an insert, update and delete to maintain synchronized copy of data.
● All OLTP insert, update, and delete operations can be stored as inserts to maintain complete transaction history.
● Add relevant information such as timestamp, transaction type, source system id, and id of user who changed the transaction.

Figure 4-8 Building a low latency operational data store

This ODS is used for companies:

- Looking to manage increasing data volumes and shrink batch windows for ETL processing, and to mitigate risk associated with extracting data from heavily used production environments.
- Needing to stream data to an operational data store, where an ETL product, such as DataStage or IBM Cognos® Data Manager, extracts, transforms, and loads data from an ODS into the data warehouse.
- Already employing ETL products, but looking to gain operational efficiencies by reducing the impact of extracts or increasing the timeliness of data for loading into data warehouse.

The business value of this setup includes such benefits as:

- Real-time operational reporting from the ODS
- More frequent ETL processing available by using the ODS as the source for loading the data warehouse

- Using your existing investment in ETL processes and tools
- Enabling more comprehensive Business Analytics by replicating update and delete operations as inserts into the ODS

There would also be technical value, such as:

- Low impact on source environments
- No changes to existing infrastructure

Another reason to use the consolidation topology is for off loading production query and reporting cycles. The Reporting server can also be used for consolidation requirements, such as consolidating financial information from multiple branches into a single corporate instance.

Replication frequency generally varies from continuous (near real time) to periodic. Table level refresh or copy can be used in addition to log based change data capture (Figure 4-9).

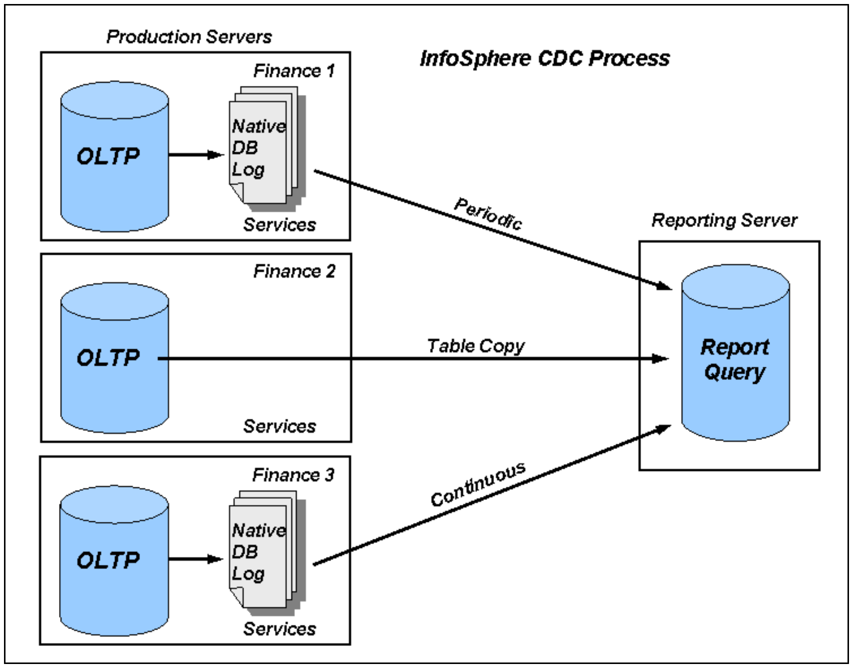
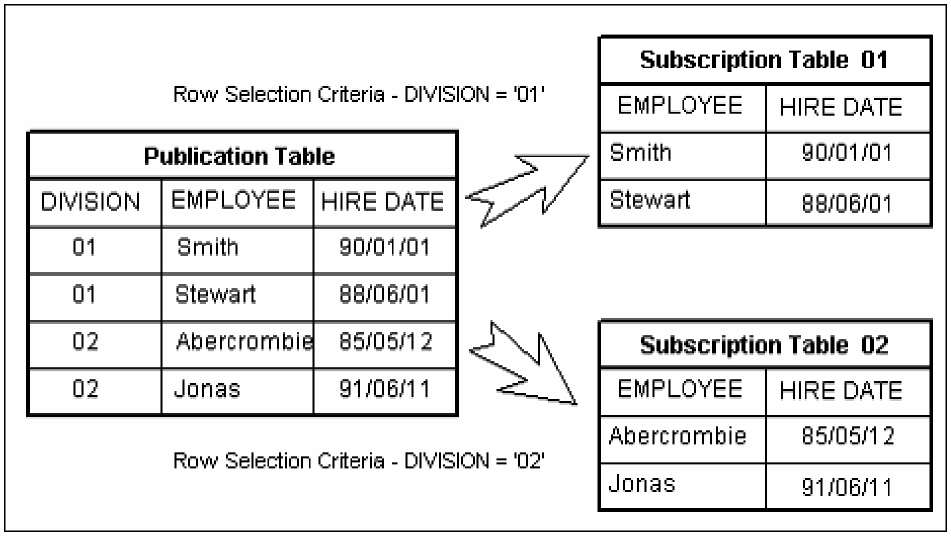


Figure 4-9 Off loading production query and reporting cycles

4.5 Data distribution replication

You can use InfoSphere CDC to implement data distribution, where a single publication table is used to update multiple subscription tables. As part of data distribution, you must transfer data that is relevant only to the subscription environment. For example, if data is distributed to separate divisions, you must apply row selection criteria so that only data relevant to a particular division is mirrored to its environment.

Frequently, the publication table has a column, such as DEPARTMENT, DIVISION, or COMPANY, that can be referenced in the row selection criteria to define the destination of replicated data (Figure 4-10). However, you can use any column or combination of columns for the selection criteria. Different row selection criteria can be defined for each possible destination. When replication occurs, each row in the publication table is evaluated for the values defined in the row selection criteria. These values determine the destination of the replicated row.


Row Selection Criteria - DIVISION = '01'
Row Selection Criteria - DIVISION = '02'

Publication Table

DIVISION	EMPLOYEE	HIRE DATE
01	Smith	90/01/01
01	Stewart	88/06/01
02	Abercrombie	85/05/12
02	Jonas	91/06/11

Subscription Table 01

EMPLOYEE	HIRE DATE
Smith	90/01/01
Stewart	88/06/01

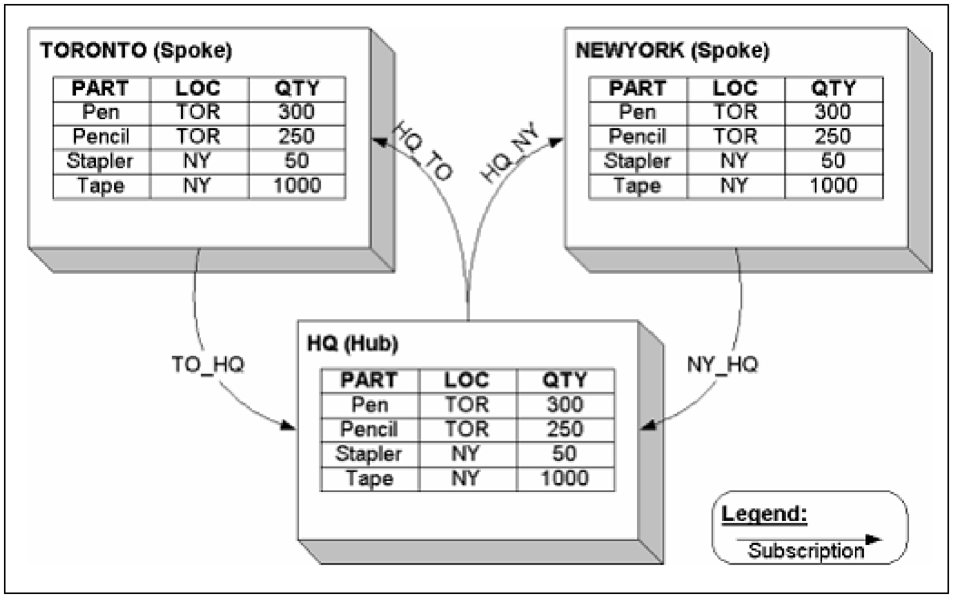
Subscription Table 02

EMPLOYEE	HIRE DATE
Abercrombie	85/05/12
Jonas	91/06/11

Figure 4-10 Data distribution implementation

4.6 Hub-and-Spoke replication with propagation

Hub-and-Spoke replication requires a configuration consisting of centrally-administered tables on a hub server that are simultaneously maintained on multiple subscription (spoke) servers (Figure 4-11). Changes applied to the tables on subscription servers (in this example, TORONTO and NEWYORK) are replicated to the same tables on the hub server (HQ), and then routed to the same tables on all other subscription servers in the Hub-and-Spoke configuration.



PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000

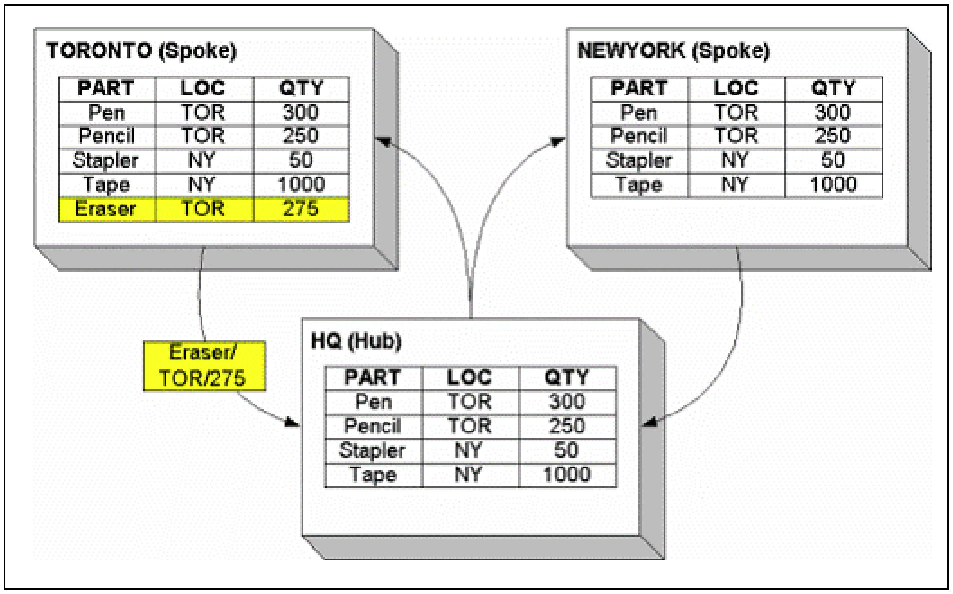
PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000

PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000

Figure 4-11 Hub-and-Spoke replication

In a Hub-and-Spoke configuration, it is possible for changes originating on one spoke to be replicated back to that spoke recursively. To prevent this situation from happening, you must configure propagation control for each hub-to-spoke subscription. Propagation control allows you to specify the corresponding spoke-to-hub subscription when defining the hub-to-spoke subscription.

Figure 4-12 shows a change on the TORONTO spoke being replicated along the spoke-to-hub subscription to the HQ hub.



PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000
Eraser	TOR	275

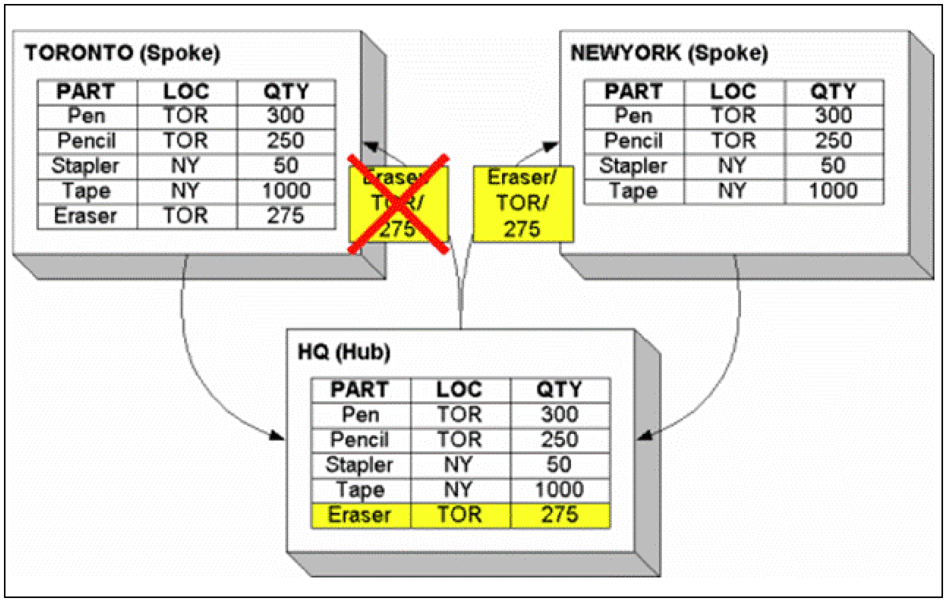
PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000

PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000

Figure 4-12 Replicating along a spoke-to-hub subscription

After applying the change to the HQ hub, the hub server then needs to replicate the change to its spokes. There are two subscriptions that replicate data from the hub, HQ_TO and HQ_NY. When defining these subscriptions, you must declare that you do not want HQ_TO to replicate changes that were originally applied to the hub by the TO_HQ subscription. The same is true for the HQ_NY and NY_HQ subscriptions.

Figure 4-13 illustrates how changes should propagate from the hub.



TORONTO (Spoke)

PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000
Eraser	TOR	275

NEWYORK (Spoke)

PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000

HQ (Hub)

PART	LOC	QTY
Pen	TOR	300
Pencil	TOR	250
Stapler	NY	50
Tape	NY	1000
Eraser	TOR	275

Figure 4-13 Propagation control

Before you can configure propagation control for a hub-to-spoke subscription, you must describe the corresponding spoke-to-hub subscription.

A good example to use for this topology is for e-commerce application synchronization. Here the topology provides continuous bidirectional synchronization between web-based applications and mission-critical business applications. It also helps organizations improve customer online shopping experience with improved visibility into inventory and customer shopping activities.

This situation is shown in Figure 4-14.

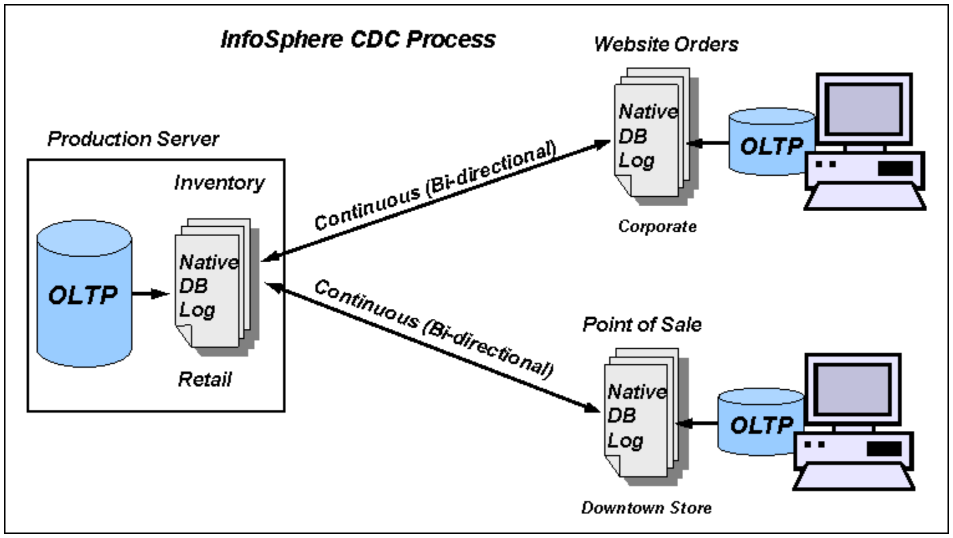


Figure 4-14 Application synchronization

4.7 Destination

This section describes the types of InfoSphere CDC destinations (targets) that are available to support InfoSphere CDC Hub (Figure 4-15). Depending on your business and environmental requirements, you use different destination options. For example, you might use a JMS Message Queue for event detection, such as a notification when inventory or balances reach a critically low level. For another example, you might use destinations, such as flat files or DataStage, to eliminate high impact nightly ETL batch processes. You can also use web services as a destination for a part of an event-driven architecture and service-oriented architecture (SOA).

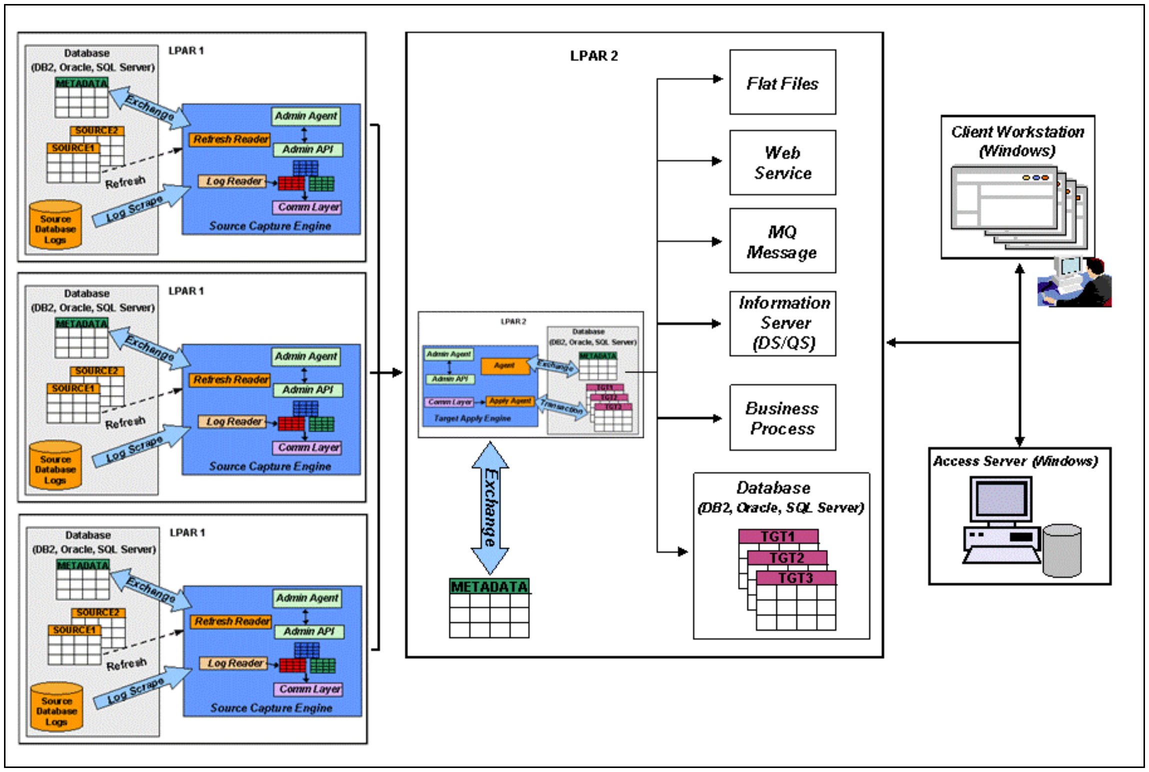


Figure 4-15 InfoSphere CDC architecture - source to InfoSphere CDC Hub

4.7.1 JMS Message Queue

InfoSphere CDC retrieves database operations from the source database and transforms data into XML messages.

Using the InfoSphere CDC Queue Targeting Mapping Designer, you can map XML elements and attributes to columns in a target table. When you start mirroring, InfoSphere CDC sends the XML message to a JMS message destination. You must install and set up an InfoSphere CDC source product that can scrape table-level operations (inserts, updates, and deletes) from a source database.

This source database represents the production database where your source tables are. When InfoSphere CDC detects a transaction, boundary, or commit operation, it sends the XML message to the JMS message destination.

Here are some examples of the kinds of business events that you can define on your production database:

- A new customer sale has been entered into the source database. You might want InfoSphere CDC to send an XML message to a JMS application that generates an event to different departments.
- A credit card balance changes significantly in a short period. You might want to track this change and notify the fraud department with real-time information about the credit card changes.
- Inventory levels are running low on a particular product. You might want to detect the low inventory and notify production management.

Figure 4-16 shows the architecture to support these business events.

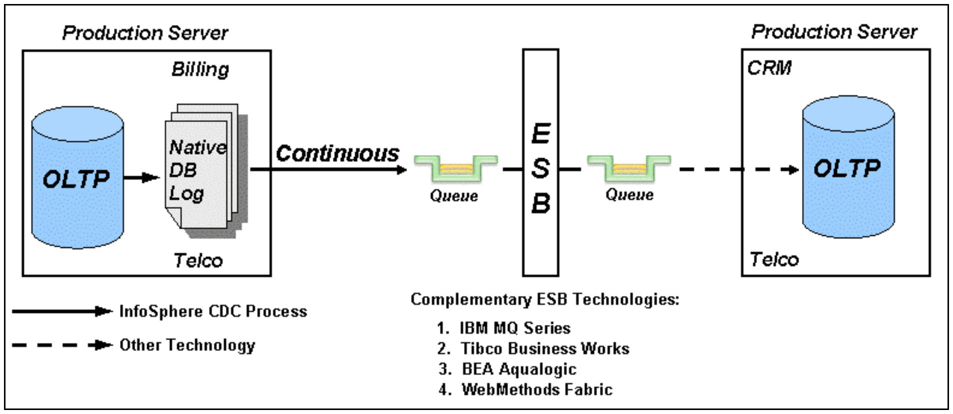


Figure 4-16 Event synchronization through an Enterprise Service Bus

Using the JMS Message Queue as a destination may be beneficial in the following situations:

- You are using a JMS compliant message-oriented middleware solution, such as IBM WebSphere MQ Server, WebSphere ESB, Tibco, or BEA WebLogic.
- You have built dashboards, KPIs, or composite applications with missing or stale data using messaging-oriented middleware, for example, Emergency Center monitoring to quickly identify patient bottlenecks, perform short-term trending, and identify process improvement and problem areas.
- You need real-time event data for business activity monitoring (BAM), for example, detecting changes to inventory in a retail store inventory system and triggering alarms for the store managers in an executive dashboard.
- You need event data to build a business process management (BPM) solution, for example, detecting unusual ATM transactions and feeding it into Fraud Prevention process built using Tibco.

4.7.2 Flat files

Flat files can be used by an existing ETL solution or can be used to feed a data warehouse appliance or relational database management system (RDBMS).

When replicating to flat files, choose the directory that holds the flat files. Set the number of rows or time threshold (seconds) when flat files are hardened (marked complete) for processing by the ETL solution. The flat file is closed and the next one is created and opened when either value is reached.

The flat file is only hardened on commit boundaries. If commit cycles are potentially larger or longer than the thresholds, the commit cycles take precedence. As a result, you might end up with more rows in the flat file than expected.

When you are using flat files, consider the following items:

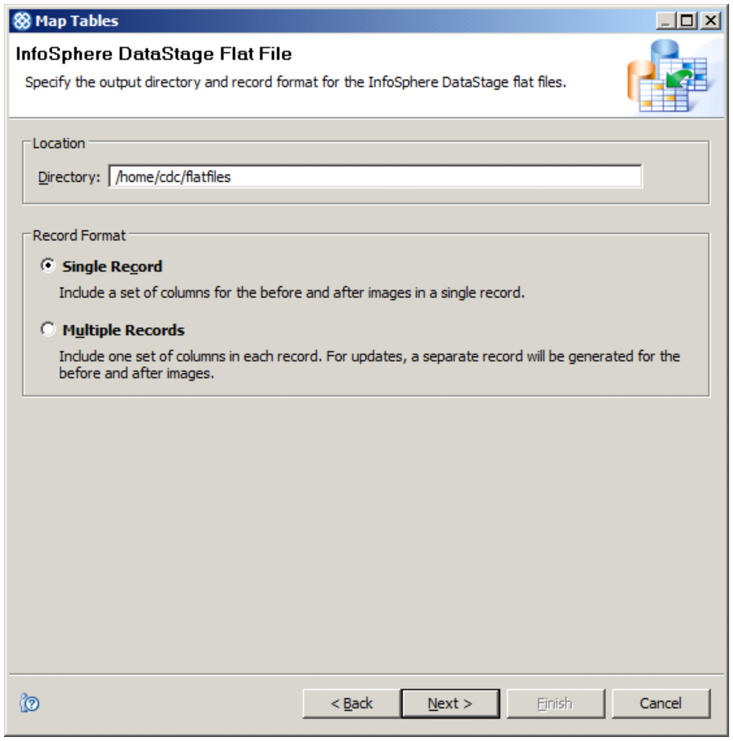
- Can be used by any ETL solution. Most ETL solutions support flat files as a source.
- Suitable for high volumes of changes.
- A flat file has a row (or multiple rows) for every table operation, which contain the following information:
  - Timestamp
  - Transaction sequence number
  - Operation type (Insert/Update/Delete (I/U/D))
  - User

– Before image of row

– After image of row

You can log the before and after images in a single row in the flat file or create one row with the before image and another row with the after image. This situation can be beneficial if the downstream process only needs the after image of an updated row.

The option for single or multiple record format stored for flat files is shown in Figure 4-17 (with the option to log the before and after image in a single row or multiple rows).



InfoSphere DataStage Flat File

Specify the output directory and record format for the InfoSphere DataStage flat files.

Location

Directory: /home/cdc/flatfiles

Record Format

[✓ **Single Record**]
Include a set of columns for the before and after images in a single record.

Multiple Records
Include one set of columns in each record. For updates, a separate record will be generated for the before and after images.

Figure 4-17 Option for single or multiple record format stored for flat files

For a single record, there is one line for an update transaction (U) containing the before update record image followed by the after update record image, as follows:

```
"2012-01-03 23:20:46","82950","U","DB2INST1","700700700","Timothy
Blitz","4 Sugar Forest Dr","IRVING","TEXAS","22598", "2012-01-03",
"77", "700700700","Timothy Blitz","80 Grandravine Dr","SAN JOSE",
"CALIFORNIA","95101"
```

For multiple records, there are two lines for an update transaction, with one line containing the before update record image (B) followed by the after update record image (A), as follows:

```
"2012-01-03 23:29:36","83161","B","DB2INST1","700700700","Timothy
Blitz","4 Sugar Forest Dr","IRVING","TEXAS","22598“

“2012-01-03 23:29:36","83161","A","DB2INST1","700700700","Timothy
Blitz","80 Grandravine Dr","SAN JOSE","CALIFORNIA","95101”
```

Choosing between single record and multiple records depends the design of the DataStage job. For example, if your DataStage job can safely process an UPDATE as a DELETE/INSERT pair, then using the multiple record format allows you to create a DataStage job that does not need to contain special logic to deal with updates. If your job does need to be aware of updates specifically, then single record format is the most convenient.

Figure 4-18 shows a situation where you replicate to flat files.

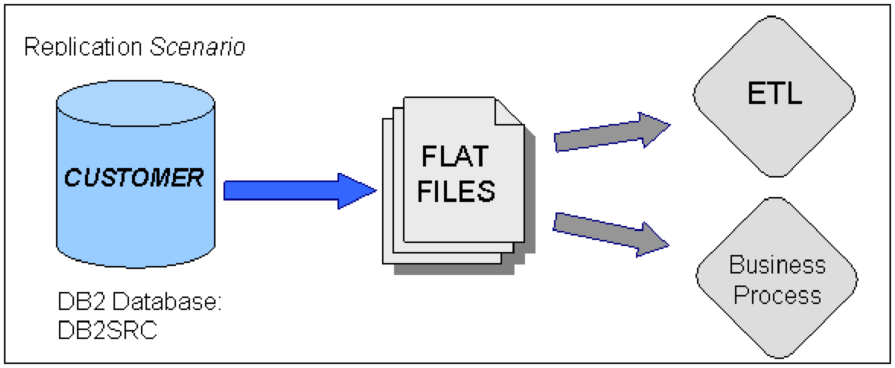


Figure 4-18 Replicating to flat files

Figure 4-19 shows replication to a business intelligence (BI) appliance.

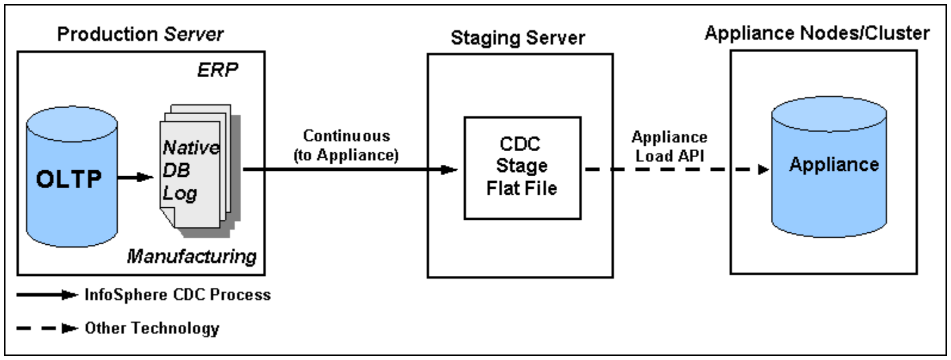
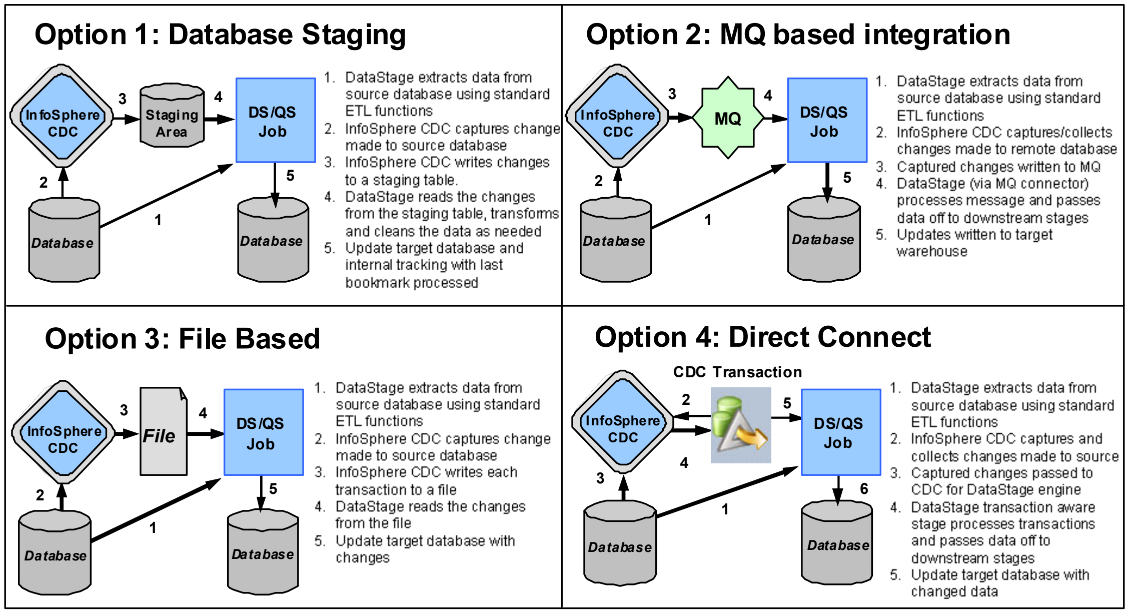


Figure 4-19 Replicating to a BI appliance

4.7.3 DataStage

This section describes the four methods of integrating InfoSphere CDC with DataStage (Figure 4-20). The most closely integrated option is Direct Connect, followed by the file-based method. There are other considerations, such as reasonable latency (time it takes from when the transaction occurred to when it was applied to the target), data volumes, and number of tables.



1. DataStage extracts data from source database using standard ETL functions
2. InfoSphere CDC captures change made to source database
3. InfoSphere CDC writes changes to a staging table.
4. DataStage reads the changes from the staging table, transforms and cleans the data as needed
5. Update target database and internal tracking with last bookmark processed

1. DataStage extracts data from source database using standard ETL functions
2. InfoSphere CDC captures/collects changes made to remote database
3. Captured changes written to MQ
4. DataStage (via MQ connector) processes message and passes data off to downstream stages
5. Updates written to target warehouse

1. DataStage extracts data from source database using standard ETL functions
2. InfoSphere CDC captures change made to source database
3. InfoSphere CDC writes each transaction to a file
4. DataStage reads the changes from the file
5. Update target database with changes

1. DataStage extracts data from source database using standard ETL functions
2. InfoSphere CDC captures and collects changes made to source
3. Captured changes passed to CDC for DataStage engine
4. DataStage transaction aware stage processes transactions and passes data off to downstream stages
5. Update target database with changed data

Figure 4-20 Integrating InfoSphere CDC with DataStage

Database staging method

For the database staging method, instead of feeding the changes from InfoSphere CDC directly to DataStage, they are written to a staging area where they are picked up by DataStage and then applied to the target. Having these changes in a staging area might be preferable for customers with multiple applications that require the raw change-only data.

Here is the process for this method:

1. DataStage extracts data for initial load using standard ETL functions.
2. InfoSphere CDC continuously captures changes made to the source database.
3. InfoSphere CDC continuously writes changes to a set of staging tables using audit type mappings
4. DataStage reads the changes from the staging tables, and transforms and cleans the data as needed.
5. Update the target database with changes.
6. Update internal tracking with the last InfoSphere CDC bookmark processed (The InfoSphere CDC bookmark is involved only to ensure that every change gets into the staging tables).

This method is ideal for the following situations:

- Low latency (minutes)
- Low / medium data volumes (a few thousand rows per second)
- Any number of tables

WebSphere MQ based integration method

The WebSphere MQ based integration is similar to the staging approach. Here, the InfoSphere CDC changes are written to WebSphere MQ, which are then picked up by DataStage and applied to the target warehouse. This approach might be preferable for clients who have an existing messaging infrastructure in their environment, or perhaps their message queue serves as the backbone for other services that also want to use the InfoSphere CDC data directly.

Here is the process for this method:

1. DataStage extracts data for the initial load using standard ETL functions.
2. InfoSphere CDC continuously captures changes made to the remote database.
3. InfoSphere CDC continuously writes change messages to WebSphere MQ through the InfoSphere CDC event server target.

4. DataStage (through the WebSphere MQ connector) processes messages and passes data off to downstream stages.
5. Updates are written to target database.

This method is ideal for the following situations:

- Near real-time integration (seconds)
- Low data volumes (hundreds of changes per second)
- When the infrastructure uses WebSphere MQ

File-based method

There is also the file-based method, where InfoSphere CDC changes are written to flat files, which are then processed by DataStage. This approach has the benefit that it provides recoverability in the sense that if the source or target machine fails, the replication process can resume from the last processed file. It does require more disk space because a landing area is required for the files.

Here is the process for this method:

1. DataStage extracts data for initial load using standard ETL functions or InfoSphere CDC can be used for refresh.
2. InfoSphere CDC continuously captures changes made to the source database.
3. InfoSphere CDC DataStage writes one file per table and periodically hardens the files.
4. DataStage reads the changes from the completed files.
5. Update the target database with changes.

This method is ideal for the following situations:

- Medium latency (a few minutes or more between periodic batches)
- High data volumes requiring parallel loading
- Less than 100 tables

Direct connect method

Direct connect is the most integrated approach. To summarize this method at a high level, the changes captured by InfoSphere CDC at the source database are streamed directly into DataStage, where they are then applied by a DataStage job.

Here is the process for this method:

1. DataStage extracts data for initial load using standard ETL functions or InfoSphere CDC can be used for the refresh.
2. InfoSphere CDC continuously captures changes made to source database and flows over TCP/IP to InfoSphere CDC Transaction Stage.
3. InfoSphere CDC Transaction Stage passes data off to downstream stages.
4. Update the target database with changed data. The bookmark persists in the target database along with the client data to maintain end-to-end transactional integrity.
5. InfoSphere CDC continuously captures the changed data from the source systems and passes it to a continuously active DataStage job.
6. The DataStage job receives transactions through the InfoSphere CDC Transaction Stage operator.
7. In the job, the data can be transformed or passed on to other downstream locations.
8. The InfoSphere CDC bookmark is maintained along with DataStage applying the changes to the target database.

This method is ideal for the following situations:

- Near real-time integration (seconds)
- Medium data volumes (hundreds to low thousands of rows per second)
- Less than 150 tables

Figure 4-21 gives an overview of the process used by InfoSphere CDC to consume data from DataStage.

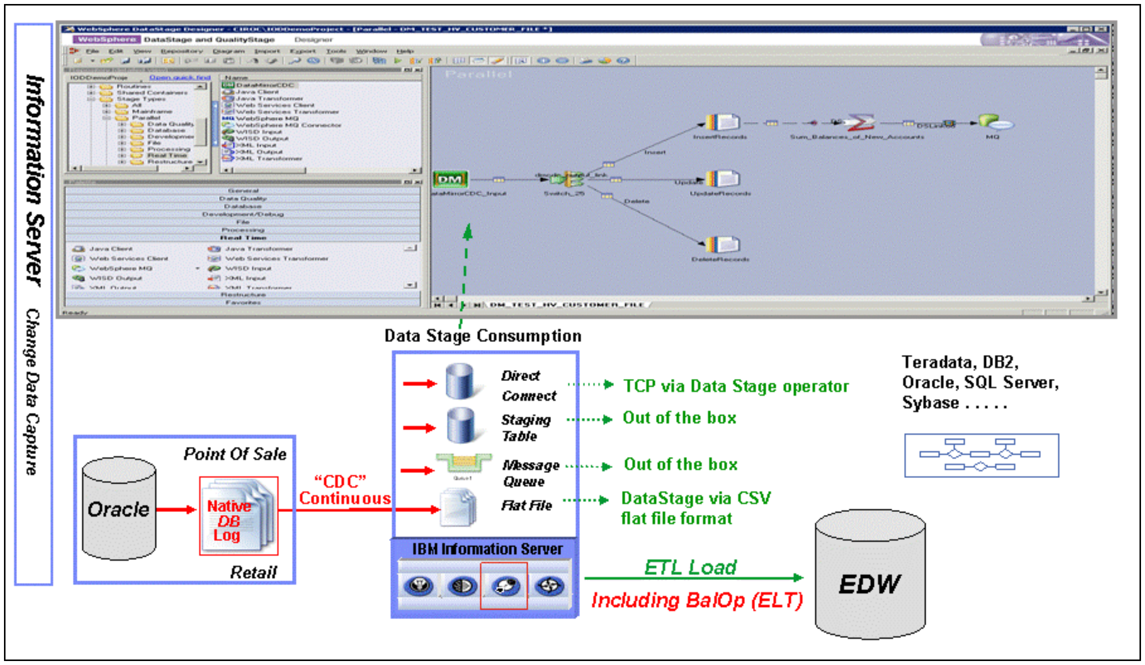


Figure 4-21   DataStage consumption by InfoSphere CDC

4.7.4 Web services

This section describes how InfoSphere CDC can be an part of an event-driven architecture and service-oriented architecture (SOA).

Event-driven companies, which acquire, deploy, and use real-time information, are most successful at sensing and responding to the changes or events that drive their businesses. This situation calls for a redefining of existing traditional IT architecture towards an event-driven architecture (EDA). The term *event-driven architecture* refers to any application that reacts intelligently to changes in conditions. Those changes could be a customer registration, a termination of services from a customer, a hardware device malfunction, or a power outage in one region of the country that causes a temporary shutdown or a sudden change in stock price. Depending on the size of the business, there are hundreds or thousands of notable events that occur every minute, every hour, and every day. By nature, some events might be positive, some negative, and some that might provide a business opportunity while others might pose a threat.

SOA is becoming the *de facto* standard for technical infrastructure in organizations across all industries. It is an approach to building software applications as collections of autonomous services that interact without regard to each other's platform, data structures, or internal algorithms. SOA equips organizations to enable improvements in their ability to react to changing business dynamics and in using their technical assets, helping them gain competitive advantage.

The term *second-generation SOA*, also referred to as Web 2.0, is the merger of SOA with EDA. The outcome is the creation of new systems that exceed the sum of their parts. EDA and SOA together enable companies to move more quickly towards becoming what are known as *real-time enterprises*. These companies compete by using the most current information to enable faster and more informed business decisions. They gain a competitive advantage by using such things as new composite applications, real-time dashboards, and key performance indicators (KPIs), and focus on enabling more automated business processes.

Event-driven architecture: An overview

Events are at the core of EDA. The way in which an event is recognized, enriched, and flowed across a business provides strategic competitive advantage. Companies must aggregate and integrate their data so they can more quickly react to those critical events as they occur.

Discrete data events

Discrete data events are events that are generated from a single system and represent granular activity in the business system. A shipping order, a change in the inventory level, or a new customer added to a customer relationship management system are discrete data events. They are analogous to a row in a table, but might be the aggregate result of a few elements from multiple rows in multiple tables. What categorizes a data event as being discrete is its linkage to a single business entity, such as a customer or a product.

An overview

An example of how InfoSphere CDC supports SOA environments is by packaging data transactions into XML documents and delivering them to applications in real time. This capability can also be used as an event detection solution, capturing mission-critical data transactions in real time and sending them to downstream applications to generate or automate business processes immediately.

Figure 4-22 shows the event detection process.

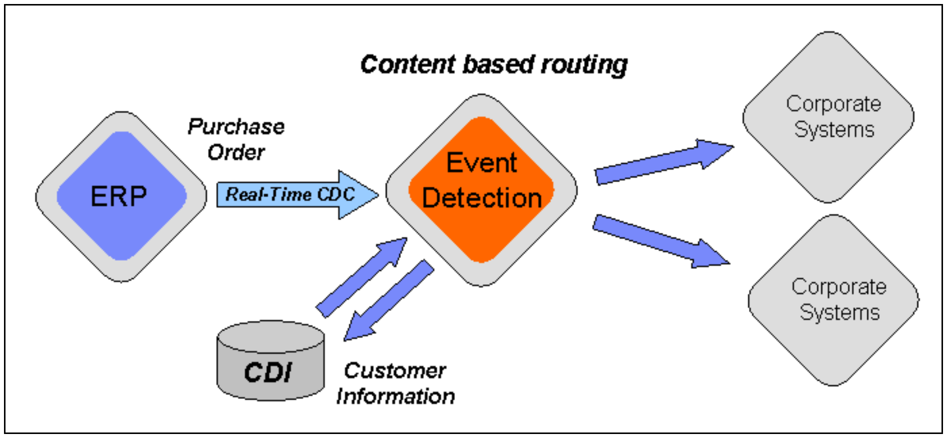


Figure 4-22 Event detection

Context-rich composite events have the following characteristics:

- Combines content associated with the event from other systems.
- Routes the data to different applications to initiate business processes based on the content of a message.

5

InfoSphere CDC features and functionality

This chapter describes features and functionality for IBM InfoSphere Change Data Capture (InfoSphere CDC). InfoSphere CDC administrators learn how the different features and functions can be used to customize the replication flow to meet the unique requirements of their environments. InfoSphere CDC operators gain knowledge about how the different features and functions affect the replication process and how that affects operations.

The topics in this chapter include the following:

- Transformational capabilities:

  This section describes each of the group types of transformations, including examples of usage. The intent is to make the reader aware of the diverse transformational possibilities available. InfoSphere CDC provides the ability to manipulate data during the replication flow. These transformations can be made against any or all columns included in a single operation. They provide functionality such as table joins, date and string manipulation, and IF / THEN / ELSE logic.

- Replication modes:

  This section describes each of the modes and provide details specific to each one. The intent is to make the reader aware of the different replication modes and when they would use each one. InfoSphere CDC provides three replication modes: two mirrored modes and one refresh mode. The mirrored modes capture data from the transactional logs of the database and then replicate that data downstream to the target. One mirrored mode runs continuously until stopped and the other mode allows scheduled stops. The refresh mode is an ETL style mode pulling data directly from the tables. It can be used to synchronize the data between the source and target tables. Different types of refreshes, including subset and differential refreshes, are covered.

- Filtering:

  The intent of this section is to make you aware of the filtering capabilities of InfoSphere CDC and how those capabilities can assist you in targeting specific subsets of data for replication. InfoSphere CDC provides functionality to filter data that is replicated at both the row and column level. The row level filtering is similar to the WHERE clause in a SELECT statement in that it can be used to include or exclude rows of information during replication. The column level filtering can be used to select which columns to include or exclude during replication. It can also be used to identify which columns are considered critical and trigger replication of the row or operation.

- Apply methods:

  This section describes the potential benefits provided by each apply method and when to use them in a replication environment. InfoSphere CDC can apply data to the target tables using six different modes, which are Standard, IBM LiveAudit™, Adaptive Apply, Summarization, Consolidation One to One, and Consolidation One to Many. Each apply method provides unique functionality that allows the user to create not only the standard replication environment but other specialized replication environments as well.

- Conflict detection and resolution:

  The last section describes when to use conflict detection and conflict resolution, how to set them up, and the usages for each of the resolution methods. The intent is to make the user aware of how InfoSphere CDC can assist in resolving conflicts between source and target data without compromising the replication flow.

InfoSphere CDC provides a Conflict Detection and Resolution functionality that can be integrated directly into the replication flow. This functionality can be used as part of a bidirectional replication or when the source and target data might not be consistent. It allows the user to identify specific columns to be analyzed for conflicts and five different methods to resolve the conflict.

5.1 Transformations

InfoSphere CDC allows businesses to share data between heterogeneous sources and targets. This heterogeneity often involves sources and targets that have different data structures, so it might require some of the following:

- Mapping between different column names
- Conversions between data types
- Changes in data sizing
- Default values
- Creating column values through derived expressions
- Accessing other tables for additional data
- Running user written code for processing outside of InfoSphere CDC

Mapping for differences in naming, type, size, or default values can typically be handled in a straight forward manner in InfoSphere CDC by using the graphical user interface provided by the Management Console (MC). Creating values through derivations or table joins, or running user written code in the form of User Exits, uses InfoSphere CDC transformational capabilities. Some of those capabilities are described in the following examples of what InfoSphere CDC provides:

- A diverse list of column functions that can be used along with standard mathematical operators to build derived expressions. Within Management Console, derived expressions can be manually scripted or built using the drag editor. These derived expressions can be saved and reused when mapping other columns.
- Access to additional data beyond that provided by the associated log entry. Both 'before' source values and 'current' target values, and journal control field values, are available to be mapped directly to target columns or used within derived expressions.
- The ability to include additional data by joining to other tables. This ability can be used to include master information from lookup tables, or it can be used to combine rows from multiple source tables into a single target row.

- Multiple points within the replication flow to perform user written user exits. These user exits can be used to provide additional and specialized functionality to meet the unique needs of the client. User exits are typically used when a single derived expression is insufficient to generate the wanted results.

Within InfoSphere CDC, transformations are performed as part of the replication flow. The transformations can be performed at the source (derived column) or at the target (derived expression). This flexibility allows the user to determine where to expend the additional processing impact required for the transformation.

5.1.1 Column functions

InfoSphere CDC provides a number of column functions that can be used within derived expressions when mapping target columns. These column functions help perform low-level data transformations during the flow of replication.

The column functions can be categorized into the following six functional groups:

- String functions can manipulate strings of text. They include functionality such as concatenation, substring, and left and right trimming. Here are examples of concatenation:
  - Denormalization of data by combining multiple columns of data into report or query friendly formats:

    ```
    %CONCAT(%RTRIM(FIRSTNAME), ' ', MI, ' ', %RTRIM(LASTNAME))
    ```
  - Retrieving subsets of data from within long character string columns, such as acquiring the person's first initial:

    ```
    %SUBSTRING(FIRSTNAME, 1,1) -
    ```
  - Minor cleaning of text data to remove trailing blanks or spaces from fixed-length source columns to variable length target columns:

    ```
    %RTRIM(FIRSTNAME)
    ```
- Date and time functions are used to manipulate date and time values. They can add a two-digit century specification to a date, or retrieve the current date and time. Here are some simple examples:
  - Acquiring current date:

    ```
    %CURDATE("*GMT")
    ```
  - Creating a time stamp field by combining date and time fields together:

    ```
    %TODATE(DATE, "*YMD", TIME)
    ```

- Conversion functions are used to manipulate data types. They can move between character, numeric, and date values. Here is an example:
  - DIVISION (VARCHAR 10) column contains 'US2003'. The data needs to be manipulated to add 1000 to the number value in the last four characters, resulting in 'US3003'. The transformation would be:

    ```
    %CONCAT(%SUBSTRING(DIVISION,1,2),
    %TOCHAR((%TONUMBER(%SUBSTRING(DIVISION,3,3)) + 1000),4))
    ```
- Conditional and variable functions provide flexible logic. Use the %IF column function to evaluate an expression and return different results. Use the %VAR function to declare a new variable, assign a value to it, or retrieve the value of an existing variable. Here is an example:
  - Load the SALARY_LEVEL column with the values 'LOW', 'MID', or 'HIGH' based upon the value in the SALARY column. The transformation would be:

    ```
    %IF(SALARY < 30000, 'LOW', %IF(SALARY < 60000, 'MID', 'HIGH'))
    ```
- Data functions are used to provide additional data beyond that provided by the actual log entry. They provide information such as before or current values. They can join to other tables to acquire additional data values. Here is an example:
  - A client had problems with hackers changing grades on a school database. They resolved the issue by replicating the before image of a grade back into the grade column if the column was changed by a user other than a special admin user ID. Effectively, as fast as the hackers changed the grades, the grades were restored to their previous value. The following transformation was used:

    ```
    %BEFORE(GRADE)
    ```
- User exit functions are used to call user written code from within a derived expression. Depending upon the InfoSphere CDC platform installed, these functions could be stored procedures or user functions, such as Java code. The following transformation could be used:

  ```
  %STPROC('CUSTNAME', CUSTOMER_ID)
  ```

  Two examples of its use could be:
  - Passing information to a secondary table not targeted by InfoSphere CDC but based upon an event encountered during InfoSphere CDC replication, such as suspected erroneous data being sent to a holding table.

  - Triggering a database event based upon input received during a InfoSphere CDC replication. In this example, a client wanted to perform a backup of certain tables immediately following the complete replication of data from a nightly batch job. The customer updated a record in a status table as the last operation of the batch job. InfoSphere CDC replicated the update for the status table, which resulted in the user exit being initiated. The user exit performed the backup and various other functions.

5.1.2 Journal control fields

Journal control fields provide information about the log entry on the source system. When a change is made on the source system, the database records the change in a log entry that contains the changed data and what type of change was made, who made the change, and when the change was made. When a relevant log entry triggers a replication event to the target system, InfoSphere CDC replicates the changed data along with the extra log entry information available through journal control fields.

InfoSphere CDC provides many journal control fields that contain log entries from your source systems. The journal control fields can be:

- Mapped directly to columns on the target system. Here are examples:
  - Map `&TIMSTAMP` and `&USER` to target columns to show when the original transaction took place and who made it.
  - Map `&CCID` (commit cycle ID) to target columns to identify transactions that belong together in a single unit of work or commit group.
- Used in row level filtering to identify log entries to include or omit:

  ```
  &JOB = 'PURGE001'
  ```
- Used in derived expressions:

  ```
  %IF(%SYSTEM = 'USHDQR', 'Atlanta', 'Toronto')
  ```

5.1.3 Joining

InfoSphere CDC provides a column function (%GETCOL) that allows you to join tables to retrieve the value of a column for a specific row. This function allows you to retrieve and use information not found in the actual log entry.

Joins: Joins performed at the source occur at the time InfoSphere CDC reads the transaction log. The data retrieved by the join matches the current values in the join table and might be the same values as when the original log entry occurred.

You can use the `%GETCOL` function in expressions to perform the following operations:

- Obtain columns from one or more keyed secondary tables and join them with an existing primary table before sending the data to the target. The primary table refers to the source table being replicated. The secondary tables refer to tables referenced in the `%GETCOL` function.
- Specify how keys of the secondary tables are populated, to allow InfoSphere CDC to perform the necessary secondary reads.
- Use columns from secondary tables that were retrieved previously to populate keys for subsequent reads (the population of key column values is not restricted to primary columns only).
- Specify the order that table reads are performed.
- Condition the table reads that are performed.
- Read tables external to InfoSphere CDC to perform dynamic translations on the target.

There are two syntax formats for `%GETCOL`:

- The long syntax format is used to read a table and return the value of the column specified, based on the key column values that are identified. If more than one row satisfies the key requirements specified, then this function returns the first row only. If the read is unsuccessful, then this function returns the default value specified.
- The short syntax format is used to return the value of the specified column from a row retrieved by a previous `%GETCOL` function invocation. The short syntax lets you retrieve more than one column from a table (that was read previously using the `%GETCOL` function), without reading the table again. The previous `%GETCOL` function invocation must be for the same log entry during continuous mirroring or the same row during refresh. Here are some examples:
  - A client wants to denormalize their target table by joining information from various tables. They use their detail table as the main table to be replicated. They then perform various joins to gather information not included in the detail record, such as customer name, address, and telephone number.

   - A client wants to keep records on a target table synchronized with changes made to master tables on a source database. In this case, the main table is replicated by InfoSphere CDC using standard apply and joins to gather the additional data from the master tables. Anytime a change to the main table is replicated, the joins are performed to gather the additional data from the master tables. The master tables are replicated to the target table, but use the Consolidation One to Many apply method. As changes to the master table are replicated, InfoSphere CDC updates as many records as necessary in the target table to correspond to the new master table values.

5.1.4 User exits for customizations

A user exit is a processing point where a user written program can be started. The program itself is called a *user exit*, and can return a result. This mechanism allows defining operations that need to be performed at the user exit point. The logic defined in the user written program determines the actions that occur.

User exit examples: User exit examples are further described in 9.7, "User exits" on page 369.

User exits can be used to:

- Define data transformations for a specific working environment
- Handle business requirements not handled by the predefined column functions provided by InfoSphere CDC
- Perform operations or provide notification in response to specific product conditions

A user exit lets you define a subroutine that InfoSphere CDC can start when a predefined event occurs. The following two types of user exits are available:

- Table-level user exits run a set of actions before or after a database event occurs on a specified table.
- Subscription-level user exits run a set of actions before or after a commit event occurs on a specified subscription.

Subscription-level user exits can work in tandem with table-level user exits to track which table-level user exits were started during a transaction. The subscription-level user exit could then use that information to apply actions based on the tables in the transaction.

5.1.5 Considerations for using transformational functionality

Some things to consider when using InfoSphere CDC transformational functionality are:

- Depending on the InfoSphere CDC engine you have installed, some column functions might not be available.
- Column function syntax may differ between platforms. For example, the syntax for `%GETCOL` is different for the IBM i platform than the one used by DB2 on Linux, UNIX, or Windows.
- Names for column functions are not case-sensitive.
- For some column functions, large object (LOB) columns cannot be specified as a function parameter.
- Character literals can be specified in their internal numeric representation, as parameters for column functions. To accomplish this task, use the double-angled bracket notation (<< >>). This notation allows both printable and non-printable characters to be used. Specifying character values as decimal integers can represent either American Standard Code for Information Interchange (ASCII) or Extended Binary Coded Decimal Interchange Code (EBCDIC) characters.

5.2 Replication modes

InfoSphere CDC provides three modes of replication:

- Refresh
- Continuous
- Scheduled End (Net Change)

These modes provide flexibility about how and when to replicate. Refresh replication provides bulk copy functionality. Continuous and Scheduled End replication provide log based and mirroring functionality.

5.2.1 Refresh

The InfoSphere CDC refresh operation is the replication mode used to capture a complete copy of the data in the source table and transfer that data to the target table. Each refresh applies all features of replication (such as row and column filtering, column mappings, and so on) during the transfer.

Because refresh is an operation applied to the source table, the manner in which refreshed data is applied to the target table depends on the type of table apply method (such as standard and audit). Before the source table data is transferred, the target table may be cleared and indexes dropped before the refresh, and rebuilt following the refresh. If the target table is cleared, the data in the source and target tables is synchronized after completion of the refresh.

Typically, a refresh is performed only once to initialize target tables by transferring all the data in the source table or if configuration changes are made that require a refresh. Mirroring is then used to replicate changes as they occur. A refresh can be used to provide snapshot images of tables for use in reporting or for tables that are changed by batch loads only. Refreshes can be scheduled to run at specific intervals or times.

Some factors that determine the amount of time required to complete a refresh are:

- The size of the tables to be replicated and the number of columns and rows selected for replication
- The width of the rows in number of columns
- The existence of Large Object (LOB) data types
- The amount of processing involved with replication (such as derived expressions and translations)
- The speed and bandwidth of communications lines
- The number of indexes on the target table

Canceling a refresh can be done at any time, but should only be done to interrupt a refresh. A controlled shutdown ends the process after the current table finishes its refresh activities. A controlled shutdown is the suggested method to end a refresh process. An immediate shutdown ends refresh processes without completing the refresh. This action might require the refresh to be redone to guarantee data consistency between the source and target tables.

Tables can become out of synchronization for various reasons, including the following issues:

- Parked tables: If a table is parked from replication to make changes (such as updating the definition of a source table), and the changes on the source table are no longer being replicated, the target table may become inconsistent with the source table.

- Configuration changes: A refresh might be necessary when a set of subscriptions is promoted from a test environment to a production environment. The promotion operation may add new transformations or other table mapping changes that require the source and target tables to be refreshed to prepare for mirroring.
- Maintenance operations: For large bulk SQL operations performed during maintenance windows on the source table that affect a majority of the rows, it might be faster to resynchronize using refresh. Refresh might be faster than mirroring to replicate millions of changes due to the ability to bulk load rows into the target database.

InfoSphere CDC refreshes all of the flagged tables within a single subscription as one sequential operation that runs to completion. Each table is refreshed individually one at a time until all flagged tables have finished refreshing. Refresh is an operation that applies to a single subscription, so while one subscription is refreshing, other subscriptions are not affected. They may continue mirroring data for different tables or refreshing tables as required. To perform a parallel refresh, multiple subscriptions can be used.

InfoSphere CDC offers two primary types of refresh operations: Standard Refresh and Differential Refresh.

- A standard refresh results in a complete copy of the data in a source table being sent to the target table. This type of refresh is typically performed to bring the entire source and target tables into synchronization.
- A differential refresh updates the target table by applying only the differences between it and the source table. This type of refresh is typically performed when the target table is already synchronized with the source table. With a differential refresh, you can choose to perform a refresh only, refresh and log differences, or log differences only.

The order in which data is retrieved from the database during a refresh depends on the type of refresh performed. During a standard refresh, no ORDER BY sort is used; the database determines the order in which the data is returned. During a differential refresh, InfoSphere CDC queries the database using an ORDER BY sort on the table keys chosen in the table mapping to sort the source and target tables and determine their differences.

When a refresh is performed with multiple tables, the order in which each individual table is refreshed is based on the group order, as set in the Refresh Order option. If no Refresh Order is set, then tables are refreshed in alphabetical order. After a refresh has successfully completed, the subscription can be restarted for mirroring. InfoSphere CDC then processes the backlog of changes.

If Referential Integrity (RI) is in effect on the target tables being replicated to, it is important to set the refresh order. If the tables are refreshed in the incorrect order, database failures can result when InfoSphere CDC tries to rebuild the foreign key constraints. It is also important to make sure all tables that make up the RI are part of the same InfoSphere CDC subscription.

Here are examples of RI usage:

- Perform initial synchronization of the source and target tables.
- Restore synchronization following the loss of data integrity on the target table.
- Identify differences between the source and target table using differential refresh.

Subset refresh

Both a standard refresh and a differential refresh can be further refined through the use of a WHERE clause to only include rows within a specified range. This situation is useful for tables where only the most recent data requires a refresh. This feature requires one of the following conditions be met:

- The table capture point has been set, either explicitly or through the table having been mirrored.
- The scraping point for the subscription has been set (using the **`SETLOGPOS`** command or **`dmsetbookmark`** command where applicable).

Here are examples of when to use subset refreshes:

- When the source table's size precludes refreshing the entire table in one process
- When only the most recent data needs to be refreshed
- When a particular subset of the data needs to be refreshed, such as a particular customer or part

Differential refresh

A differential refresh updates the target table by applying only the differences between it and the source table. Instead of the target table being cleared at the start of the refresh and repopulated with data, the differential refresh compares each row in the target table with each row in the source table to identify missing, changed, or additional rows. The primary advantage of the differential refresh is that the target table stays online during the refresh operation.

There are three possible methods for the differential refresh:

- Refresh Only: Performs a differential refresh by changing any target rows that differ from the source rows.
- Refresh and Log Differences: Performs a differential refresh and also creates a log table in the target replication engine metadata to track all changes during the refresh. The log table is identical to the target table, with the addition of a column to indicate the actions taken during the refresh, such as inserting a row, deleting a row, or updating a row. For an update, both the source and target row images are logged. This log table is created in the same database and table space as the TS_CONFAUD table (or DMMD_DMCONFAUD on z/OS), with the same owner as the metadata. The name of the log table is created by combining the subscription name, the target table name, and the refresh start date and time.
- Only Log Differences: Creates and populates a log table in the target replication engine metadata to identify all differences between the source and target tables. The target table is not updated. This method allows you to evaluate what the differences are between the target and the source. If you then decide to refresh the table, you can go back to the subscription and select Refresh Only to update the target table or update the target table manually based on the contents of the log table.

Performing a differential refresh has some requirements and restrictions:

- Differential refresh is only available for tables that use Standard replication.
- The collation sequence of the source and target tables must be identical.
- Derived columns on the source table are not supported.
- Any target columns that are mapped to derived expressions, constants, or journal control fields are ignored.
- The key columns of the target table must be mapped directly to columns on the source table.

Differential refresh sends all the data from the source to the target to perform the compare. Clients might want to consider combining the differential refresh with subset refresh to reduce the impact and timing of a full differential refresh. The differential refresh can also be performed in its own subscription, minimizing the impact to existing subscriptions.

5.2.2 Continuous mirroring

Continuous mirroring replicates changes to the target on a continuous basis. It is used when business requirements dictate replication to be running continuously without a clearly defined reason to end replication now.

Continuous mirroring can be ended with the following options:

- Normal: Completes in-progress work and then ends replication. This option might take some time if there are in-progress transactions. The subscription ends in a confirmed and stable state.
- Immediate: Stops all in-progress work and then ends replication. Starting replication after using the Immediate option can be slower than using the Normal option.
- Abort: Stops all in-progress work and then ends replication rapidly. This option is the fastest way to end replication. Starting replication after using the Abort option can be much slower than using the Normal option.
- Scheduled End: Processes all committed changes to the indicated point in the database log and then ends replication normally. You can use the current date and time (Now), a specified date and time, or a specific log position.

5.2.3 Scheduled end (net change)

Scheduled end (net change) mirroring replicates changes to the target up to a user specified point in the source database log and then ends replication. It is used when business requirements require data replication periodically and a clearly defined endpoint for the state of the target database.

Scheduled end (net change) mirroring allows replication to end based upon the following points in your source database log:

- Current time (or now): This option can be used for subscriptions that are latent. The subscriptions replicate from their bookmark position up to the log position at the time the subscription is started. This option is useful when working with subscriptions that you want to guarantee have finished processing the logs before ending.
- User specified date and time: This option guarantees that the subscription processes all transactions up to a certain time of the day. Typically this option is used when the user wants to maintain a set amount of latency between the source and target. An example is a reporting instance that is kept 24 hours behind the production database.
- User specified log position: This option allows the user to replicate data from one log position to another. An example would be where the user needs to recover transactions up to a certain log position.

These user specified end points ensure that the target database is in a known state when replication ends.

Ending replication allows preparation for transitional activities in business environments and allows moving to the next step in the business process.

5.3 Filtering

InfoSphere CDC provides filtering at both the row and column level. Row level filtering can be used to include or omit rows in the replication flow. Critical columns can be used to filter rows based upon changes to specific columns. Column level filtering can be used to limit which columns are replicated.

5.3.1 Row level

Row level filtering supports both standard InfoSphere CDC and SQL SELECT WHERE expressions. Standard InfoSphere CDC expressions include column functions, arithmetic and Boolean operators, column names, and journal control fields. All row selection expressions must return a Boolean result.

The following are valid row selection expressions:

- `1 = 1`

  This expression always returns a true result, and so either all or no rows are replicated. The Boolean constants `TRUE` and `FALSE` are not supported. An example is `NAME = 'Monica Flanagan'`.

- `(SALES < 10000) OR (SALES > 90000)`

  Use parentheses to group Boolean expressions. Short forms, such as `SALES < 10000 OR > 90000`, are not allowed. In this case, you must specify `SALES` twice.

- `NOT((AIRPORT = 'ATL') OR (AIRPORT = 'CLT'))`

  Short forms, such as `NOT(AIRPORT = 'ATL' OR 'CLT')`, are not allowed.

- `%RTRIM(DEPT) = 'IT'`

  Column manipulation functions can be used in row selection expressions, and they can be applied to any operand in a Boolean expression.

- `PRINCIPAL *(1 + INTRATE) > 20000`

  Basic numeric operators, such as multiplication (*) and addition (+), can be included in row selection expressions.

- `%IF(COUNTRY = 'US', PRICE, PRICE * 1.2) > 50`

  The `%IF` column function does not allow you to return a Boolean result, and so the result that is returned by the function must be compared with another value in the row selection expression.

- `STATE IN ('CA', 'AZ', 'AL', 'GA', 'CT')`

  The IN operator is valid in **`SQL SELECT WHERE`** clauses. This example can be used when working with an InfoSphere CDC product that supports this type of expression.

- `&JOB = 'QTRPURGE'`

  Journal control fields can be used to identify transaction-specific information. In this scenario, `&JOB` can be used to identify a specific job and then include or omit records pertaining to the job.

InfoSphere CDC provides functionality to declare critical columns. You can use critical columns to control the updates replicated and reduce the workload on the network and target database. When you select a column as critical, InfoSphere CDC only replicates update operations when any critical column value has changed.

Here is an example row level filtering:

Replication is wanted only when the account balance is updated in the customer account table. In this scenario, the CUST_ACCT_BAL column is selected as a critical column. InfoSphere CDC only replicates this row when there are updates made to the CUST_ACCT_BAL column.

Period end processing reads all of the records in a very large table and updates a time stamp field. If the time stamp field is not turned off as a critical column, InfoSphere CDC replicates the entire table, row by row. By making the time stamp field non-critical, that situation is avoided.

5.3.2 Column level

By default, InfoSphere CDC replicates all mapped and unmapped source columns to the target table, because unmapped source columns could be used on the target in derived expressions or used by a user exit. If there is a source column that should be excluded from replication, it can be cleared on the Filter Tab in Management Console. If you configure column level filtering, be aware that a change in a non-replicated column causes the change to be dismissed by CDC, saving bandwidth.

Examples of when columns might be excluded are:

- The target table contains only 20 columns of the 150 columns in the source table. By selecting only the 20 columns required, the amount of data replicated to the target is reduced, saving network bandwidth and processing impact.

- The source table contains confidential information such as social security numbers and salary. The target table should not have access to this information, so these columns are excluded from replication.
- Changes might occur only to columns excluded by the column level filter. In this case, InfoSphere CDC does not replicate the change, saving bandwidth.

5.4 Apply methods

InfoSphere CDC provides six apply methods that can determine how replicated data is applied to the target table. These methods are Standard, LiveAudit, Adaptive Apply, Summarization, Consolidation One to One, and Consolidation One to Many.

5.4.1 Standard

Under Standard replication, InfoSphere CDC replicates the effect of the source operation to the target table. A row insert operation on the source table would result in an inserted row on the target table, and so on.

Management Console provides two mechanisms for mapping using Standard replication.

- One-to-One Mappings: Map multiple source tables to multiple target tables where the source and target tables share a table structure and similar table names.
- One table mapping (Standard): Map one source table to one target table using standard replication. These tables can have different table structures and names.

Standard replication can be used for scenarios such as the following:

- Synchronize data between two databases to allow two different environments to work on the same data. This scenario can be used during parallel testing of applications.
- Provide a subset of source data to a reporting instance. This scenario allows reports to be run against the reporting instance and frees up resources on the source system.
- Stream data from source systems to a dynamic ODS. This scenario allows nightly ETL jobs to be performed at the ODS system and frees up the source systems to run 24x7.

- Consolidate data from multiple source systems into a single target system. This scenario provides a single view of the data and reduces the number of systems required to be accessed for reporting.
- Distribute data from one source system to multiple target systems. This scenario could allow subsets of data to be distributed to target systems based upon filtering criteria.

5.4.2 LiveAudit

LiveAudit replication provides an audit trail of source table operations. When data is replicated using LiveAudit, the target tables contain rows that track insert, update, delete, and clear (truncate) operations applied to the mapped source table.

LiveAudit maintains an audit table containing a row for each source operation. Additional information is included in the target row, such as operation type (insert / update / delete), time stamp for the transaction, and user that created the transaction. Journal control fields can be mapped to additional target columns to meet specific business needs.

With LiveAudit, target tables have the potential to grow to be large. Sufficient disk space must be allocated or regular maintenance must be performed to accommodate large subscription tables that ware being used for auditing.

Management Console provides two mechanisms for mapping using LiveAudit replication.

- LiveAudit Mappings: Map multiple source tables to multiple target tables using LiveAudit replication.
- One table mapping (LiveAudit): Map one source table to one target table using LiveAudit replications. This mechanism allows greater granularity of the options available for LiveAudit mappings.

Examples of where LiveAudit replication can be used are:

- Regulatory requirements to keep audit trails of all operations against the data.
- Change tracking of sensitive information.

- Providing operations processed against source data to downstream applications. For example, an ETL application can use the time stamp values to determine which operations to process to ensure that target data is consistent with a specific point in time. The operation type is used to determine what type of operation needs to be processed.
- Providing test data operations. An application can be used to process the audit data and generate a "replay" of the operations against a test environment.

5.4.3 Adaptive Apply

Adaptive Apply replication provides flexibility when replicating operations to target tables that may not be synchronized with the source tables. With Standard replication, the operations are database dependant upon the absence (inserts) or existence (updates and deletes) of the target rows. An insert operation replicated from the source results in a database error if a matching target row is found (duplicate key error). With Standard replication, this action results in the subscription ending.

Adaptive Apply replication provides upsert functionality. For a scenario where a matching target row is found, the insert operation converts into an update operation. Replication does not encounter an error and continues. When an update operation cannot find a matching target row, the operation is converted to an insert operation. When a delete operation cannot find a matching target row, the operation is ignored.

Examples of where Adaptive Apply replication can be used are:

- When external applications modify target tables independent of the source tables.
- When you are restoring the contents of a subscription table from recorded log entries. By setting the log position to a specific entry or point in time, Adaptive Apply can be used to populate an empty target table so that it contains the latest data.

5.4.4 Summarization

Summarization replication allows you to maintain numerical totals in selected target table columns. Under summarization, the target table is a repository of numerical data that has been accumulated or deducted in response to source row level operations transferred by refresh or mirroring activity.

Accumulation ensures that numeric changes applied to the target column are directly proportional to changes applied to the corresponding source columns. Deduction ensures that numeric changes applied to the target columns are inversely proportional to changes applied to mapped source columns.

Examples of where Summarization replication can be used are:

- Summarization replication could be used to simplify accounting, statistical, and other business operations that require intensive addition and subtraction of numeric data. Instead of applying resource-intensive programs to determine totals from an accumulated stack of transactions, aggregates can be maintained in a target table as source row level operations are replicated. As a result, applications designed to generate more complex calculations for reports and other purposes can work directly with the currently maintained totals.
- Summarization replications can also be used to create a soft delete environment. See 5.4.6, “Soft deletes” on page 95 for more information.

5.4.5 Row consolidation

InfoSphere CDC provides two types of row consolidation: Consolidation One to One and Consolidation One to Many. Row consolidation allows flexibility with row ownership of the target table.

Using Standard replication, data warehousing can be implemented by configuring a target table to receive rows from multiple source tables. Each row in the target table can only be inserted, updated, or deleted by one of the source tables contributing data to the warehouse. Each row is effectively “owned” by a source table.

Row ownership by a source table can be too restrictive in some environments. When multiple source tables need to be merged to create a single target table row, row ownership does not work. Consolidation One to One replication allows this merger without the row ownership issues.

Consolidation One to One allows merging different information about a common entity, such as a person, a customer, or a product part, into a single row on the target table. It is intended for environments where information about the entity is scattered across different tables, databases, or servers, but must be centralized to facilitate report generation, data management, data security, and other business objectives and activities.

Consolidation One to Many allows data changes to a lookup table to be applied to all target rows affected by the change. A single change on the source lookup table might require hundreds, thousands, or more rows on the target table to be changed. Consolidation One to Many replication gives you the ability to keep the target rows current with changes not only against the primary source table, but also changes made to secondary tables used to provide supplemental information.

Consolidation One to Many reacts to operations in the following manner:

- Inserts: Inserts to the lookup table do not generate operations on the target table.
- Updates: Updates to the lookup table generate updates to rows matching the consolidation key value.
- Deletes: Deletes to the lookup table do not generate operations on the target table.

5.4.6 Soft deletes

InfoSphere CDC provides support for soft deletes. A soft delete is an operation where the target row is not deleted following a delete operation on the source row. Instead, the target row is inserted if it does not exist, or updated if it does. A flag field or entry type field is also updated to indicate that this row was deleted from the source.

All Databases - Soft deletes can be configured for tables using the Summarization apply type by completing the following steps:

1. Choose **Summarization** as the apply type.
2. Add a flag column on the target table (varchar(2)).
3. Do not select any columns for summarization.
4. Map the journal control field `&ENTTYPE` to the flag column.

Operations on the source row result in the following actions on the target row:

- Insert: InfoSphere CDC attempts to update the row. If the row does not exist, InfoSphere CDC inserts it, and PT is placed in the flag column.
- Update: InfoSphere CDC attempts to update the row. If the row does not exist, InfoSphere CDC inserts it, and UP is placed in the flag column.
- Delete: InfoSphere CDC attempts to update the row. If the row does not exist, InfoSphere CDC inserts it, and DL is placed in the flag column.
- Insert using deleted row key values: The delete flagged target row is overwritten with the values from the insert and PT is placed in the flag column.

5.4.7 Custom apply methods (user exits)

InfoSphere CDC can use user written and maintained user exits to control the target apply process.

A few examples of where clients have chosen to use user exits to perform applies are:

- Web Service: A user exit can be used to interface InfoSphere CDC directly with a web service. The user exit performs the apply process to the web service while InfoSphere CDC manages scraping the transactions from the source transaction logs and performs low-level transformations to the data.
- Full Unit of Work to message queue: By default, InfoSphere CDC includes one operation per message when targeting a JMS message queue. Some customers want to see the entire Unit of Work within a single message queue. This action can be facilitated through a user exit when using the InfoSphere CDC Event Server engine.
- Soft Delete: To use a standard apply process with soft deletes, a user exit is required. The user exit manages the processing required to not delete the row and update a flag field. See Example 9-83 on page 383 for more details.

5.4.8 Flat files

InfoSphere CDC can use flat files through the InfoSphere CDC DataStage engine. With this engine, two options for connecting to InfoSphere DataStage are available: flat file and direct connect.

With the flat file method, InfoSphere CDC produces a file containing information about one or more records and database operations. The flat file can be configured with the Single Record option so that the before and after images are included together in a single line followed by a delimiter, or with the Multiple Record option, where each record may occupy two lines (before and after images of an update).

InfoSphere CDC flat files can be used to quickly and efficiently capture information from source table transactions and pass the information along to any ETL engine that can read a flat file. This action allows InfoSphere CDC to complement many of the ETL products available on the market by:

- Providing transactions capture throughout the day as opposed to at set times
- Providing low impact to the source by reading the transaction logs and not the tables directly

- Removing the need for batch ETL processing on the source that requires tables locked or users off the system
- Providing transactions for just the data that changed as opposed to having to read through the entire table

Configurable options for flat files are:

- Location of flat files
- Threshold limits in number of operations or time, which ever comes first, for hardening the flat file

InfoSphere CDC continues to write to a temporary data file until one of the threshold limits are met, or in the case of refreshes, after the entire table is contained in the flat file. After the threshold is met or the refresh is completed, InfoSphere CDC hardens the temporary data file and make it available for consumption. It adds a time stamp to the file name.

InfoSphere CDC creates a `<`*`Table_Name`*`>.stopped` status file when the refresh operation or mirroring is ended. When InfoSphere CDC restarts, the bookmark is used to determine where the scrape process should begin in the transaction log, so no data is lost.

Important: If the refresh or mirroring operation is terminated using the **`dmterminate`** command, the temporary data file may not be hardened and no `<`*`Table_Name`*`>.stopped` status files may be generated for the tables in the subscription. After the replication process is restarted, the subscription uses the last-saved bookmark to reposition the log reader and start generating new data files. The temporary file is not cleaned up. To ensure that the temporary data files are hardened, and the `<`*`Table_Name`*`>.stopped` status files are created, use a Normal or Scheduled End shutdown in Management Console, or a **`dmshutdown`** command with the appropriate flags for the severity level.

5.4.9 DataStage direct connect

InfoSphere CDC provides integration to connect directly with InfoSphere DataStage. This action allows:

- Integrated control of both the InfoSphere CDC and InfoSphere DataStage products
- Template jobs with metadata information to be created and passed from InfoSphere CDC to InfoSphere DataStage
- Commit status on target through bookmarks

- The ability to synchronize the restart point for an InfoSphere CDC subscription to guarantee no loss of data
- Ability to autostart and securely connect to InfoSphere DataStage jobs

The process for the Direct Connect connection method is similar to the Flat File connection method. The size and time limits set in the InfoSphere DataStage Properties dialog box determine when data is committed to the target database.

With the Direct Connect connection method, you can enable the autostart feature to run in active mode, which allows InfoSphere DataStage to start a job when appropriate and begin to stream data to InfoSphere DataStage. Running with autostart enabled requires both InfoSphere CDC and InfoSphere DataStage to be installed on the same server. If autostart is not enabled, you must start the DataStage jobs using a different mechanism before the InfoSphere CDC process can start replicating the changes. You can start the InfoSphere CDC subscription before starting the DataStage job; the replication waits for the DataStage job to become active.

Examples of when to use InfoSphere CDC with Direct Connect to InfoSphere DataStage are:

- As part of the Change Data Delivery (CDD) configuration for using InfoSphere CDC to replicate multiple sources to InfoSphere DataStage (typical with retail customers with multiple stores)
- When complex transformations are required to the data through InfoSphere DataStage
- To optimize the data capture process for InfoSphere DataStage and minimize the impact to the source database

5.4.10 JMS message queues

InfoSphere CDC Event Server is the target engine used to create XML documents that are placed into JMS message queues. Using Event Server with an InfoSphere CDC source engine, transactions can be captured and delivered to a JMS message queue. The messages are then available to be processed by any application, such as WebSphere MQ, that can reading a JMS message queue. The requirement for the ESB application is that it is able to read and process a JMS message queue.

Options for InfoSphere CDC Event Server are:

- Map source columns to XML elements and attributes.
- Configure the header information for the XML document.
- Load to staging table then convert to XML.
- Perform low-level transformations to the data.

- Import and export mapping projects and XML schemas.
- Build XPath expressions.
- Query columns from other tables.
- Set runtime options.

There are two methods for mapping to a JMS message queue:

- Message Destination Mappings: Uses the Map Tables wizard to map a source table to a JMS message destination. InfoSphere CDC Event Server receives the row-level operation and transforms this row into XML. The XML message is sent to a JMS application supported by InfoSphere CDC Event Server.
- One table mapping of any type:
  - Standard: Allows source tables to be mapped to a target table within a staging database before being converted to an XML document. This type allows transformations to be done to the data through a user exit on the target and frees up resources on the source.
  - Adaptive Apply: Similar to Standard, but allows for replication to an empty target table. Adaptive apply functionality converts inserts to updates or updates to inserts as needed.

5.5 Conflict detection and resolution

Conflict detection and resolution let you detect, log, and act on inconsistent data on the target. This function ensures that the replication environment handles data conflicts automatically and in accordance with business rules. Set conflict detection so that InfoSphere CDC can detect and resolve conflicts as they occur. As conflicts are detected and resolved, InfoSphere CDC logs them in a conflict resolutions audit table.

During replication, InfoSphere CDC detects conflicts when you:

- Insert a row and the row's key exists in the target table. This action violates the unique key constraint.
- Update a row and the row's key does not exist in the target table.
- Update a row and the contents of the rows in the source table and target table, before the update, do not match.
- Delete a row and the row's key does not exist in the target table.
- Delete a row and the contents of the rows in the source table and target table, before the delete, do not match.

InfoSphere CDC does not detect conflicts in target columns that are:

- Populated with expressions using the `%BEFORE`, `%CURR`, `%GETCOL`, `%STPROC`, and `%USER` column functions
- Populated with journal control fields
- Not populated by a value
- Contain a Large Object (LOB) data type

Conflict detection and resolution can be applied to individual columns for tables configured for Standard replication. There are five possible conflicts and detection resolutions:

- Source Wins: The source row overwrites the target row. If the target row does not exist, the source row is inserted. This resolution helps maintain consistency between the source and target tables.
- Target Wins: The target row remains intact and the source row information is discarded.
- Largest Value Wins: The largest value for a column is used to determine whether to use the source row information or the target row information. For example, if the source row contains a time stamp that is newer than the target row's value, the source row overwrites the target row. Null values are treated as the smallest value possible. Therefore, if the target row does not exist, the source row is inserted. If the source and target values are the same, InfoSphere CDC resolves the conflict using the Target Wins method where the target row remains as is.
- Smallest Value Wins: The smallest value for a column is used to determine whether to use the source row information for the target row information. If the target row does not exist, InfoSphere CDC uses Null as the smallest value and the row is not inserted.
- User Exit: When InfoSphere CDC resolves conflicts with a user exit program, it applies the image returned by the user exit program to the target table. A user exit program can be configured to specify the row InfoSphere CDC uses to resolve the conflict on the target table.

Here are examples of when to use conflict detection and resolution:

- Bidirectional replication: Changes can be made to the same data on both sides of the replication. However, this action could result in conflicts.
- Expected inconsistencies in the data: The target data is not synchronized with the source data, and business rules need to be applied to record and address the differences.

- Refreshes performed outside of InfoSphere CDC: When an application other than InfoSphere CDC is used to refresh data, it is common to start InfoSphere CDC at a log position before the time of the refresh. This action allows InfoSphere CDC to pick up incomplete transactions not reflected in the data refreshed. Conflict detection and resolution can be turned on for the subscription to handle the resulting duplicate transactions. Once past the refresh point, Conflict Detection and Resolution can be turned off to optimize throughput.

6

Understanding the architecture

This chapter provides an introduction to the general architecture of IBM InfoSphere Change Data Capture (InfoSphere CDC). This chapter introduces some key InfoSphere CDC terms, or specify the usage of terms within InfoSphere CDC where they have a specific meaning within the context of InfoSphere CDC.

An architecture is a set of defined terms and rules that are used as instructions to build products. In computer science, architecture describes the organizational structure of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact though interfaces include classes, components, and subsystems.

The major components of InfoSphere CDC are described here to provide an orientation to the main parts and pieces of the product and provide an understanding of how they interact to form the whole of the InfoSphere CDC replication system.

In addition to this high-level view, this chapter describes some more fundamental architectural concepts within InfoSphere CDC and go into some detail regarding the implementation of some of the key components, to provide you with a more in-depth understanding.

However, this chapter does not describe specific features, functionality, and usages of InfoSphere CDC in this chapter; those descriptions are the purview of the other chapters in this book. And there is a great deal of information that is also available in the product documentation. This chapter also does not describe platform-specific topics to any great degree, other than to point out some of the differences in their implementation of InfoSphere CDC.

By the end of this chapter, you should have a good understanding of the major components of InfoSphere CDC and the nuances of how it operates. With this understanding, you are able to envision how the product could be set up in your environment to move your data efficiently and rapidly from one database system to another one.

6.1 Component overview

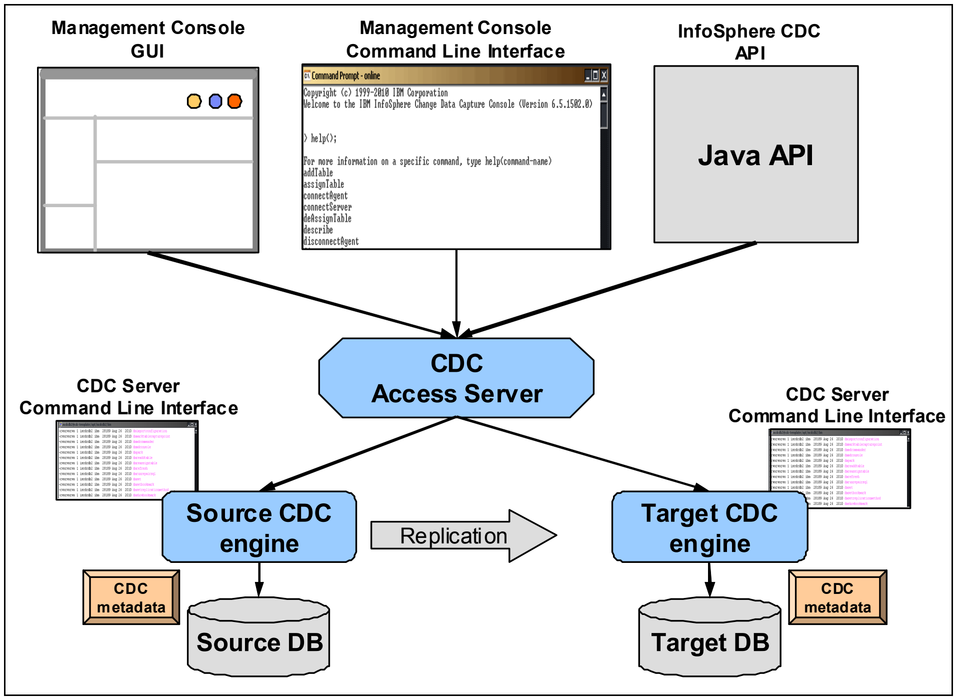
InfoSphere CDC assumes that you have a supported database that has tables that you want to replicate to a target. This replication could be accomplished two ways:

- In snapshot form, where all current data in a table or tables at some point is moved to a target.
- Capturing changes to the data as soon after they occur and moving only the changed data to the target, either as it is or with specific changes made to the data. This replication process could also include some metadata about the changes.

To accomplish this replication, the InfoSphere CDC implementation includes an InfoSphere CDC source engine and an InfoSphere CDC target engine to send, receive, and apply data changes. Most InfoSphere CDC engines can serve as a source engine, capturing database changes from the source database, and as a target engine capable of receiving change data and applying it to the designated target database or other destination, such as DataStage or a JMS queue.

An InfoSphere CDC engine is also commonly referred to as an InfoSphere CDC instance because all the work associated with the engine is held in a process or set of operating system processes that are combined. From the Management Console and Access Manager perspective, an InfoSphere CDC engine is a data store.

Only source and target engines are required for replication to occur, but there are additional configuration and control interfaces used primarily for configuration, control, and monitoring (Figure 6-1).



```
Command Prompt - online
Copyright (c) 1999-2010 IBM Corporation
Welcome to the IBM InfoSphere Change Data Capture Console (Version 6.5.1502.0)


> help();

For more information on a specific command, type help(command-name)
addTable
assignTable
connectAgent
connectServer
deAssignTable
describe
disconnectAgent
```

Figure 6-1 Replication landscape

Here are brief descriptions of the InfoSphere CDC components:

- Source and target InfoSphere CDC engines: These engines send, receive, and apply data changes. An InfoSphere CDC engine is also commonly referred to as a InfoSphere CDC instance. All of the work associated with the engine is held in a process or set of operating system processes that are combined. From the Management Console and Access Manager perspective, an InfoSphere CDC engine is seen as a data store.
- InfoSphere CDC metadata: Configuration information that is associated with a specific InfoSphere CDC instance, such as database connection information, subscriptions, and table mappings. The subscription and table mapping definitions are distributed across the source and target InfoSphere CDC engines.

- InfoSphere CDC Server command-line interface: Native commands that are run on the server that runs the InfoSphere CDC engine. These commands start and stop the entire InfoSphere CDC instance and subscriptions. This command-line interface (CLI) can only control the engine that provides the commands.
- InfoSphere CDC Access Server: This service controls all configuration, control, and monitoring access to the InfoSphere CDC engines other than the InfoSphere CDC Server CLI. The InfoSphere CDC Management Console GUI, Management Console CLI, and Java API all pass through the InfoSphere CDC Access Server to obtain access to the source and target InfoSphere CDC engines. Access Server also has a CLI to control the access of users to the various engines in the environment, but this section does not describe this tool. For more information about the Access Server CLI, see the *InfoSphere Management Console Administration Guide* at the following address:

  `http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.mcadminguide.doc/concepts/overview_of_cdc.html`
- InfoSphere CDC Management Console GUI: The most commonly used interface to configure and control InfoSphere CDC. All configuration, control, and monitoring activities are contained in this interface. The Management Console GUI connects to the InfoSphere CDC engines through the InfoSphere CDC Access Server.
- InfoSphere CDC Management Console Command Line Interface: A less deployed but useful interface to control InfoSphere CDC operations. The Management Console CLI connects to the InfoSphere CDC engines through the InfoSphere CDC Access Server.
- InfoSphere CDC API: This interface is sometimes used in large and complex environments. The InfoSphere CDC API provides full control over configuration, control, and monitoring by using Java classes and methods. This interface connects to the InfoSphere CDC engines through the InfoSphere CDC Access Server.

The Management Console GUI, Management Console CLI, and InfoSphere CDC API all must connect to the Access Server first before being able to do anything with the InfoSphere CDC engines. When connecting to the Access Server, a user name and password must be provided, which controls the access level of the user.

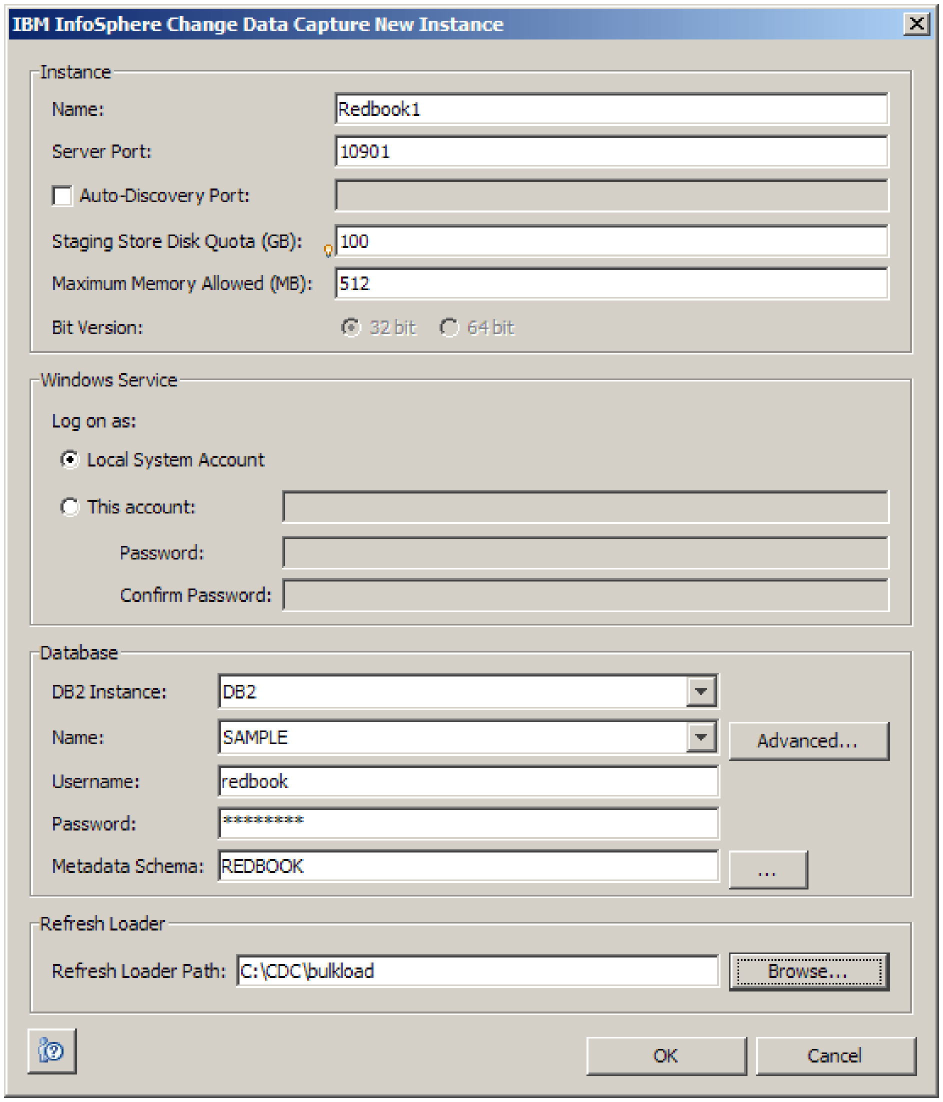
6.1.1 InfoSphere CDC instances

An InfoSphere CDC instance is a change data capture process that is associated with a particular database instance of a specific type (for example, DB2 or Oracle), with a message queue (InfoSphere CDC Event Server), or with IBM DataStage. There are InfoSphere CDC engines available for a wide range of database products and versions, on a wide variety of hardware platforms and operating system environments. An instance can handle both the source and target sides of replication, and can simultaneously be both a source and a target. For example, in a three-tiered scenario, InfoSphere CDC instance A captures change data from database X and sends it to InfoSphere CDC instance B acting as a target and applying the changes to database Y. Instance B can also act, at the same time, as a source that scrapes the logs for the same or a different set of tables and sendd the change data to InfoSphere CDC instance C as a target, which applies the change data to database Z.

An InfoSphere CDC instance has a CLI consisting of native commands that are run on the server that runs the InfoSphere CDC engine. These commands start and stop the entire InfoSphere CDC instance and subscriptions. The CLI can only control the engine that provides the commands. Many of the commands that can be run through this CLI can also be run through the Management Console interface.

In addition, each InfoSphere CDC instance has a metadata store specific to the instance. The configuration metadata that is associated to a specific InfoSphere CDC instance includes such information as database connection information, subscriptions, and table mappings. It is important to know that the subscription and table mapping definitions are distributed across the source and target InfoSphere CDC engines.

After InfoSphere CDC has been installed, one or more instances can be created, with each instance running as a separate process. Instances are created after an installation of InfoSphere CDC, or by running (for Java-based engines) the InfoSphere CDC configuration tool **`dmconfigurets`** and adding an instance (Figure 6-2).


IBM InfoSphere Change Data Capture New Instance

Instance

Name:	Redbook1
Server Port:	10901
Auto-Discovery Port:	
Staging Store Disk Quota (GB):	100
Maximum Memory Allowed (MB):	512
Bit Version:	[✓ 32 bit] 64 bit

Windows Service

Log on as:

[✓ Local System Account] This account:

This account:	
Password:	
Confirm Password:	

Database

DB2 Instance:	DB2
Name:	SAMPLE
Username:	redbook
Password:	*********
Metadata Schema:	REDBOOK

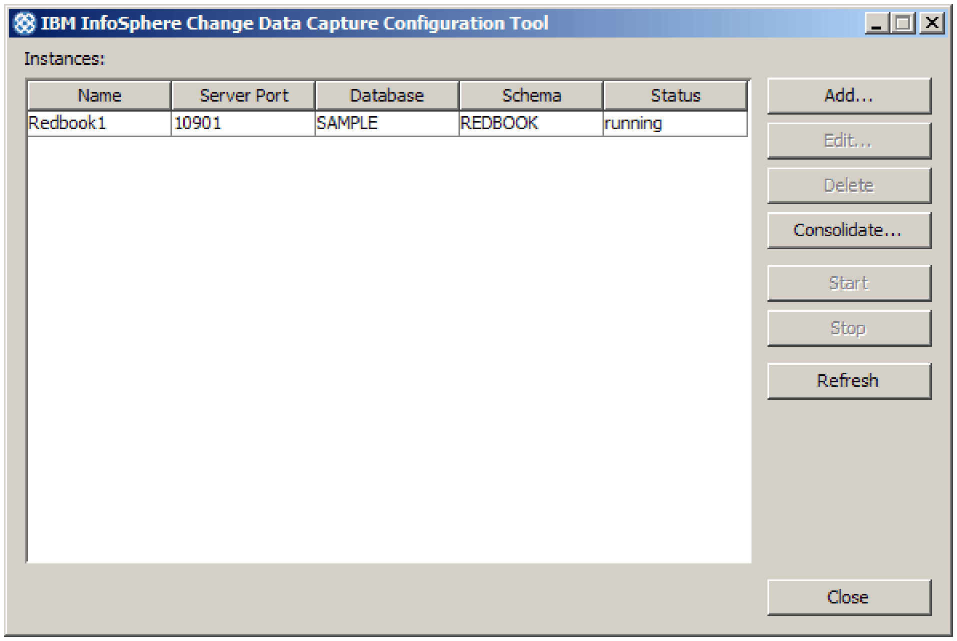
Refresh Loader

Refresh Loader Path:	C:\CDC\bulkload

Figure 6-2 Creating a instance

An instance can be controlled (started and stopped) from the Configuration Tool as well (Figure 6-3). There are separate InfoSphere CDC command tools that can also be used for this purpose, but the Configuration Tool is a convenient interface for this task.

Although the GUI interface to the configuration tool is shown here, there is also a console version for cases where it is preferred or where a GUI cannot be used.



Name	Server Port	Database	Schema	Status
Redbook1	10901	SAMPLE	REDBOOK	running

Figure 6-3 The InfoSphere CDC configuration tool

An InfoSphere CDC source instance and a target instance are all that is required to run InfoSphere CDC to capture change data and move it to a target after the system has been configured.

Because InfoSphere CDC incorporates a wide set of features and functionality, configuration is an important aspect of the product. One of the key modules in InfoSphere CDC is the Management Console, a GUI tool that runs on Microsoft Windows and is used to define what data on the source side is replicated, and how and to where it is replicated.

In addition to configuring InfoSphere CDC, there are also operational aspects of the Management Console, such as starting and stopping of replication activity, and monitoring activity while it is in process.

As a graphical user interface, the Management Console makes many otherwise complex tasks intuitive and straightforward. InfoSphere CDC also provides other tools that can be used for these tasks as well, for example, when you want or must automate some aspects of both configuration and control of a replication environment. These additional interfaces to InfoSphere CDC are described elsewhere in this chapter, and in the product documentation.

Although the basic tasks of developing and administering an InfoSphere CDC replication environment is made straightforward by the Management Console, the Management Console is a powerful tool that is likely to b used by multiple personnel within an organization. You should plan and assign privileges and roles appropriately. In a replication system, there might be multiple InfoSphere CDC source engines and target instances, which might be configured and controlled by one or more Management Console installations.

The last major component of InfoSphere CDC is the Access Server, which serves as a relay point between InfoSphere CDC instances and Management Console. Access Server knows about InfoSphere CDC data stores and InfoSphere CDC users, both defined in the Management Console. An InfoSphere CDC data store is essentially an InfoSphere CDC instance seen through the Management Console GUI. It is a source of changed data or a target that consumes that data. Users have specific types (which define the tasks they may perform) and are attached to specific data stores. As such, Access Server serves as the validation point for Management Console user logins.

Access Server can run on a separate machine than the InfoSphere CDC engine (and usually does), and is supported on Linux, UNIX, and Windows platforms.

6.1.2 Interoperability between the InfoSphere CDC components

InfoSphere CDC is designed to allow different versions of the product to work together. As a heterogeneous replication solution, a version of InfoSphere CDC for DB2 on System i must, as an example, be able to replicate to and from InfoSphere CDC for Microsoft SQL Server running on Windows Server 2008. In addition, InfoSphere CDC Access Server must be able to communicate with both components, and this is true for all InfoSphere CDC versions that can serve as a replication source or target. As expected, there are limits to this cross-release interoperability, and matrixes of compatibility are readily available in the product documentation.

6.2 Management Console fundamentals

InfoSphere CDC Management Console is a Java-based and rich client interface for an InfoSphere CDC replication system, and consists of four major functional subdivisions:

- Access control
- Configuration
- Operation
- Monitoring

It is also available as a command-line interface (CLI). The Management Console connects to the InfoSphere CDC engines through the InfoSphere CDC Access Server.

Multiple Management Consoles may be concurrently active for the same system, allowing a team approach to both developing and controlling a replication system with InfoSphere CDC in daily operations. Only one Management Console may be active on a particular machine, and team members should have Management Console installed on their own workstations.

This section describes Management Console concepts to provide a high-level overview of the functionality of InfoSphere CDC, and a framework for the introduction of some key architectural concepts. This section does not describe specific details about the use or operation of Management Console, or specifics of configuration options. For more information, consult the online documentation installed with the product, or visit the InfoSphere CDC Knowledge Center for InfoSphere CDC 6.3 at the following address:

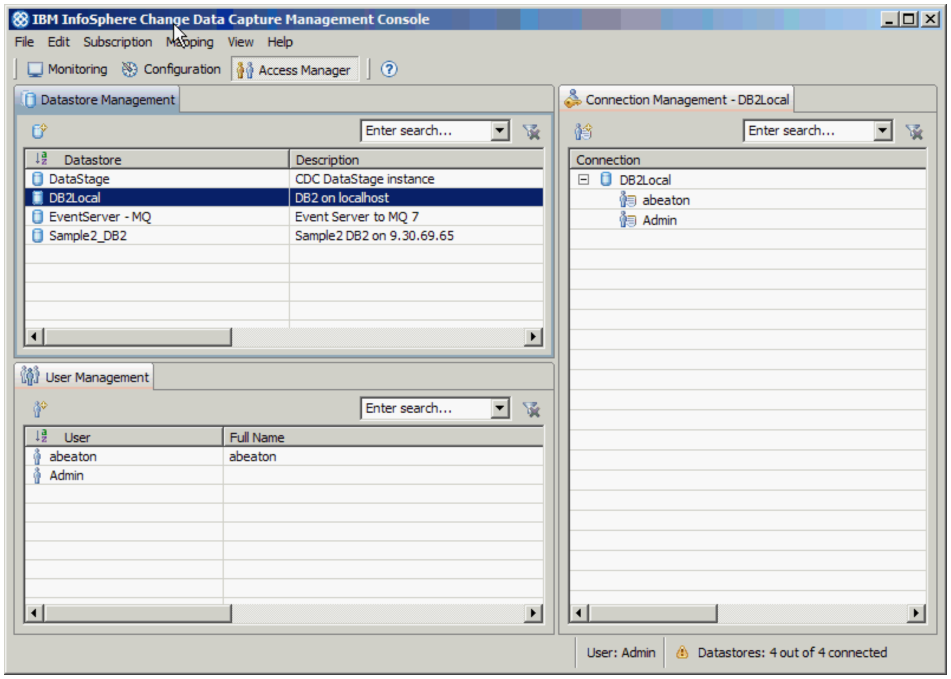
`http://publib.boulder.ibm.com/infocenter/cdc/v6r3m3/index.jsp`

There is also an InfoSphere CDC Knowledge Center for InfoSphere CDC V6.5 at the following address:

`http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/index.jsp`

6.2.1 Access Manager Interface

The Access Manager Interface in Management Console creates and configures InfoSphere CDC data stores, and creates InfoSphere CDC users and associate them with specific data stores (Figure 6-4).



Datastore	Description
DataStage	CDC DataStage instance
DB2Local	DB2 on localhost
EventServer - MQ	Event Server to MQ 7
Sample2_DB2	Sample2 DB2 on 9.30.69.65

User	Full Name
abeaton	abeaton
Admin	

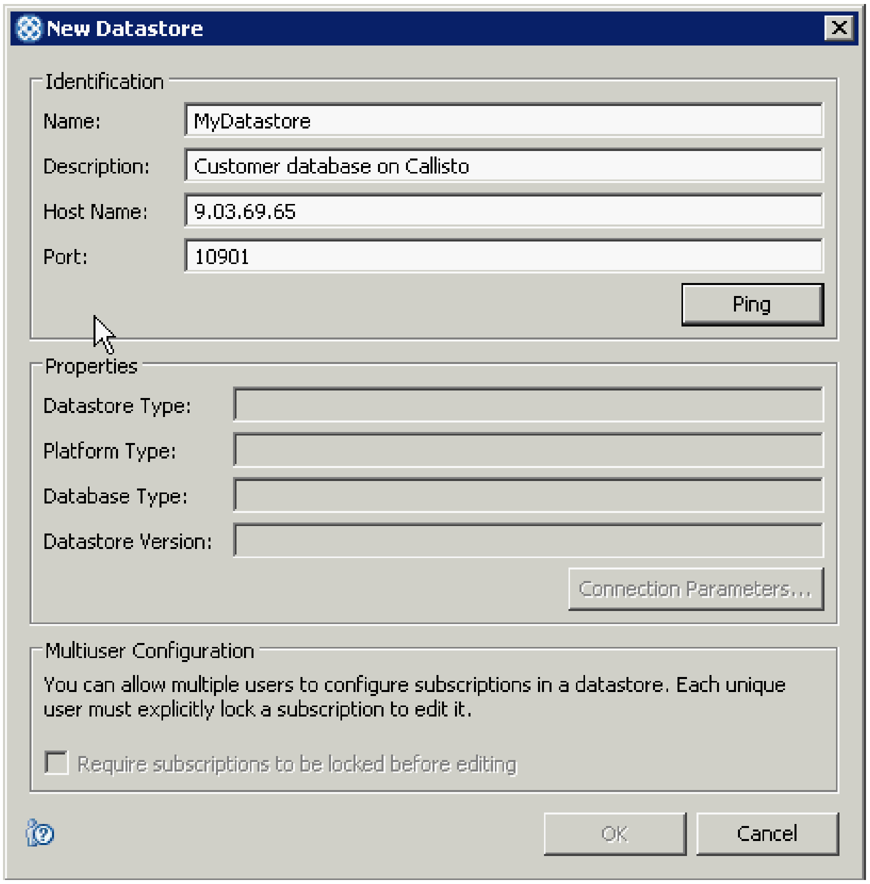
Connection
- DB2Local
  - abeaton
  - Admin

User: Admin   Datastores: 4 out of 4 connected

Figure 6-4 Access Manager Interface

Data stores

To design a replication configuration, InfoSphere CDC needs to have access to a source database from which it captures change data, and know the target to which the change data is delivered. An InfoSphere CDC data store is an InfoSphere CDC component that describes a source or target system, the host name on which InfoSphere CDC is running, and a database user ID and password for the system. In operation, InfoSphere CDC replicates to or from a data store. A data store is essentially a representation of an InfoSphere CDC instance (Figure 6-5).


New Datastore

Identification

Name:	MyDatastore
Description:	Customer database on Callisto
Host Name:	9.03.69.65
Port:	10901

Properties

Datastore Type:	
Platform Type:	
Database Type:	
Datastore Version:	

Multiuser Configuration

You can allow multiple users to configure subscriptions in a datastore. Each unique user must explicitly lock a subscription to edit it.

☐ Require subscriptions to be locked before editing

Figure 6-5 Data store component

InfoSphere CDC users

The Access Manager component of InfoSphere CDC allows the creation and control of InfoSphere CDC users.

There are distinct roles for users. One role is the System Administrator role, which can optionally create and manage data stores and users. Two other roles are the Operator and the Monitor roles, which can view replication state and status, view statistics, and events and table mappings, but cannot start or stop replication or perform any configuration.

Users are associated with data stores, and a user's privileges are active for the data stores with which they have been associated.

6.2.2 Configuration Interface

The Management Console configuration interface is used to define what tables from a source system are replicated, in what way (that is, what if any changes should be applied to the data), and which tables on the target system are recipients of the source data.

Types of replication

The two main types of replication in InfoSphere CDC are refresh and mirroring.

Refresh

A refresh operation, also known as a snapshot, generally involves a truncation of a target table and the insertion of the rows in the source table to the target. Under the heading of Refresh, there is a refresh operation called Differential Refresh, where differences between the source and target tables are applied to the target, bringing the two into the same state in a different manner from a full refresh. This type of refresh also has the option of logging any changes that are found and applying them, or logging the differences while not applying any of the changes required to make the source and target identical. There is also a range refresh where only rows from a specified range are brought over to the target. An InfoSphere CDC Refresh operation does not involve capturing change data from the source database log file, but rather reads from the source table and sends rows across to the target as inserts.

Mirroring

Mirroring involves the capture of change data from the source database log files, and moving the change data over to the target. Under the heading of mirroring, there is the Mirror Continuous option, where an InfoSphere CDC source engine runs continually, capturing change data on an ongoing basis and moving it to the target engine that is also continuously running.

There is also Mirroring Scheduled End where mirroring is run periodically to capture and move change data since the last time InfoSphere CDC was run.

Having a subscription in mirroring mode means that InfoSphere CDC tracks change data for the subscription. When log entries are read for tables in the subscription that have a Mirroring mode, they are processed and sent to the target (when a commit is read for the transaction of which the change is part.) If a table is not going to be mirrored, putting it into a state of Refresh Park ensures that it has no impact on mirroring activity. When a table is set to Mirroring mode, InfoSphere CDC enables additional logging on the table. When you set a table to Refresh Park, additional logging is disabled, and the stored bookmark position is discarded.

For the InfoSphere CDC engine on z/OS, additional logging (DATA CAPTURE CHANGES) is enabled when the table is first selected for replication (if not already configured on the table). Logging is turned off when the table is removed from a subscription, provided it is not included in other subscriptions.

A common way to begin replicating a table is to set it to refresh before mirroring. This action performs a refresh of the target table so that it is initially synchronized with the source, and when this action completes, mirroring automatically commences.

Subscriptions

A *subscription* is a logical container that describes a replication configuration from tables in a source data store to a target data store. All tables in a subscription are kept synchronized together. In addition to metadata about the subscription as a whole, such as latency thresholds for notification, a subscription consists of replication table mappings. A table mapping maps one source table to one target table, and a source table may be mapped once in any subscription. A table mapping also defines the type of replication (such as mirroring or refresh). In the place of a source column, you can have an expression or a Journal Control Field, which is a value derived from the source log file related to the operation, such as the record modification time or record modification user.

A subscription should group tables in such a way as to maintain referential integrity. For example, if table T1 has a foreign key that references table T2 and these tables are mapped to corresponding tables T1 and T2 on the target that have the same RI constraints, then these mappings should be in the same subscription. Within a subscription, operations are applied on the target in the same order that they are run on the source. It is important to understand that change data replicated as part of a subscription are all committed in the same transaction, thus ensuring data consistency.

There are many other powerful features that can be configured, such as row filtering based on some criteria or data translations or wanted encoding changes.

A subscription is shown in Figure 6-6.

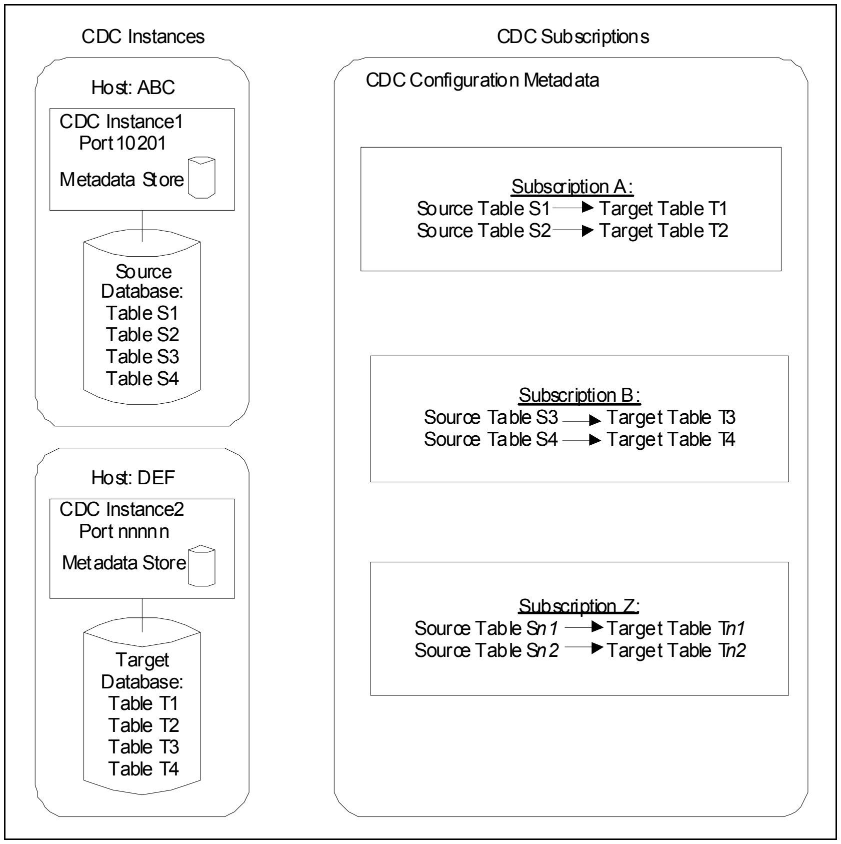
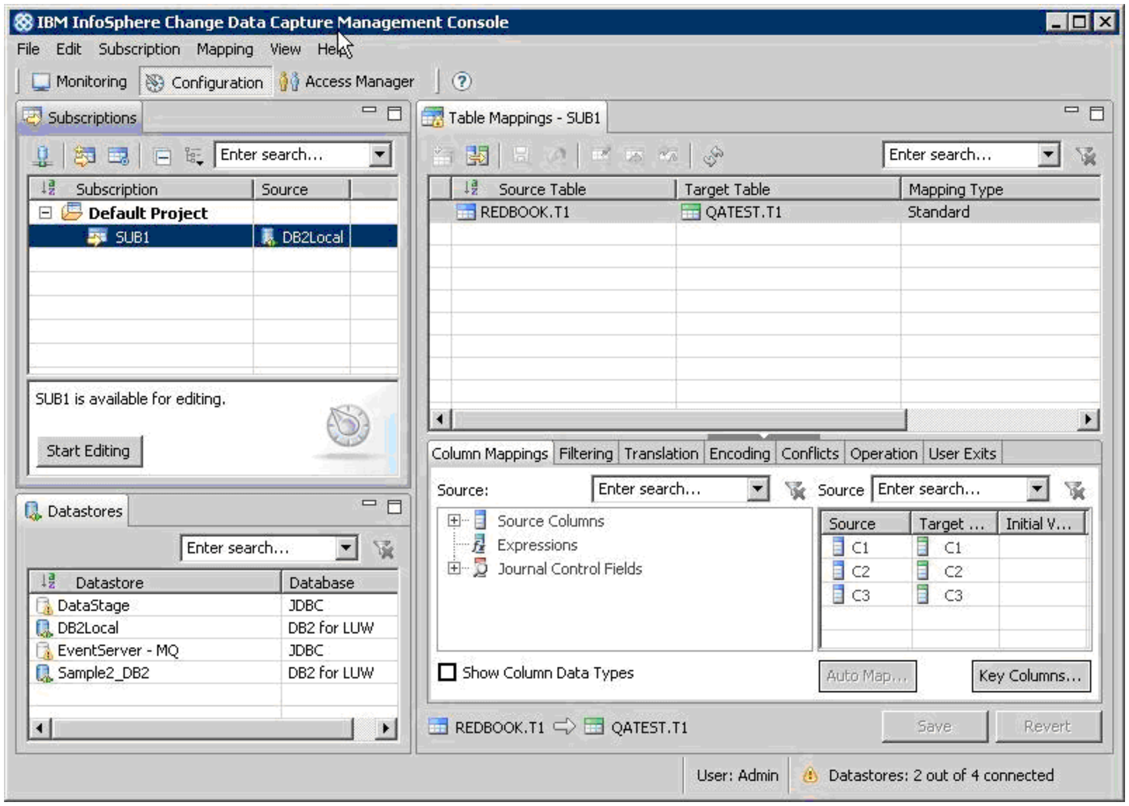

CDC Instances
CDC Subscriptions
Host: ABC
CDC Instance1
Port 10201
Metadata Store
Source Database: Table S1 Table S2 Table S3 Table S4
Host: DEF
CDC Instance2
Port nnnnn
Metadata Store
Target Database: Table T1 Table T2 Table T3 Table T4
CDC Configuration Metadata
Subscription A:
Source Table S1 → Target Table T1
Source Table S2 → Target Table T2
Subscription B:
Source Table S3 → Target Table T3
Source Table S4 → Target Table T4
Subscription Z:
Source Table Sn1 → Target Table Tn1
Source Table Sn2 → Target Table Tn2

Figure 6-6 InfoSphere CDC subscription

For ease of administration, InfoSphere CDC subscriptions can be contained within a Management Console project. A project is not a InfoSphere CDC component, but a way to group subscriptions within the Management Console. A Management Console project is transferable between Management Console installations using import / export.

You might need to have more than one person modify an InfoSphere CDC configuration at the same time, and for this reason, subscriptions can be locked for editing to prevent collisions among multiple users operating with different instances of Management Console.

The Management Console Configuration interface is used to create and configure subscriptions, which map source tables to target tables (Figure 6-7).



Subscription	Source
Default Project	
SUB1	DB2Local

SUB1 is available for editing.

Datastore	Database
DataStage	JDBC
DB2Local	DB2 for LUW
EventServer - MQ	JDBC
Sample2_DB2	DB2 for LUW

Source Table	Target Table	Mapping Type
REDBOOK.T1	QATEST.T1	Standard

Source	Target ...	Initial V...
C1	C1	
C2	C2	
C3	C3	

REDBOOK.T1 ⇨ QATEST.T1

User: Admin

Datastores: 2 out of 4 connected

Figure 6-7 Management Console configuration interface

6.2.3 Monitoring Interface

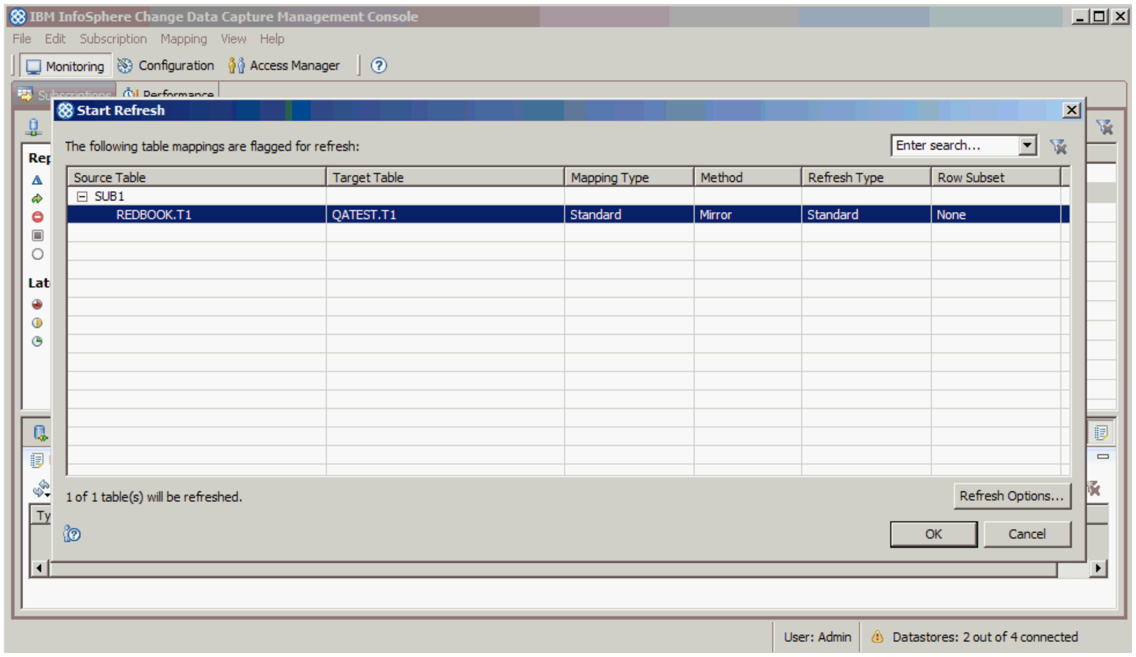
One of the properties of a mapping within an InfoSphere CDC subscription is its replication method, which can be set to either Mirror or Refresh (the replication method can be changed using Management Console). After a subscription has been fully set up, it is necessary to start the replication activity, and have InfoSphere CDC begin replicating data.

There are several means of accomplishing this task, but the Monitoring interface of Management Console is the most commonly used means. Other means include the InfoSphere CDC engine command-line tools and the Management Console API.

A subscription may be started in Refresh or Mirroring mode. In Refresh mode, table mappings that have a replication method of Refresh, and tables with a replication method of Mirroring that have been flagged for refresh, are refreshed one at a time until all the refreshes have completed. When starting in Mirroring mode, any tables mappings that have been flagged for refresh are refreshed before the start of mirroring replication.

While the subscription is replicating, there might be other subscriptions defined t are in an idle or inactive state.

The Management Console Monitoring interface can be used to initiate Refresh operations and start mirroring (Figure 6-8).


IBM InfoSphere Change Data Capture Management Console
File Edit Subscription Mapping View Help
Monitoring
Configuration
Access Manager
Start Refresh
Enter search...
Refresh Options...
OK
Cancel
User: Admin
Datastores: 2 out of 4 connected

The following table mappings are flagged for refresh:

Source Table	Target Table	Mapping Type	Method	Refresh Type	Row Subset
⊟ SUB1					
REDBOOK.T1	QATEST.T1	Standard	Mirror	Standard	None

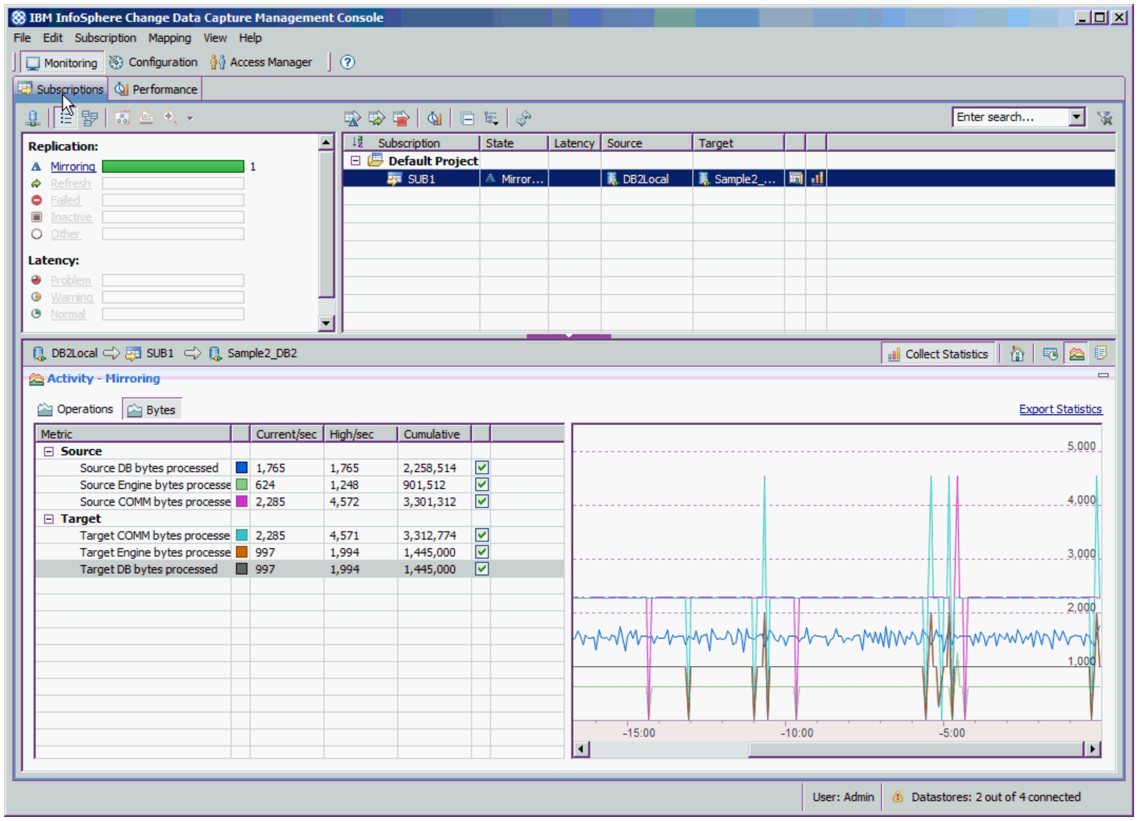
1 of 1 table(s) will be refreshed.

Figure 6-8 Refresh

When Refresh or Mirroring begins, the status of the subscription in the Monitoring interface changes to reflect the replication state.

The Management Console Monitoring interface provides many monitoring views of mirroring activity and InfoSphere CDC events, and is shown in the Monitoring interface. If you click **Collect Statistics**, you can monitor replication activity in terms of operations or bytes, latency alerts are shown, you can view a graph of latency or operations, and view the InfoSphere CDC event log for detailed messages.

InfoSphere CDC provides other mechanisms for starting and stopping mirroring activity, but Management Console provides a simple intuitive tool that serves most needs.



Subscription	State	Latency	Source	Target
Default Project				
SUB1	Mirror...		DB2Local	Sample2_...

Metric		Current/sec	High/sec	Cumulative	
Source					
Source DB bytes processed	🟦	1,765	1,765	2,258,514	✓
Source Engine bytes processe	🟩	624	1,248	901,512	✓
Source COMM bytes processe	🟪	2,285	4,572	3,301,312	✓
Target					
Target COMM bytes processe	🟦	2,285	4,571	3,312,774	✓
Target Engine bytes processe	🟧	997	1,994	1,445,000	✓
Target DB bytes processed	⬛	997	1,994	1,445,000	✓

Figure 6-9 Monitoring interface

6.2.4 InfoSphere CDC API

Sometimes used in large and complex environments, the InfoSphere CDC API provides full control over configuration, control, and monitoring with Java classes and methods. This interface connects to the InfoSphere CDC engines through the InfoSphere CDC Access Server, and can be used to enable automated configuration and control. Further information about the use of the API is in 9.1, “Options for managing InfoSphere CDC” on page 232.

6.2.5 Access Server fundamentals

Access Server is a service that controls all configuration, control, and monitoring access to the InfoSphere CDC engines other than the InfoSphere CDC Server CLI. The InfoSphere CDC Management Console GUI, Management Console CLI, and Java API all pass through the InfoSphere CDC Access Server to obtain access to the source and target InfoSphere CDC engines. Access Server also has a command-line interface to control access of users to the various engines in the environment. It supports the running of multiple Management Consoles. For more information about the Access Server CLI, see the *InfoSphere Management Console Administration Guide* at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.mcadminguide.doc/concepts/overview_of_cdc.html

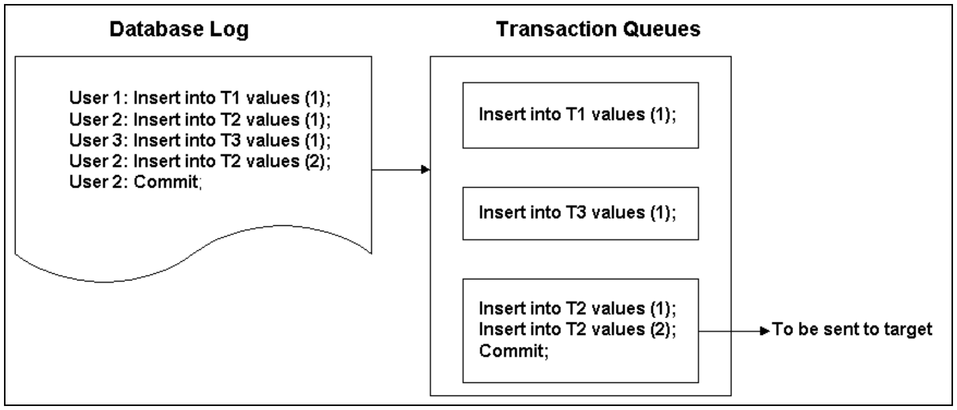
6.3 The InfoSphere CDC engine

Because InfoSphere CDC is a heterogeneous product (it can replicate between a wide range of hardware platforms, operating systems, and database products), there are a few different types of engines, all of which can act as a source or target. Because the internal architectures vary to some degree, they are described separately in this section after this section describes the InfoSphere CDC bookmark concept that is common to all of the types.

6.3.1 Bookmarks

The source database system can have many independent connections making concurrent changes to the data. At any time, one or more of these connections may have an open transaction containing all the changes that have been made on that connection since it last committed or rolled back. All these uncommitted changes also have been written to the database log.

InfoSphere CDC maintains these uncommitted changes in its Transaction Queues. There is a separate queue for each open transaction where InfoSphere CDC accumulates the changes for that transaction as they are read from the database log. InfoSphere CDC stores the uncommitted changes it has read from the log in to transaction queues until a commit is seen (Figure 6-10). Changes are removed from these queues when rollbacks are done in the source database.


Database Log
Transaction Queues
User 1: Insert into T1 values (1);
User 2: Insert into T2 values (1);
User 3: Insert into T3 values (1);
User 2: Insert into T2 values (2);
User 2: Commit;
Insert into T1 values (1);
Insert into T3 values (1);
Insert into T2 values (1);
Insert into T2 values (2);
Commit;
To be sent to target
Figure 6-10 Transaction queues

InfoSphere CDC only sends committed changes to the target to be applied. As each transaction is committed, InfoSphere CDC sends the changes for that transaction (taken from its transaction queue) to the target. The flow of change data from the source to the target can be seen as a stream of complete transactions being sent in the order in which they were committed in the source database. This flow is called the Replication Stream.

InfoSphere CDC maintains a “bookmark” as part of this Replication Stream. This bookmark is persisted into the target database along with the application data. It contains all the information necessary for InfoSphere CDC to be able to continue replication from that point. It is committed as part of the same transaction in which the change data is written to the target database.

The two primary pieces of information in this bookmark are:

- Restart Position: InfoSphere CDC needs to begin rereading the database logs to recreate the transaction queues. This action generally corresponds to the beginning of the oldest open database transaction at the time the bookmark was constructed by InfoSphere CDC. Using this Restart Position, InfoSphere CDC is able to recreate the Replication Stream.
- Stream Position: After InfoSphere CDC has recreated the Replication Stream, the stream position indicates exactly where InfoSphere CDC was in that stream when it last applied data to the target.

The InfoSphere CDC bookmark contains all the information necessary to recreate the replication stream on the source side and position it to resume replication at exactly the point of the last operation applied on the target.

Bookmark information is stored on the target side and is updated whenever data is applied to the target. It is communicated to the source engine while running to keep the source informed about log dependency. For restart, it is sent at the time mirroring starts to recreate the replication stream on the source at exactly the right point. It is this key mechanism that allows InfoSphere CDC to always know where scraping must be resumed, to not resend change data already applied on the target, and to not miss rescraping changes that were possibly previously read but lost in-flight as a result of an abnormal termination.

6.3.2 The InfoSphere CDC Linux, UNIX, and Windows engine

The InfoSphere CDC for Linux, UNIX, and Windows engine is a version ported to database products that run on these platforms.

InfoSphere CDC captures change data on a source from the source database log files. This scenario involves two separate components:

- A log reader
- A log parser

In Figure 6-11, these components are grouped as one, as the granularity of operations performed is not essential to an understanding of the activity at this level.

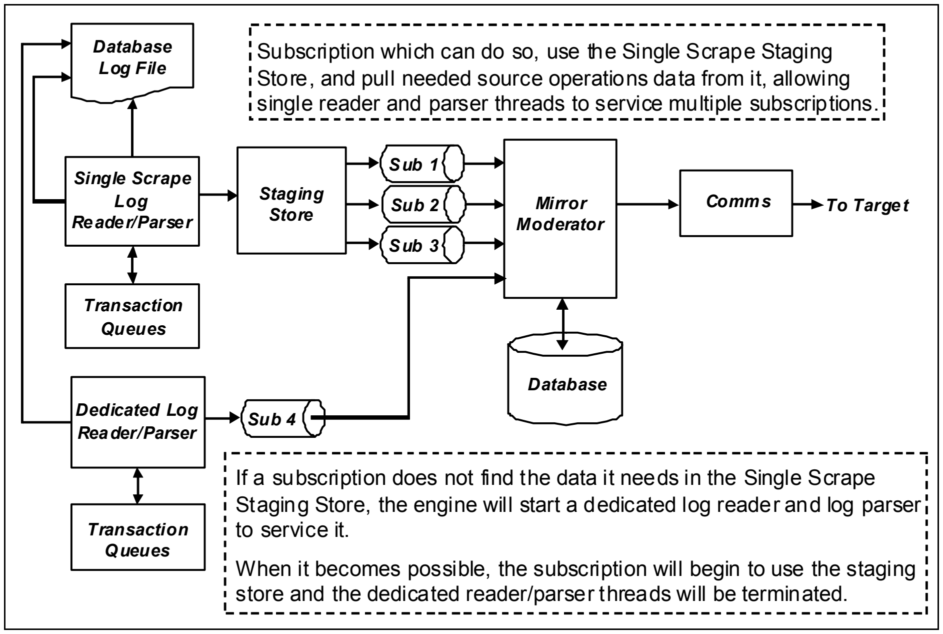

Database Log File
Subscription which can do so, use the Single Scrape Staging Store, and pull needed source operations data from it, allowing single reader and parser threads to service multiple subscriptions.
Single Scrape Log Reader/Parser
Staging Store
Sub 1
Sub 2
Sub 3
Mirror Moderator
Comms
To Target
Transaction Queues
Database
Dedicated Log Reader/Parser
Sub 4
If a subscription does not find the data it needs in the Single Scrape Staging Store, the engine will start a dedicated log reader and log parser to service it.
When it becomes possible, the subscription will begin to use the staging store and the dedicated reader/parser threads will be terminated.
Transaction Queues

Figure 6-11 Source operations component data flow for the Linux, UNIX, and Windows engine

During mirroring, a source InfoSphere CDC engine only sends committed data to a target. A source engine reads the log files, parses the log records and some metadata, such as the transaction ID and the table name, and stores them temporarily in the transaction queues before sending a number of committed transactions to the target.

InfoSphere CDC optimizes the resource utilization on the source server by, in most circumstances, using a single scrape component to only read the log records once and retain the operations derived from them for a time in a staging store. In certain circumstances, such as when the needed log records are not in the staging store, InfoSphere CDC employs a private scrape to get the needed log records. InfoSphere CDC dynamically determines if and when to employ a private scrape and when to remove one by having the subscription rejoin the single scrape.

LOBs do not participate in single scrapes, as they are retrieved directly from the database by the Mirror Moderator.

For refresh operations, the source engine retrieves rows using JDBC and puts them through the normal source processing before sending them to the target as insert operations.

Linux, UNIX, and Windows engine single scrape

To minimize resource consumption, InfoSphere CDC attempts to only read and parse the source database log files once. Employing single scrape optimization provides considerable overall efficiency improvements when replicating multiple subscriptions.

If a source system has constrained resources that might be taxed by having multiple log reader / log parser threads concurrently active, using single scrape in this scenario can be beneficial.

Because much of the log data is not table-specific, it is better not to have multiple threads processing and discarding the data. This situation is especially true where subscriptions are replicating only a small portion of the data being changed. For example, if two subscriptions have no tables in common but each is replicating only 10% of the total changes, then single scrape provides a huge benefit.

Single scrape is enabled by default in Version 6.5.1 for all InfoSphere CDC source instances (except for the Oracle Trigger version, which does not read the Oracle log to obtain source operations). This situation only applies to the Linux, UNIX, and Windows engine versions.

If you do not keep a single scrape cache of sufficient size to allow the subscription to use single scrape, the subscription will have its own parser. This situation results in a more manageably sized cache.

These situations include where a subscription has fallen too far behind the single scrape process and the content the subscription needs is no longer available. This situation could occur if, for example, a subscription was not running or idle for a period, and then later restarted.

If a subscription had been idle (that is, not mirroring) for a period and is then started, the log entries the subscription needs might not be found in the staging store. In this case, the subscription has its own log reader and log parser. If at some point the subscription caught up so that needed content is available in the staging store, the subscription joins single scrape and its dedicated reader, and the parser would go away.

A subscription might be too far ahead and waiting for single scrape to advance to its position is too inefficient compared to having the subscription run on its own. This situation could occur when a subscription has refreshed all of its tables, and so is scraping near the head of the log while Single Scrape is still scraping an older portion of the log.

In either case, for a subscription running with its own reader and parser, InfoSphere CDC can decide to have the subscription join (or rejoin) single scrape when it becomes more efficient than maintaining the reading and parsing activity separately.

InfoSphere CDC combines the best advantages of a single reader / parser thread with the flexibility of per subscription reader parser threads, and can automatically compute and run with the most advantageous arrangement.

However, there may be cases where you want to manually force a certain mode. You can decide if InfoSphere CDC allows subscriptions to run independently or to turn Single Scrape off, and thus force all subscriptions to run independently.

There are implications to turning Single Scrape off and forcing all subscriptions to use it. Those implications, and the InfoSphere CDC system parameters required to force all the subscriptions, are described later in this chapter.

Linux, UNIX, and Windows engine staging store

The source instance component diagram shown in Figure 6-11 on page 123 maintains a staging store of change data that can be used by active subscriptions. To maintain maximum throughput, to the degree that memory constraints allow it, InfoSphere CDC maintains this staging store in memory.

However, InfoSphere CDC might need to move some of this staged data to disk. This situation could occur if, for example, some active subscription was unable to move the data to the target as rapidly as it was accumulating, causing the staging store size to grow. This situation could occur if the network between the source and that target temporarily became busy, the target database became busy, or because mirroring for that subscription was stopped temporarily.

If you allow a subscription to remain idle for a long period, it is possible that the staging store size on disk might become considerable. For this reason, InfoSphere CDC provides a configuration option to set the disk quota for this storage, which defaults to 100 GB, but can be as small as 1 GB, or as large as you want to set it. Subscriptions that are not actively mirroring are said to be idle. InfoSphere CDC Single Scrape excludes tables that are in a table mapping that has been marked for Refresh mode and parked. Tables in mappings that are Mirror / Parked are still included for scraping.

During a controlled shutdown of InfoSphere CDC, the portion of the staging store in memory is persisted to disk, allowing for faster restart of mirroring. If there is an abnormal termination of InfoSphere CDC due to a system crash, the logs are read from the correct point so mirroring can resume with no loss of data. The persistence of the staging store on disk is not essential for restarting replication, as long as the database logs holding the required entries are still available. The format of persisted staging store data is internal to InfoSphere CDC and cannot be used by another application to read change data.

During mirroring of change data, each subscription retrieves the changes for its set of tables from the staging store. Data is removed from the staging store after it has been sent to all subscriptions that are using or could use the store, or when the staging store size threshold has been reached.

It is a common practice among users to create a test subscription or subscriptions. These subscriptions should be parked or deleted after work with them is completed so that single scrape does not take them into account and maintain data for them in the staging store.

Linux, UNIX, and Windows Engine Staging store size considerations

The value used for **`staging_store_disk_quota_gb`** only guarantees that InfoSphere CDC does not exceed that amount of storage for the staging store and does not pre-assign disk space. You must ensure that the configured amount of disk space is available to InfoSphere CDC. Should the staging store be below the disk quota and you still run out of disk space, an error occurs and mirroring halts.

In general, InfoSphere CDC only adds data to the staging store when at least one subscription is mirroring. For situations where subscriptions are only run periodically, but you want to have InfoSphere CDC capturing changes all the time, you can use command-line commands to control the InfoSphere CDC continuous capture feature. For more information about this feature, see Chapter 8, “Performance analysis and design considerations” on page 211. When enabled, InfoSphere CDC reads logs and adds change data to the staging store all the time, even when no subscriptions are running. This setting affects the amount of data that must be stored in the staging store, and almost certainly causes the staging store to grow bigger than the available memory and thus be partially persisted to disk.

If the InfoSphere CDC staging store disk quota is reached, subscriptions are forced into using their own dedicated log reader and the advantages of having multiple subscriptions use Single Scrape are lost until the situation is rectified.

It is important to avoid out-of-disk space situations, as these situations result in abnormal termination of InfoSphere CDC and require rescraping of the database logs. The staging store is only affected by subscriptions with tables configured for mirroring. The store contains data for all tables being mirrored (used in a table mapping with a replication method of mirror). Any subscriptions used solely for refreshing tables (where all the table mappings have a replication method of refresh) do not affect the behavior of single scrape or contribute to the size of the staging store.

When all the source subscriptions are running and have little latency, the volume of data kept in the staging store is relatively small, corresponding essentially to the delay in apply speed between the fastest and the slowest subscriptions.

When all the source subscriptions are running but some are latent, then the staging store needs to contain all the data for the latent subscriptions. For example, if one of the subscriptions is one hour latent, then that hour's worth of data for that subscription needs to be kept in the staging store along with that hour's data for all subscriptions.

Persisting the staging store to disk during operation

The data in the staging store is organized into a sequence of data blocks. If the staging store grows to a size that cannot be kept in memory, then some of these data blocks are persisted to disk and removed from memory. InfoSphere CDC is optimized to select blocks to write out based on the most likely best performance outcome, and there are no available tuning parameters for this situation.

Persisting the staging store to disk at shutdown

When the InfoSphere CDC engine is shut down by running **`dmshutdown -c`** (normal shutdown), InfoSphere CDC persists all the data in the staging store to disk so that the data is available when the engine restarts.

Single scrape events and errors

Single scrape errors and events are visible in the event log. A list of these events and errors are provided in Appendix A, “Single scrape events and errors” on page 431.

Transaction queues

Uncommitted transactions accumulate on the source until they are committed. If memory requirements dictate that some of the transaction queue data needs to be moved to disk to free some memory, some of the transaction queue data may be persisted in temporary files.

Persisting transaction queues at shutdown

When a normal shutdown is called for a subscription that is mirroring with a private parser, the uncommitted data that InfoSphere CDC maintains in memory is written out to the InfoSphere CDC transaction queue storage. This storage is maintained in a repository specific to each InfoSphere CDC instance, located in the instance configuration directory, and is called txqueue.

If all subscriptions running with private parsers are shut down normally (controlled) at the same time, then all uncommitted transaction data needs to be stored in the transaction queue repository at the same time.

Shared scrape also has transaction queues. These queues are persisted whenever the last subscription running with shared scrape stops.

If a subscription is using single scrape, it has no parser and no transaction queue data. If it is stopped, no transaction queue data is persisted at that time. If a subscription that is using a private scraper is stopped, its uncommitted transaction data is persisted. When all subscriptions that are using shared scrape have stopped, the single scrape parser persists its uncommitted transaction data at that time.

Because all private parsers and shared scrape are using the same repository, when all subscriptions are stopped, the size of the repository grows to the size needed to store all of this data at the same time.

The txqueue repository grows to be as large as the total amount of uncommitted transaction data for all subscriptions that existed when they were all shut down.

A long-running large uncommitted transaction, many subscriptions that are shut down (controlled), and subscriptions that are added over time can cause the txqueue database to grow to a large size at shutdown time.

This internal repository does not shrink files after they have grown. It might appear that there is much content in the txqueue database, but that might not be the case. If the files have grown large and disk space is a concern, the txqueue database can be deleted.

Failure to persist transaction queues

Abnormal termination of InfoSphere CDC prevents it from being able to persist the in-memory transaction queue data to disk. Because the data is persisted to disk when a subscription does a normal shutdown, if a subscription that is mirroring is stopped using the shutdown immediate option, or if the machine crashes or the InfoSphere CDC source process is terminated, there is no transaction queue data available for the subscription at startup. InfoSphere CDC must always be able to revert to the source database log files and start from the first operation of the oldest open transaction that existed when the last source transaction committed (that was most recently applied on the target). If there were no open transactions at the time of the controlled shutdown, then InfoSphere CDC starts from the commit of the transaction that was most recently applied on the target.

Persisting the transaction queue data during a controlled shutdown is an optimization that permits a faster startup of the mirroring continuous mode, because the data can be read from the persisted data much faster than it can be reread from the database log files.

This situation also means that the txqueue files can always be deleted from the instance configuration directory. The only effect of this action is that when mirroring is started for a subscription, it needs to begin reading the database log further back than it otherwise would, and so might take longer to start replicating.

If the only subscriptions that are not running are subscriptions that were not stopped using shutdown controlled the last time they were run, if these subscriptions are refreshed before mirroring them again, or the user does not care about these subscriptions (for example, if all their tables are currently parked or the subscription is never used), then the persisted txqueue files have no effect.

Linux, UNIX, and Windows engine metadata in brief

InfoSphere CDC for the Linux, UNIX, and Windows engine stores most of its metadata in its own separate repository, independent of other repositories, with a small portion of the metadata maintained in the host database. This operational metadata includes the TS_AUTH table with some instance metadata, and the TS_BOOKMARK table that is used to store the InfoSphere CDC bookmark for the last applied transaction on the target. This setup enables InfoSphere CDC to always be able to compute the correct restart position in the source log files.

The InfoSphere CDC configuration metadata for the InfoSphere CDC Linux, UNIX, and Windows engine is stored in a separate container from the database system from which it is replicating or to which it is replicating. InfoSphere CDC uses the IBM PointBase pure Java light-weight database, which is used for several purposes by InfoSphere CDC. In addition to the configuration metadata storage on the source and target sides, IBM PointBase is also used to store InfoSphere CDC events, InfoSphere CDC statistics, and persisting transaction queues at shutdown.

The InfoSphere CDC configuration metadata can be viewed by running **`dmmdconsole -I <instancename>`**. However, the metadata is not directly editable at the user level and the connection obtained by the command l is read-only. The metadata is defined by configuring InfoSphere CDC through the use of Management Console or the InfoSphere CDC tools. Changes to the metadata must be made through these tools only.

Generally speaking, metadata is an internal implementation detail, which you do not have to understand to understand the product. However, you might find it helpful to understand the metadata structure, and so these details are provided. A listing of the main InfoSphere CDC metadata tables is available as a PDF for downloading and viewing from the IBM Redbooks website. For information about how to access that PDF, see Appendix B, “Additional material” on page 435.

The configuration metadata should be backed up periodically by running **`dmbackupmd`**.

As mentioned previously, the replication subscription and table mapping definitions are distributed across the source and target engine metadata stores. After your configuration changes, you should back up the metadata on both the source and target sides, because recovering an InfoSphere CDC system to a previous state requires restoring the metadata on both sides. It is best practice to back up the InfoSphere CDC metadata along with and at the same time as the user database.

Linux, UNIX, and Windows engine as a target engine

All source operations are moved to the target engine and become target operations. A target operation contains both the before and after image for an update, and maintains all state information about the object (derived expressions and user exit calls). The target engine generates SQL statements and writes the bookmark information to the TS_BOOKMARK table.

The target flow is shown in Figure 6-12.

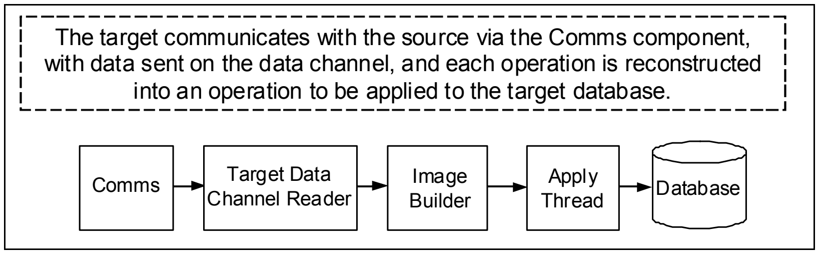

The target communicates with the source via the Comms component, with data sent on the data channel, and each operation is reconstructed into an operation to be applied to the target database.
Comms
Target Data Channel Reader
Image Builder
Apply Thread
Database

Figure 6-12 Target operations component data flow

6.3.3 The InfoSphere CDC for System i engine

InfoSphere CDC on System i uses the same concepts as used in other InfoSphere CDC versions. On System i, the equivalent functionality is implemented as different jobs for each component, such as the mirror driver, scraper, source control, target control, and apply.

There is no single scrape mechanism. Instead, each subscription has its own log reader process using the receive journal entry API to capture change data for the tables that are in scope for that subscription. There is one log reader (DMSSCRAPER) per subscription and per journal, because on System i you can have one subscription with tables journaled to different journals. There is one mirror process that merges the changes coming in from the different scraper jobs into source operations to send to the target side.

As change data is read from the log, it is stored in a System i user space object until a commit is read for the transaction. However, most applications on System i do not use commit control and therefore the database performs auto-commits when operations are applied to the DB2 tables. For most applications, there is no staging at the source and operations read from the journal are directly sent to the target. The entries in the user space are not kept after the subscription ends mirroring. When mirroring resumes for the subscription, InfoSphere CDC obtains the bookmark information for the last applied operation from the target and begins reading the journals from that position.

As a target, InfoSphere CDC on System i uses native apply to write to DB2. It builds up the buffer of the operation image and then writes it natively. The bookmarks for all subscriptions targeting a System i engine are kept in the user space in the InfoSphere CDC product library; the user space is called JRN_STATUS.

For refresh, as a source, InfoSphere CDC writes user-defined entries into the journal to mark the refresh start and end for the refresh while active logic. When mirroring is restarted, change data between these markers is read and sent across to the target to be applied there, as is done with other versions of InfoSphere CDC.

For refresh as a target, refresh messages received from the source system are processed as insert records, becoming a series of row by row inserts using native apply.

When a user initiates a (normal) controlled shutdown of a subscription on System i, a \S user-defined journal entry is written into the journal. When the log process reads this entry, it initiates the shutdown procedure. Any processing of entries before that point is completed, and InfoSphere CDC processes then end.

InfoSphere CDC on System i maintains its metadata files directly in DB2 tables (physical files), which are kept in the InfoSphere CDC product library.

6.3.4 The InfoSphere CDC for z/OS engine

The InfoSphere CDC for DB2/z engine shares many of the same concepts with the Linux, UNIX, and Windows engine previously described.

InfoSphere CDC parameters cannot be set from the Management Console as with the other engines. Parameters must be set in the control data sets for InfoSphere CDC. System parameters for this engine are not dynamic and are only read when the instance is started (started task).

InfoSphere CDC metadata for both source and target descriptions is stored in the DB2 database, and can be accessed through SQL. User access to it is available through reporting utilities that allow documenting of the InfoSphere CDC environment. Metadata can thus be saved and restored for product roll-out.

InfoSphere CDC for z/OS has a single scrape component that acts as a log reader and log parser. The component checks and analyzes the source database logs for all of the subscriptions on the selected data store, and uses Hiperspaces for staging data that it has read until it reads a COMMIT DB2 Log record. Single scrape on this engine is designed to reduce MIPS and should only be enabled if there are two or more subscriptions.

Hiperspaces are stored in z/OS Central Storage that is above the bar (64-bit addressable storage), and may be paged to z/OS Auxiliary Storage if the page frames in Central Storage are taken by the operating system on behalf of another address space.

This scraper component, the database synchronous log task (DSL), reads the DB2 log blocks for in-scope tables and reconstructs before and after images for operations on those tables. The images are sorted into ascending date and time sequence and added to a memory file.

The Database Log Scraper (DLS) task reads the memory file written by the DSL task and gathers the before and after row images into commit groups, which are written to a memory file as logical transactions.

A Communication Initialization and Termination (CIT) task reads the logical transactions from the memory file and sends them across to the InfoSphere CDC target engine to be converted to target operations and applied to the target database.

InfoSphere CDC supports concurrently running product address spaces under the same image of IBM MVS™ / ESA. Each InfoSphere CDC address space (or instance) is associated with only one DB2 subsystem. A DB2 subsystem can have multiple associated InfoSphere CDC address spaces. For multiple InfoSphere CDC address spaces, there is no communication between them other than InfoSphere CDC source-target communication.

The source flow in a z/OS environment is shown in Figure 6-13.

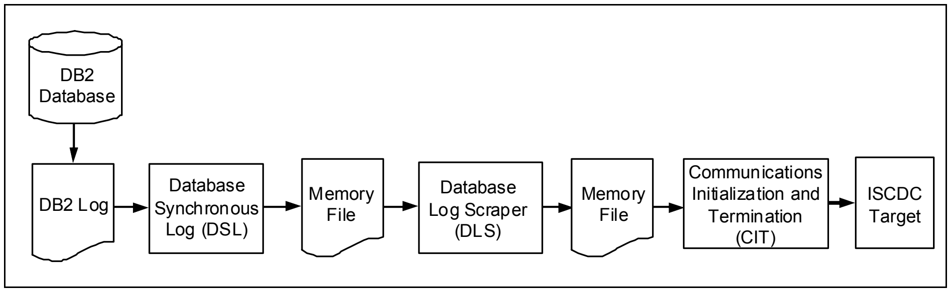

DB2 Database
DB2 Log
Database Synchronous Log (DSL)
Memory File
Database Log Scraper (DLS)
Memory File
Communications Initialization and Termination (CIT)
ISCDC Target

Figure 6-13 z/OS source flow

As a target, InfoSphere CDC uses storage above the bar (64-bit addressable storage) to cache changes that are applied to tables at the target. As the changes are received from the source environment by the CIT, they are cached and applied to the target tables by a database table change task. This task issues a SQL request to write the table row changes to the DB2 target table, after first applying any transformations specified in the metadata and starting any user exits.

When the applied changes are committed, the changes are purged from the cache.

If DB2 backs out the logical unit of work before the changes can be committed (for example, due to a deadlock or timeout condition), then InfoSphere CDC rereads the changes from the cache and reapplies them.

For refresh, as a source, a Database Table Refresh (DTR) task uses an SQL bulk read to retrieve rows from the source table. Checkpoint information is written into the InfoSphere CDC metadata table, causing DB2 to write a corresponding DB2 log record that marks the refresh operation begin point.

After a commit group is assembled on the source, a CIT task pushes the data to the InfoSphere CDC target.

For refresh as a target, refresh rows are received from the InfoSphere CDC source by the CIT task and passed to the Database Table Change (DTC) task. This task applies any transformations as specified in the metadata, starts any user exits, and then issues a SQL request to write the table rows into the target table.

6.4 Communications between source and target

The InfoSphere CDC communications component (Comms) includes a monitor component that coordinates and performs health-check monitoring of other InfoSphere CDC components.

Each InfoSphere CDC instance has both a control channel using two TCP/IP sockets and each subscription has a data channel using two TCP/IP sockets (each being used unidirectionally).

The control channel allows for the sending of control messages between the source and target independent of what might currently be on the data channel. This situation allows such things as the target knowing that mirroring is shutting down for the subscription while there is still data in the data channel for it to process.

For example, when a command is received by a subscription to begin mirroring, the Comms component requests the bookmark position for the subscription from the target engine. The return value is used to compute the starting position in the logs on the source engine side, taking into account data that might exist in the single scrape staging store and in the persisted transaction queue data.

6.5 Summary

This chapter introduced the general architecture of InfoSphere CDC, defined and described some key InfoSphere CDC terms, and provided an introduction to the primary InfoSphere CDC components.

Now you should have a good understanding of the InfoSphere CDC approach to replication. It should be sufficient to enable you to understand how InfoSphere CDC could be (or is being) used in your environment. And it should enable you to delve deeper into the workings on the platform you are operating with, and better understand the details of the wide range of functionality provided by InfoSphere CDC.

7

Environmental considerations

This chapter outlines some of the specific environmental considerations you need to be aware of, including:

- Dealing with different time zones or encodings is described in 7.1, “Globalization with InfoSphere CDC” on page 138.
- An IBM InfoSphere Change Data Capture (InfoSphere CDC) landscape can consist of several servers that must interconnect to be able to perform configuration and replication activities. It is common that firewalls have been put into place to protect these servers from unauthorized access. As such, firewalls contribute to almost all InfoSphere CDC connectivity issues. The configuration of these firewalls must be considered when designing your replication landscape with InfoSphere CDC, as described in 7.2, “Firewall configurations” on page 149.
- If and how you can use InfoSphere CDC when access to the source or target is restricted is described in 7.4, “Remote processing capabilities” on page 162.
- When replicating from a resilient environment or when InfoSphere CDC must be part of the resilient environment you have set up or been asked to design, see 7.5, “Using InfoSphere CDC in resilient environments” on page 168.

- How to cooperate with changes in your database environment or replication needs is outlined in 7.6, “Change management” on page 190.
- This chapter describes the initial load aspects when you start (or reinitiate) InfoSphere CDC.

Whenever your environment has some requirements for settings, the information in this chapter should help you with design considerations and InfoSphere CDC suggestions.

This information can be beneficial in the design phase, but also has some hands-on examples that can assist during the implementation of the designed architecture.

7.1 Globalization with InfoSphere CDC

In this fast growing economic environment, globalization is an important factor to consider when combining information from around the world. Two key aspects that this section addresses are:

- How do you combine data from around the world into a centralized time zone?
- How do you translate the different characters into a single character set?

The good news is that current IT environments have been provided with the technical means to solve these questions:

- Time zones and Coordinated Universal Time (UTC)
- Unicode (an international character encoding standard)

When systems with different time zones or code pages interact with each other, you need to take care of proper conversions. This section describes the means and methods to deal with these issues while using InfoSphere CDC.

7.1.1 Time zone considerations

Time zone considerations can be made both from a business and a technical viewpoint.

For certain business requirements, the time of the creation of the data or the time of consolidating the data is important. For example, for a banking environment, you might want to only create the planning and forecasting report when all data from the different worldwide business units is consolidated from all business units' at noon local time.

From a technical point of view, you could be investigating a data consistency problem by comparing source logging with the target log. You must be sure to compare the events that happened at the same time, independent of the time zone where the event occurred.

Technical considerations when dealing with time zones

Within InfoSphere CDC, you have date and time information in the following areas:

- Date and time fields within the data.

  These fields are the date and time fields within the table. They can either be generated by the source application or the database (for example, in current time fields.). From an InfoSphere CDC perspective, when they are part of the selected columns for replication, they are not modified. So the time that was put in the column is reflected without any time-zone conversions into the target database.

- Time stamp within the log.

  This time stamp is provided by the database and is put in the database logging mechanism.

- Time stamp from the events.

  This time stamp is put in the event log by InfoSphere CDC.

Time zone considerations for replicating data

Consider timestamps within the log. This section uses the example shown in Figure 7-1. Assume that the worldwide travel policy is that travel can only be booked after approval has been received from the travel manager. The travel policy is audited on regular basis by the worldwide expense auditor, who does not approve any expenses when the travel policy has not been followed.

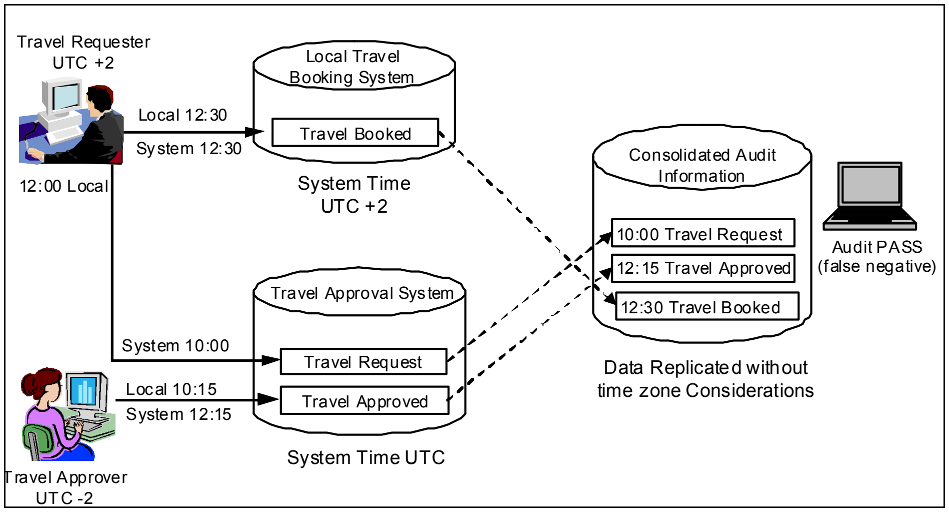


Figure 7-1 Considerations without time zone

As outlined in Figure 7-1, both the Travel Requester and Travel Approver use the Travel Approval System for the Approval process, which keeps its system time in UTC. At 12:30 local time, the Travel Requester (who is in an area where the local time is UTC+2) puts in a Travel request. This travel request is approved by the Travel Approver (who is located in an UTC-2 time zone) at 10:15 local time. Assume that the records do not keep a date and time and that the operation time stamp in the database log is used by InfoSphere CDC to generate the time of request and approval in the Consolidated Audit Information, which uses the system time.

The Travel Requester did not wait for the Travel Approver to book his trip. However, due to the time zone differences, the invalid sequence of events are not reflected in the Consolidated Audit Information. When the auditor compares the time of approval with the time of booking, and without having the location information of each of the roles, no irregularities are found and the booking passes.

When the time zone of the Travel Requester is considered when the data is replicated, the sequence of events becomes clear. In this case, assume that the log entries are changed to the UTC time zone. This time, the Consolidated Audit Information reflects that the Travel Requester did not wait for approval before booking his trip (Figure 7-2).

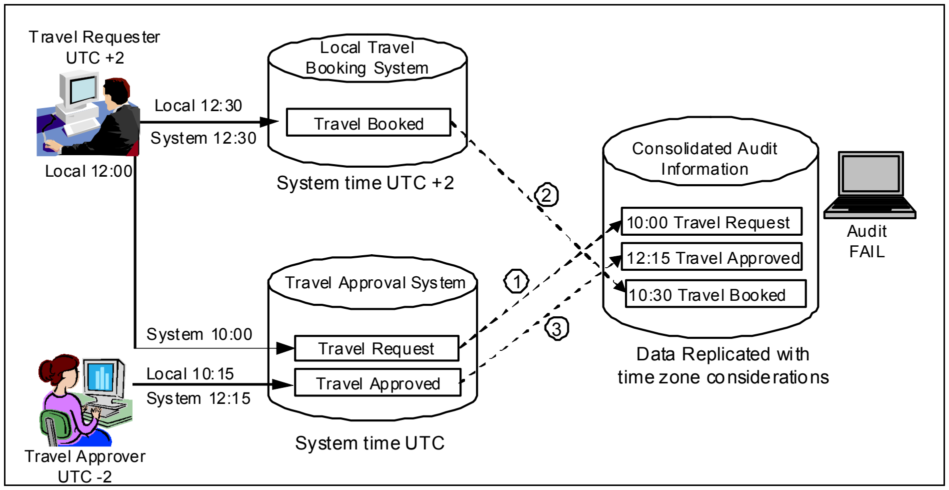


Figure 7-2 Considerations with time zone

With the correct timing sequence, it is clear to the worldwide auditor that the process for Travel Request has not been done in the correct sequence and therefore fails the audit.

To illustrate the situation, consider the sample scenario outlined in the following sections.

Booking system

The booking system is set to UTC+2, and has the following database table definition:

```
CREATE TABLE "CDCDEMO"."STD_TRAVEL_BOOKED" ("TRAVELID" NUMBER(10)
  NOT NULL, "TRAVEL_INFO" VARCHAR2(30),
  CONSTRAINT "TRAVELID" PRIMARY KEY("TRAVELID"))
  TABLESPACE "USERS"
```

In this sample, the STD_TRAVEL_BOOKED table represents the booking application. It records TRAVEL_INFO but does not have a booking time in the table itself. The booking time is taken from a journal control field.

Within InfoSphere CDC, the instance refers to the CDC_Oracle_Redo data store.

Approval system

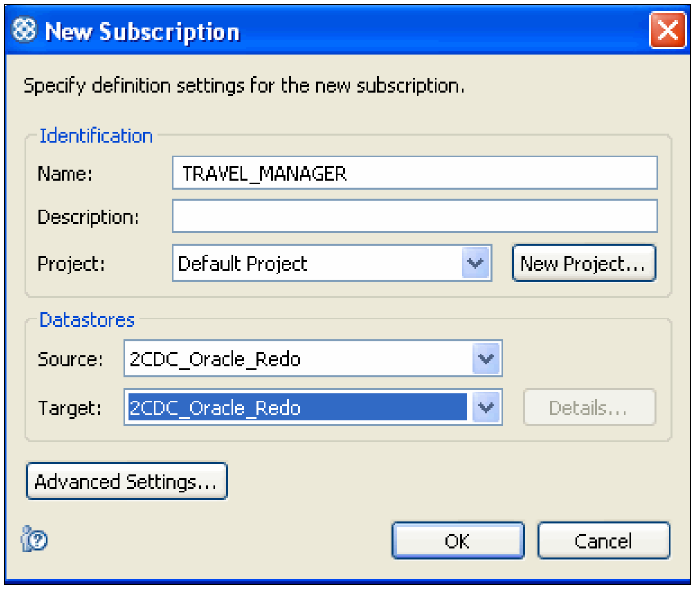
This system is set to UTC-7, and has the following database table definitions:

```
CREATE TABLE "CDCDEMO"."STD_TRAVEL_AUDIT" ("TRAVELID" NUMBER(10)
  NOT NULL, "REQUEST" DATE NOT NULL, "APPROVED" DATE,
  "BOOKED" DATE,
  CONSTRAINT "TRAVELID" PRIMARY KEY("TRAVELID"))
  TABLESPACE "USERS";

CREATE TABLE "CDCDEMO"."STD_TRAVEL_APPROVE" ("TRAVELID"
  NUMBER(10) NOT NULL, "REQUEST" DATE, "APPROVED" DATE, PRIMARY
  KEY("TRAVELID"))
  TABLESPACE "USERS"
```

In this sample, STD_TRAVEL_AUDIT represents the auditors audit application, which contains the Request, Approved and Booked date. The STD_TRAVEL_APPROVE table represents the managers approval system.

Two subscriptions need to be created:

- The TRAVEL_MANAGER (Figure 7-3) replicates the approval date and time to the audit application.


New Subscription

Specify definition settings for the new subscription.

Identification

Name: TRAVEL_MANAGER

Description:

Project: Default Project

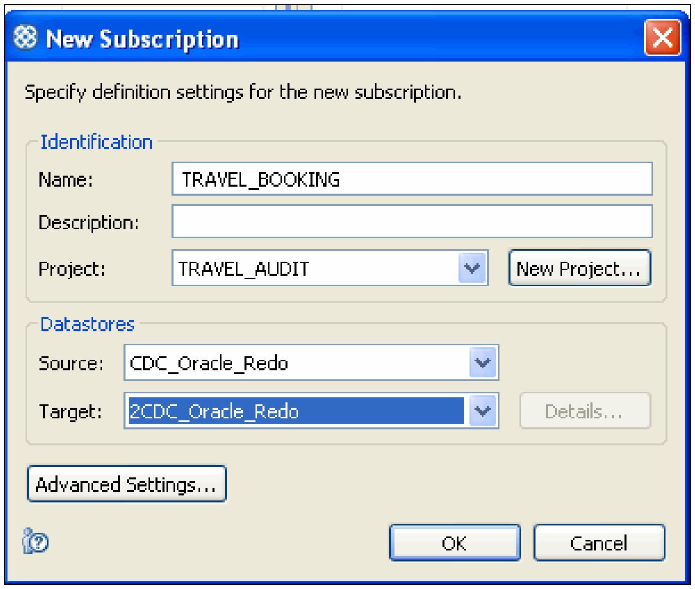
Datastores

Source: 2CDC_Oracle_Redo

Target: 2CDC_Oracle_Redo

Figure 7-3 TRAVEL_MANAGER

- The TRAVEL _BOOKING (Figure 7-4) replicates the booking date and time to the audit application.


New Subscription

Specify definition settings for the new subscription.

Identification

Name:	TRAVEL_BOOKING
Description:	
Project:	TRAVEL_AUDIT

Datastores

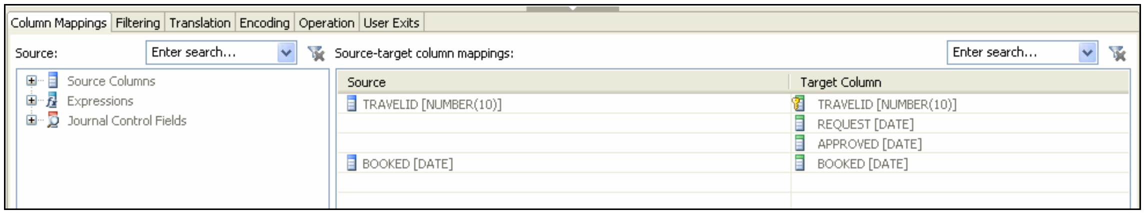
Source:	CDC_Oracle_Redo
Target:	2CDC_Oracle_Redo

Figure 7-4 TRAVEL_BOOKING

The tables are mapped with one-to-one consolidation and the TRAVELID is used as the key.

Wrong mapping without time zone considerations

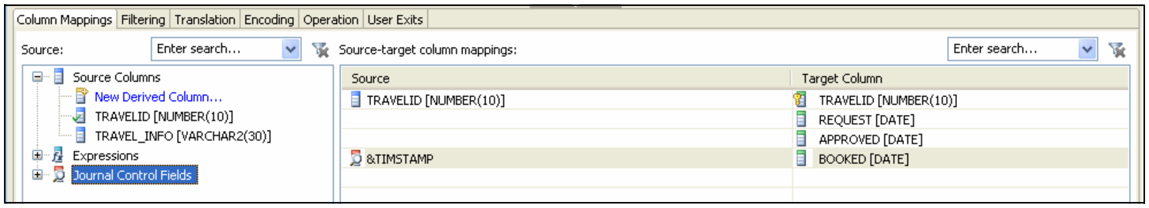
STD_TRAVEL_APPROVE is mapped to STD_TRAVEL_AUDIT (Figure 7-5).



Source	Target Column
TRAVELID [NUMBER(10)]	TRAVELID [NUMBER(10)]
	REQUEST [DATE]
	APPROVED [DATE]
BOOKED [DATE]	BOOKED [DATE]

Figure 7-5 Wrong mapping approval

STD_TRAVEL_BOOKED is mapped to STD_TRAVEL_AUDIT (Figure 7-6). As there is not a booked date in the source table, the journal time stamp is used.



Source	Target Column
TRAVELID [NUMBER(10)]	TRAVELID [NUMBER(10)]
	REQUEST [DATE]
	APPROVED [DATE]
&TIMSTAMP	BOOKED [DATE]

Figure 7-6 Wrong mapping booking

Besides using Consolidation One-to-One, we did not take anything else into consideration and automapped the available fields.

To simulate the application, complete the following steps:

1. An employee files a booking request in the Application Approval system. The application uses the local application time, which is put in the table shown in Figure 7-7.

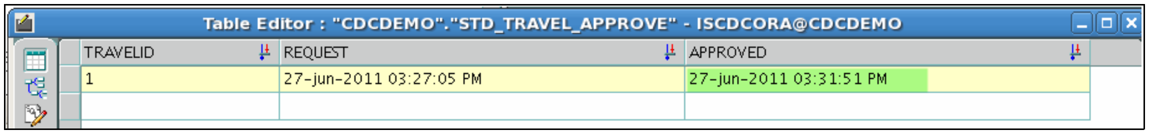


Table Editor : "CDCDEMO"."STD_TRAVEL_APPROVE" - ISCDCORA@CDCDEMO

TRAVELID	REQUEST	APPROVED
1	27-jun-2011 03:27:05 PM	27-jun-2011 03:31:51 PM

Figure 7-7 Approval table

2. The employee decides not to wait and uses the booking application to confirm the booking (Figure 7-8). The local system has been added for information, but is not replicated.

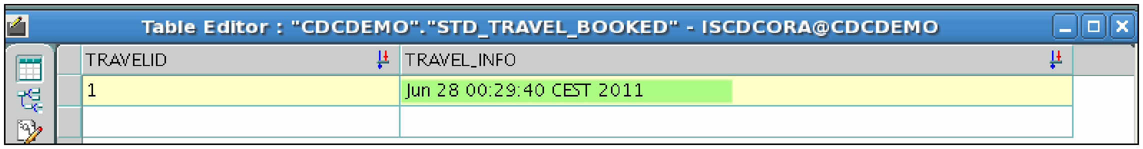


Table Editor : "CDCDEMO"."STD_TRAVEL_BOOKED" - ISCDCORA@CDCDEMO

TRAVELID	TRAVEL_INFO
1	Jun 28 00:29:40 CEST 2011

Figure 7-8 Booking table

3. The manager approves the travel (Figure 7-9).

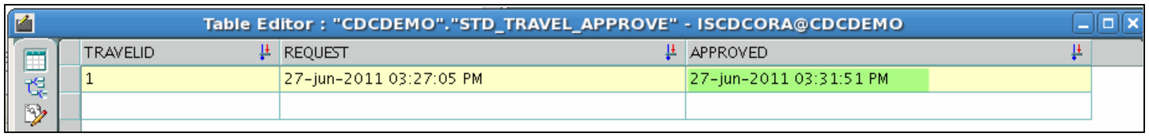
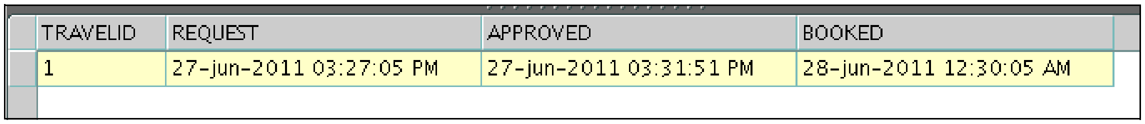


Table Editor : "CDCDEMO"."STD_TRAVEL_APPROVE" - ISCDCORA@CDCDEMO

TRAVELID	REQUEST	APPROVED
1	27-jun-2011 03:27:05 PM	27-jun-2011 03:31:51 PM

Figure 7-9 Manager approval

The resulting travel table opens (Figure 7-10).



TRAVELID	REQUEST	APPROVED	BOOKED
1	27-jun-2011 03:27:05 PM	27-jun-2011 03:31:51 PM	28-jun-2011 12:30:05 AM

Figure 7-10 Result table

Although in the simulation we made sure that we made the booking before the approval data and time was entered in the approval system, the audit table incorrectly reflects that all was booked in the correct sequence.

Date and time fields within the data

When replicating across time zones, it is important to understand how the different time zone's are understood by InfoSphere CDC.

Complete the following steps:

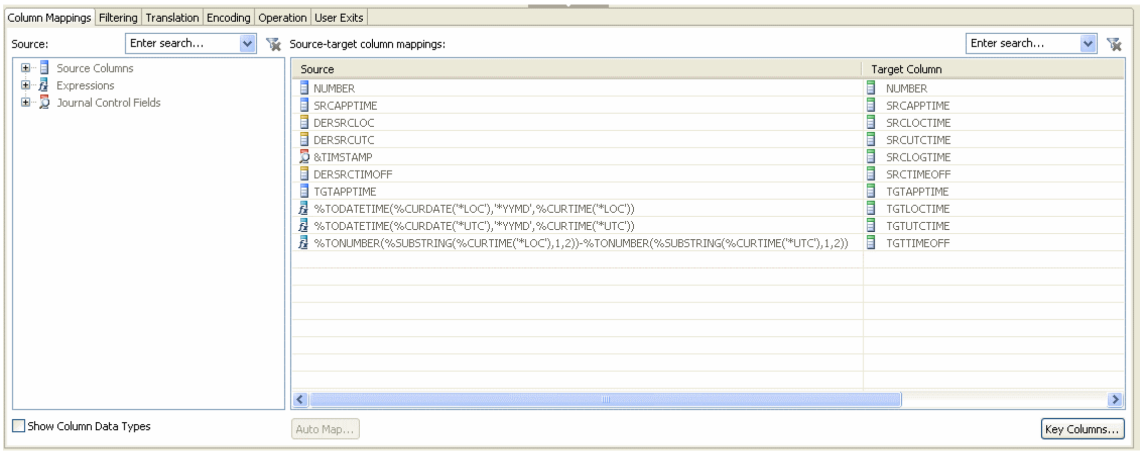
1. On the source database, create a table by running the following command:

```
CREATE TABLE "CDCDEMO"."REP_TIME" ("NUMBER" NUMBER(10) NOT NULL,
    "SRCAPPTIME" DATE, "SRCLOCTIME" DATE, "SRCUTCTIME" DATE,
    "SRCLOGTIME" DATE, "TGTAPPTIME" DATE, "TGTLOCTIME" DATE,
    "TGTUTCTIME" DATE)
    TABLESPACE "USERS";
```

2. On the target database, create a table by running the following command:

```
CREATE TABLE "CDCDEMO"."TimeOverView" ("NUMBER" NUMBER(10),
    "SRCAPPTIME" DATE, "SRCLOCTIME" DATE, "SRCUTCTIME" DATE,
    "SRCLOGTIME" "TIMESTAMP(6)", "SRCTIMEOFF" NUMBER(10),
    "TGTAPPTIME" DATE, "TGTLOCTIME" DATE, "TGTUTCTIME" DATE,
    "TGTTIMEOFF" NUMBER(10))
    TABLESPACE "USERS";
```

3. Create a subscription and set up an audit replication with mappings (Figure 7-11).



Source-target column mappings:

Source	Target Column
NUMBER	NUMBER
SRCAPPTIME	SRCAPPTIME
DERSRCLOC	SRCLOCTIME
DERSRCUTC	SRCUTCTIME
&TIMSTAMP	SRCLOGTIME
DERSRCTIMOFF	SRCTIMEOFF
TGTAPPTIME	TGTAPPTIME
%TODATETIME(%CURDATE('*LOC'),'*YYMD',%CURTIME('*LOC'))	TGTLOCTIME
%TODATETIME(%CURDATE('*UTC'),'*YYMD',%CURTIME('*UTC'))	TGTUTCTIME
%TONUMBER(%SUBSTRING(%CURTIME('*LOC'),1,2))-%TONUMBER(%SUBSTRING(%CURTIME('*UTC'),1,2))	TGTTIMEOFF

Figure 7-11 TimeOverViewMappings

The derived column, DERSRCLOC, contains the following expression:

```
%TODATETIME(%CURDATE('*LOC'),'*YYMD',%CURTIME('*LOC'))
```

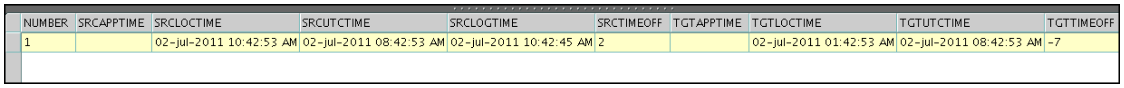
The derived column, DERSRCUTC, contains the following expression:

```
%TODATETIME(%CURDATE('*UTC'),'*YYMD',%CURTIME('*UTC'))
```

The derived column, DERSRCTIMOFF, contains the following expression:

```
%TONUMBER(%SUBSTRING(%CURTIME('*LOC'),1,2))-%TONUMBER(%SUBSTRING(%CURTI
ME('*UTC'),1,2))
```

When you start the replication and add a record in the source table, you have a target table record (Figure 7-12).



NUMBER	SRCAPPTIME	SRCLOCTIME	SRCUTCTIME	SRCLOGTIME	SRCTIMEOFF	TGTAPPTIME	TGTLOCTIME	TGTUTCTIME	TGTTIMEOFF
1		02-jul-2011 10:42:53 AM	02-jul-2011 08:42:53 AM	02-jul-2011 10:42:45 AM	2		02-jul-2011 01:42:53 AM	02-jul-2011 08:42:53 AM	-7

Figure 7-12 TimeOverViewTargetRecord

When the user, or user application, inputs a time stamp date or time, the time defined by the application or user is used. This time is either the system time or application time.

For InfoSphere CDC, replicate this time as is, and if you need to centralize this time, you need to convert it. As only the system where the data is created is aware of the time zone, any conversions needed should be done on that system.

If possible, you could modify the data model so the data and time field can hold the time zone where the data is created. You could also pass the date and time by using one of the following techniques:

- Delivering a centralized UTC time stamp
- Delivering local time overview table

Time zone considerations for monitoring

InfoSphere CDC uses local time for events, so when using the event viewer in the Management Console, both source and target events are shown with their local time.

7.1.2 Encoding conversions

In most circumstances, InfoSphere CDC handles code page conversions automatically. When you want to override the automatic code page conversions within InfoSphere CDC, you need to determine the code pages to transform from and to.

How InfoSphere handles encoding

InfoSphere CDC dynamically determines the code page from the source and target during the startup of the subscription. InfoSphere CDC detects the data encoding based either on the column encoding or database encoding and assigns the detected encoding to each column detected in the database. The encoding is based on what the database knows about the encoding of the data found in a specific column. Before assigning the encoding to a column, InfoSphere CDC normalizes the encoding knowledge found to an Internet Assigned Numbers Authority (IANA) encoding name. For more information about the IANA, go to the following address:

http://www.iana.org/assignments/character-sets

Every database has its specific knowledge about data encoding. That knowledge is represented by either a number (code page, Coded Character Set Identifier (CCSID), or Oracle charset ID) or a string (Oracle charset name or Sybase encoding name). Some databases can have a different encoding for each column, and some of them might have a database encoding for non-Unicode columns and then the Unicode national encoding for columns supporting that encoding.

When a table is added to the catalog, the column code page is requested from the database and stored in the internal metadata. InfoSphere CDC always keeps the encoding at the column level and uses this encoding during the replication.

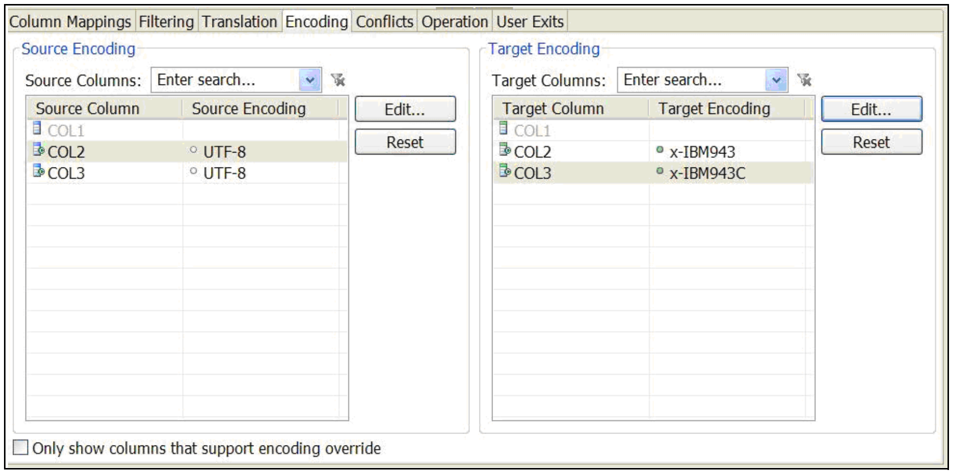
For mainframe, any necessary encoding is done using the Unicode Conversion Services (UCS), which must be enabled on the system.

During startup of the subscription, the source and target negotiate which side does the encoding conversion for each column and the encoding information is put into memory.

Overriding code page conversions

When you have applications that write data into character columns with a different encoding from what has been defined in the database, InfoSphere CDC provides the flexibility to override the column encoding to specify the encoding of the actual data. Besides specifying the encoding conversion for character data, you can override the encoding to specify that no conversion should be done (binary data) or specify the conversion for binary columns, making InfoSphere CDC treat them as character data.

To override the automatic code page conversion, you need to go to the Encoding window of the table mapping. All character type data is shown there. You can map source column data to different column encodings. For example, in Japanese environments with different types of the same encoding (such as IBM-943 and IBM-943C), this capability can be useful. An example is shown in Figure 7-13.



Source Column	Source Encoding
COL1	
COL2	UTF-8
COL3	UTF-8

Target Column	Target Encoding
COL1	
COL2	x-IBM943
COL3	x-IBM943C

Figure 7-13 Overriding code page

You can also override the encoding to specify that it is a binary column or you can specify encodings for binary columns, making them character columns. InfoSphere CDC then treats the source column as through it were a binary or character column from the perspective of what target columns it can be mapped to, and this setting changes the functions in which InfoSphere CDC can be used.

7.2 Firewall configurations

An InfoSphere CDC landscape can consist of several servers that must interconnect to be able to perform configuration and replication activities. If the servers that host any of the InfoSphere components (Management Console, Access Server, or Replication instance) are in different networks or network segments, firewalls are usually in place to protect these servers from unauthorized access.

Firewalls contribute to almost all InfoSphere CDC connectivity issues. The configuration of these firewalls must be considered when designing your replication landscape.

7.2.1 How InfoSphere CDC uses TCP/IP

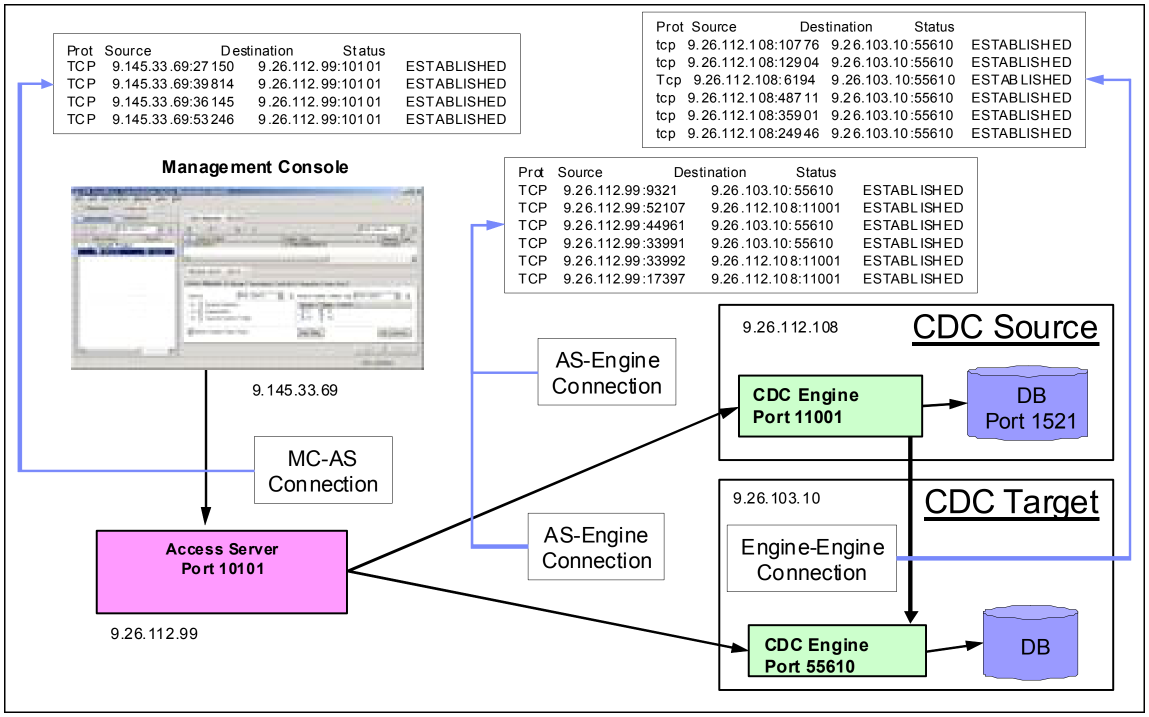
Before elaborating about how firewalls affect your replication configuration, it is important to understand how InfoSphere CDC uses the TCP/IP application protocol. As with all TCP-based applications, InfoSphere CDC uses a client-server model. A server is an application that offers a service to Internet users. In addition, a client is a requester of a service and aa server is a program that receives a request, performs the required service, and sends back the results in a reply.

Each process (client or server) that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports. This port is used by the operating system to determine to which application program (process) it must deliver incoming messages. After it is identified to the TCP/IP protocol, a socket is registered. The socket is uniquely identified by an address that consists of the transport protocol (TCP), local network address, and the local port number, for example TCP, 172.16.5.14:10901. The process of registering a socket is also called *binding*.

When a client socket communicates with a server socket, the connection is identified by an association that completely specifies the two processes that make up the connection using the following attributes: transport protocol (TCP), local network address:local port number, foreign / destination and network address:foreign / destination port number. An example is tcp, 172.16.5.10:47112, 172.16.5.14:10901.

A server can usually deal with multiple requests and multiple requesting clients at the same time. Most servers wait for requests at a well-known port so that clients know which port (and in turn, which application) they must direct their requests to. The client typically uses an arbitrary port called an ephemeral port for its communication. A connection that is established between a client and a server is full duplex, meaning that data streams can flow in both directions concurrently.

InfoSphere CDC typically requires multiple connections from a client to a server. Each connection is identified by its own association where the destination port number is well-known (fixed) and the source port number is variable. For example, if a subscription is running to replicate data from an Oracle to a DB2 database, InfoSphere CDC establishes two connections, one to replicate the change data and the other one to send / receive control information. If you include the Management Console and Access Server in the picture, the connections that are active at a given point in time could look as shown in Figure 7-14.



```
Prot  Source               Destination         Status
TCP   9.145.33.69:27150    9.26.112.99:10101   ESTABLISHED
TCP   9.145.33.69:39814    9.26.112.99:10101   ESTABLISHED
TCP   9.145.33.69:36145    9.26.112.99:10101   ESTABLISHED
TCP   9.145.33.69:53246    9.26.112.99:10101   ESTABLISHED
```

```
Prot  Source               Destination         Status
tcp   9.26.112.108:10776   9.26.103.10:55610   ESTABLISHED
tcp   9.26.112.108:12904   9.26.103.10:55610   ESTABLISHED
Tcp   9.26.112.108:6194    9.26.103.10:55610   ESTABLISHED
tcp   9.26.112.108:48711   9.26.103.10:55610   ESTABLISHED
tcp   9.26.112.108:35901   9.26.103.10:55610   ESTABLISHED
tcp   9.26.112.108:24946   9.26.103.10:55610   ESTABLISHED
```

```
Prot  Source               Destination          Status
TCP   9.26.112.99:9321     9.26.103.10:55610    ESTABLISHED
TCP   9.26.112.99:52107    9.26.112.108:11001   ESTABLISHED
TCP   9.26.112.99:44961    9.26.103.10:55610    ESTABLISHED
TCP   9.26.112.99:33991    9.26.103.10:55610    ESTABLISHED
TCP   9.26.112.99:33992    9.26.112.108:11001   ESTABLISHED
TCP   9.26.112.99:17397    9.26.112.108:11001   ESTABLISHED
```

Figure 7-14 InfoSphere CDC connections

You can see the connections and their associations and the fact that multiple connections are activated. The InfoSphere CDC Management Console client process connects to the Access Server server process on port 10101. It establishes at least two connections for configuration and monitoring activities and an additional connection for each connected data store. In this example, the source port for each connection has been requested to and assigned by TCP/IP. This port is dynamically assigned based on availability and does not follow a specific sequence.

The Access Server routes all Management Console requests to the respective CDC engines. In the context of setting up firewall rules, it is important to know that there are no direct connections between the Management Console and the CDC engines.

7.2.2 Firewalls

Typically, firewalls protect a network from the outside (untrusted) environment (Internet and intranet). Firewalls can consist of one or more functional components, the most important CDC-supported component being the packet-filtering router.

Packet-filtering routers that can forward or discard packets according to filtering rules. When a packet (network data and a header which includes address and routing information) arrives at the packet-filtering router, the router extracts information from the packet header and decides, according to the filter rules, whether the packet passes through or is discarded. The following information can, and other information, be extracted from the packet header:

- Source IP address
- Destination IP address
- TCP/UDP source port
- TCP/UDP target port

Because most server applications run on a specific server and use well-known TCP/UDP port numbers, it is possible to allow or deny services by using the related destination IP address and target port in the filter. Most firewalls are configured this way: They shield servers from unauthorized inbound connection attempts coming from a client.

In a number of cases, firewalls are also configured to only forward packets coming from a certain network, indicated by the source IP address.

Rarely, firewalls are configured to forward packets coming from a certain source port or range of source ports. Configuring a firewall in this manner requires client applications to always bind to a certain port or a port within a certain range.

7.2.3 InfoSphere CDC in a firewalled network environment

InfoSphere CDC can operate in most firewalled network environments, including ones where you have packet filtering based on a source IP address and source ports. However, when firewalls are configured with source port filtering, you must carefully compare your intended InfoSphere CDC configuration against the restrictions put in place by the firewall. Failure to do this comparison might cause replication to not start successfully and it might be difficult to determine the cause. Generally, ensure that when a firewall exists anywhere on the route between a client and server that its configuration must be known and validated against the InfoSphere CDC configuration.

Looking at the different components in Figure 7-15, the following connections must be reviewed when being used by firewalls:

- Management Console connections to the Access Server
- Access Server connections to the source and target CDC instances
- Source CDC instances to target CDC instances (subscriptions)
- Replication instances to database instances (remote log reading and remote apply)

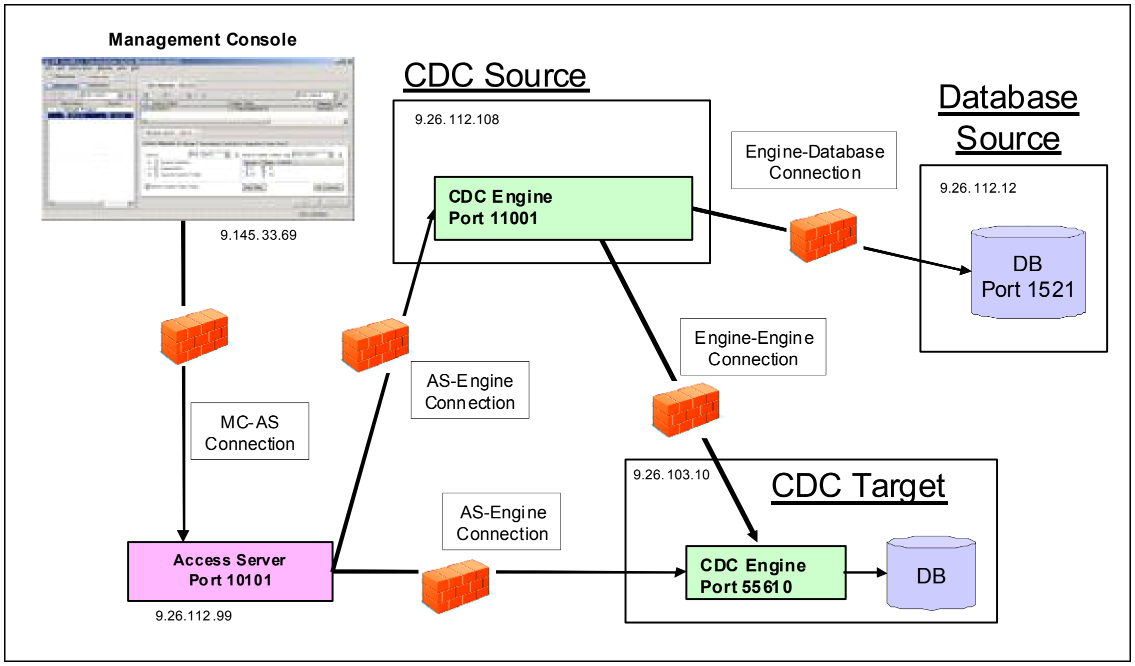


Figure 7-15 InfoSphere CDC firewalled network

All connections between the InfoSphere CDC Management Console and the Access Server are initiated from the workstation running Management Console. For every connection, the target port is the port on which the Access Server listens. Connections are established when the Management Console connects to the Access Server (after the user and password are entered) and when data stores are connected. Traffic on all connections is mostly bidirectional.

Similarly, all connections between the InfoSphere CDC Access Server and the CDC instances are initiated from the server running the Access Server. Connections are established when a user connects to a data store from the Management Console and new connections are established if monitoring functions are used from within the Management Console. For every connection, the target port is the port that has been configured for the CDC instance. Traffic on these connections is mostly bidirectional.

Connections between CDC source and target instances are initiated from the server running the CDC source instance. These connections are established when subscriptions are started.

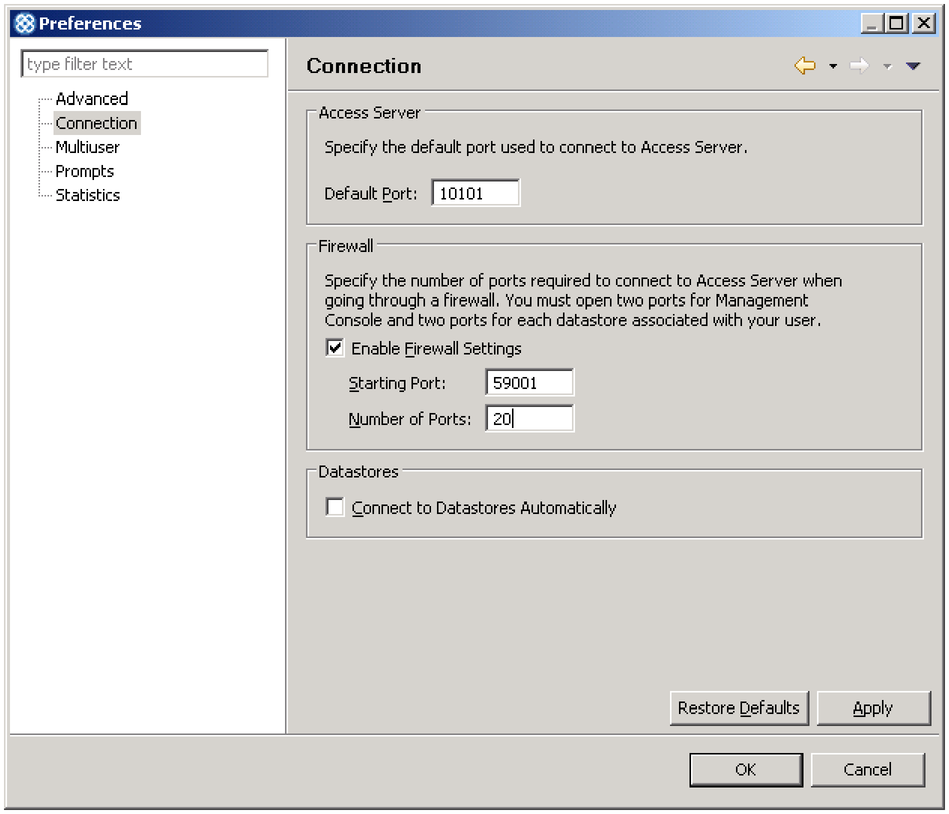
Connections between CDC instances (source and target) and the database are initiated from the server running the CDC instance in question. These connections are established when the CDC instance is started and when subscriptions are active.

7.2.4 Configuring source port restrictions

If your firewall also filters based on source port, you can and should configure a starting source port for the InfoSphere CDC component in question, whether it is Management Console, Access Server, or Subscription. When specifying a source port, the component in question establishes the first connection using this port. Any subsequent connection uses the previously issued port number + 1.

Management Console to Access Server

The starting source port and number of ports Management Console can use is configured in the Connection preferences of your Management Console client (Figure 7-16). When connecting to the Access Server, which listens on port 10101, the first connection has the following association: tcp, source_ip:59001, target_ip:10101. The next source port is 59002, and so on.



Connection

Access Server

Specify the default port used to connect to Access Server.

Default Port: 10101

Firewall

Specify the number of ports required to connect to Access Server when going through a firewall. You must open two ports for Management Console and two ports for each datastore associated with your user.

[✓] Enable Firewall Settings

Starting Port: 59001

Number of Ports: 20

Datastores

[ ] Connect to Datastores Automatically

Figure 7-16 Connection preferences

When planning for the firewall configuration, the total number of connections that will be established can be calculated by the following formula:

2 + (number of connected data stores * 2)

For example, with starting port 59001 and 20 ports, the source ports to be opened in the firewall would range from 59001 through 59020. The Management Console is able to connect to the Access Server and have a maximum of 19 data stores connected simultaneously.

Access Server to CDC instances

The starting source port and number of ports Access Server can use is configured in the `dmaccessserver.vmargs` file that is located in the Access Server `config` directory. You can edit the text file and restart the Access Server, after which the changes take effect. The format of the arguments is:

```
jar lib/server.jar local_port:<first_port>
local_port_count:<number_available_ports> <Access_Server_Listener_Port>
```

The number of source ports to open when defining Access Server source ports depends on the number of Management Consoles that are routed through to the different InfoSphere CDC instances. This number can be calculated using the following formula:

2 * (number of management consoles + (number of management consoles * number of connected data stores) + number of data stores)

CDC source to target instances (subscriptions)

The starting source port must be configured for each subscription in the Advanced settings. Each subscription potentially establishes up to six connections to the target CDC instance. This situation depends on the type of CDC engine that is running on the source and target.

The number of source ports to open in your firewall configuration can be calculated using the following formula:

6 * number of subscriptions

CDC instance to database

The source port for connections established from the CDC instance to the database cannot be configured. As a best practice, keep the CDC engine close to the database server and do not use firewall rules with source port restrictions.

7.2.5 Troubleshooting CDC connection issues

This section describes basic troubleshooting steps to determine whether a firewall is blocking traffic between CDC components.

The first step in analyzing connectivity issues is determining whether the client can find the server by its configured IP address or host name. Often, connectivity issues are caused by a mismatch on the host tables of the server that runs Access Server and the CDC source server.

For example, the host name for the CDC target server, tgtsvr, might be resolved to it real address of 172.16.5.1 in your DNS. The server running the Access Server might have a host table that causes the tgtsvr host name to resolve to the address 10.1.1.5. This configuration causes the Access Server to be unable to reach the target server.

Another example is when the tgtsvr host name is resolved to address 10.1.1.5. The Access Server uses the DNS and resolves to address 171.16.5.1 for the target CDC server. You can map source and target tables up to the point where the subscription tables are described and the source CDC engine attempts to establish a connection to the tgtsvr host name. The description fails because of the invalid 10.1.1.5 address that is listed in the source server's host table.

If the environment allows this action and firewalls do not block ICMP requests, you can run **ping** from each of the servers running a CDC component to ensure that host names are resolved to the expected IP addresses. Alternatively, you can configure the data stores in your Access Server by their IP addresses.

If there are still connectivity issues after having confirmed the IP addresses are correct, a firewall might block traffic coming from a certain server. Each of the CDC components, Access Server, and CDC engines, respond to basic connection requests coming from a telnet client. Simple tests can be run to determine if the route between any of the components blocked by a firewall or different issue.

The command to test the activity of a server port is:

```
telnet <ip address of server> <target port>
```

If a CDC component is active on the specified IP address and listening on the target port, the connection is established successfully and opens a telnet session. You can return to the telnet command line by pressing ^] (Ctrl+]) and then run **quit** to exit.

When testing connectivity, it is important to remember which CDC component establishes the connection. If a connectivity issue occurs when tables are being mapped, you need to determine whether the source CDC engine can connect to the target CDC engine. This situation means that the telnet test must be done on the CDC source server. Should you be able to connect to the Access Server but not be able to connect to a certain data store, run the test on the server hosting the Access Server.

7.3 Log retention

As described in Chapter 1, “Introduction and overview” on page 1, InfoSphere CDC depends on the logging mechanism for its push capabilities. This section describes some general guidelines for log retention in combination with InfoSphere CDC. If you require some specific platform or database guidelines, see 7.3.2, “Log retention platform-specific guidelines” on page 159.

7.3.1 Log retention general guidelines

Each database has an initial fast logging mechanism, typically referred to as an online log. The DBA has tuned the online log by providing fast storage. Having highly performed storage means having higher costs, so the storage space reserved for the online log is limited. When the online log has used all predefined settings, it is moved to what is referred to as archive logging. These predefined settings are mostly dependent on available fast storage and the number of transactions made by the database or different databases using the same storage space.

By implementing InfoSphere CDC, you can read from the online log during normal operations. If there are frequent cases where InfoSphere CDC has a guaranteed latency (for example, with back-up procedures and in environments where InfoSphere CDC is running periodically), it is necessary to provide storage for the archive log that also has an adequate read speed. This action ensures that performance of the InfoSphere CDC scraping process can stay within limits to cope with the replication needs.

Long open transactions in the source database can cause InfoSphere CDC to require a log much older than necessary. For example, a user who logged in to the database and issued a command without having issued a subsequent commit or rollback can cause this situation. If the user does not exit their session, a transaction can be open for hours, days, or even weeks. InfoSphere CDC must retain information about all open transactions in the database in the staging store on Linux, UNIX, and Windows or the log cache on z/OS. As the transaction ages, it can be removed from this storage as room is needed for current transactions. Ending replication for a subscription while this situation exists causes InfoSphere CDC to require the log where the long open transaction began when it is restarted. A standard practice when running InfoSphere CDC is to ensure that there is a procedure in place to end open transactions in the database after a certain period of inactivity. At minimum, a policy should be in place requiring users to log out at the end of the work day.

Make sure that all log data stays available until InfoSphere CDC has been able to replicate the required data to the defined destinations.

7.3.2 Log retention platform-specific guidelines

As described in 7.3.1, “Log retention general guidelines” on page 158, usage of the database log is key to replicating data using InfoSphere CDC in near real-time mode. Some client environments require regular and frequent cleanup or archiving of the logs to meet their business / disk space requirements. If database logs are removed before InfoSphere CDC has completed processing the transactions within them, mirroring fails. The following sections outline considerations for log retention / management in a replication environment.

Log retention for DB2 on Linux, UNIX, and Windows

For this type of log retention, InfoSphere CDC requires the following items:

- Log retention must be enabled for any DB2 for Linux, UNIX, and Windows database to be replicated.
- A database backup is required before an initial connection is established from InfoSphere CDC.
- Physical access to the DB2 online logs and archive logs on disk is required.
- Retention of the DB2 online logs and archive logs for as long as InfoSphere CDC is shut down or latent.

Log retention for DB2 for System i

For this type of log retention, InfoSphere CDC requires the following items:

- InfoSphere CDC replicates data from physical files. The files must be journaled with journal images on both.
- When tables are not journaled when selected within InfoSphere CDC for mirroring, InfoSphere CDC activates the journaling for this table in the default journal. The standard default journal for InfoSphere CDC is DMIRROR/DMCJRN, but this journal can be (and should) be modified to a client-specific journal.
- Retention of the DB2 online logs and archive logs for as long as InfoSphere CDC is shut down or latent.

Log retention for z/OS

For this type of log retention, InfoSphere CDC requires the following items:

- Retention of archive logs for as long as any of the subscriptions are inactive or latent.
- Fast access to archive logs if they are stored on tape. However, the preferred configuration is archive logs on disk.
- Where a log cache is configured, archive logs are required only when a subscription is inactive or latent longer than the oldest data retained in the cache.

Here are some guidelines about InfoSphere CDC on z/OS:

- Long open transactions (Units of Recovery (URs))

  Messages are issued to the event log when a UR has been open in the database for a long period (60 minutes by default, which can be change by using the **`OPENCOMMITWARNAGE`** parameter). If the subscription is shut down while the UR remains open, InfoSphere CDC needs to reread the logs from the beginning of that UR. You should ensure that there are no long open transactions before ending a subscription. These messages can optionally be issued to the system console by using the **`CONSOLEMSGS`** parameter.

- Log cache

  With a log cache configured, requirements to reread data from the DB2 logs are reduced. The log cache is a circular buffer and retains information about all URs and all data for tables with **`DATA CAPTURE CHANGES`** configured. The cache reader keeps up with the head of the DB2 log and removes the oldest data to make room for new URs as the DB2 log is extended.

  Suppose a subscription is inactive and requires data older than the data stored in the log cache, or a subscription experiences such extreme latency that it is removed from the cache. In that case, it requests the data from the DB2 instrumentation facility interface (IFI) directly and returns to reading from the log cache when the data it needs is available there.

Log retention for Oracle platforms

InfoSphere CDC requires access to both the redo and the archive logs, which need to be configured for supplemental logging.

Retention of the logging is required for both redo and archive logging for as long as InfoSphere CDC is shut down or latent.

Log retention for Microsoft SQL Server

For this type of log retention, InfoSphere CDC requires the following items:

- Physical access to the online transaction logs on disk. Multiple physical log files are supported.
- Physical access to the transaction log backups on disk. Transaction log backups may not be compressed or encrypted or otherwise transformed from the format output by SQL Server.
  - InfoSphere CDC can only read files in the original unaltered format generated by the built-in SQL backup utility. Log files may not be moved after they have been saved
  - When using third-party back-up tools, you must make sure that they do not compress or move the transaction logs from the original location.
- Retention of the logging is required for both redo and archive logging for as long as InfoSphere CDC is shut down or latent.

Log retention for Sybase

For this type of log retention, InfoSphere CDC requires the following items:

- Physical access to the online logs on disk.
- Physical access to the archive logs on disk. All archive logs must be located in one directory and InfoSphere CDC must have read permission for this directory.
- Retention of the online logs and archive logs for as long as InfoSphere CDC is shut down or latent.

Here are some guidelines:

- The online logs and archive logs can be on different disks.
- Ensure the truncate log option on checkpoint is disabled. If the truncate log option is enabled, the database log is truncated automatically without backup every time a database checkpoint is performed
- Never run truncate_only operations. Running a memory dump transaction with the truncate_only option deletes inactive transactions from the log without creating a backup.
- Use only data or log segments, not a combination. If a database has mixed segments, log backup is not allowed and only full database backups can be performed.

7.4 Remote processing capabilities

Although a general preferred practice is to install InfoSphere CDC on the server where the source or target database server is located, in some environments business reasons may dictate that it must be installed on a separate or remote server. This requirement might exist because of a lack of resources on the source / target database server sufficient to run InfoSphere CDC, corporate security standards, or other vendor application environments.

This section outlines implementation methods that can be used to implement InfoSphere CDC in a remote capture or apply configuration.

Some options for these configurations include:

- InfoSphere CDC is installed for both the source and target on the target database server and configured to remotely scrape the logs of the source database system (see 7.4.1, "Remote source" on page 163 for more details).
- InfoSphere CDC supports log shipping for Oracle databases. This functionality has been designed to cater to environments where log retention policies conflict with the use of replication solutions against those logs or when access to the original database logs is not possible or is forbidden (see 7.4.1, "Remote source" on page 163 for more details).
- InfoSphere CDC is installed for both the source and target on the source database server and configured to remotely apply database changes to the target database (see 7.4.2, "Remote target" on page 164 for more details).
- InfoSphere CDC is installed on an intermediate server for either remote source scrape or target apply (see 7.4.3, "Remote source and target" on page 165 for more details).
- This section also describes InfoSphere CDC support for log shipping, which is available only on the Oracle environments (see 7.4.4, "Log shipping" on page 166 for more details.

When InfoSphere CDC is installed on a different server from the database server it replicates changes from or to, there is always a requirement that a client for the accessed database be installed on the CDC server.

7.4.1 Remote source

When referring to a remote source in the context of InfoSphere CDC, the CDC engine is on a different server from the database that is generating the logs (Figure 7-17).

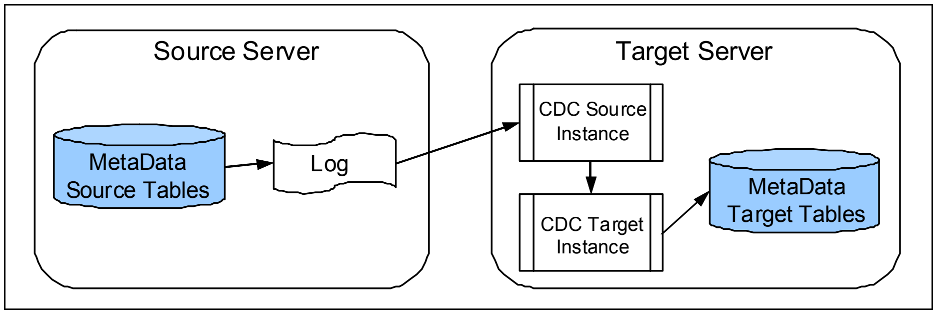


Figure 7-17 InfoSphere CDC reading logs from a remote server

Remote log reading is supported for DB2 on Linux, UNIX, and Windows, DB2 on System i, Sybase, and Oracle databases, and the mechanics for setting remote log reading are different for all databases. The following sections provide additional information regarding DB2 and Oracle as examples.

DB2 databases

When processing the DB2 logs from a different server, the DB2 client must be installed on the server that hosts InfoSphere CDC.

Before creating the InfoSphere CDC instance, the DB2 source database from which the log entries are read must be identified to the DB2 client. This task can be accomplished by completing the following steps:

1. Catalog the remote database node on the DB2 client (CDC) server.
2. Catalog the remote database on the DB2 client (CDC) server.

Specific commands and more details can be found at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.cdcfordb2luw.doc/concepts/configuringremotelogreading.html

Oracle databases

When remotely reading Oracle logs, the server that InfoSphere CDC is running on must be the same brand and chipset (for example, reading Oracle logs on IBM System p® from an HP server is not supported). Also, the endian of the database and CDC server must be the same (processing database logs generated on a Linux server from a System p server is not supported).

The following items must be considered:

- The archive and redo log files are stored in a shareable file system (SAN or NFS mount). If available, a SAN is preferable over NFS for performance impact and throughput reasons; when log volumes are high, NFS mounts are not practical. It is not possible for redo / archive files stored on RAW devices.
- If the first set of online and archive logs cannot be shared, consider having Oracle multiplex the archive logs (and, optionally, the online logs too).
- The directories that hold the archive and redo log files must be accessible from the server on which InfoSphere CDC runs. The InfoSphere CDC user must have read permissions on archive and redo logs.

Some specifics to consider when setting up the remote source configuration are:

- Create the dba group on the InfoSphere CDC server with the same GID as the source database server and make the InfoSphere CDC UNIX account part of this group. This configuration eliminates having to open too many permissions for others.
- Create a mount point on the InfoSphere CDC server that resembles the directory referenced in the v$archived_log and v$logfile views.
  - For example, the archive logs are stored in `/oradata/cdcdemo/archive`, and if possible, create the following mount point on the InfoSphere CDC server:

    ```
    /oradata/cdcdemo/archive
    ```
  - When InfoSphere CDC determines the name of the log it must read, it can go to the directory without having to replace the path component.
- If the mount point cannot be created, the following parameters can be set to override the directories from which the log files are read:
  - `mirror_archive_log_directory` (location of the archive logs).
  - `mirror_online_log_directory` (location of the online logs).

  If remote log reading must be performed in a RAC configuration and the original directories cannot be mounted the same way, you must ensure that archive logs of all RAC nodes are in the same directory.

7.4.2 Remote target

In a remote target environment, InfoSphere CDC must apply the transactions to a database that is on a different server. On the InfoSphere CDC server, make sure that you are able to connect to the remote target database as though it was on the same server.

Database logs are read locally on the source server and changes are sent to the target CDC instance. This instance then applies them remotely to the target server (using the database client) (Figure 7-18).

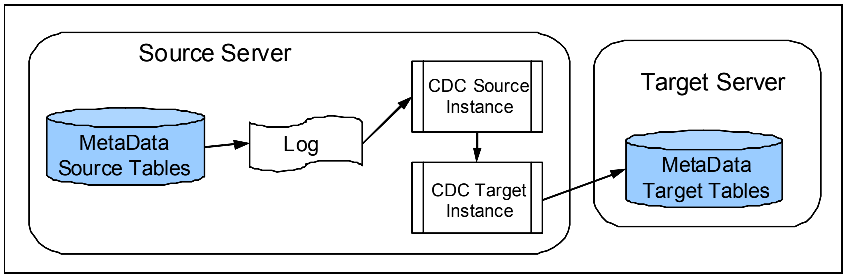


Figure 7-18 Remote target

Remote apply is supported for DB2 on Linux, UNIX, and WIndows, Sybase, SQL Server, and Oracle, and requires that the database client for the applicable database is installed on the server that hosts InfoSphere CDC. Additionally, the database server that is targeted must be identified in the configuration of the client so that it can be reached by CDC. For DB2, the database server and the remote database must be cataloged; for Oracle, the remote database should exist as a TNS entry in the SQL*Net configuration of the client.

7.4.3 Remote source and target

In this scenario, InfoSphere CDC is installed on a dedicated server that hosts the source and target engine. Besides the CDC engines, the source and target databases must have their database client installed on this server.

Database logs are read remotely from the source server and sent to the target CDC instance. This instance then applies them remotely to the target server (using the database client) (Figure 7-19).

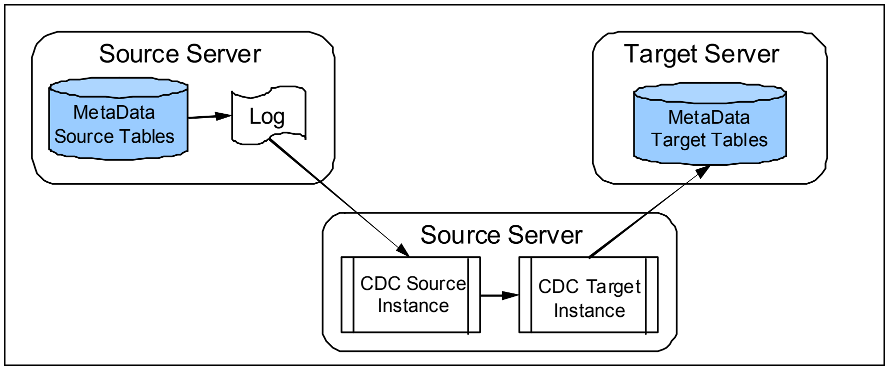


Figure 7-19 Remote source and target

7.4.4 Log shipping

Log shipping is available for Oracle environments only.

In log shipping, InfoSphere CDC uses the database logs coming from another system (Figure 7-20).

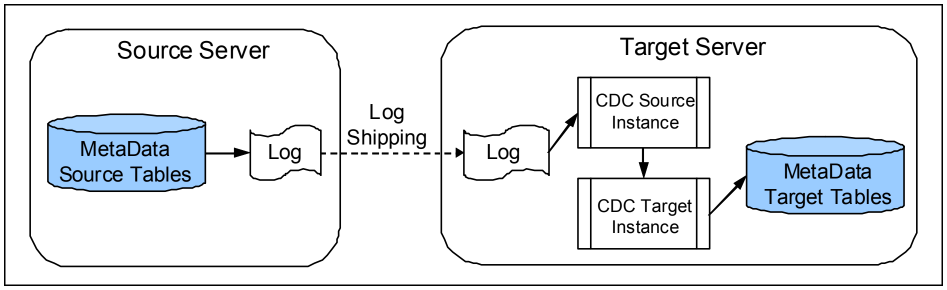


Figure 7-20 InfoSphere CDC log shipping

InfoSphere CDC does not take care of the shipping of the logs. It is the responsibility of the DBA to make sure that the logs are sent to the defined system. Log shipping is only possible by using archive logs. Therefore, for a near real-time scenario, this option might not be the best solution.

Here is what you need to use log shipping in an Oracle environment. The InfoSphere CDC system parameters should be set as follows:

- `Oracle_log_shipping=true`
- `Oracle_archive_logs_only=true`

When changing these system parameters, the InfoSphere CDC instance must be restarted for them to take effect.

The shipping of the archive logs can be performed by either the Oracle Data Guard log shipping feature or by a client-created log shipping script.

When you use the Oracle Data Guard log shipping feature, InfoSphere CDC can safely pick up the archive log files stored in the `oracle_archive_dir` directory while Data Guard ensures the consistency of the files. The following considerations must be met:

- Archive logs are shipped to a secondary destination by Oracle Data Guard.
- InfoSphere CDC must be configured to read the logs from the secondary destination.
- The extra data store system parameters to be set are:
  - `oracle_using_log_transport_services=true`
  - `oracle_archive_destination_id=<destination_id_from_oracle>`
  - `oracle_archive_dir=<directory_holding_archive_logs>`

  When changing these system parameters, the InfoSphere CDC instance must be restarted for them to take effect.

When creating your own script to perform the log shipping, you must include the following steps:

1. Select new archive files from `v$archived_log` with `status="A"` (marked available by Oracle). Ensure that you do not ship archive logs that do not have the “A” status, as this situation exposes the risk of incomplete shipment and losing transactions.
2. Run a checksum of each archive file before sending the file to the target.
3. Transfer the archive files to the InfoSphere CDC server.
4. Run a checksum of each archive file on the InfoSphere CDC server and compare with the checksum of source server.
5. Notify InfoSphere CDC for each archive file using the **`dmarchivelogavailable`** command.

The following system parameters must be set when using custom scripting for archive log shipping:

- `oracle_archive_dir= <directory_holding_archive_logs>`

  or

- `oracle_log_path_userexit=<class_name>`

  Class name must be in the `lib` directory under the InfoSphere CDC home and must implement ArchiveLogPathUserExit and hold method ArchiveLogPath.

Here are some considerations for the log shipping scripts:

- If possible, run the entire log shipping from the source server.
- All steps, from picking up the archive log and to making the archive log available for InfoSphere CDC, can then be handled by one single script. This situation allows for full control over comparison of checksums and availability of the archive logs.
- If sending the archive logs and processing on the target cannot be done in a singe script, consider the following items:
  - Send the archive log under a different name.
  - Rename the archive log file when the transmission is complete (using the **`SFTP`** subcommand).
  - The script that makes the archive logs available for InfoSphere CDC only picks up archive logs with the original name.
- When refreshing tables or marking table capture points, ensure that you switch online log files to archive and ship the logs. Otherwise, the starting point for replicating the table may not be reached until the online logs are archived.

7.5 Using InfoSphere CDC in resilient environments

In many organizations, information is the most important business asset, which makes the database that holds it an important IT asset. Also important is having the data be highly available. There are two options to implement a highly available (HA) architecture for databases:

- Hardware mirroring
- Software mirroring (replication)

Hardware mirroring is implemented at the disk level, and is supplied by disk arrays vendors. For software mirroring, there are dedicated solutions available that ensure a backup server is kept synchronized (asynchronously) with production. This section describes the implementation of InfoSphere CDC in resilient environments, which includes typical configurations for the various platforms.

Database clusters are typically implemented for high availability, disaster recovery, load balancing, or a combination of those options. In a cluster scenario, one of the nodes can stop due to a planned activity or due to a failure. The process of stopping a node and turning over services to another node is called *failover*. InfoSphere CDC works directly with the database and can be installed on the node that might fail over at some point, which affects InfoSphere CDC functionality. While designing an InfoSphere CDC implementation in a cluster environment, two main points should be considered:

1. If InfoSphere CDC processing is moved from one node to another during failover, how do the external clients, such as Management Console and subscriptions, reach this instance?
2. How is replication processing by CDC started after a backup node takes over control? This situation includes finding the binary files and configuration (metadata). Both binary files and metadata of the instance should be accessible.

7.5.1 InfoSphere CDC reachability: Virtual IP

In a high availability cluster, a common resource is the virtual IP address. A virtual IP (VIP) is an IP address that is not connected to a specific computer or network interface card on a computer. Incoming packets are sent to the VIP address, but they are redirected to physical network interfaces.

InfoSphere CDC is typically implemented in an active / passive configuration where the replication is only active on one node at a time. The virtual IP address is directed to the InfoSphere CDC node and is used in other instances and in Management Console. When InfoSphere CDC has a failover to another node, nothing should be changed for external users, and all connections can be restored to the same (virtual) IP and port (Figure 7-21).

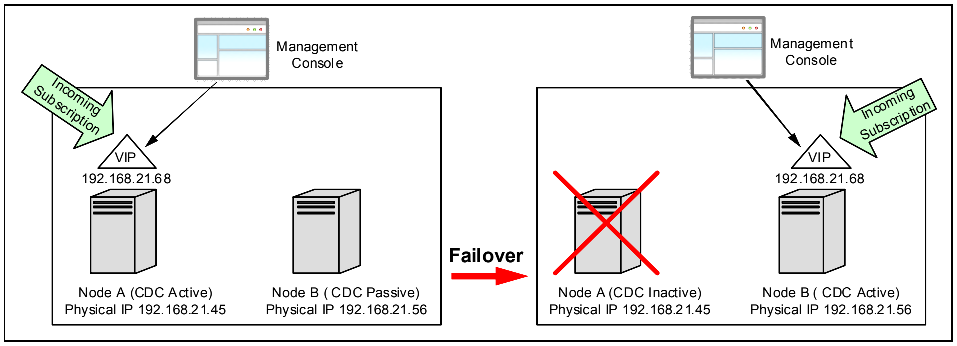


Figure 7-21 CDC cluster reachability

In the left side of Figure 7-21, Management Console connects to the data store using (virtual) IP address 192.168.21.68 which is directed to physical address 192.168.21.45 (node A). Also, data packets coming from subscriptions that are targeting this data store are redirected to the physical IP address. When a failover of node A is performed, the traffic targeting virtual IP address is redirected to physical IP address 192.168.21.56, which is on node B (Figure 7-21).

7.5.2 InfoSphere CDC binary files and metadata for the Linux, UNIX, and Windows engine

The location of the InfoSphere CDC installation binary files and metadata for the common (Java) engine has a significant impact on how the InfoSphere CDC cluster is maintained. It is important to know that the configuration of InfoSphere CDC consists of two components:

- Configuration metadata (subscriptions and mapped tables): These items are kept in a directory on the file system under the `cdc_home/instance` directory.
- Operational information, such as instance signature and bookmarks. These items are kept in tables in the database from or to which CDC is replicating data.

The configuration metadata and operational information are related through the signature and cannot be maintained independent from each other. The signature is established when the instance is created. You cannot recreate an instance with the same location of the operational information without invalidating the old instance.

The next sections describe three possible topologies of an InfoSphere CDC cluster implementation of the Linux, UNIX, and Windows engine and how it should be maintained in each one of them. These topologies are:

- InfoSphere CDC on a shared volume
- InfoSphere CDC on separate nodes with a shared database
- InfoSphere CDC on separate servers with separate databases

7.5.3 InfoSphere CDC on a shared volume

To implement this topology for the Linux, UNIX, and Windows engine, you need a volume that can be accessed by all nodes in the cluster. Usually the shared volume is provided by a storage area network (SAN) or Network Attached Storage (NAS). In some configurations, the disk volume is only mounted on and accessible to the active node. Having a shared volume that keeps the installation of CDC provides the simplest topology for implementation and maintenance.

InfoSphere CDC binary files and configuration metadata is stored on the shared volume and any changes made to the configuration on the active node are centrally stored and available at failover time. InfoSphere CDC operational information is stored centrally in the database, for which clustering must have been set up. All files in the CDC directory structure must be owned by the same user ID on all nodes in the cluster and also the same group ID if it is installed on UNIX / Linux. Because there is only one copy of the configuration metadata, all nodes on which this InfoSphere CDC instance runs must have same database, services, database connection parameters, and database log locations.

The topology for a shared mount point installation is shown in Figure 7-22.

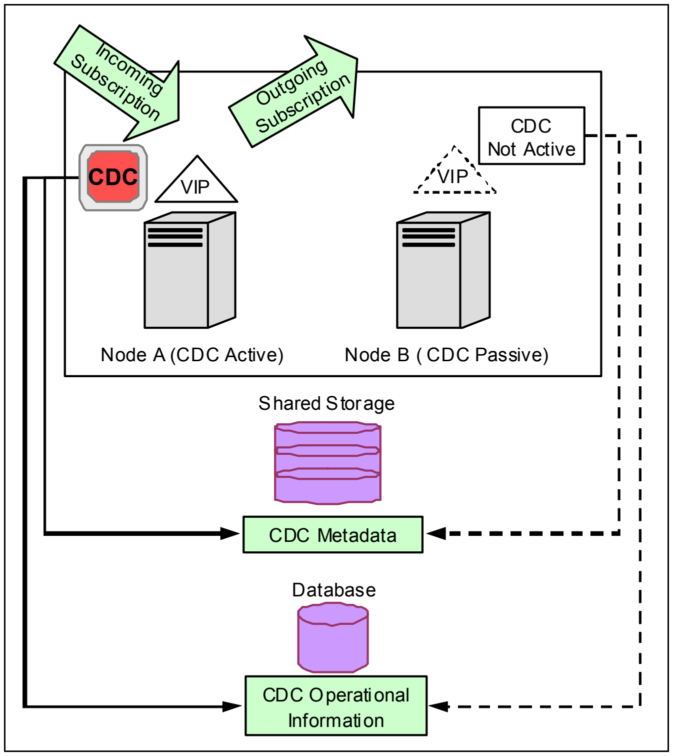


Figure 7-22 InfoSphere CDC on SAN

Installation and configuration steps

Here are the installation and configuration steps:

1. Install InfoSphere CDC once on the active node. The product home directory (referred to as `$CDC_HOME`) must be on a shared volume.
2. Create an InfoSphere CDC instance on the active node. Configuration metadata is created in the file system under the product home directory and operational metadata in the database.
3. Start InfoSphere CDC instance on the active node.
4. Configure subscriptions and table mappings.

Failover steps

If failover is planned, complete the following steps:

1. Stop replication for all subscriptions (run **`dmendreplication`**). If this InfoSphere CDC instance is a target, replication must be stopped from the source.
2. Stop InfoSphere CDC instance (run **`dmshutdown`**).

After the shared volume is available on the failover node:

- Start the CDC instance (run **`dmts32`** or **`dmts64`**).
- Clear the staging store for the instance (run **`dmclearstagingstore`**).
- Restart subscriptions (run **`dmstartmirror`**). If this InfoSphere CDC instance is a target, replication must be started from the source.

7.5.4 InfoSphere CDC on separate nodes with a shared database

In this topology for the Linux, UNIX, and Windows engine (Figure 7-23), InfoSphere CDC is installed on each node in the cluster and there is no shared volume available to contain the CDC installation. The attached database is clustered using a shared volume. Although it is straightforward to implement, it requires additional maintenance and failover steps.

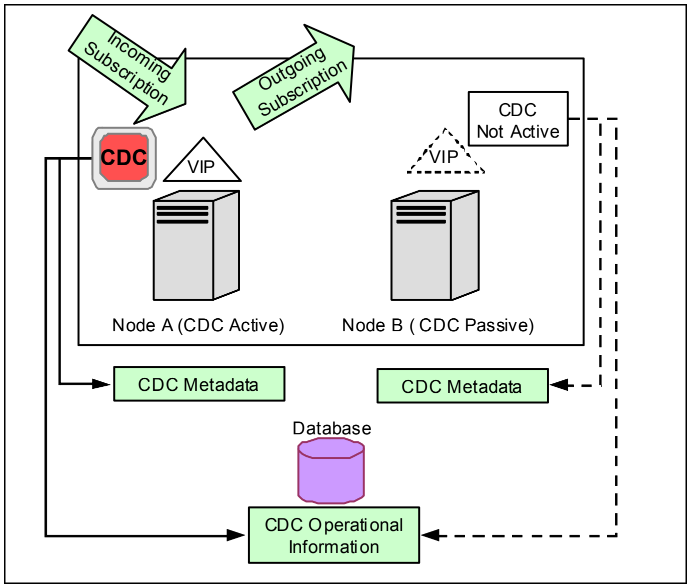


Figure 7-23 CDC resilient installation - separate nodes

InfoSphere CDC binary files and configuration metadata are kept independent from each other on both nodes, so you need to synchronize all the changes that are made to the configuration from the primary to the backup node. Files that are kept in the CDC `home` directory and its tree should have the same user ID and group ID when installed on UNIX and Linux. This setup ensures that the user running the instance on the backup node can actually read and update the configuration when failed over.

Because operational information is stored in the database and there is a link (through a signature) between this data and the configuration metadata, it is not possible to create a CDC instance on the backup node and redo the changes. Therefore, the configuration metadata on the primary node should periodically be backed up and restored onto the backup node. The configuration metadata can be synchronized as frequently as wanted. At a minimum, you should do this task after making configuration changes (mapped tables, subscriptions, and system parameters) and after the initial refresh of mapped tables has taken place and the table status has gone to active.

Because configuration metadata is replicated and operational metadata is the same, as in the previous topology, all nodes that this InfoSphere CDC instance runs on must have the same database, services, database connection parameters, and database log locations.

Installation and configuration steps

Here are the installation and configuration steps:

1. Install InfoSphere CDC on all nodes. Keep `$CDC_HOME` the same on the servers.
2. Create an InfoSphere CDC instance on the active node. The product metadata is created under the product home directory and in the database.
3. Copy the entire `$CDC_HOME/instance` directory to the backup server. Do not create an instance on the passive node, because it invalidates the instance created on the active node.
4. Start the InfoSphere CDC instance of the active node.
5. Configure subscriptions and map tables.
6. Periodically copy the file system metadata and event logs from the active to the passive node. Metadata is kept in the `$CDC_HOME/instance/<instance_name>/conf/md*` file. You can run the **`dmbackupmd`** command to back up the files to the `$CDC_HOME/instance/<instance_name>/backup/bnn` directory. To keep the log history when the instance becomes active on the second node, the events database should be also copied. Events are kept in the `$CDC_HOME/instance/<instance_name>/events/*` file.

Failover steps

If a failover is planned, complete the following steps:

1. Stop replication for all subscriptions (run **`dmendreplication`**). If this InfoSphere CDC instance is a target, replication must be stopped from the source.
2. Stop InfoSphere CDC instance (run **`dmshutdown`**).

After the InfoSphere CDC instance is stopped (or has failed) on the active node and the Virtual IP becomes available on the passive node:

1. Restore the latest version of the file system metadata and events.
2. Start the instance (run **`dmts32`** or **`dmts64`**).
3. Clear the staging store for the instance (run **`dmclearstagingstore`**).
4. Restart subscriptions (run **`dmstartmirror`**). If this InfoSphere CDC instance is a target, replication must be started from the source.

7.5.5 InfoSphere CDC on separate servers with separate databases

In this topology for the Linux, UNIX, and Windows engine, there are two separate instances of the database that use a technology, such as DB2 High Availability Disaster Recovery (HADR) or Oracle Data Guard, to keep a backup (standby) database synchronized with the primary database.

When the high availability solution is implemented on the CDC source, the log entry sequence numbers (LSN for DB2 and SCN for Oracle) are the same on the primary and the backup databases. The backup database is typically in standby mode or only opened for read activity (reporting).

If the CDC target is a database for which such a high availability solution is active, the synchronization of the log entry sequence numbers is not relevant. However, in such a scenario, the reachability of the server is an important consideration. Typically, a virtual IP address cannot be assigned because the primary and backup databases are remote from each other and on different network segments.

InfoSphere CDC configuration and operational metadata are different and kept in pairs due to the signatures between them. This topology should only be implemented if a virtual IP cannot be assigned.

This configuration is shown in Figure 7-24. The InfoSphere CDC binary files and configuration metadata are kept independent from each other on both servers, so you need to synchronize all changes that are made to the configuration from the primary to the backup server. Files in the InfoSphere CDC `home` directory and its tree should have the same user ID and group ID when installed on UNIX and Linux. This setup ensures that the user running the instance on the backup node can actually read and update the configuration when failed over.

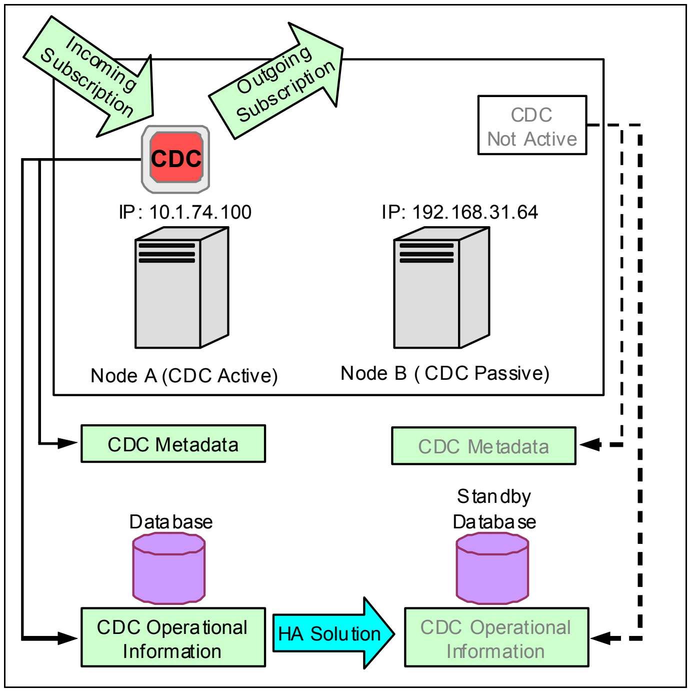


Figure 7-24 CDC in a HADR / Data Guard environment

The configuration metadata on the primary node should periodically be backed up and restored onto the backup node. The configuration metadata can be synchronized frequently, but at a minimum you should do this task after making configuration changes (mapped tables, subscriptions, and system parameters) and after the initial refresh of mapped tables has taken place and the table status has gone to active.

Because configuration metadata is replicated and operational metadata is the same, as described in 7.5.4, “InfoSphere CDC on separate nodes with a shared database” on page 173, all nodes that this InfoSphere CDC instance runs on must have the same database, services, database connection parameters, and database log locations.

Installation and configuration steps

Here are the installation and configuration steps:

1. Install InfoSphere CDC on all nodes. Keep `$CDC_HOME` the same on the servers.
2. Create an InfoSphere CDC instance on the active server. The product metadata is created under the product home directory and in the database.
3. Copy the entire `$CDC_HOME/instance` directory to the backup server.
4. Start the InfoSphere CDC instance on the active server.
5. Configure subscriptions and map tables.
6. Periodically, copy the file system metadata and event logs from the active to the backup server. Metadata is kept in the `$CDC_HOME/instance/<instance_name>/conf/md*` file. You can run the **`dmbackupmd`** command to back up the files to the `$CDC_HOME/instance/<instance_name>/backup/bnn` directory. To keep the log history when the instance becomes active on the backup node, the events database should be also copied. Events are kept in the `$CDC_HOME/instance/<instance_name>/events/*` file.

Failover steps when the server is the source for InfoSphere CDC

If the failover is planned, complete the following steps:

1. Stop replication for all subscriptions (run **`dmendreplication`**). If this InfoSphere CDC instance is a target, replication must be stopped from the source.
2. Stop the InfoSphere CDC instance (run **`dmshutdown`**).

After the InfoSphere CDC instance is stopped (or fails) on the primary server and the database on the backup server is active, complete the following steps on the backup server:

1. Restore the latest version of the file system metadata.
2. Start the instance (run **`dmts32`** or **`dmts64`**).
3. Clear the staging store for the instance (run **`dmclearstagingstore`**).
4. Restart subscriptions (run **`dmstartmirror`**).

Failover steps when the server is the target for CDC

If the failover is planned, complete the following steps:

1. Stop replication for all subscriptions (run **`dmendreplication`**) on the source server.
2. Stop the InfoSphere CDC instance (run **`dmshutdown`**).

After the InfoSphere CDC instance is stopped (or fails) on the primary server and the database on the backup server is active, complete the following steps on the backup server:

1. Restore the latest version of the file system metadata.
2. Start the instance (run `dmts32` or `dmts64`).

Then, the subscriptions need to be redirected to the backup server. Complete the following steps:

1. From the Access Manager perspective, change the IP address (or host name) of the target data store.
2. Connect to the target data store from the Management Console Configuration perspective.
3. Right-click the connected data store, click **Properties**, and click the **Aliases** tab.
4. Add the IP address or host name of the backup server to the aliases.
5. Go to each subscriptions' properties and click the **Details** button next to the target data store. You should be able to select the backup server's IP address or host name as the new destination for the subscription

7.5.6 System i resilient environments

There are different scenarios for InfoSphere CDC resiliency when an IBM System i server (formerly known as AS/400 and IBM eServer™ iSeries®) is part of the replication landscape. There are two methods for providing resilience in a System i environment, one similar to the clustering solutions that are available for other platforms and the other using software-based high availability solutions.

This section focuses on how the two methods pertain to the configuration failover of InfoSphere CDC.

Resiliency using Independent Auxiliary Storage Pools

Independent Auxiliary Storage Pools (IASPs) are a System i supported means of storing business data and applications separately from the internal storage that comes with the server. IASPs lend themselves to using SANs and disk-based replication solutions for resiliency of data and applications. IASPs can be detached from one server and then attached to a backup server so that the backup server can take over production work.

A simple scenario of an IASP that can be attached to local servers is shown in Figure 7-25.

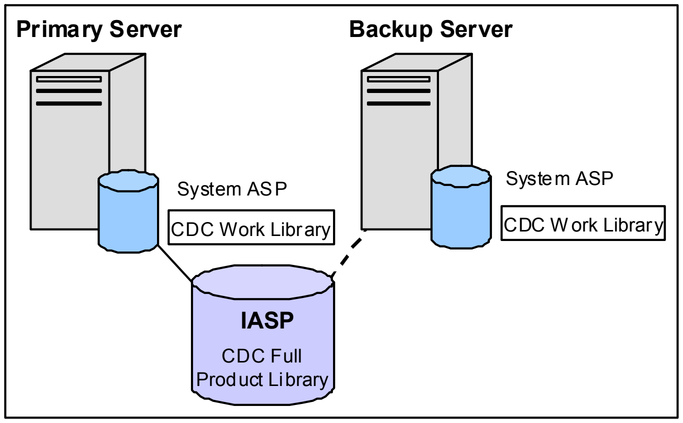


Figure 7-25 IASP attached to local servers

When using a disk-based mirroring solution, an IASP can have a physical backup that is remote from the primary production site. If there is a failure of the primary server or a site disaster, the server on the backup site can take over the production work.

A mirrored IASP is shown in Figure 7-26.

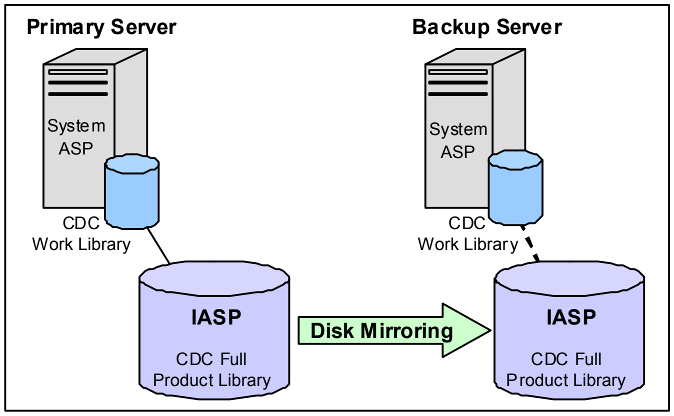


Figure 7-26 IASP attached to local servers

An IASP can only be attached to one System i server at a time. Also, when disk mirroring is used to synchronize to a backup site, the backup IASP cannot be attached to the backup server until synchronization is stopped.

Using InfoSphere CDC in an IASP environment

InfoSphere CDC supports installation in environments where IASPs are used. The product has a number of objects that must be kept in internal storage (`*SYSBAS ASP`), such as a user profile (D_MIRROR), the subsystem description, and a job queue.

When installing InfoSphere CDC in a resilient environment, you must perform the installation on the active server that is connected to the IASP. During the installation, you are prompted to specify the name of the IASP and the product library; this library is where the full product, including the configuration tables, is installed, and which is kept on the IASP. The installation automatically creates another library on `*SYSBAS ASP` (the name is automatically determined by the installation program, being the first eight characters of the product library name, appended with 01), which is the work library. The work library holds all the objects that cannot be kept on the IASP.

After you have installed InfoSphere CDC on the active system, you must perform a work library installation on the inactive system. The work library installation just creates the library and objects that cannot be kept on the IASP and that are only used when CDC must be activated on the backup server.

If you change the D_MIRROR user profile or the objects in the InfoSphere CDC work library, these changes must also be made on the inactive server. The system ASP holds the service table entries, which define the port that the product listens on. The service table entry must be available on the backup server, or the listener does not start successfully when the subsystem is started. Configuration, such as mapped tables, subscriptions, and so on, are kept in the full product library, which is on the IASP, and shared (or mirrored) between the primary and backup server.

If there is a failover or switchover of InfoSphere CDC in an IASP environment, you must stop the subscriptions replicating from or to the primary System i server, and then start the subsystem in the CDC work library. This action automatically starts the TCP/IP listener. Once active, the replication can be resumed and should incur no loss or duplication of data.

Besides the InfoSphere CDC library, the IASP needs to hold the tables that are being replicated from or to. Also, if you are replicating from tables that are on an IASP, the journal and journal receivers of these tables must also be on the same storage. This action should already be part of your implemented recovery strategy.

Using InfoSphere CDC with software-based high availability solutions

Many clients with System i servers employ software-based high availability solutions to provide for disaster recovery. An example of a software-based high availability solution is IBM iCluster® for i.

Software-based HA solutions operate by reading journal entries generated by the production application and applying these entries onto the target tables. These target (backup) tables are also journaled and therefore generate journal entries similar to the ones on the production side. Journaling on the backup server might be different from the journaling on the primary server (but typically not). The journal receivers containing the backup server's tables journal entries might (and probably do) have a different name from the primary server's journal receivers. In some cases, journaling on the backup server might even be disabled while the server is not running any production processes.

This situation adds additional complexity to the journal management and switchover process if InfoSphere CDC uses the System i production server as a source. This section lists the considerations when implementing InfoSphere CDC in such a landscape.

There are two main items that must be considered in a software-based HA implementation:

- Switchover and failover of InfoSphere CDC
- Journal management (purging of obsolete journal receivers)

These considerations only apply when the System i server is used as a source for InfoSphere CDC. When targeting this server platform, switchover is much more straightforward and journal management considerations do not apply.

Switchover and failover

In high availability terminology, switchover is when switching control to a backup server is a planned activity. You can stop applications on the primary server and perform preparative actions on the backup server before resuming production activity, for example, when migrating to a new server. A failover occurs when there is an unplanned event that causes the primary server to be unavailable, such as a power outage or a site disaster.

If there is a switchover, the preparative actions that must be taken on the primary server are:

1. Stop business application activity so that journal entries are no longer generated.
2. Stop the InfoSphere CDC subscriptions.

3. Restart the subscriptions with scheduled end of now (net change mirroring) to ensure that all transactions have been replicated to the target server.
4. Optionally, stop the subsystem.

After these steps have been performed, you should complete preparative steps on the backup system before resuming business application activity:

1. Start the InfoSphere CDC subsystem, but do not start any subscriptions yet.
2. Set the bookmark (SETJRNPOS) to any entry in the current journal receiver.
3. Mark table capture points for all tables in the subscriptions; this action marks the point from which changes are being replicated again.
4. The business applications can be restarted.
5. At any time, the subscriptions can now be started; any transactions that were generated after the marking of table capture points are replicated to the target server.

When there is a failover, you might not exactly know up to which point the transactions in the primary server's journals have been replicated to the target server. The next few sections elaborate on the challenges that you are presented with in the event of a failover and how to overcome them to be able to resume replication without data loss.

Journals and journal receivers

On System i servers, the subscription bookmarks that are kept by InfoSphere CDC are identified by the name of the journal (and its library), the journal receiver (and its library), and a journal entry sequence number within the receiver.

Tables (physical files) on the primary (production) server are journaled and the journal has one or more receivers associated with it, storing database transactions in the current, attached receiver. If tables are also journaled on the backup server (this setup is optional, but is a best practice to facilitate failover), the attached journal should have the same name and be on the same library as the one on the production server.

The production journal could have the following journal receivers associated with it:

- RCV0578
- RCV0579
- RCV0580

On the backup server, the equivalent journal almost certainly has different journal receivers associated with it, for example, RCV1593 and RCV1594. Naming and switching of the journal receivers on the backup server is independent of the journal receivers on the primary server. The likelihood of journal receivers having the same name is small and cannot be enforced.

An example scenario for an InfoSphere CDC implementation coexisting with a software-based HA solution is shown in Figure 7-27.

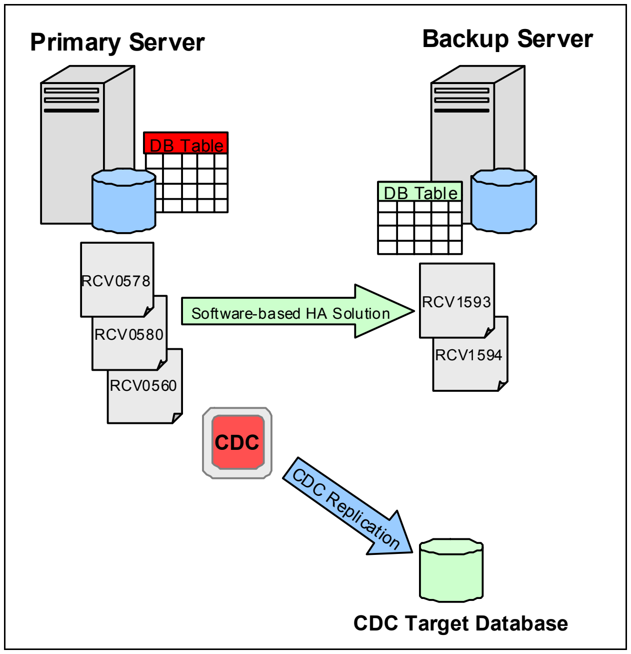


Figure 7-27 Software-based HA solution

Bookmark and failover

In Figure 7-27, a journal entry 110 in receiver RCV0580 might correspond to the journal entry 81524 in receiver RCV1594 on the backup server. When InfoSphere CDC applies the primary server's journal entry to the target database, the bookmark that is written would be RCV0580-110.

Should production have to be switched over from the primary to the backup server and the subscription restarted from the backup server, the bookmark information would not be valid anymore. Most likely, the journal entry that is identified by the bookmark does not exist in the backup server's journal and the subscription fails to start.

Before starting the subscriptions on the backup server, the bookmark (journal position) must be reset on the backup server. Because the source server might no longer be available and the bookmark information kept in the database targeted by the subscription cannot be associated exactly with a journal entry on the backup server, you must use a strategy that replicates all entries. You cannot prevent some table operations from being replicated more than once, and it is likely that primary key violations occur when applying entries to the target tables. If the replication is targeting tables without unique keys, such as audit tables, or applications, such as DataStage or a JMS provider, there will be duplicates in the entries that are sent, and these duplicates might have to be resolved afterward.

A common strategy to set the journal position after a failover is to keep a time stamp for every subscription in the database / application that is targeted. A technical table is created on the source System i server and this table holds a (dummy) key and a time stamp column. The technical table is included in every subscription and resolves in a target table that is populated with the last update that was done on the source table. On the source server, a simple program is scheduled to run that updates the row in the source table with the current system's date and time.

By following this strategy, you have a table in the target database that holds the date and time of the last applied entry. This date and time can then be used to find the journal entry in the backup server's journal and set the bookmark using the **SETJRNPOS** command.

In this scenario, assume that the mirroring by the high availability solution is up to date and has transferred all changes from the primary server to the backup server.

To determine if no changes are skipped for replication, you could set the bookmark to a journal entry that is a few minutes before the recorded time stamp. This setting covers situations where the system times of the primary and backup server are not synchronized and entries on the backup server have an earlier time stamp than the corresponding entries on the primary server.

If the target tables have primary keys, you can keep the subscription from stopping with a duplicate key or missing record error by choosing one of the following options:

- Activate conflict detection on the replicated tables and specify “source wins”. If an insert cannot be applied to the target, it is turned into an update. An update of a row that does not exist results in an insertion, and deletion of a non-existing row is ignored. Additionally, the conflicts are logged in a conflict auditing table so they can be reviewed afterward.

- Remap the tables with the Adaptive Apply mapping type. This action produces the same result as activating conflict detection, but it does not log conflicts.
- Instruct the InfoSphere CDC target engine to continue on error during mirroring. A system parameter can be set for this option. The name of the system parameters is dependent on the engine type; for the Linux, UNIX, and Windows engine, set `mirror_end_on_error=false`.

The configuration changes should only be used to allow the subscription to continue beyond the point where the source and target are synchronized again. All three options might have an adverse effect on the apply throughput and you should consider changing the settings back to the original ones as soon as possible.

Journal management

If there is a failover, the journal on the backup server must have all the entries that were not replicated to the subscription's target server. The journal receivers must only be removed from the backup server once they do not hold any data that might have to be replicated in a switch.

As a simple strategy for ensuring that journal entries are available at failover time, implement a clean-up strategy that keeps a number of journal receivers on the backup server, for example, for one day.

If there is insufficient space on the backup server, you must correlate the journal position (bookmark) of the subscriptions with a backup journal receiver. `RTVDMJENT` returns the oldest journal receiver that is still needed by any of the subscriptions; it must be run on the primary server. Through the journal receiver that is returned, you can determine the time stamp of the first journal entry. The time stamp can be used to find the oldest journal receiver that is still required on the backup server. You should subtract a few minutes from the time stamp if this procedure is also done when the bookmark is set after failover.

Mirroring scope for a high availability solution

In addition to managing the journal receivers and setting the bookmark, consider the configuration of the high availability solution. On the backup server, a fully functional copy of the InfoSphere CDC installation must be available to be used when control is switched to it. At the same time, not all objects in the product library may be available for mirroring (save / restore) due to object locks.

Objects listed in Table 7-1 must be included in the mirroring scope of the high availability solution.

Table 7-1 Mirroring scope objects

Library	Object	Object Type	Include or Exclude	Comments
InfoSphere CDC Product Library	*ALL	*ALL	Include	N/A.
InfoSphere CDC Product Library	*ALL	*USRQ	Exclude	These objects are exclusively locked when subscriptions are running.
QSYS	D_MIRROR	*USRPRF	Include	InfoSphere CDC User Profile.

Initially, all objects in the InfoSphere CDC product library should be saved on the primary server and restored onto the backup server. There are objects such as communications user spaces that must exist on the target. Stop all subscriptions and the subsystem when doing the initial synchronization of the InfoSphere CDC product library to the backup server.

7.5.7 z/OS / Sysplex and InfoSphere CDC in resilient environments

Resiliency consists of being able to recover and restart replication if any of the main source or target components fail.

The overall architecture is shown in Figure 7-28. The components in solid boxes are the ones in active use; the components in dashed-line boxes become active should the current components fail. In general, each component can be physically on a different machine. Any active component could fail, requiring a failover to a backup machine.

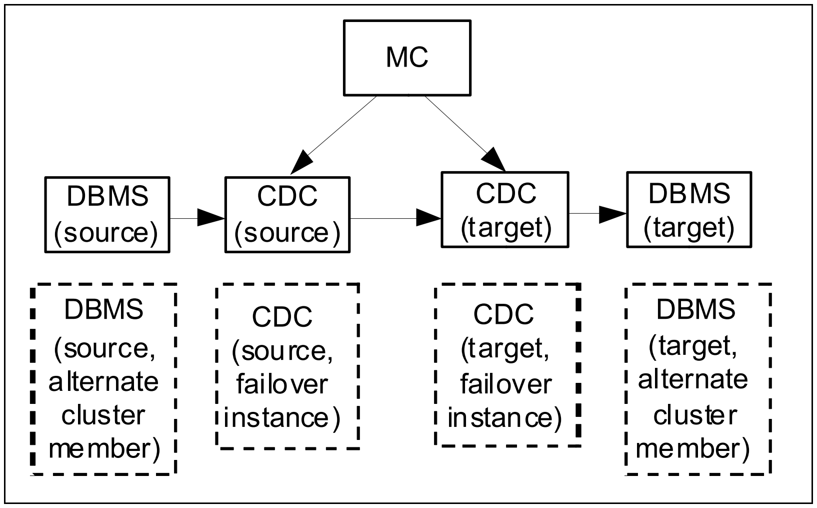

MC
DBMS (source)
CDC (source)
CDC (target)
DBMS (target)
DBMS (source, alternate cluster member)
CDC (source, failover instance)
CDC (target, failover instance)
DBMS (target, alternate cluster member)

Figure 7-28 Failover architecture

A failover entails the following actions:

- Ensuring the configuration and operational metadata on the failed over machine matches the metadata is required to continue replication. This action includes ensuring that database log restart positions are properly set.
- Re-establishing communications.
- Ensuring the required software is running on the failover machine.
- Restarting replication within the new failover environment.

Resiliency with InfoSphere CDC z/OS can be achieved by working within a DB2 z/OS data sharing environment. The InfoSphere CDC z/OS address space must run in a z/OS instance corresponding to one of the members of the data sharing group. Should that instance of z/OS fail, or should that DB2 member not be operational, InfoSphere CDC z/OS can be restarted on another z/OS instance containing a member of the data sharing group. Because the data is shared, the failover instance of InfoSphere CDC z/OS uses the same metadata (including operational metadata) as the failed instance, so failover considerations are primarily around connectivity.

To prepare InfoSphere CDC z/OS to restart on a different member, ensure that the following tasks are accomplished:

- The DB2 subsystem parameter SSID in the CHCDBMxx configuration member must refer to the group attachment name, not the subsystem ID of an individual member.
- The Host Name data store, as configured in the Management Console, must resolve to a virtual IP address before configuring any subscriptions.
- The InfoSphere CDC z/OS source machine's TCP/IP configuration must be configured to use the same virtual IP address for the InfoSphere CDC target configured in the Management Console.
- Upon failover, the virtual IP address used by the Management Console to connect to the source or the target engine must refer to the new z/OS instances on which the InfoSphere CDC z/OS address space is running. The data store must be configured using virtual IP addresses in the Management Console before creating subscriptions.
- Subscriptions are configured as persistent by right-clicking the subscription and clicking **Properties** → **Advanced Settings...** → **Mark subscription as persistent**.

The four main resiliency scenarios are

- Source database management system (DBMS) failure
- InfoSphere CDC source failure
- InfoSphere CDC target failure
- Target DBMS failure

The InfoSphere CDC source failure and InfoSphere CDC target failure scenarios also cover the case of overall OS or hardware failures for the z/OS instance on which they are running.

Should the source DBMS become unavailable, all subscriptions are automatically ended by the product. With persistent subscriptions, the product detects when the DBMS becomes available and automatically restarts the ended subscriptions. If there is a hard DBMS failure and the DBMS member is unusable, then the InfoSphere CDC source needs to be failed over to use another member.

Failover of the InfoSphere CDC z/OS source entails ending the InfoSphere CDC z/OS address space if it is still running and restarting it with the same user ID on another member of the data sharing group. If the address space was ended while the subscriptions were running, then the subscriptions that have been marked persistent are restarted automatically when the address space is restarted. Otherwise, the subscriptions need to be restarted manually.

Because the target's IP address is unchanged and correctly configured in the shared metadata, only the virtual IP address for the InfoSphere CDC source needs to be redirected to the failover member for the Management Console to access it.

Resiliency for an unavailable InfoSphere CDC z/OS target could entail surviving network outages or, if the target becomes unusable, entail failing over the target to another member. For network outages, the persistent subscription feature of InfoSphere CDC z/OS provides a means of reconnecting automatically. Should the InfoSphere CDC target become unreachable through communication channels, the InfoSphere CDC z/OS source tries connecting to the target at intervals set by the AUTORESTARTINTERVAL configuration parameter. After the InfoSphere CDC target becomes available, the subscriptions marked as persistent are automatically restarted.

If a failover of the target is required, the persistent subscription tries simplify failover procedures. The InfoSphere CDC z/OS target address space needs to be ended (if it is still running) and then restarted using the same user ID on another member in the data sharing group. The virtual IP address used by the source and Management Console to reach the target needs to be updated to resolve to the failover member. With the target instance running and virtual IP address for the target redirected, the persistent subscription automatically reconnects to the new target. The InfoSphere CDC z/OS source can be left running during a target failover. For InfoSphere CDC sources that do not support the persistent subscription feature, the subscriptions need to be manually restarted. Also, if the failover is run in a controlled manner, that is, by explicitly ending subscriptions, then the subscriptions need to be restarted manually. The automatic restart feature is designed to restart subscriptions that shut down due to loss of connectivity, not subscriptions that were shut down normally or due to a DBMS error on the target.

If the target DBMS becomes unavailable, the InfoSphere CDC z/OS target remains up and suspends its operations. For reliable suspension behavior, the retry cache must be configured through the **`RECOVERYRETRYLIMIT`** and **`RETRYCACHESIZE`** configuration parameters. The suspension results in data backing up and eventually leads to the product stalling. This situation is a normal recovery situation until the target DBMS becomes operational once again. The product detects that the DBMS is up, mirroring resumes automatically, and the product processes its backlog.

7.6 Change management

It is essential to know what changes to your environment have an impact on your InfoSphere CDC environment. You need to incorporate InfoSphere CDC into your change management procedures. This section describes the change management workflow for InfoSphere CDC, what are its important steps, the role of bookmark information, and how certain activities can be automated.

Basically, there are two main topics that trigger change management in a InfoSphere CDC environment:

1. Database table changes that occur in the environment InfoSphere CDC is replicating from or to.
2. The configuration changes that are done within InfoSphere CDC. New business requirements or applications can demand changes in your replication environment.

Changes within just the InfoSphere CDC configuration typically do not require special treatment. Subscriptions to be deployed can be stopped, changed, or replaced (import) with a new definition, and restarted.

Alterations of tables, which are part of the InfoSphere CDC replication environment, could have a significant effect on the behavior of InfoSphere CDC. A possible cause for changed table structures is new application releases. Within InfoSphere CDC, you map specific columns of a source table to specific columns of a target table. When changes to either the source or target table structure take place, it can affect replication. The following examples of DDL changes affect replication:

- Adding new columns
- Dropping in-scope columns
- Dropping out of scope columns
- Modifying column formats, such as data type, length, and precision

When the structure of a table from which InfoSphere CDC is replicating is changed, the format of the database log entries changes accordingly. If the log reader is actively reading entries and is not aware of the structure change, it does not correctly interpret the log entries and unexpected errors could occur. The InfoSphere CDC engines can detect DDL changes from the database log and take corrective actions in such cases. There are slight differences in how a InfoSphere CDC engine behaves in case of source table changes.

If the structure of a target table changes, InfoSphere CDC behaves as follows:

- Add new column: The new column is not mapped by InfoSphere CDC, but is populated with the database default value.
- Delete previously mapped column: InfoSphere CDC encounters an apply error because InfoSphere CDC attempts to write to a non-existent column.
- Alter column definition: InfoSphere CDC might encounter mapping issues because of a changed target column definition.

7.6.1 Understanding InfoSphere CDC bookmarks

When implementing changes to source tables, understanding the bookmarks that InfoSphere CDC keeps is key to choosing the best method for a safe deployment of subscription changes in your environment. Here are the following two types of bookmarks that are maintained by InfoSphere CDC:

- Subscription bookmark: This bookmark is the bookmark that InfoSphere CDC keeps per subscription and is updated on the target side as transactions are applied. The bookmark provides for the resilience of the product. If there is any disruption in the processing that causes the replication to stop, InfoSphere CDC resumes reading the logs from the bookmark position from that point onwards so that no transactions are lost or duplicated.
- Table capture point: This bookmark is the commit point in the database from which operations for the table that has its capture point marked are processed. All changes that are committed after the marked point are replicated. The table capture point is set when you run the **`mark table capture point`** routine for a table in a subscription. When the subscription is restarted after marking the table capture point, it resumes from its (subscription) bookmark position. However, changes to tables that have their table capture point marked are ignored until the log scraper advances and reaches the capture point. Each table can have its own table capture point; they do not have to correspond.

If there is a service window available where database and application changes can be applied, and you can ensure that there are no source database transactions during that time frame, this window provides an ideal way of deploying subscription changes. If you ensure that all transactions before the service window are sent and applied to the target side, the InfoSphere CDC configuration, including subscriptions, can be fully recreated and table capture points marked before the first application activity after the service window starts.

Figure 7-29 shows the deployment within the service window.

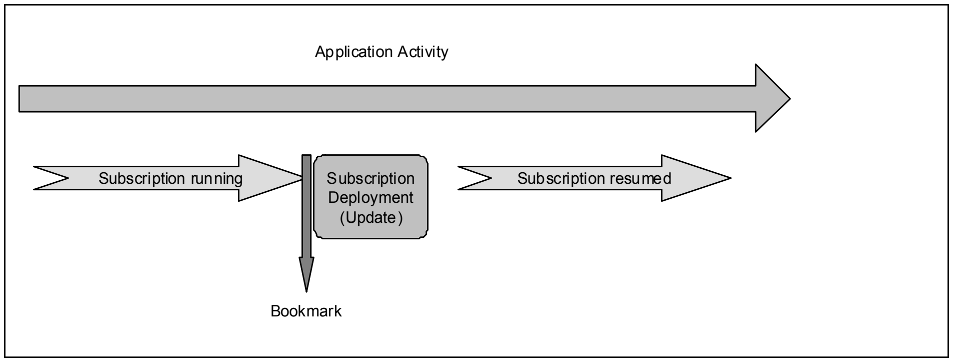


Figure 7-29 Deployment within a service window

Suppose you have a situation where there is not a service window or if the InfoSphere CDC configuration changes cannot wait to be applied during a time of no activity. In that case, you want the subscriptions to continue processing changes following the last successfully applied transaction after the subscription changes have been deployed, without skipping any changes (Figure 7-30).

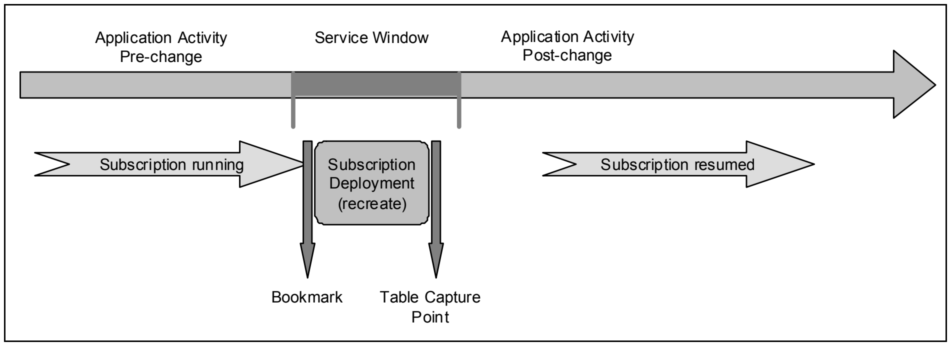

Application Activity Pre-change
Service Window
Application Activity Post-change
Subscription running
Subscription Deployment (recreate)
Subscription resumed
Bookmark
Table Capture Point

Figure 7-30 Deployment with no service window

The focus in this section is on deploying changes when a service window is in place. This section also briefly touches on the considerations for the different InfoSphere CDC engines when there is no service window.

7.6.2 Change Management sample environment

For our change management samples, we use the client environment shown in Figure 7-31. The architecture is split into a development (APPSRCDEV and APP_TGT_DEV) and a production environment (APPSRCPRD and APP_TGT_PRD), both replicating between the two locations in Toronto and Sydney. CM_DEV and CM_PROD are the two subscriptions that are identical in structure and are deployed in the client's development and production environment. Both subscriptions contain several source tables that are mapped through standard replication in a 1:1 fashion to their corresponding target tables, with no transformations.

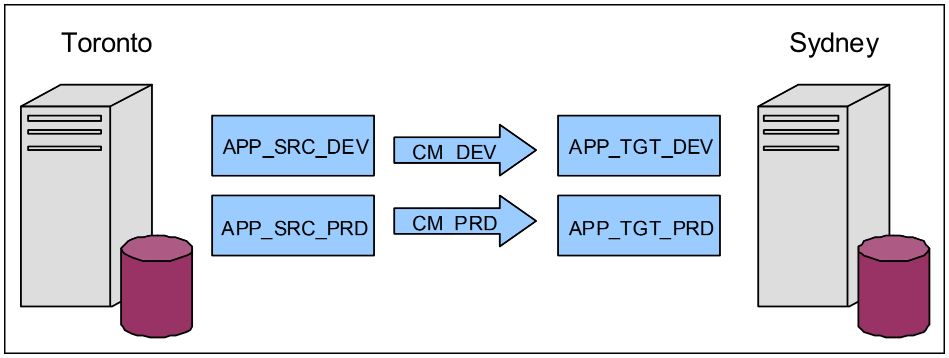


Figure 7-31 Sample client environment

Changes are first made in the development environment and then deployed to the production environment. In the development environment, there are no constraints with regards to refreshes and so on, but in the production environment, no refreshes can be done; the procedure must be such that no operations are lost or duplicated.

7.6.3 DDL changes in a service window

For our change management scenario, we assume that at least one table within the subscription needs to be changed in structure due to new business requirements (such as column added and column data type modified). These changes need to be run for the source and the target table.

In our planned scenario, we perform these changes in the development environment first. Complete the following steps:

1. Stop the CM_DEV subscription in the development environment.
2. To implement the changes in the development environment, run the DDL statements on the table in question to make the table structure changes for the source and target table.
3. Update the source and target table definitions for InfoSphere CDC through the Management Console. This step is necessary to make InfoSphere CDC aware of the structural changes and enable it to correctly interpret the database log entries (Figure 7-32 Figure 7-33 on page 195) and the target table (Figure 7-34 on page 195 and Figure 7-35 on page 196).

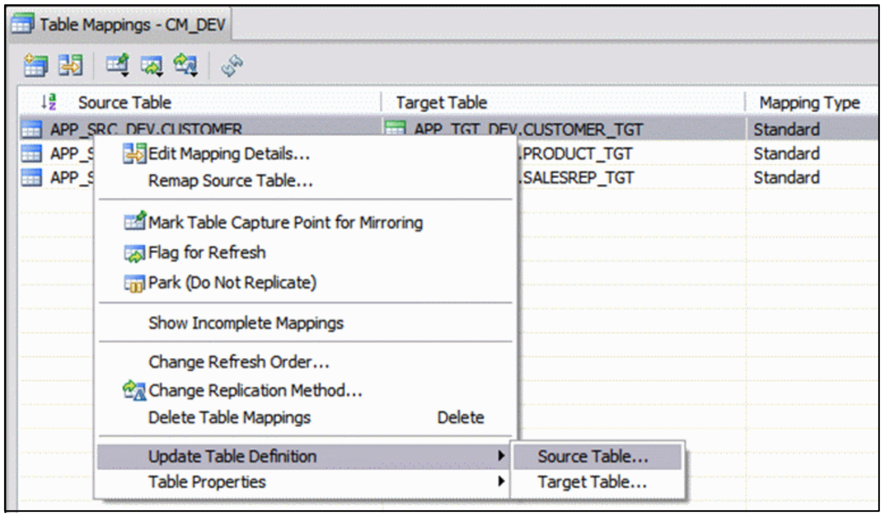
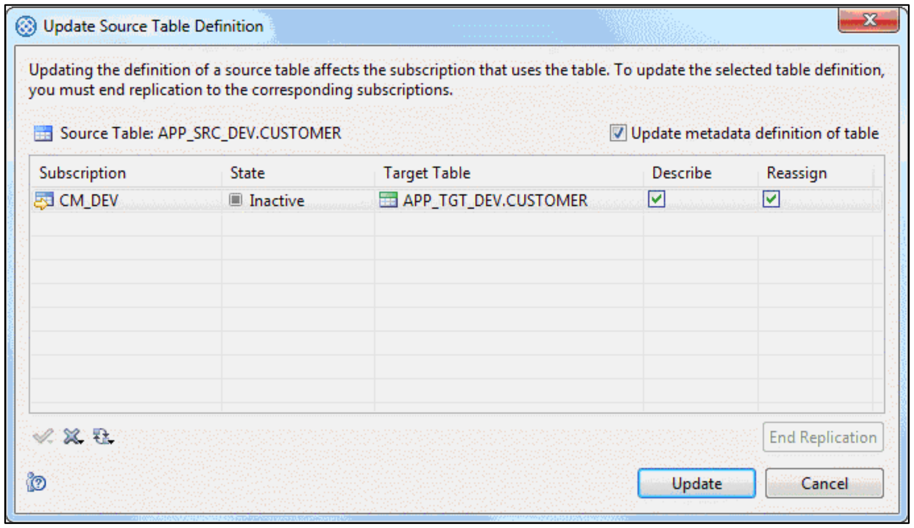


Table Mappings - CM_DEV

Source Table	Target Table	Mapping Type
APP_SRC_DEV.CUSTOMER	APP_TGT_DEV.CUSTOMER_TGT	Standard
APP_S	.PRODUCT_TGT	Standard
APP_S	.SALESREP_TGT	Standard

Figure 7-32 Update source table - 1



Update Source Table Definition

Updating the definition of a source table affects the subscription that uses the table. To update the selected table definition, you must end replication to the corresponding subscriptions.

Source Table: APP_SRC_DEV.CUSTOMER ☑ Update metadata definition of table

Subscription	State	Target Table	Describe	Reassign
CM_DEV	Inactive	APP_TGT_DEV.CUSTOMER	✓	✓

Figure 7-33 Update source table - 2

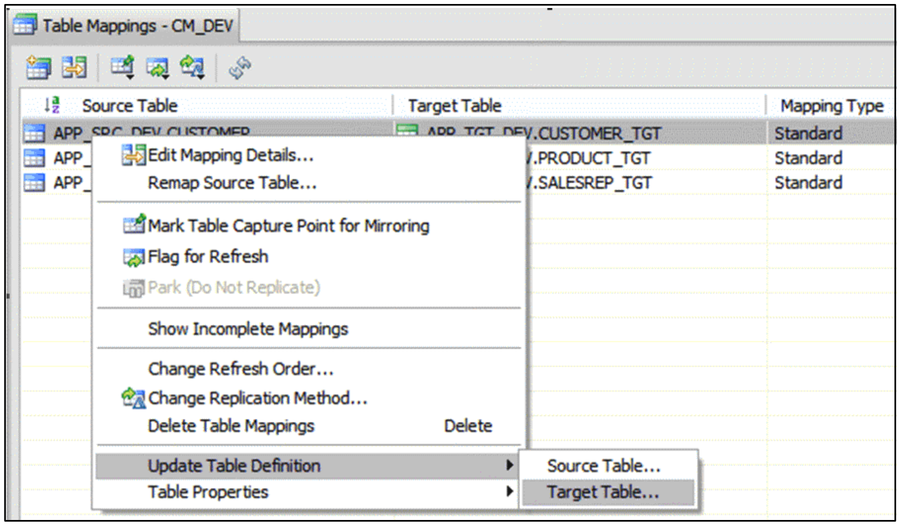
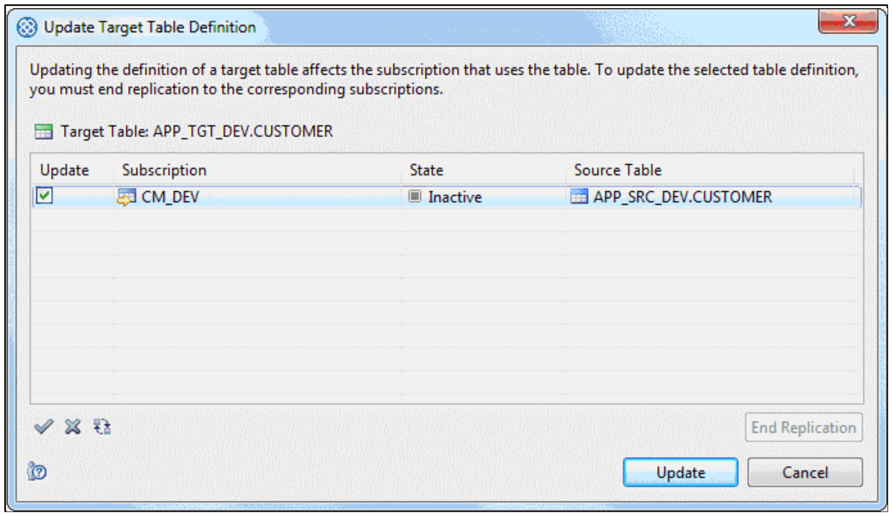


Table Mappings - CM_DEV

Source Table	Target Table	Mapping Type
APP_SRC_DEV.CUSTOMER	APP_TGT_DEV.CUSTOMER_TGT	Standard
APP_	.PRODUCT_TGT	Standard
APP_	.SALESREP_TGT	Standard

Figure 7-34 Update target table - 3


Update Target Table Definition

Updating the definition of a target table affects the subscription that uses the table. To update the selected table definition, you must end replication to the corresponding subscriptions.

Target Table: APP_TGT_DEV.CUSTOMER

Update	Subscription	State	Source Table
✓	CM_DEV	Inactive	APP_SRC_DEV.CUSTOMER

Figure 7-35 Update target table - 4

4. Adjust the mappings in the development subscription. After you update the table definitions for the source and target tables, you must alter the table mappings for newly added columns that have a different name or data type on the source and target. The status of the changed tables has been set to Parked (Figure 7-36).

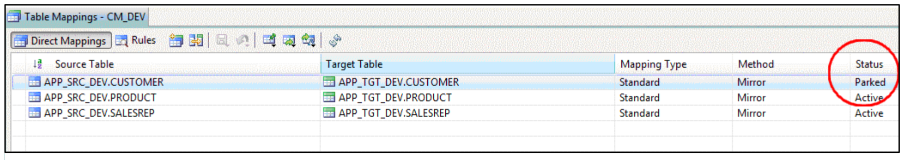


Table Mappings - CM_DEV

Source Table	Target Table	Mapping Type	Method	Status
APP_SRC_DEV.CUSTOMER	APP_TGT_DEV.CUSTOMER	Standard	Mirror	Parked
APP_SRC_DEV.PRODUCT	APP_TGT_DEV.PRODUCT	Standard	Mirror	Active
APP_SRC_DEV.SALESREP	APP_TGT_DEV.SALESREP	Standard	Mirror	Active

Figure 7-36 Changed tables are parked

To keep the example simple, assume that the changed table is refreshed in the development environment (Figure 7-37).

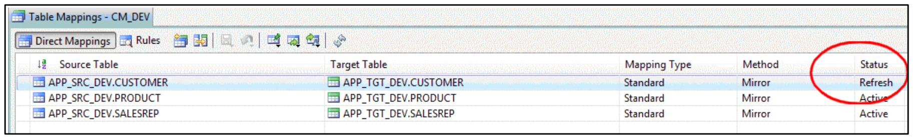


Table Mappings - CM_DEV

Source Table	Target Table	Mapping Type	Method	Status
APP_SRC_DEV.CUSTOMER	APP_TGT_DEV.CUSTOMER	Standard	Mirror	Refresh
APP_SRC_DEV.PRODUCT	APP_TGT_DEV.PRODUCT	Standard	Mirror	Active
APP_SRC_DEV.SALESREP	APP_TGT_DEV.SALESREP	Standard	Mirror	Active

Figure 7-37 Refresh changed tables

5. To deploy the changed subscription in the production environment, export the updated subscription CM_DEV out of the development environment to an XML file for import in production (Figure 7-38).

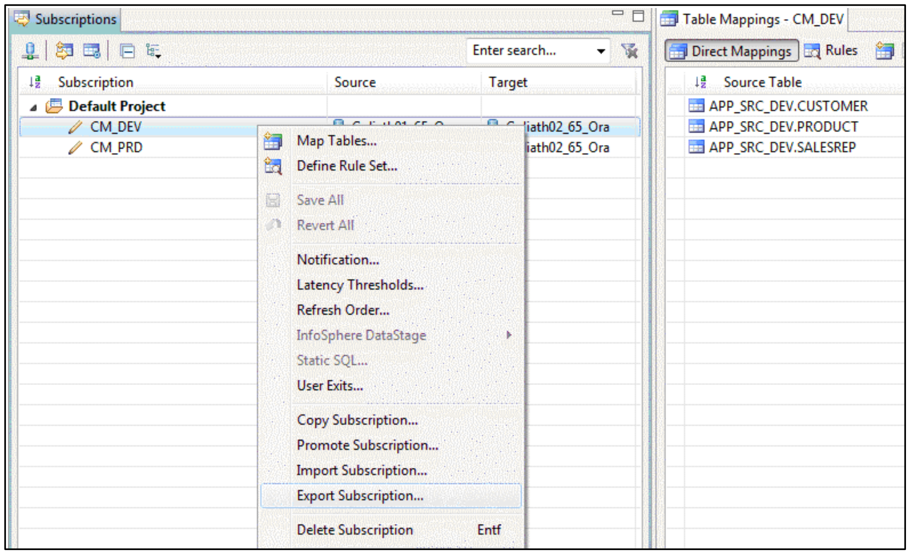


Figure 7-38 Export development subscription

A prerequisite for the successful importation of a subscription that has table mappings using an index is that a named index (not a system generated name) must be specified and that index exists in the import environment. Alternatively, the individual key columns can be specified. An unsupported system generated index is shown in Figure 7-39.

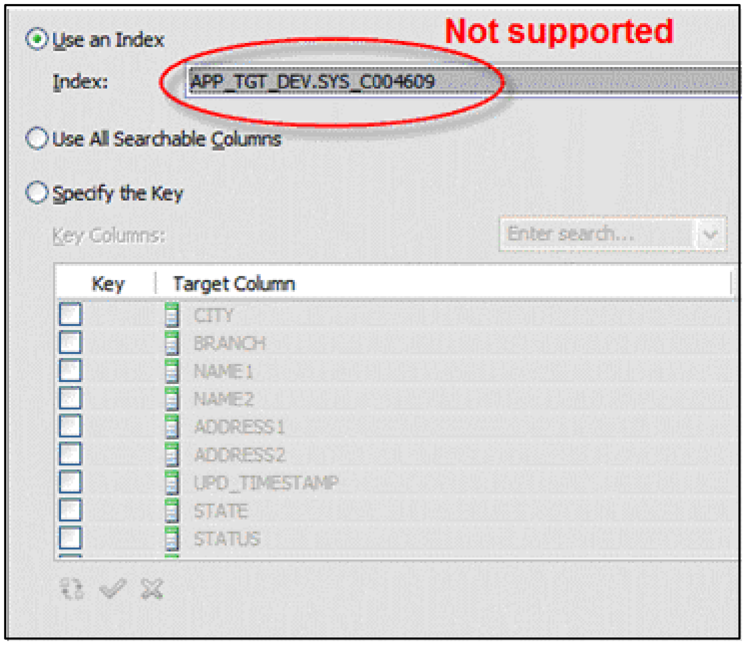
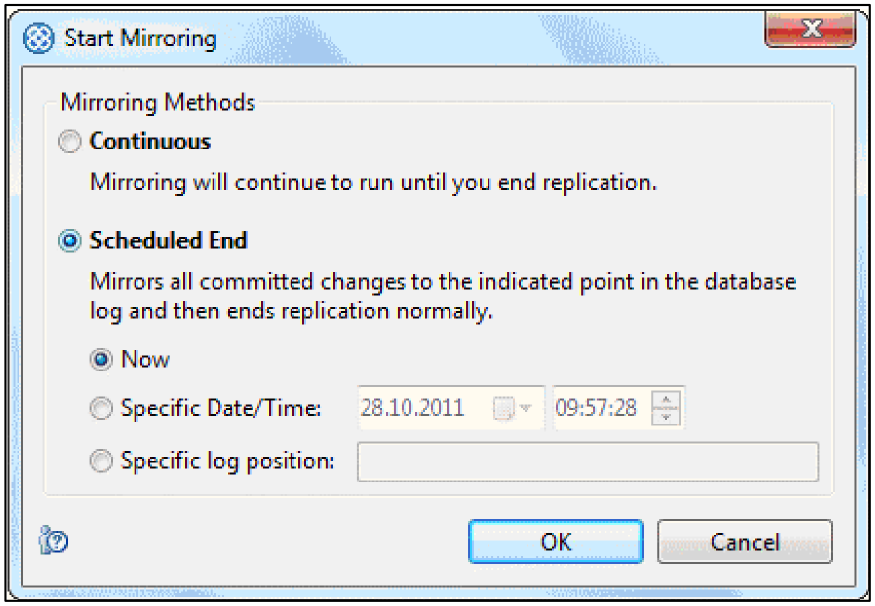


Figure 7-39 System generated index

For the deployment in the production environment, you are not allowed to refresh the changed tables after making the DDL changes. Instead, mirroring should continue at the last confirmed bookmark without losing any transactions. To accomplish this task, ensure that replication for the tables in question is and remains active up to the point where the first DDL change is made. Ensure that all transactions to the tables in question have been successfully replicated to the target before making any DDL changes to them.

It is important, for the production environment, that the sequence of steps in this procedure is followed. If DDL changes are done to a source table without first having processed all entries in the database log, the only possible recovery is to refresh the table in question. This procedure is focused on keeping the downtime for the replication at a minimum. Therefore, the update of the target table definition and the import of the new subscription are the last steps. It also allows for automation of the change management process from the production side.

6. Stop activity on the tables to be altered. Ensure that the application has stopped so that no more transactions are generated on the tables for which the structure changes.
7. Stop the subscription and restart it in net change mirroring mode to apply any pending transactions. This action ensures that all the transactions up to the DDL change have been applied to the target tables. When the subscription automatically stops, the replication is up-to-date (Figure 7-40).


Start Mirroring

Mirroring Methods

( ) **Continuous**

Mirroring will continue to run until you end replication.

[✓ **Scheduled End**]

Mirrors all committed changes to the indicated point in the database log and then ends replication normally.

[✓ Now]

( ) Specific Date/Time: 28.10.2011 09:57:28

( ) Specific log position:

Figure 7-40 Restart replication in net change mode

8. Implement the DDL change at the source.

9. Update source table definitions to ensure that the source table definitions have been updated for the data store. Repeat this step for all tables that had DDL changes applied (Figure 7-41 through Figure 7-44 on page 201).

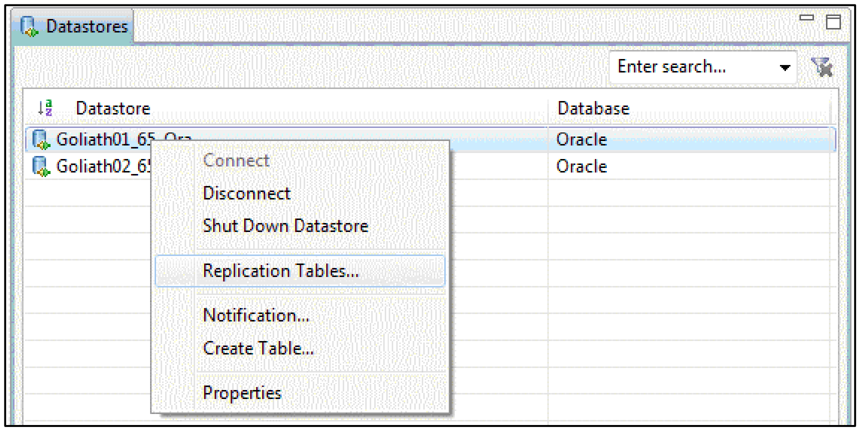


Figure 7-41 Update Source Table definition - 1

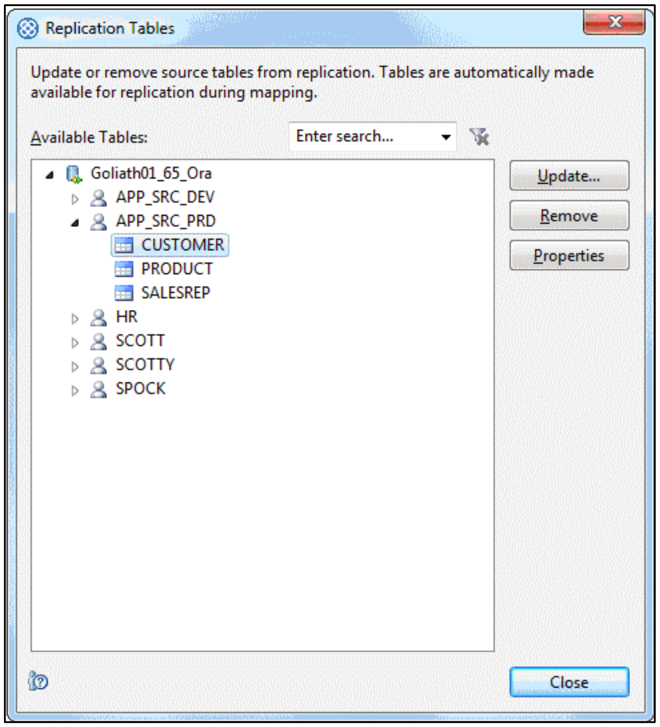
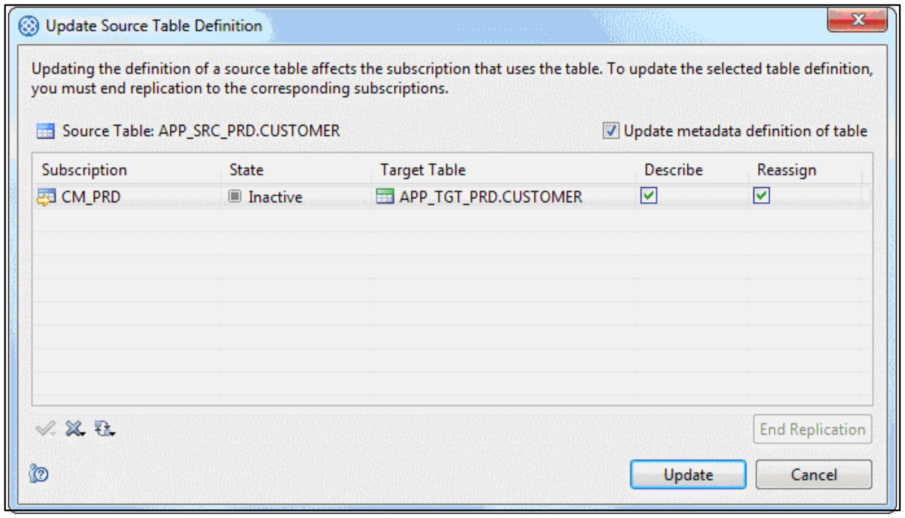

Update or remove source tables from replication. Tables are automatically made available for replication during mapping.

Figure 7-42 Update Source Table definition -2


Update Source Table Definition

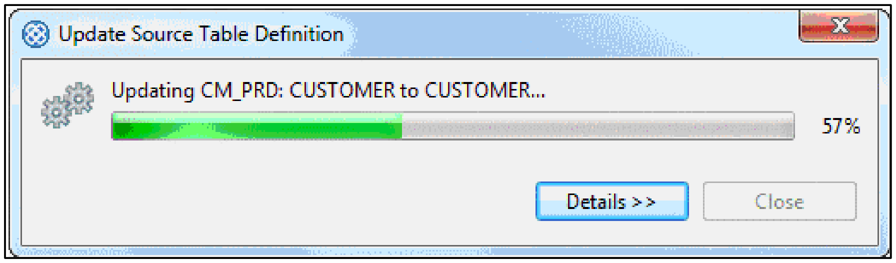
Updating the definition of a source table affects the subscription that uses the table. To update the selected table definition, you must end replication to the corresponding subscriptions.

Source Table: APP_SRC_PRD.CUSTOMER

☑ Update metadata definition of table

Subscription	State	Target Table	Describe	Reassign
CM_PRD	Inactive	APP_TGT_PRD.CUSTOMER	☑	☑

Figure 7-43   Update Source Table definition - 3

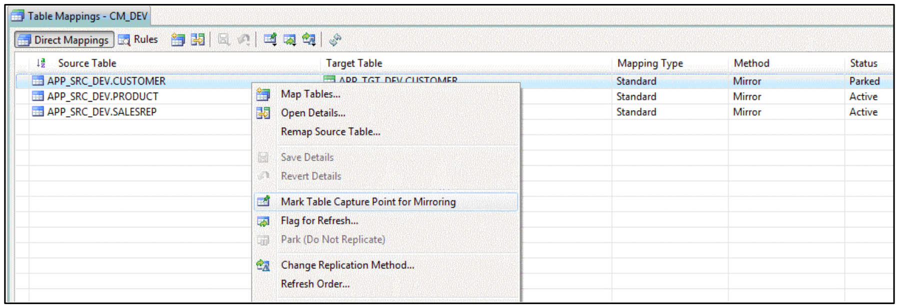

Update Source Table Definition

Updating CM_PRD: CUSTOMER to CUSTOMER...

57%

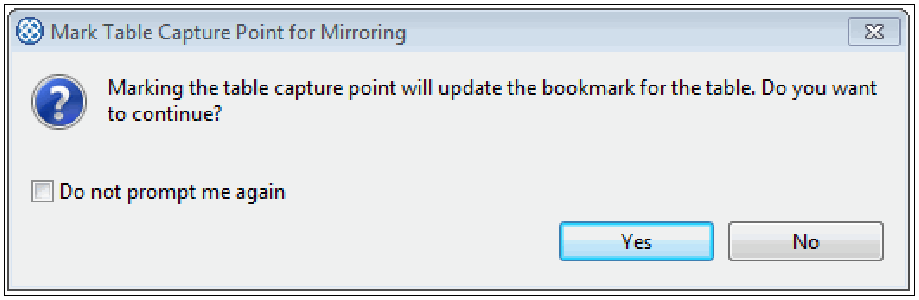
Figure 7-44   Update Source Table definition - 4

10. The last step before resuming business application activity is to set the changed tables to active again. Select the tables and click **Mark Table Capture Point for Mirroring** (Figure 7-45 and Figure 7-46).



Source Table	Target Table	Mapping Type	Method	Status
APP_SRC_DEV.CUSTOMER	APP_TGT_DEV.CUSTOMER	Standard	Mirror	Parked
APP_SRC_DEV.PRODUCT		Standard	Mirror	Active
APP_SRC_DEV.SALESREP		Standard	Mirror	Active

Figure 7-45 Mark Table Capture Point for Mirroring - 1

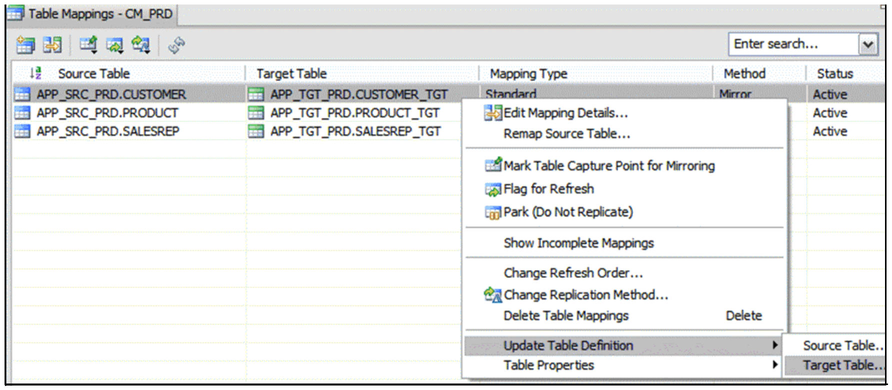


Marking the table capture point will update the bookmark for the table. Do you want to continue?

Figure 7-46 Mark Table Capture Point for Mirroring - 2

Now the business application can resume activity on the tables in question. Log entries are written in the tables' new format and also must be collected by InfoSphere CDC in the correct format.

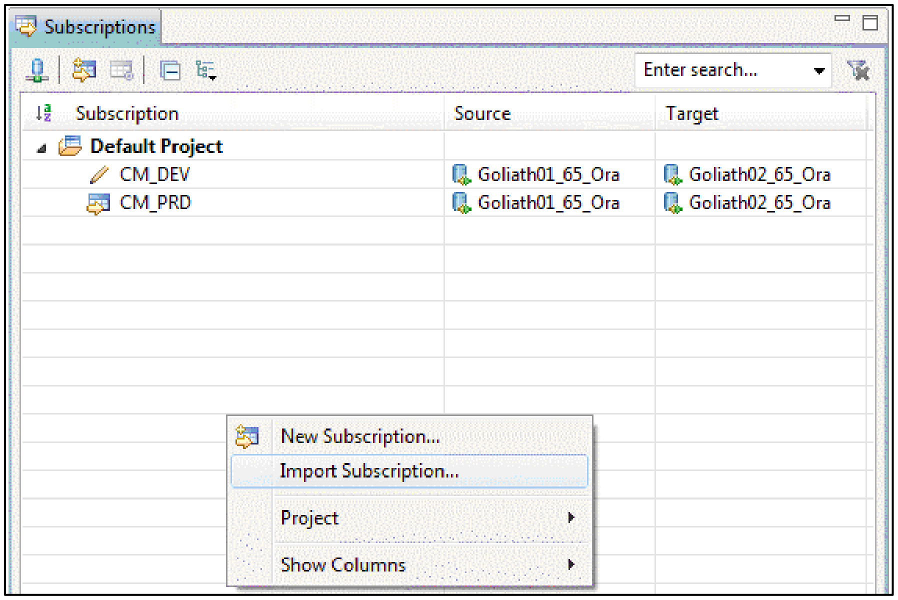
11. Implement the necessary DDL changes at the target system. Then, update the target table definition through Management Console (Figure 7-47).



Source Table	Target Table	Mapping Type	Method	Status
APP_SRC_PRD.CUSTOMER	APP_TGT_PRD.CUSTOMER_TGT	Standard	Mirror	Active
APP_SRC_PRD.PRODUCT	APP_TGT_PRD.PRODUCT_TGT			Active
APP_SRC_PRD.SALESREP	APP_TGT_PRD.SALESREP_TGT			Active

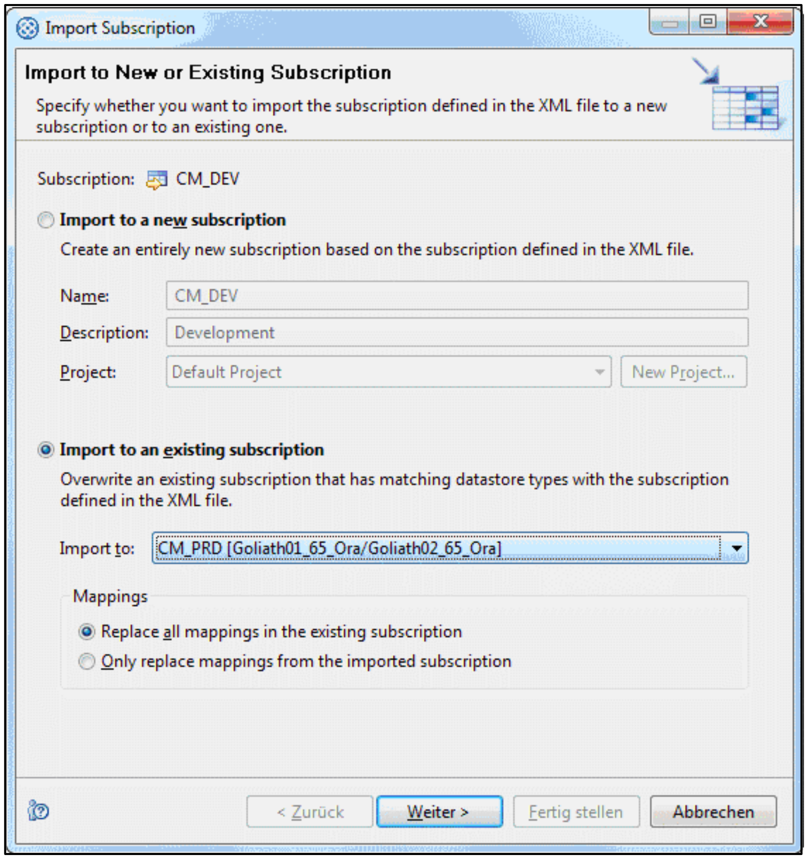
Figure 7-47 Update target table definition

12. Deploy the new version of the subscription with altered table mappings for the new columns that were exported from the development environment into the production environment. You can choose to replace the existing CM_PRD subscription and override the different schema names between the development and production environments, as shown in Figure 7-48 to Figure 7-51 on page 207.



Subscription	Source	Target
Default Project		
CM_DEV	Goliath01_65_Ora	Goliath02_65_Ora
CM_PRD	Goliath01_65_Ora	Goliath02_65_Ora

Figure 7-48 Import Altered Subscription from development environment



Import to New or Existing Subscription

Specify whether you want to import the subscription defined in the XML file to a new subscription or to an existing one.

Subscription: CM_DEV

( ) **Import to a new subscription**

Create an entirely new subscription based on the subscription defined in the XML file.

Name: CM_DEV

Description: Development

Project: Default Project

(●) **Import to an existing subscription**

Overwrite an existing subscription that has matching datastore types with the subscription defined in the XML file.

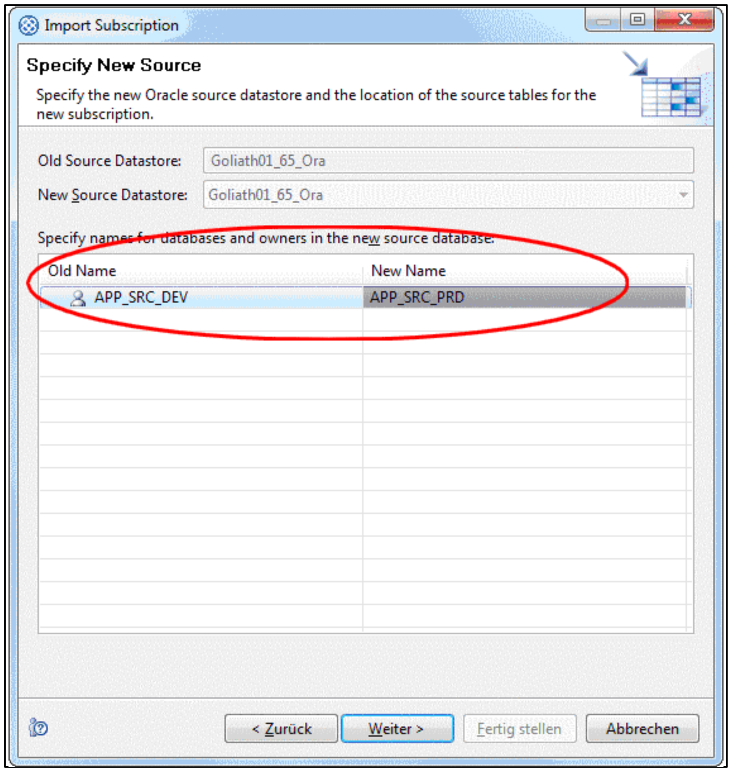
Import to: CM_PRD [Goliath01_65_Ora/Goliath02_65_Ora]

Mappings

(●) Replace all mappings in the existing subscription

( ) Only replace mappings from the imported subscription

Figure 7-49 Import Altered Subscription from development environment



Import Subscription

Specify New Source

Specify the new Oracle source datastore and the location of the source tables for the new subscription.

Old Source Datastore: Goliath01_65_Ora

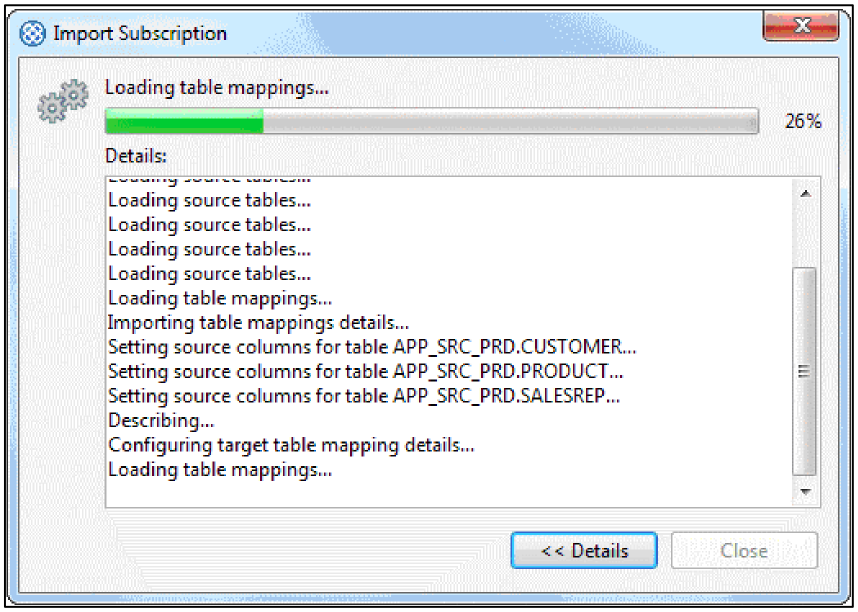
New Source Datastore: Goliath01_65_Ora

Specify names for databases and owners in the new source database:

Old Name	New Name
APP_SRC_DEV	APP_SRC_PRD

< Zurück | Weiter > | Fertig stellen | Abbrechen

Figure 7-50 Import Altered Subscription from development environment



```
Loading source tables...
Loading source tables...
Loading source tables...
Loading source tables...
Loading table mappings...
Importing table mappings details...
Setting source columns for table APP_SRC_PRD.CUSTOMER...
Setting source columns for table APP_SRC_PRD.PRODUCT...
Setting source columns for table APP_SRC_PRD.SALESREP...
Describing...
Configuring target table mapping details...
Loading table mappings...
```

Figure 7-51 Import Altered Subscription from development environment

13. Resume replication by restarting the subscription in Continuous Mirroring mode. This action replicates all pending and any new transactions from source to target.

7.6.4 DDL changes without a service window

There are differences in how source table DDL changes are dealt with across the different engines. This section elaborates on how InfoSphere CDC engines behave when DDL operations on in-scope tables are encountered in the database log.

To ensure that replication can continue without having to refresh the table, the database must log full image data for all columns in a table, including newly added columns. InfoSphere CDC relies on the full row images to be present in the database logs to ensure this action.

Linux, UNIX, and Windows engine behavior and procedure

InfoSphere CDC common (Java) engines can detect DDL statements in the database logs and, when encountered, they stop the replication. Then, the definition of the table in question must be updated before replication can be resumed.

If a discrepancy in the table structure is detected when the replication is started, the subscription fails and requires the table definition to be updated before continuing. When updating the table definition for a table whose structure has changed, the table is parked and is no longer replicated if the subscription is resumed.

To avoid losing transactions when applying DDL changes, complete the following steps:

1. Ensure that the subscription is active with no latency.
2. Complete the DDL changes on the source tables; the subscription stops and reports that a DDL change has taken place.
3. Update the source table definition; the tables are then parked.
4. Mark the table capture point for the tables in question to change the status to active again.
5. Get the current bookmark from the target side of the subscription (by running **`dmshowbookmark`**).
6. Set the subscription bookmark to the retrieved bookmark and ensure that the **`-a`** option is used to apply the bookmark to all active tables.
7. Restart the subscription.

This procedure only works if DDL statements are not mixed with DML statements for the replicated tables.

Table 7-2 shows some example statement sequences. In that example, Tables A, B, and C are replicated, while Table D is not.

Table 7-2 Sample statement sequence

Statement sequence	Comments	DDL change without refresh?
DML TABLE A DML TABLE B DML TABLE C DML TABLE D *DDL TABLE A* *DDL TABLE B* DML TABLE A DML TABLE B DML TABLE C DML TABLE D	You can continue to replicate without refreshing. Replication must be active or current until the time the first DDL change is made. If replication is running, it then stops and InfoSphere CDC definitions can be adjusted to accommodate the new table structures.	OK

Statement sequence	Comments	DDL change without refresh?
DML TABLE A DML TABLE B DML TABLE C *DDL TABLE A* DML TABLE D *DDL TABLE B* DML TABLE A DML TABLE B DML TABLE C DML TABLE D	You can continue to replicate without refreshing. Replication must be active or current until the time the first DDL change is made. If replication is running, it then stops and InfoSphere CDC definitions can be adjusted to accommodate the new table structures. The only DML statement between the two DDL changes to the tables in scope belongs to Table D, which is not replicated.	OK
DML TABLE A *DDL TABLE A* DML TABLE B DML TABLE C *DDL TABLE B* DML TABLE A DML TABLE B DML TABLE C	The subscription does not start until the definitions of both table A and B have been adjusted in the InfoSphere CDC definitions. DML for table B before the DDL for table B must be skipped to avoid malformed data and unexpected behavior of the replication. Refresh the tables in this case.	NO

Oracle considerations

One of the key elements in limiting the downtime for source table changes in an Oracle environment is making sure that the Oracle log entries are dynamically adapted to the table structure changes. As of Oracle 10, the database allows you to specify supplemental logging for all columns at the table level. In environments where the deployment of the changes in production is critical and downtime dependency between production and InfoSphere CDC must be reduced, ensure that table-level supplemental logging for all columns has been activated.

Here are SQL examples to activate table-level supplemental logging for all columns:

```
SQL> ALTER TABLE APP_SRC_PRD.CLIENT ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
SQL> ALTER TABLE APP_SRC_PRD.PRODUCT ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
SQL> ALTER TABLE APP_SRC_PRD.SALESREP ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

Table changes for DB2 on z/OS

The behavior of InfoSphere CDC for z/OS regarding DDL operations performed on in-scope tables depends on several factors:

- DDL statement type
- DB2 DATA CAPTURE CHANGES setting

- Version of InfoSphere CDC
- Version of DB2

DDL statements only impact InfoSphere CDC for z/OS if the types of DDL operations affect its ability to read the log. Here are examples of these types of DDL statements:

- Adding columns
- Modifying column formats (such as data type, length, or precision)
- Changing column types (such as CHAR to VARCHAR)
- Dropping and recreating tables
- Turning off DATA CAPTURE CHANGES

These operations cause data to be written to the log that is not received by InfoSphere CDC for z/OS or cause the log record format of the table (the layout of the data in the database logs) to change. For operations where the log record format changes, the log reader must be directed about how to proceed when a change is encountered. InfoSphere CDC metadata must be modified to accommodate the new log record format, or the log reader fails to properly decode log records after the point of the DDL change. Changes that do not affect the physical structure of the table in the log or the Data Capture Changes setting for the table do not interrupt replication.

InfoSphere CDC for z/OS can detect most DDL changes on in-scope tables only if DATA CAPTURE CHANGES is enabled on the SYSIBM.SYSTABLES system catalog table in DB2. If it is not enabled, InfoSphere CDC for z/OS is aware of DDL changes when a log record is encountered that does not match the definition in the metadata, but its actions depend on the type of DDL operation and the version of DB2.

Detailed information about how InfoSphere CDC for z/OS DB2 deals with DDL changes can be found in the Schema Evolution section in the IBM Information Center at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.cdcforzos.doc/concepts/schemaevolution.html

8 Performance analysis and design considerations

This chapter describes the high-level concepts around performance tuning and optimization in IBM InfoSphere Change Data Capture (InfoSphere CDC). This chapter includes information about understanding the InfoSphere CDC architecture, highlighting the potential bottlenecks, and documenting monitoring methods and considerations for designing InfoSphere CDC systems in large environments.

More information about CDC performance tuning can be found at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.performancetuning.doc/concepts/overviewofinfospherecdc.html

8.1 High volume between two systems

Throughput and latency are the two main considerations in large environments. It is important to first understand the concept of throughput and latency in the context of InfoSphere CDC.

8.1.1 Latency and throughput

Latency, sometimes referred to as replication lag, is the amount of time between an update applied to the source system and the same update applied to the target system. The shorter the duration, the lower the latency.

There are three types of latency:

- Data latency is the time delay in data transfer from source systems to target systems
- Analysis latency is the time delay that includes data creation and processing (ETL/ELT)
- Action latency is the time delay in getting information from data by using BI tools.

Figure 8-1 shows the types of latency.

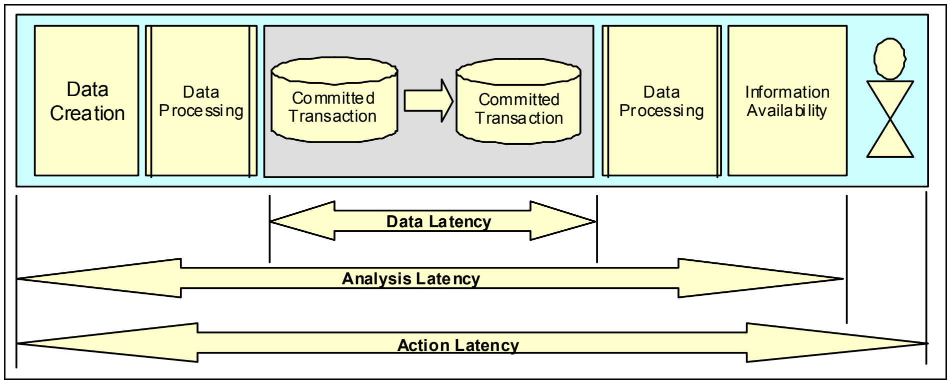


Figure 8-1 Types of latency

Throughput is the quantity of data processed within a given duration of time. High volumes of data changes from the source system require higher throughput if lower latency is wanted.

Latency is not throughput, but in high volume environments, latency is directly affected by available throughput.

8.1.2 InfoSphere CDC architecture

There are multiple components in the InfoSphere CDC architecture and there could be bottlenecks in each of these components. The major components in InfoSphere CDC include the following:

- Log reader: The log reader reads entries from the database logs (including redo logs and transaction logs) and checks to make sure that the transaction falls in-scope. When in-scope, the entries are placed in a transaction queue. Uncommitted transactions may be read, as these transactions might happen in parallel and are typically committed at different times. The transaction queue acts as a staging area for these transactions.
- Log parser: The log parser reads committed transactions from the transaction queue and extracts individual columns from the row level operations.
- Staging store: The staging store is an in-memory staging area for records processed by the log parser. The data remains in the staging store until all subscriptions have completed.

Figure 8-2 shows a high-level view of the architecture.

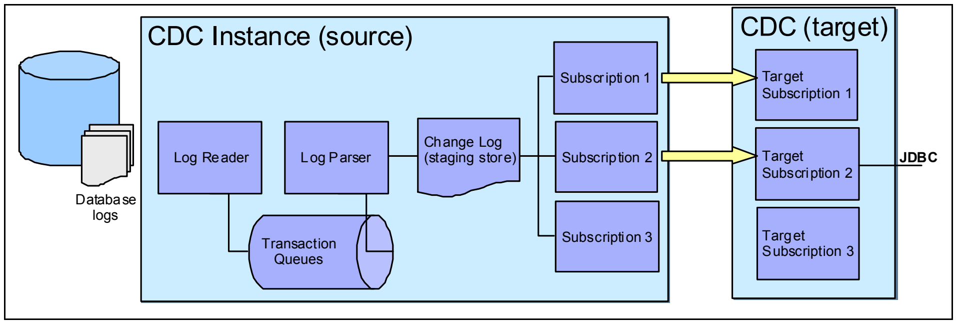


Figure 8-2 InfoSphere CDC architecture

Potential bottlenecks

InfoSphere CDC based data replication operates as a pipeline of components that move data changes along their way from the source database to the target system. As with any system, InfoSphere CDC is only as fast as the slowest component in the architecture. The following components could be potential bottlenecks in the replication pipeline:

- Log reader
- Source engine
- Network communication
- Target engine
- Target database

Latency is usually the first indication of a performance issue in a data replication environment. Here is a list of some factors that have a direct relationship to increased latency and reduced throughput:

- Size of source database transactions
- Source database delay in writing record changes to the database logs
- Relative size and performance of the source database and target systems
- Hardware factors, such as available physical memory and processor, and disk I/O performance
- Complexity of data transformations
- Available communication bandwidth

8.2 Identification of potential bottlenecks

There are three main areas that can have performance bottlenecks. They are the source systems, communications, and the target systems.

The source side bottlenecks could be due to one of the following items:

- Log reader: Reading the logs involves I/O and, in scenarios where the database logs are placed on an NFS mount, all this log data needs to be transferred across the network. Bottlenecks in the log reader can be identified by comparing the elapsed time of copying the log file to another volume to the amount of time taken to scrape the logs. This time should be comparable.
- Log parser: If the source tables have many columns or wide records, parsing can take a long time and might even reach the processor boundaries.
- Derived expressions: Usage of derived columns and expressions can result in additional processor usage. Bottlenecks due to derived expressions can be identified by replacing derived expressions with constant values.

There could also be bottlenecks in the communication layer, depending on the available bandwidth for InfoSphere CDC. These types of bottlenecks can be identified by testing an intra-system data replication. Another way of identifying a communication bottleneck is by copying a large file from the source to the target system using FTP, which should give an indication of the data replication speed. If FTP cannot use all of the available bandwidth, the same constraint applies for InfoSphere CDC.

A `ping` command should not be used to measure throughput, as the command is used for small packets of data.

There is also a potential for bottlenecks in the target side. Target system-related bottlenecks can be identified by disabling insert, update, and delete operations for individual tables. This method does all the processing in the InfoSphere CDC layer except for modifying the target system tables.

The InfoSphere CDC Management Console has a feature, the Performance Monitor, which helps identify bottlenecks in the described areas. This feature can be used if the InfoSphere CDC version of both the source and target engine is Version 6.5 or higher.

8.3 Performance monitoring in InfoSphere CDC environments

Diagnosing and resolving performance issues is an iterative process in InfoSphere CDC. High latency and resulting performance issues can be identified using one of the performance monitoring tools described in this section.

8.3.1 Performance monitoring using the Management Console

Management Console includes an event log and monitoring tools. The event log can be used to examine InfoSphere CDC event messages. The monitoring tool allows you to continuously monitor replication operations and latency. The monitor in Management Console is meant for time-critical working environments that require continuous analysis of data movement and all statistics are real-time statistics. The monitoring data can be exported for historical analysis.

After configuring the replication environment and starting the data replication process in InfoSphere CDC, you can monitor and analyze replication activities using the Monitoring perspective.

8.3.2 System monitoring tools

Any system monitoring tool can be used to identify high-level performance issues attributed to processor utilization, memory usage, and any I/O bottlenecks.

8.4 Using workflow for performance issues

There are three high-level steps to identifying and resolving performance issues by analyzing the InfoSphere CDC workflow. These steps are:

1. Configure visual cues to highlight latency issues.

   InfoSphere CDC shows a visual cue in the Monitoring perspective when subscription latency exceeds a predetermined value based on business requirements. The latency threshold can be configured in InfoSphere CDC Management Console by completing the following steps:

   a. Click **Configurations** → **Subscriptions**.

   b. Right-click a subscription and select **Latency Thresholds**.

   c. Select the **Notify when latency crosses these thresholds** check box.

   d. Specify a value in **Warning threshold (minutes)**.

   e. Specify a value in **Problem threshold (minutes)**.

   f. Click **Set Notification**.

2. Overview of data replication problem identification.

   Look at the high-level system overview to monitor the data replication process for initial analysis of the data latency issues. To see a data replication system overview, complete the following steps:

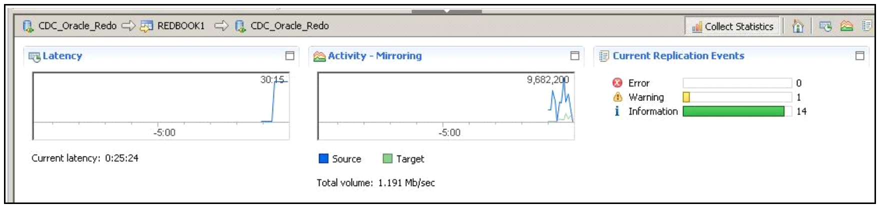
   a. Click **Monitoring** → **Subscriptions**.

   b. Select the subscription that needs to be monitored.

   c. Click **Mirroring** or **Refresh** for the selected subscription.

   d. Right-click the subscription and select **Collect Statistics**.

   e. Double-click the titles at the upper right to show an overview of the replication environment. This action also enables quick identification of latency issues that might affect replication performance.



Current latency: 0:25:24

Total volume: 1.191 Mb/sec

Error	0
Warning	1
Information	14

Figure 8-3 Problem identification

3. Analyze performance statistics.

   Further analysis can be performed by collecting additional metrics at the instance level. To collect additional metrics, complete the following steps:

   a. Select a subscription that is already collecting real-time statistics.

   b. Right-click the subscription and click **Show in Performance View**.

   c. Select **Statistics Checkboxes**. Up to 10 metrics can be chosen.

   d. Click **Collect Data**. Any bottleneck can be shown at the top of this view. A bar next to the name of the InfoSphere CDC component identifies the bottleneck. With InfoSphere CDC, only one component can have a bottleneck at any given point in time.

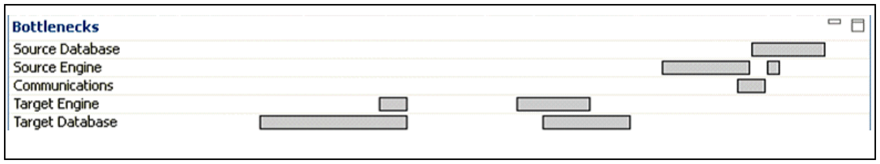


Figure 8-4 Bottlenecks

8.5 Installation considerations

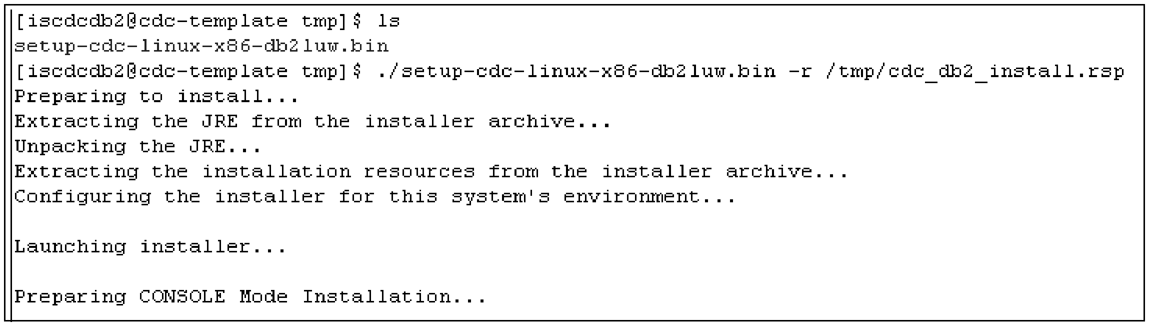
Large-scale distribution and consolidation implementations of InfoSphere CDC have many source and target instances. Each of these source and target systems need to have a InfoSphere CDC instance installed in them. It might not always be feasible to manually deploy the solution for every server and database in the entire landscape.

8.5.1 Silent installations and instance creation

The InfoSphere CDC engine supports automated deployment and instance configuration using scripting. InfoSphere CDC uses a response file to create an InfoSphere CDC instance for all databases it needs to use as source or target systems. Before being able to remotely deploy InfoSphere CDC in an automated way, the following preparatory steps need to be conducted:

1. Create template databases (for source and target systems) for the installation of InfoSphere CDC. The InfoSphere CDC user that holds the metadata must be created and the databases must have the correct settings for (supplemental) logging and archiving the transaction logs. A script to do all the database preparation steps enables this step to be automatically performed at the remote site.

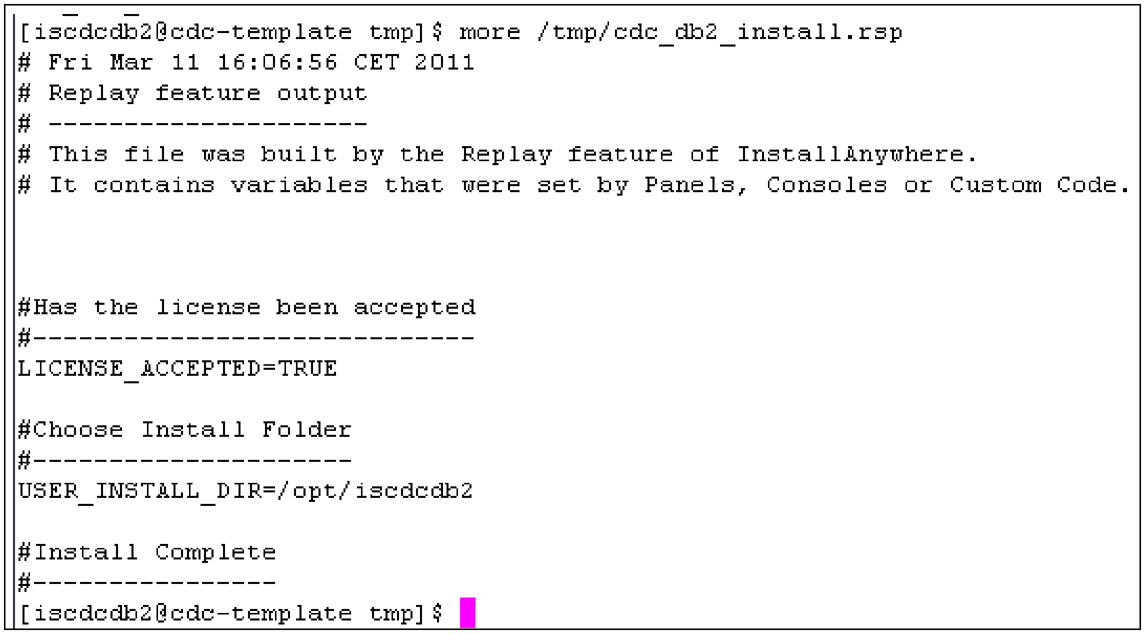
2. Install InfoSphere CDC on the server that holds the template database and specify the **-r responsefile** option to create a response file to be used to install the production servers. You need to do this step for both the source and target template servers. An example response file for installation is shown in Figure 8-5.



```
[iscdcdb2@cdc-template tmp]$ ls
setup-cdc-linux-x86-db2luw.bin
[iscdcdb2@cdc-template tmp]$ ./setup-cdc-linux-x86-db2luw.bin -r /tmp/cdc_db2_install.rsp
Preparing to install...
Extracting the JRE from the installer archive...
Unpacking the JRE...
Extracting the installation resources from the installer archive...
Configuring the installer for this system's environment...

Launching installer...

Preparing CONSOLE Mode Installation...
```

Figure 8-5 Response file installation

After the installation is finished, a response file is generated. An example is shown in Figure 8-6.



```
[iscdcdb2@cdc-template tmp]$ more /tmp/cdc_db2_install.rsp
# Fri Mar 11 16:06:56 CET 2011
# Replay feature output
# ---------------------
# This file was built by the Replay feature of InstallAnywhere.
# It contains variables that were set by Panels, Consoles or Custom Code.



#Has the license been accepted
#-----------------------------
LICENSE_ACCEPTED=TRUE

#Choose Install Folder
#---------------------
USER_INSTALL_DIR=/opt/iscdcdb2

#Install Complete
#----------------
[iscdcdb2@cdc-template tmp]$
```

Figure 8-6 Sample response file

3. Create the InfoSphere CDC instance for the template databases.

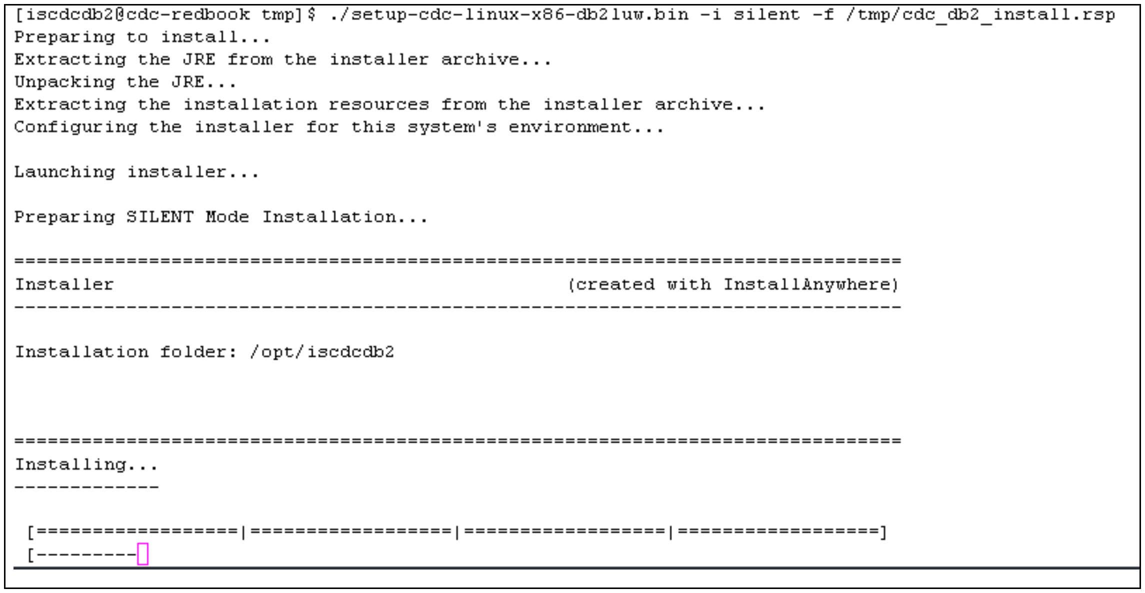
4. Configure the subscriptions and table mappings.
5. Export the instance configuration by running **`dmexportconfiguration`**. Running this command prompts you for the password of the database user that is the owner of the InfoSphere CDC metadata that will be used on the production installation when the instance is imported. The password is encrypted in the generated XML file. You need to do this step for both source and target instances.

   If needed, parameters such as <name>, <tcpPort>, <dbName>, and <dbSchema> can be overridden in the XML file to accommodate the production database settings. As the database user name and password are encrypted in the XML file, these items should not be changed. Optionally, remove the InfoSphere CDC instances and installation from the template servers.

For unattended deployment at the remote site, complete the following steps:

1. Prepare the production databases (source and target) for the installation of InfoSphere CDC. The InfoSphere CDC user that holds the metadata must be created and the database must have the correct settings for (supplemental) logging and archiving transaction logs. If a script has been created for this task, run the script.

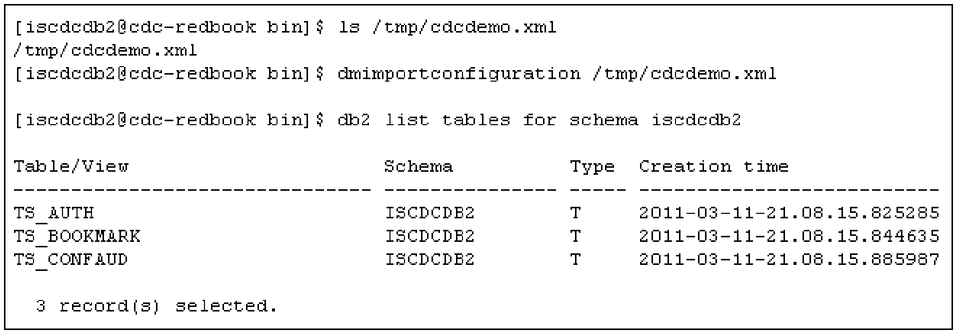
2. Install InfoSphere CDC on the production servers using the response files that were created on the template servers (specify **`-i silent -f responsefile`**). An example of the response file generation is shown in Figure 8-7.



```
[iscdcdb2@cdc-redbook tmp]$ ./setup-cdc-linux-x86-db2luw.bin -i silent -f /tmp/cdc_db2_install.rsp
Preparing to install...
Extracting the JRE from the installer archive...
Unpacking the JRE...
Extracting the installation resources from the installer archive...
Configuring the installer for this system's environment...

Launching installer...

Preparing SILENT Mode Installation...

===============================================================================
Installer                                        (created with InstallAnywhere)
-------------------------------------------------------------------------------

Installation folder: /opt/iscdcdb2



===============================================================================
Installing...
-------------

 [==================|==================|==================|==================]
 [----------
```

Figure 8-7 Response file generation

3. After the installation is finished, change the directory to the `cdc_home/bin` directory and import the instance configuration by running **`dmimportconfiguration`**. This step needs to be done for both source and target instances. After the import of the instance configuration is finished, the metadata tables are created in the database. An example of the instance configuration is shown in Figure 8-8.



```
[iscdcdb2@cdc-redbook bin]$ ls /tmp/cdcdemo.xml
/tmp/cdcdemo.xml
[iscdcdb2@cdc-redbook bin]$ dmimportconfiguration /tmp/cdcdemo.xml

[iscdcdb2@cdc-redbook bin]$ db2 list tables for schema iscdcdb2

Table/View                      Schema          Type  Creation time
------------------------------- --------------- ----- --------------------------
TS_AUTH                         ISCDCDB2        T     2011-03-11-21.08.15.825285
TS_BOOKMARK                     ISCDCDB2        T     2011-03-11-21.08.15.844635
TS_CONFAUD                      ISCDCDB2        T     2011-03-11-21.08.15.885987

  3 record(s) selected.
```

Figure 8-8 InfoSphere CDC instance configuration

Upon finalizing the import, the InfoSphere CDC instance is automatically started and is ready to be used. For manual and automated configuration options, see Chapter 9, “Customization and automation” on page 231.

8.6 Design considerations

Sometimes a standard InfoSphere CDC configuration might not scale based on the workload or the latency might be too high. In such instances, alternative architectures need to be considered. Some of these alternative architectures might include using:

- Multiple parallel subscriptions
- Multiple InfoSphere CDC instances
- An n-tiered architecture
- Cascading replication
- Continuous scraping

Each of these alternative architectures is applicable and is suitable under certain circumstances. However, not all platforms support all of these alternative architectures. The remainder of this section goes through each of the alternative architectures and highlights any exceptions or challenges associated with them.

At the time of the publication of this book, the Performance Monitoring and Tuning guide for InfoSphere CDC is available at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.performancetuning.doc/concepts/overviewofinfospherecdc.html

See this document for more advanced monitoring and tuning techniques.

8.6.1 Using multiple parallel subscriptions

Data replication from a single table can be optimized by using multiple subscriptions. The available data can be partitioned by using row filtering so multiple subscriptions can operate concurrently to replicate changes to the target systems. This scenario is most suitable when there is a numeric column that uniquely identifies every row in the table. This column can be used to partition the data across multiple subscriptions.

This scenario does not work if the column used to partition the data is updated.

A Java based user exit (see 9.7, “User exits” on page 369 for more information about user exits) can be used to tag rows of data to be directed to corresponding subscriptions. This task is achieved by implementing a MODULO function on the numeric column that uniquely identifies every row of data in the table. As an example, to partition the data, such as the data shown in Table 8-1, into three subscriptions, the Java based user exit performs a MODULO 3 on the numeric key column and uses the value as a predicate filter for the three subscriptions. Subscription 1 has a predicate of MODULO_CUSTOMER_ID=1, Subscription 2 has a predicate of MODULO_CUSTOMER_ID=2, and Subscription 3 has a predicate of MODULO_CUSTOMER_ID=0.

Table 8-1 Sample source table

Customer ID	MODULO_CUSTOMER_ID	Customer name
31	1	Customer 31
32	2	Customer 32
33	0	Customer 33
42	2	Customer 42
76	1	Customer 75

The MODULO_CUSTOMER_ID column is derived by applying the MODULO 3 function on the Customer ID column from within a Java based user exit.

The three subscriptions for this source table have row-based filtering on the MODULO_CUSTOMER_ID column. Subscription 1 handles rows with Customer IDs (31,76), Subscription 2 handles rows with Customer ID (32,42), and Subscription 3 handles the row with Customer ID 33. All three subscriptions populate the same target system, so the throughput is much higher, which reduces latency and increases performance. The architecture for this configuration is shown in Figure 8-9.

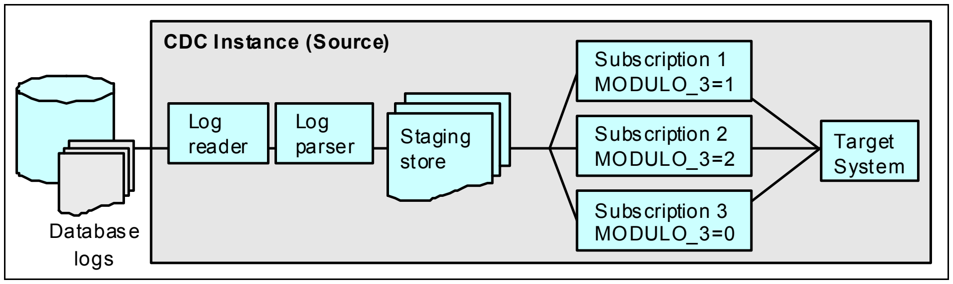


Figure 8-9 Multiple parallel subscription architecture

At the time of the publishing of this book, InfoSphere CDC V6.5.2 was released, which introduced fast apply techniques to improve the throughput when applying transactions on the target database. More information can be found at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m2/topic/com.ibm.cdcdoc.performancetuning.doc/concepts/improvingtargetdatabaseperformancewithfastapply.html

8.6.2 Using multiple InfoSphere CDC instances

Distributing the workload across multiple InfoSphere CDC instances can also be used to reduce latency and increase throughput (Figure 8-10).

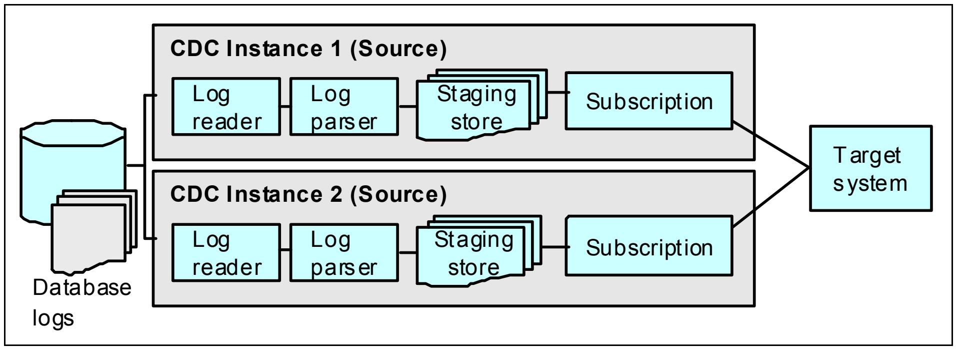


Figure 8-10 Multiple InfoSphere CDC instances architecture

Every InfoSphere CDC instance has its own log reader, which might have some performance implications on the source side, but the overall benefit of higher throughput in large data replication environments out-weighs the additional work done in the source systems.

Each InfoSphere CDC instance in the architecture shown in Figure 8-10 could either be set up to replicate different tables or they might be set up to replicate the same table with different row-based filters. Depending on the data volumes in the individual tables, one or a combination of both of the methodologies can be used.

8.6.3 Using an n-tiered architecture

N-tiered data replication in InfoSphere CDC means that the InfoSphere CDC instance is on a different server than the database it sources or populates. An n-tiered architecture can operate as follows:

- InfoSphere CDC reads logs coming from a different server (remote log reading).
- InfoSphere CDC applies transactions to a database that is on a different server (remote apply).

Typical applications of an n-tiered architecture

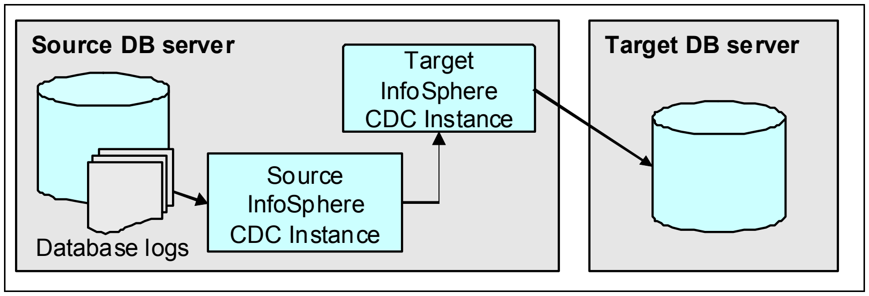
The three common scenarios for the n-tiered architecture are:

- Compliance, security, and ownership reasons, including:
  - Database administrators not comfortable installing third-party applications (InfoSphere CDC) on the database server.
  - A database server is owned by the business and they want to mitigate any risk in the disruption of the database servers.
- Software restrictions on the database server. The operating system patch levels might not be at a required level for InfoSphere CDC.
- Source system is at capacity.
  - No additional processing capability for InfoSphere CDC.
  - The source system might be an existing system that does not have sufficient cycles to process high volumes of database logs.

The InfoSphere CDC n-tiered architectures are differentiated by the location of the source and target instances.

InfoSphere CDC instances on source database server

In this configuration, the InfoSphere CDC instance for both the source and target systems are located in the source database server. This configuration is called Option 1 (Figure 8-11).

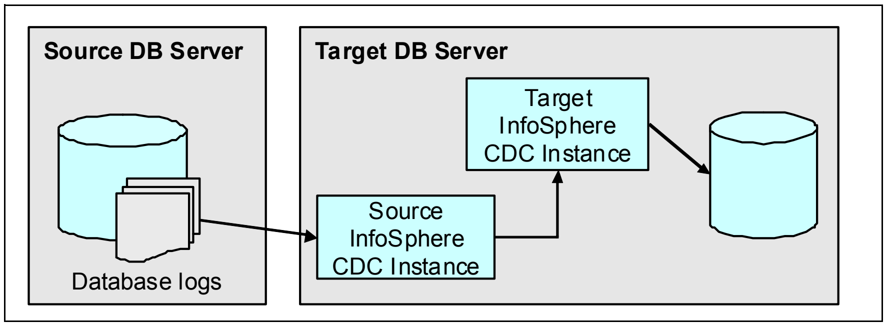


Source DB server
Target InfoSphere CDC Instance
Source InfoSphere CDC Instance
Database logs
Target DB server

Figure 8-11 N-tiered architecture - Option 1

InfoSphere CDC instances on the target database server

In the configuration shown in Figure 8-12, the InfoSphere CDC instance for both the source and target systems are on the target database server. This option is called Option 2.



Source DB Server
Database logs
Target DB Server
Target InfoSphere CDC Instance
Source InfoSphere CDC Instance

Figure 8-12 N-tiered architecture - Option 2

InfoSphere CDC instances on separate database servers

In the configuration shown in Figure 8-13, the InfoSphere CDC instances for both the source and target systems are on a centralized InfoSphere CDC server.

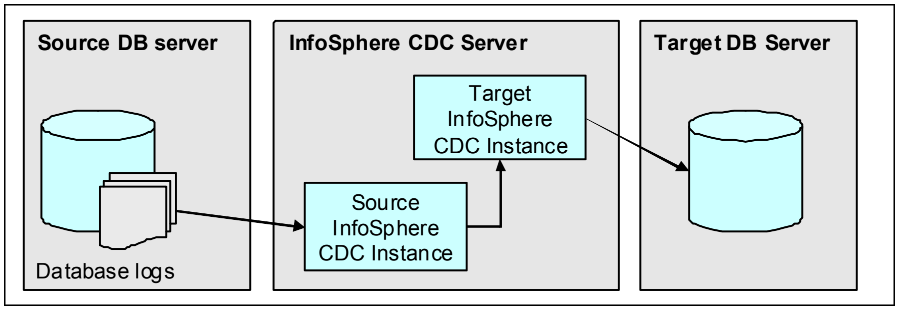


Figure 8-13 N-tiered architecture - Option 3

8.6.4 Using cascading replication to spread the workload

Using cascading replication, the performance and latency can be vastly improved as a result of the multiplier effect. Similar to having multiple subscriptions for a given table with row level filters, this method distributes the workload across multiple subscriptions. This method can be used with the multiple subscription method. Use the Java based user exit to perform a MODULO on a numeric column in the source table and replicate the data based on different predicate values to different target systems. These target systems can then be used as a source to load and consolidate data rows in the final destination table. Cascading replication is shown in Figure 8-14.

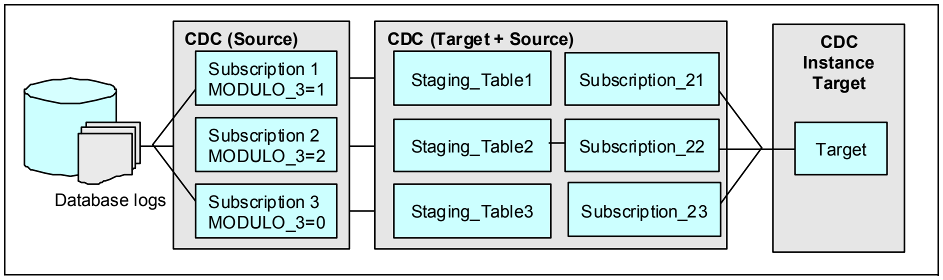


Figure 8-14 Cascading replication

8.6.5 Continuous scraping

In large distribution environments, one of the biggest challenges is purging source database logs. If all target systems are not simultaneously active or if there are unstable network connections between source and target, there could be some target systems that are not synchronized with the source.

InfoSphere CDC needs access to the database log entry that is represented by the oldest bookmark of any of the subscriptions. When there is a network outage, InfoSphere CDC stops reading the database transaction logs and the bookmark of the subscription is no longer advanced. After it is restarted, the subscription requests the bookmark from the target system and restarts from the position corresponding to the bookmark in the log. If this log entry is no longer available because log files have been purged, the replication does not start. The log files must be restored for normal operation to continue. In some instances, the database logs may no longer be recovered and a resynchronization (refresh) of all tables must be done.

Besides the risk of losing changed data, a prolonged outage of the replication could result in resource utilization spikes when subscriptions are restarted. When replication is started in mirror mode, InfoSphere CDC always attempts to catch up to the head of the database transaction log and thus use any resource it can obtain from the operating system to achieve this goal. To optimize this operation and reduce the impact of InfoSphere CDC in large distribution environments, you should consider enabling the Continuous Capture feature, which is available in all engines supporting single scrape.

Enabling continuous capture causes the log reader process to be separated from the replication of the logical database operations. The InfoSphere CDC source engine actively reads the database logs at all times and processes the in-scope entries. Entries (operations) are sent to transaction queues and, once committed, they are parsed and accumulated into the staging store. The staging store is kept until all the subscriptions have completed processing. Even when there are no subscriptions active, the engine continues to read and process the database log entries and strives to stay at the head of the log.

When the inactive subscriptions are restarted, the parsed log entries are already in the staging store and can be processed immediately by the subscriptions, saving a fair amount of processing time. The staging store entries are cleaned up as soon as subscriptions no longer need them.

As the staging store on the source server is accumulating the log changes, its size increases. You must plan your source environment to have sufficient disk space. More importantly, the maximum size of the staging store that was specified when the instance was created must be equal to or lower than the available disk space.

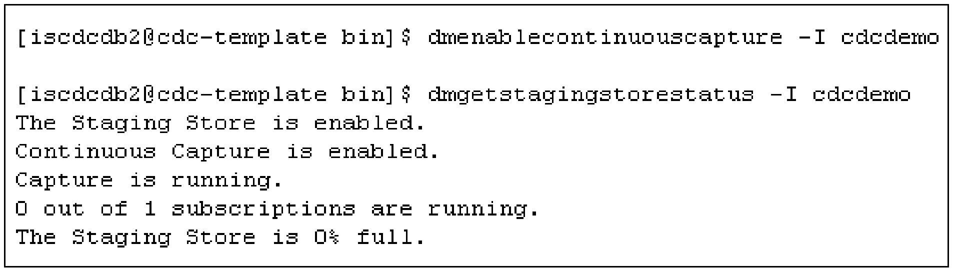
Quota and disk capacities: Running out of quota for the staging store pauses scrape activities. However, running out of physical disk space causes the log scraper to end abnormally and render the staging store corrupted. In such an event, the staging store must be cleared (by running **`dmclearstagingstore`**) and the continuous capture process must start reading the logs from the oldest bookmark.

Continuous scraping of the database logs is incompatible with the independent staging store capability of InfoSphere CDC. If you choose to enable continuous scraping, all subscriptions sourcing the instance must share the single scrape staging store. Should your environment need a more flexible setup where some subscriptions must be able to have their own independent staging store, consider creating an additional InfoSphere CDC instance to run those subscriptions.

The data store system parameter **`staging_store_can_run_independently`** controls the use of a single scrape staging store. The default setting of this parameter is `true`, which means that subscriptions can create their own log reader process if a database log entry cannot be found in the staging store. Changing the parameter to `false` causes all subscriptions to work off the same staging store, in effect creating a single scrape staging store.

After the single scrape staging store has been activated, the **`dmenablecontinuouscapture`** and **`dmdisablecontinuouscapture`** commands can be used to start and stop continuous scrape. At all times, the staging store status can be monitored using the **`dmgetstagingstorestatus`** command.

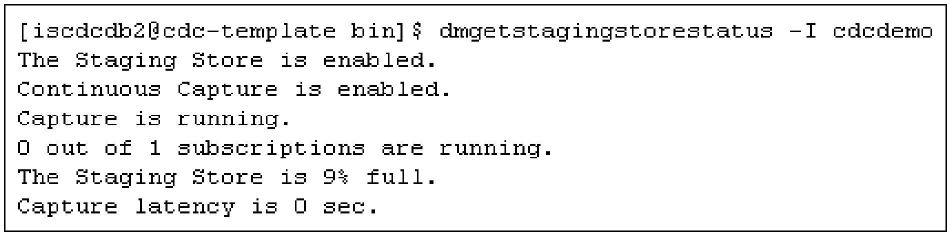
Enabling continuous capture and check status is shown in Figure 8-15.



```
[iscdcdb2@cdc-template bin]$ dmenablecontinuouscapture -I cdcdemo

[iscdcdb2@cdc-template bin]$ dmgetstagingstorestatus -I cdcdemo
The Staging Store is enabled.
Continuous Capture is enabled.
Capture is running.
0 out of 1 subscriptions are running.
The Staging Store is 0% full.
```

Figure 8-15 Enable continuous capture and check status

One of the subscriptions that is sourcing this InfoSphere CDC instance has a table mapped for Mirror and transactions are being generated, but the subscription is inactive. Now, after transactions are generated and the staging store status is monitored, it gradually starts filling up. In Figure 8-16, the staging store is 9% full and the capture process is reading the head of the log, given the fact that the capture latency is 0%.



```
[iscdcdb2@cdc-template bin]$ dmgetstagingstorestatus -I cdcdemo
The Staging Store is enabled.
Continuous Capture is enabled.
Capture is running.
0 out of 1 subscriptions are running.
The Staging Store is 9% full.
Capture latency is 0 sec.
```

Figure 8-16   Staging store status

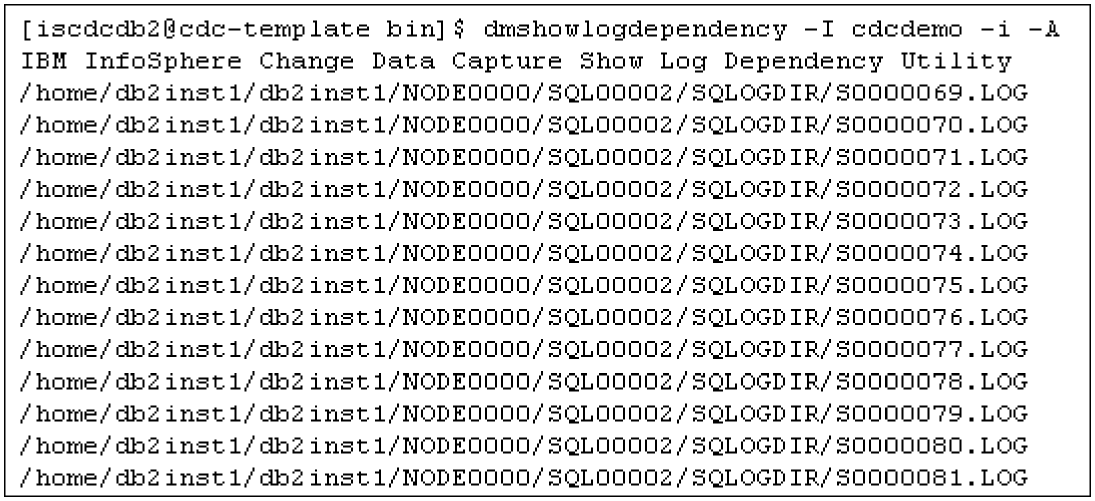
Should the InfoSphere CDC instance be stopped at some point, the single scrape staging store is persisted to disk. When restarting the instance, the continuous capture is automatically restarted and continues from the last read log entry.

Abnormal termination: Abnormal termination of the InfoSphere CDC instance renders the staging store corrupted. Always shut down the InfoSphere CDC instance by running **`dmshutdown`** to avoid the situation where the staging store must be cleared and the log reader must restart from the earliest bookmark position.

Consider the log retention policy that can be adapted when using continuous capture. The suggested approach for log retention is to always keep the database logs until transactions have been applied onto the target to avoid restoring them in the event the InfoSphere CDC instance terminates abnormally. However, in large distribution environments where the target systems are not always active, imposing the recommended log retention policy might incur challenges in terms of disk space and manageability of the database server.

The **`dmshowlogdependency`** command can be used to list the database logs that are still required by InfoSphere CDC. The **`-i -A`** option of this command lists all database logs starting from the oldest bookmark.

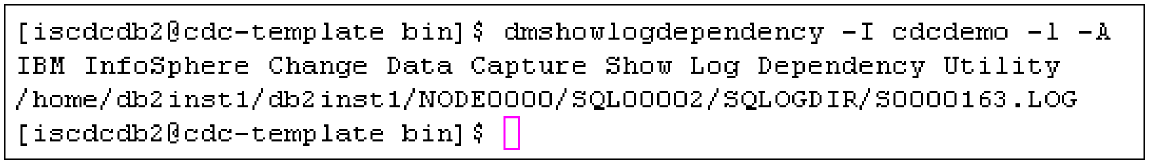
Figure 8-17 shows a list of database logs that are still required by InfoSphere CDC.



```
[iscdcdb2@cdc-template bin]$ dmshowlogdependency -I cdcdemo -i -A
IBM InfoSphere Change Data Capture Show Log Dependency Utility
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000069.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000070.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000071.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000072.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000073.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000074.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000075.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000076.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000077.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000078.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000079.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000080.LOG
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000081.LOG
```

Figure 8-17 Identifying database logs required by InfoSphere CDC

As an alternative to choosing a log retention policy that keeps all database logs on the system that potentially must be read later, you could consider purging all database logs before the log file that is currently read by the continuous capture process. The **dmshowlogdependency -l -A** command shows the currently read log file (Figure 8-18).



```
[iscdcdb2@cdc-template bin]$ dmshowlogdependency -I cdcdemo -l -A
IBM InfoSphere Change Data Capture Show Log Dependency Utility
/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLOGDIR/S0000163.LOG
[iscdcdb2@cdc-template bin]$
```

Figure 8-18 Identifying logs that can be purged

Purging the database logs: If the database logs are purged up to the currently processed log file (disregarding the bookmark), when there is a system or product failure, the logs must be restored or all tables need to be refreshed.

For more information about database log retention, see Chapter 7, “Environmental considerations” on page 137.

9 Customization and automation

This chapter elaborates on the various methods to configure, operate, and monitor IBM InfoSphere Change Data Capture (InfoSphere CDC).

This chapter first provides a brief introduction to the different components that can be used to manage the solution and how they are related, including the Management Console and Access Server. You should be aware of how InfoSphere CDC can be controlled by each of the components and where these components are located.

This chapter goes into a higher level of detail about the configuration and monitoring tasks using the command line and Java API. Sample code is provided to make it easier for you to create your own customized scripts or code to fit your environment.

Finally, this chapter describes how to implement user exits as part of the replication process for the various platforms supported by InfoSphere CDC. Again, various examples are provided to help you develop your own development and deployment activities.

9.1 Options for managing InfoSphere CDC

InfoSphere CDC offers various options to configure and monitor the replication environment, each having their own use and applicability, depending on your requirements.

The following sections describe the various interfaces that are delivered as part of the InfoSphere CDC product, providing a high-level overview of the common uses of each interface and a more comprehensive explanation of the automation options that are provided.

9.2 Management Console GUI

The InfoSphere CDC Management Console GUI is the most commonly used interface for configuration, operation, and monitoring of InfoSphere CDC. It provides various powerful functions to create subscriptions, map tables, start and stop subscriptions, and monitor replication throughput and latency.

All clients use the GUI in the InfoSphere CDC landscape, at least for development of subscriptions and table mappings. For most clients, this interface is the sole interface by which InfoSphere CDC is configured, operated, and monitored.

This section does not elaborate on all the features of the Management Console GUI in this chapter, but describe some of the common practices in larger scale environments.

For monitoring throughput and replication latency, the Management Console is the preferred interface in most environments. In larger scale environments with many source and target servers, many subscriptions or substantial numbers of mapped tables, other methods for monitoring subscriptions, such as the InfoSphere CDC Server Command-Line Interface (CLI) or the API, might be preferred.

In many environments, clients implement a predefined path for the deployment of their business applications. The deployment starts with changes in a development environment, promoting those changes through acceptance testing, and moving those changes into production. The changes that are made to the business application logic or underlying database might also affect the InfoSphere CDC configuration. Promoting applications and database changes through a succession of development stages must also be done for InfoSphere CDC.

Before doing any promotion to the successive environments, the subscription definitions and table mappings are changed in a development environment until they satisfy the requirements of the design. The InfoSphere CDC Management Console GUI is the preferred method for performing any of the development changes, using an interface that gives you both high-level and detailed insight into the replication configuration. When business application changes are made in the development environment, the InfoSphere CDC configuration in the associated stage is typically reviewed and adjusted as necessary.

When the business application or database changes and changes to the InfoSphere CDC configuration have been tested and approved, they are promoted to the successive development environment by, for example, a versioning solution. The Management Console offers the ability to promote subscriptions and even individual table mappings to the next environment. A wizard that leads you through a number of choices for the new source data store, the new target data store, and even different schemas to be used in the source and target tables. Another method for promoting subscriptions is to export the definitions to an XML file from within the Management Console GUI and import the XML file in the successive environment, again using the GUI.

In larger scale environments, the promotion wizard or export / import function cannot be used to deploy changes into testing and production environments because of restrictions in company policies or because of timing constraints when implementing these changes. This situation is especially true if there is a requirement for lights-out operations and deployments.

9.3 Management Console commands

The Management Console and Access Server provide a CLI for scripted control of subscriptions and the InfoSphere CDC configuration without needing physical or command-line access to the servers that have the source and target InfoSphere CDC engines. The Management Console CLI is included in Access Server installations, which are available on Linux, UNIX, and Windows platforms, and also on Management Console installations, which are available only on Windows platforms.

This section describes the common uses for the Management Console commands and provides an example of how a subscription can be started using this interface. More information about the available commands and scripting possibilities can be found in the API and Commands Reference at the following address:

http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.mcapiandcommands.doc/concepts/commandsreference.html

9.3.1 Common uses for the Management Console commands

Usually, the Management Console CLI is used is when the InfoSphere CDC operators do not have access to the source (production) server, but are responsible for starting and stopping InfoSphere CDC at given times and monitoring its status. By using this interface, InfoSphere CDC can be operated using scripts that are suitable for unattended execution without access to the production servers.

The Management Console CLI offers limited abilities for configuring subscriptions; it allows you to add tables to the catalog, select tables for subscriptions, describe subscriptions, and assign source tables to target tables as long as table structures are alike. It is difficult to create scripts with conditional processing based on the outcome of individual Management Console commands. Therefore, you should use the Java API to control the InfoSphere CDC configuration in an automated manner.

9.3.2 Compiling Management Console command scripts

If you decide to use Management Console command scripts to control InfoSphere CDC, consider first stepping through the consecutive commands using the **`online.bat`** (or **`online`** for Linux and UNIX) interface. This procedure validates the steps and provides feedback on every step.

For example, you might want to control the start of subscriptions from your client environment by completing the following steps:

1. Connect to the Access Server (similar to starting the Management Console and entering the user and password for the Access Server).
2. Connect to the Source data store.
3. Start the subscription.
4. Shows the subscription status.
5. Disconnect from the Source data store.
6. Disconnect from the Access Server.

The script that accomplishes these steps is shown in Example 9-1.

Example 9-1 Script for starting subscriptions

```
C:\IBM\InfoSphere CDC_MC> type \MCScripts\startREDSCRIPT.txt
// Connect to the Access Server
connectServer(admin,passw0rd,172.16.74.130,10101);
// Connect to the Source data store
connectAgent(CDC_Oracle_Redo);
// Start the subscription
startMirror(CDC_Oracle_Redo,REDSCRIPT,continuous);
// Get the subscription status
getSubscriptionStatus(CDC_Oracle_Redo,REDSCRIPT);
// Disconnect from the Source data store
disconnectAgent(CDC_Oracle_Redo);
// Disconnect from the Access Server
disconnectServer();
// Exit the script
exit();
```

To run the script, you can run the **`online`** command (**`online.bat`** for Windows) from the Management Console or Access Server directory and redirect the input to the command as follows:

```
C:\IBM\InfoSphere CDC_MC> online.bat < \MCScripts\startREDSCRIPT.txt
Copyright (c) 1999-2010 IBM Corporation
Welcome to the IBM InfoSphere Change Data Capture Console (Version
6.5.1505.0)
>
REDSCRIPT Starting
```

As you can see, the command script has user readable user names and passwords, which is a security exposure. To avoid this exposure, a compilation command is provided to translate the base command script to an executable program, a batch file for Windows, or a shell file for Linux or UNIX. Only store the compiled executable programs on the server that has Management Console or Access Server, and keep the readable scripts in a secure place.

Perform the compilation by running the following command:

```
C:\InfoSphere CDC_MC>online.bat -c
\MCScripts\startREDSCRIPT.txt_\MCScripts\compiled\startREDSCRIPT.bat
```

The **startREDSCRIPT.bat** script has the same commands as the source, but the users and password are encrypted. The compiled executable program is shown in Example 9-2.

Example 9-2 Compiled batch executable for starting subscriptions

```
C:\InfoSphere CDC_MC>online.bat -c \MCScripts\startREDSCRIPT.txt
\MCScripts\compiled\startREDSCRIPT.bat
The startREDSCRIPT.bat script has the same commands as the source, but
the users and password are xxxxx encrypted.
C:\InfoSphere CDC_MC>type \MCScripts\compiled\startREDSCRIPT.bat
// Connect to the Access Server
connectServer(admin,b2lxbhYQ9RgcFlwEElpKCw==,172.16.74.130,10101);
// Connect to the Source data store
connectAgent(CDC_Oracle_Redo);
// Start the subscription
startMirror(CDC_Oracle_Redo,REDSCRIPT,continuous);
// Get the subscription status
getSubscriptionStatus(CDC_Oracle_Redo,REDSCRIPT);
// Disconnect from the Source data store
disconnectAgent(CDC_Oracle_Redo);
// Disconnect from the Access Server
disconnectServer();
// Exit the script
exit();
```

Now that you have a compiled script, you can run it by running **online -b** (**online.bat** for Windows) as follows:

```
C:\InfoSphere CDC_MC>online -b \MCScripts\compiled\startREDSCRIPT.bat
REDSCRIPT Starting
```

As you can see, in both cases the REDSCRIPT subscription has the status Starting, meaning that it is still in its startup phase. The Management Console commands cannot build in delays or looping. If you want to develop a more sophisticated procedure, a controlling shell script is needed to call individual Management Console scripts and analyze the standard output of these Management Console scripts. Develop advanced operations of your InfoSphere CDC environment using the engine commands or the Java API.

9.4 InfoSphere CDC engine commands (CLI)

In many situations, you want to integrate InfoSphere CDC with the business applications and other processes that run on the servers that have InfoSphere CDC installed. Typical system management operations that must be aligned with InfoSphere CDC are the shutdown and restart of a server or the database on which InfoSphere CDC operates. For example, if the database for which a InfoSphere CDC instance has been started must be shut down, first stop all subscriptions and then shut down the InfoSphere CDC instance to avoid abnormal termination of active instances.

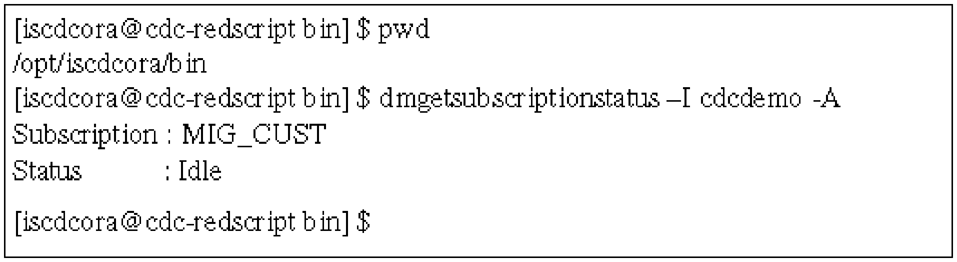
This section describes a few prerequisites for running InfoSphere CDC engine commands and provides some examples about how InfoSphere CDC commands can be scripted and integrated into existing operational environments.

Commands for starting and stopping subscriptions must always be run against the InfoSphere CDC source engine. Other commands, such as showing the bookmark of a subscription, can only be run for the InfoSphere CDC target engine, as this engine is where bookmarks are kept.

9.4.1 Running commands for the Linux, UNIX, and Windows engine

Any InfoSphere CDC Linux, UNIX, and Windows engine command that is run from the command line must be run from the `cdc_home/bin` directory or preceded by the `cdc_home/bin/` prefix. Having the `bin` directory in the path is not sufficient for running the command, and the commands do not run.

As an example, assume that you want to list the active subscriptions for a InfoSphere CDC instance. Change to the `cdc_home/bin` directory and then run **`dmgetsubscriptionstatus -I cdcdemo -A`**. See Example 9-2 on page 236 for an example of running a command for the Linux, UNIX, and Windows engine.



```
[iscdcora@cdc-redscript bin] $ pwd
/opt/iscdcora/bin
[iscdcora@cdc-redscript bin] $ dmgetsubscriptionstatus –I cdcdemo -A
Subscription : MIG_CUST
Status       : Idle

[iscdcora@cdc-redscript bin] $
```

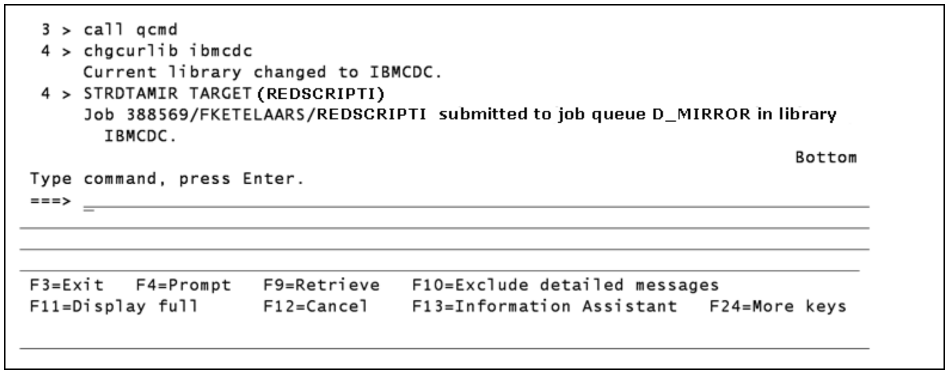
Figure 9-1 dmgetsubscriptionstatus

9.4.2 Running CL commands for System i

When running InfoSphere CDC commands for the InfoSphere CDC for DB2 on System i engine, the job that runs the commands must have the InfoSphere CDC product (instance) library as its current library. Not changing the job's current library but putting the InfoSphere CDC product library into the library list causes the command to fail.

If you intend to write CL programs to perform periodic activities, such as stopping the replication before a system IPL and restarting it afterward, the current library must be set in the CL program before any InfoSphere CDC command is run.

Figure 9-2 shows an example of running a command on System i to start the subscription in continuous mirror mode.



```
 3 > call qcmd
 4 > chgcurlib ibmcdc
     Current library changed to IBMCDC.
 4 > STRDTAMIR TARGET(REDSCRIPTI)
     Job 388569/FKETELAARS/REDSCRIPTI submitted to job queue D_MIRROR in library
       IBMCDC.
                                                                       Bottom
Type command, press Enter.
===> _

F3=Exit   F4=Prompt   F9=Retrieve   F10=Exclude detailed messages
F11=Display full      F12=Cancel    F13=Information Assistant   F24=More keys
```

Figure 9-2 CDC_command_System_i

9.4.3 Running console commands for IBM System z

InfoSphere CDC engine commands on z/OS are submitted from the operator console and appropriate messages are returned by InfoSphere CDC. The console commands that are provided as part of the InfoSphere CDC installation can be run by running **MODIFY** (or **F**) for the InfoSphere CDC started task:

```
MODIFY CHCPROC,STRTSMIR,SUBSCR=ALL
```

The The IBM Time Sharing Option (TSO) user must have authority to issue console commands. This authority is controlled by a product such as the IBM Resource Access Control Facility (RACF®). There is a difference between TSO commands and console commands. InfoSphere CDC for IBM System z® accepts console commands only.

9.4.4 Sample scripts

When deploying InfoSphere CDC, a number of implementation topics need to be addressed, such as starting and stopping of the replication, the management of the database archive logs, and monitoring the activity of the replication processes.

This section provides sample scripts to perform repeating tasks in InfoSphere CDC and enables you to implement these tasks to align with other processes running in your environment.

Starting and stopping the engines and subscriptions

Before a subscription can be started, InfoSphere CDC requires that both the source and target InfoSphere CDC data store be active. Depending on the platform and type of engine, starting and stopping InfoSphere CDC engines can require different commands. This section focuses on platforms and operating systems that require scripting to start processes. Windows is excluded here because a InfoSphere CDC instance is started and stopped with Windows services, and these services are controlled at the operating system configuration level.

Linux, UNIX, and Windows engine examples for starting and stopping

Example 9-3 shows a script to start a InfoSphere CDC Java instance. All Java InfoSphere CDC engines have the same command interface and starting and stopping techniques. Although the command is started as a daemon process (by running the **nohup** command), it writes any messages to `nohup.out` in the current directory. The script monitors for messages arriving in `nohup.out` and sends them to `stdout` before returning to the command prompt.

The script only starts the first InfoSphere CDC instance that has been defined. If you must start multiple or all InfoSphere CDC instances, the script must be modified.

Example 9-3 Script to start a InfoSphere CDC Java instance

```
#!/bin/bash

# set -x

# CDC instance variables
CDC_AGENT=DB2
CDC_HOME=/opt/InfoSphere CDCdb2
CDC_USER=InfoSphere CDCdb2
CDC_INSTANCE=`ls ${CDC_HOME}/instance | head -1`
```

```
# Start CDC instance
echo "Starting CDC ${CDC_AGENT} instance ${CDC_INSTANCE} for user
${CDC_USER} ..."
rm ${CDC_HOME}/bin/nohup.out 2> /dev/null
su - ${CDC_USER} -c "cd ${CDC_HOME}/bin;nohup ./dmts64 -I
${CDC_INSTANCE} &" > ${CDC_HOME}/bin/nohup.out

# Wait for CDC instance to be started or error to be issued (max 60
seconds)
for i in {1..60}
  do
    if [ -e ${CDC_HOME}/bin/nohup.out ];
      then
        nLines=$(wc -l ${CDC_HOME}/bin/nohup.out | awk '{print $1}')
        if (( ${nLines} != 0 ))
          then
            break
          fi
      fi
    sleep 1
  done

# Determine if instance was started correctly
nLines=$(grep -c "IBM InfoSphere Change Data Capture is running."
${CDC_HOME}/bin/nohup.out)

if (( ${nLines} != 0 ))
  then
    # Do post-instance action such as deselecting the staging store or
setting system parameters
    echo post-start > /dev/null
fi

cat ${CDC_HOME}/bin/nohup.out

echo
echo "Press Enter to continue ..."
read -t 5
```

Using a similar method, the CDC engine can be stopped.

```
#!/bin/bash

# CDC instance variables
```

```
CDC_AGENT=DB2
CDC_HOME=/opt/InfoSphere CDCdb2
CDC_USER=InfoSphere CDCdb2
CDC_INSTANCE=`ls ${CDC_HOME}/instance | head -1`

# Stop CDC instance
echo "Stopping CDC ${CDC_AGENT} instance ${CDC_INSTANCE} ..."
su - ${CDC_USER} -c "${CDC_HOME}/bin/dmshutdown -I ${CDC_INSTANCE}"
echo
echo "Press Enter to continue ..."
read -t 5
```

System i examples for starting and stopping

On most System i servers, InfoSphere CDC is started in the system's startup program, which is identified by the system value QSTRUPPGM.

Example 9-4 shows a sample CL program that could be called from the server's startup program. The program starts the InfoSphere CDC instance (IBMCDC subsystem), waits a few seconds to allow the InfoSphere CDC TCP/IP listener job to start, and then starts all the subscriptions replicating from the current system.

Example 9-4 CL program to start InfoSphere CDC

```
*************** Beginning of data *************************************
              PGM

 /*           Start the InfoSphere CDC subsystem                   */
               STRSBS     SBSD(IBMCDC/IBMCDC)
              MONMSG     MSGID(CPF1010)

 /*           Wait for the TCP listener to come up                 */
              DLYJOB     DLY(20)

 /*           Start subscriptions                                  */
              CHGCURLIB  CURLIB(IBMCDC)
              STRDTAMIR  TARGET(*ALL)

              ENDPGM
 ****************** End of data ***************************************
```

If you choose to include the startup of InfoSphere CDC in the server startup program, ensure that the system's TCP/IP service has been started before you start the InfoSphere CDC subsystem. The listener fails to start if TCP/IP is not yet active and you are not be able to connect to the InfoSphere CDC instance using the Management Console or start any replication processes that target the system. A common practice is to place the startup of InfoSphere CDC at the end of the startup program or to build in a delay or check for TCP/IP activity using the QtocRtvTCPA API.

Similarly, a CL program could shut down InfoSphere CDC before the system must be powered down (Example 9-5).

Example 9-5   CL program to shut down InfoSphere CDC

```
*************** Beginning of data *************************************
             PGM

/*           End all subscription target processes                 */
             CHGCURLIB  CURLIB(IBMCDC)
             DMENDPROC  PUBID(*ALL)

/*           End all subscriptions (*CNTRLD at first)              */
             CHGCURLIB  CURLIB(IBMCDC)
             ENDDTAMIR  TARGET(*ALL)
             DLYJOB     DLY(120)

/*           If still subscriptions active, end with *IMMED        */
             ENDDTAMIR  TARGET(*ALL) ENDTYP(*IMMED)
             DLYJOB     DLY(10)

/*           End the subsystem immediately                         */
             ENDSBS     SBS(IBMCDC) OPTION(*IMMED)
             MONMSG     MSGID(CPF1054)

             ENDPGM
****************** End of data ***************************************
```

The script first ends all subscription target processes by running **DMENDPROC** and then issues a controlled end of the subscriptions. A controlled end allows processing of the remaining journal entries by the subscription. If there is a large backlog of transactions to be replicated, the controlled ending of the subscriptions could take a long time and delay the shutdown of the system. Therefore, the program waits for 2 minutes and then ends the replication immediately.

You could make the program a bit more sophisticated by, for example, checking the monitoring activity of the subscription to shorten the fixed delay.

System z example for starting

The JCL to start the InfoSphere CDC instance (the default name is CHCPROC) is distributed with the product in the `SHCCNTL` library (see Example 9-6).

Example 9-6 JCL to start a InfoSphere CDC instance

```
//CHCPROC  PROC CONFIG=<ConfigSuffix>,
//            COMM=COMM,
//            DBMS=DBMS
//*
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//*                                                                   *
//*     LICENSED MATERIALS - PROPERTY OF IBM                          *
//*     5655-U96                                                      *
//*     COPYRIGHT IBM CORP 1998,2008  ALL RIGHTS RESERVED.            *
//*                                                                   *
//*     US GOVERNMENT USERS RESTRICTED RIGHTS -                       *
//*     USE, DUPLICATION OR DISCLOSURE RESTRICTED                     *
//*     BY GSA ADP SCHEDULE CONTRACT WITH IBM CORP.                   *
//*                                                                   *
//*     Sample JCL to run InfoSphere Change Data Capture for z/OS *
//*                                                                   *
//*  Instructions:                                                    *
//*                                                                   *
//*  1. If the Language Environment product was not installed into    *
//*     data sets with the default name prefix, change the value of   *
//*     the _ICONV_UCS2_PREFIX keyword in the PARM field of the EXEC  *
//*     statement to specify the prefix chosen for the Language       *
//*     Environment data sets.  InfoSphere Change Data Capture for    *
//*     z/OS will use this item when requesting code page conversion
*
//*     using Language Environment provided services.                 *
//*                                                                   *
//*  2. Change <ConfigSuffix> to the 2 character configuration suffix *
//*     appended to the "CHCCFGxx" member of the CHCCNTRL data set.   *
//*                                                                   *
//*  3. Change <CHC.HCHC620> to the high-level data set name          *
//*     qualifier to be used for the InfoSphere Change Data Capture   *
//*     for z/OSexecution data sets.                                  *
//*                                                                   *
//*  4. Change 'DSNxxx.SDSNEXIT' to the name of the DB2 APF           *
//*     Authorized exit library.                                      *
//*                                                                  *
//*  5. Change 'DSNxxx.SDSNLOAD' to the name of the DB2 APF         *
//*     Authorized execution library.                                *
//*                                                                  *
//*  6. Change <MetaDatacluster> to the name of the InfoSphere Change *
//*     Data Capture for z/OS Meta-Data VSAM Cluster.                *
//*                                                                  *
//*  7. Change <PALcluster> to the name of the InfoSphere Change Data *
//*     Capture for z/OS Product Administration Log (PAL) VSAM       *
//*     Cluster.                                                     *
//*                                                                  *
//*  8. Change <CACHE.QUALIFIER> to the high-level data set name     *
//*     qualifier to be used for the DB2 Log Cache data sets.        *
//*     Remove these lines if you do not want to use a DB2 Log Cache. *
//*                                                                  *
//*  9. Change <UserExitLoadLib> to the name of the execution data set
*
//*     for user exits.                                              *
//*                                                                  *
//* 10. Change 'TCPIP.SEZAINST(TCPDATA)' to the name of the z/OS     *
//*     TCP/IP Component's TCPIP.DATA data set.  If the installation  *
//*     is using a TCP/IP Resolver address space, this DD statement   *
//*     can be removed.                                              *
//*                                                                  *
//* Note:                                                            *
//*   The first step, named DELETE, removes the prior SYSMDUMP data   *
//*   set, if it exists.  The SYSMDUMP data set is then allocated     *
//*   anew when InfoSphere Change Data Capture for z/OS starts        *
//*   execution.  This newly allocated SYSMDUMP data set must have a  *
//*   disposition of MOD, so that, should multiple memory dumps occur,
they  *
//*   will be appended in sequence, and not successively overlay the  *
//*   prior memory dumps in the data set.
*
//*                                                                  *
//* Note:                                                            *
//*   The REGION parameter on the EXEC statement of the IEFPROC step  *
//*   is set to OM.  This value implies a default value for the       *
//*   MEMLIMIT parameter (which is not coded) of "NOLIMIT".  This     *
//*   permits the use of storage "above the bar".  If the REGION      *
//*   parameter is changed to a non zero value or removed, you must    *
//*   add a MEMLIMIT keyword to the EXEC statement specifying enough  *
//*   storage to serve InfoSphere Change Data Capture for z/OS's      *
//*   requirements.                                                   *
//*                                                                  *
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//*
//DELETE   EXEC PGM=IEFBR14
//SYSMDUMP DD DSNAME=<CHC.HCHC620>.SYSMDUMP,DISP=(MOD,DELETE),
//            UNIT=SYSALLDA,
//            SPACE=(CYL,(1))
//*
//IEFPROC  EXEC PGM=CHCMIT,
//            PARM=('/ENVAR(_ICONV_UCS2_PREFIX=CEE)',
//            '/&COMM,&DBMS,CONFIG=&CONFIG'),
//            REGION=0M <== SEE THE NOTE ABOVE
//STEPLIB  DD DSNAME=<CHC.HCHC620>.LOAD,DISP=SHR
//         DD DSNAME=DSNxxx.SDSNEXIT,DISP=SHR
//         DD DSNAME=DSNxxx.SDSNLOAD,DISP=SHR
//CHCCNTRL DD DDNAME=IEFRDER
//IEFRDER  DD DSNAME=<CHC.HCHC620>.DATA,DISP=SHR
//CHCPRINT DD SYSOUT=*
//CHCAUDIT DD SYSOUT=*
//CHCREPRT DD SYSOUT=*
//CHCMTDTA DD DSNAME=<MetaDatacluster>,DISP=SHR
//CHCPALOG DD DSNAME=<PALcluster>,DISP=SHR
//CHCCACHE DD DSNAME=<CACHE.QUALIFIER>.CHCCACHE,DISP=SHR,
//            AMP=('BUFND=256,ACCBIAS=SW')
//CHCCHCTL DD DSNAME=<CACHE.QUALIFIER>.CHCCHCTL,DISP=SHR
//CHCUXLIB DD DSNAME=<UserExitLoadLib>,DISP=SHR
//SYSTCPD  DD DSNAME=TCPIP.SEZAINST(TCPDATA),DISP=SHR
//UTPRINT  DD DUMMY
//* SEE THE NOTE ABOVE ABOUT THE DISPOSITION CODED FOR THIS DATA SET  *
//SYSMDUMP DD DSNAME=<CHC.HCHC620>.SYSMDUMP,DISP=(MOD,CATLG), *NOTE*
//            UNIT=SYSALLDA,
//            SPACE=(CYL,(150,50))
//ABNLIGNR DD DUMMY
//*
```

The following is an example on how to issue InfoSphere CDC console commands as part of a JCL.

```
//WCA008CM JOB (3WCA000),'BIN 376 - CXA XM1',MSGCLASS=H,
//REGION=2M,CLASS=A,NOTIFY=&SYSUID
//ISPFPROC EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//F CHCPROC,STRTSMIR,SUBSCR=TESTSUB1
//F CHCPROC,ENDRPLCT,SUBSCR=TESTSUB2,NORM
```

9.4.5 Checking an InfoSphere CDC engine and subscriptions activity

When InfoSphere CDC has been deployed on a production system, it is typically treated as a business critical application and it is therefore included in standard checks for activity. This section lists the methods for determining the activity of the InfoSphere CDC instance and its subscriptions, using operating system commands.

If you are looking for a more centralized solution to monitor activity of InfoSphere CDC instances, subscriptions, and more details, such as latency, the Java API provides better abilities to retrieve operational information. See 9.5, "InfoSphere CDC API" on page 262 for more information about customizing the configuration and monitoring of InfoSphere CDC.

Linux, UNIX, and Windows engine activity checking

There is no CLI command for checking the activity of a InfoSphere CDC instance. Most system administrators use standard operating system techniques to verify if a process has been started, and the same can be used for activity of the InfoSphere CDC engine.

There are at least three methods by which the activity of a InfoSphere CDC instance can be checked:

1. The **dmts64** (or **dmts32**) process is running for the named instance.
2. There is a listener active for the port that has been configured for the instance.
3. An engine command, such as **dmgetsubscriptionstatus**, returns a result when run and exits with code 0.

Most commonly, customers use the UNIX **ps** command to determine if processes are running on the system. Example 9-7 shows a script that checks whether there is a **dmts** process active for the first (and only) InfoSphere CDC instance configured for an engine.

Example 9-7 Script to check if a dmts process is active

```
#!/bin/bash

# set -x

# CDC instance variables
CDC_AGENT=DB2
CDC_HOME=/opt/InfoSphere CDCdb2
CDC_USER=InfoSphere CDCdb2
CDC_INSTANCE=`ls ${CDC_HOME}/instance | head -1`
```

```
# Find the process in the system using the ps command
nLines=$(ps aux | grep dmts | grep $CDC_HOME | grep $CDC_INSTANCE | wc
-l)

if (( ${nLines} != 0 ))
  then
    echo "$CDC_AGENT CDC instance $CDC_INSTANCE is running."
    exit 0
  else
    echo "$CDC_AGENT CDC instance $CDC_INSTANCE is not active."
    exit 1
fi
```

Even though it might be interesting to know whether a InfoSphere CDC instance is active, most clients prefer to know if subscriptions are up and running (if subscriptions are active, the instance is up too). For the Linux, UNIX, and Windows engine, the full status can be retrieved by running **dmgetsubscriptionstatus** from a script or the command line. Examples of those commands are:

```
[InfoSphere CDCora@cdc-redscript bin]$ dmgetsubscriptionstatus -I
cdcdemo -A -p
   Subscription : REDSCRIPT
   Status       : Idle
[InfoSphere CDCora@cdc-redscript bin]$
```

If the instance is running, the active (Active) and inactive (Idle) subscriptions are shown. The command returns an error when trying to run it against an instance that is not running.

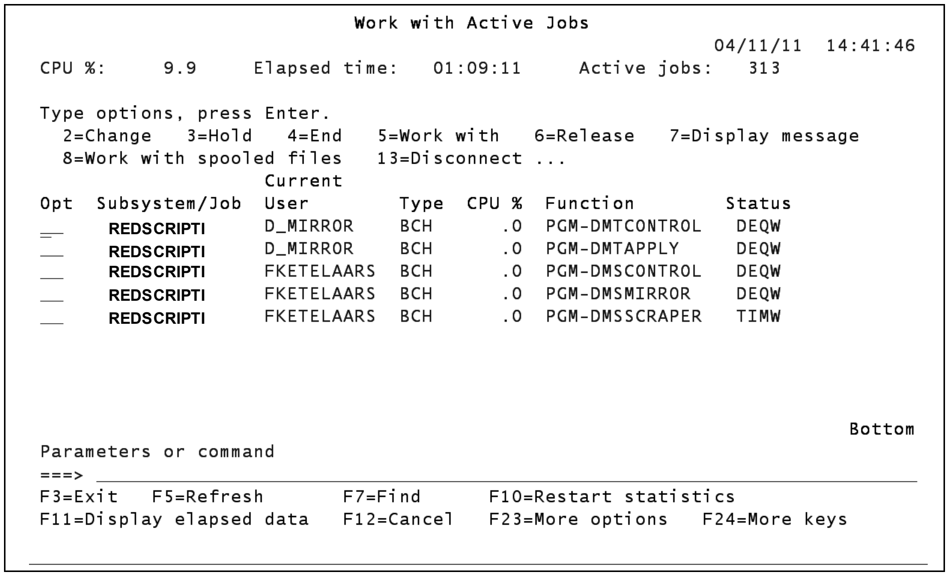
```
[InfoSphere CDCdb2@cdc-redscript bin]$ dmgetsubscriptionstatus -A -p
   IBM InfoSphere Change Data Capture instance "cdcdemo" is not
   started.
   Start the instance to resolve this error.
   [InfoSphere CDCdb2@cdc-redscript bin]$ echo $?
   253
```

A shell script could be written to analyze the output from the **dmgetsubscriptionstatus** command and take relevant actions. Also, the status check could be integrated with a monitoring solution such as IBM Tivoli® Monitoriing.

System i activity checking

On System i, a subscription's status can be checked through the InfoSphere CDC subsystem. If the InfoSphere CDC subsystem is active and there is a TCPLISTEN job in the subsystem, the instance is running.

Every subscription that is active for continuous mirroring has at least three source jobs in the InfoSphere CDC subsystem (more if multiple journals are processed by a subscription). When targeting a System i server, there are two or more jobs active per subscription (there is one DMTAPPLY job per journal processed on the source). Figure 9-3 shows the REDSCRIPTI subscription, which replicates intra-system, activating a total of five subscription jobs in the subsystem.



```
                              Work with Active Jobs
                                                               04/11/11  14:41:46
CPU %:     9.9     Elapsed time:   01:09:11     Active jobs:   313

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...
                       Current
Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__     REDSCRIPTI   D_MIRROR    BCH      .0  PGM-DMTCONTROL   DEQW
__     REDSCRIPTI   D_MIRROR    BCH      .0  PGM-DMTAPPLY     DEQW
__     REDSCRIPTI   FKETELAARS  BCH      .0  PGM-DMSCONTROL   DEQW
__     REDSCRIPTI   FKETELAARS  BCH      .0  PGM-DMSMIRROR    DEQW
__     REDSCRIPTI   FKETELAARS  BCH      .0  PGM-DMSSCRAPER   TIMW




                                                                           Bottom
Parameters or command
===> __________________________________________________________________________
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

Figure 9-3 Replicating intra-system

Most monitoring solutions can base actions on the activity of certain jobs in a subsystem. It should be fairly simple to configure activity checking by monitoring for subsystem jobs that have the same name as the subscription. You could also consider writing a CL program that employs the QUSLJOB API to list the jobs in the InfoSphere CDC subsystem. Providing a sample program to list subsystem jobs goes beyond the scope of this book.

System z activity checking

The **DSPACT** console command can be used to show activity in the InfoSphere CDC/z instance (Example 9-8).

Example 9-8 DSPACT console command

```
CHC9600I (CHCPROC) "MODIFY" command accepted, Diagnostic information
("DSPACT")
CHC9733I (CHCPROC) V5 Communications: Connections = 5
CHC9800I (CHCPROC) Target Data name = CUST10K, Sends = 6, Send Bytes =
310, Recvs = 2, Recv Bytes = 78, CPU Used = 0.011415
CHC9801I (CHCPROC) Source Data name = CUST10K, Sends = 2, Send Bytes =
78, Recvs = 6, Recv Bytes = 310, CPU Used = 0.010111
CHC9802I (CHCPROC) Monitor Agent name = DMCMVSA:37504, Sends = 18, Send
Bytes = 2,031, Recvs = 20, Recv Bytes = 1,625, CP ...
CHC9802I (CHCPROC) ... U Used = 0.012863
CHC9803I (CHCPROC) Source Control name = DMCMVSA.TOROLAB.IBM.COM, Sends
= 20, Send Bytes = 1,625, Recvs = 18, Recv Bytes  ...
CHC9803I (CHCPROC) ... = 2,031, CPU Used = 0.011747
CHC9804I (CHCPROC) Target Control name = MNOEST2.TOROLAB.IBM.COM, Sends
= 18, Send Bytes = 1,384, Recvs = 14, Recv Bytes  ...
CHC9804I (CHCPROC) ... = 1,410, CPU Used = 0.013526
CHC9778I (CHCPROC) Agent Communications: Connections = 1,
Admin(Act/Pnd) = 1/0, State = Processing, Shutdown = *N/A*
CHC9818I (CHCPROC) Medium = TCP/IP, State = Active, Shutdown = *N/A*
CHC9788I (CHCPROC) Datastore name = DMC0038, Medium = TCP/IP, Paths =
1, Sends = 21, Recvs = 38
CHC9832I (CHCPROC) DBMS: Repositories = 1, State = Processing, Shutdown
= *N/A*
CHC9742I (CHCPROC) DBMS: Repository Type = DB2 , ID = AD91:DMC0038,
Log(Monitor/Scraper) = 1/1,
        Replication(admin/source/ ...
CHC9742I (CHCPROC) ... apply) = 1/1/1, HoL RBA = X'00000A9FC229FB91'
CHC0290I (CHCPROC) The DB2 Log Cache is processing with a range of
X'0000000000000000' through
        X'0000000000000000'
CHC9751I (CHCPROC) Replication: Log(Monitor/Scraper) = 1/1,
Replication(source/apply) = 1/1
CHC9753I (CHCPROC) Replication: 1 active Target Subscription(s)
CHC9844I (CHCPROC) Subscription name = CUST10K, Repository type = DB2,
Activity = Start_Mirror, State = Processing, Shutd ...
CHC9844I (CHCPROC) ... own = *N/A*, Current RBA = X'00000A9FC229FB91',
Written at = 2011-04-13-13.29.22.179712, Scrape to ...
CHC9844I (CHCPROC) ...  = *N/A*
CHC9753I (CHCPROC) Replication: 1 active Source Subscription(s)
```

```
CHC9857I (CHCPROC) Source name = CUST10K, Activity = , State =
Processing, Shutdown = *N/A*
```

Example 9-8 on page 249 shows various bits of information under several major headings, as follows:

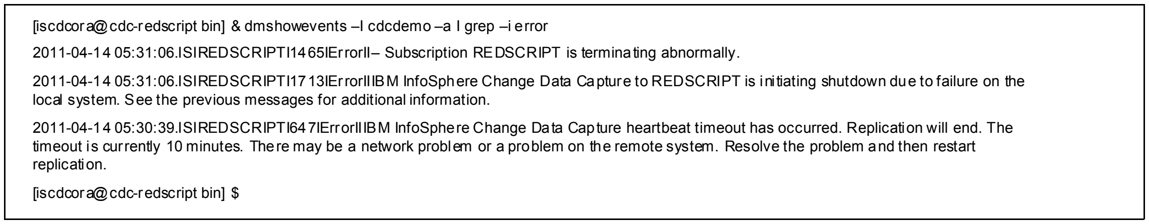
- V5 Communications shows the number of communications connections currently active. In this case, a loopback subscription called CUST10K and a single Management Console user.
- Agent Communications shows the single Management Console user.
- DBMS shows that you are connected to a DB2 subsystem called AD91 with user ID CMS0038.
- You are running one log monitor (it should always be one) and one log scraper (it should be one per source subscription).
- You are also running one source / apply because you are running one loopback subscription.
- HoL RBA shows the current DB2 head of log position (either RBA or LRSN).
- The log cache is not active and shows no position.
- Replication shows details about the source and target subscriptions currently active. Of special interest here is the Current RBA, which shows where the scraper for subscription CUST10K is currently reading the log.

Monitoring the event logs

Besides monitoring the activity of subscriptions, you typically want to automate the monitoring of event logs. Most monitoring solutions offer the ability to run system commands and analyze the output of those commands. The next section describes CLI commands that can be used in scripts or started from a monitoring solution to automate monitoring of InfoSphere CDC event logs.

Linux, UNIX, and Windows engine event log monitoring

If you plan to monitor the event logs from the InfoSphere CDC server using a command line, the Linux, UNIX, and Windows engine provides a command to list the contents of the event log. By using basic commands, you can filter out the error messages from the information messages that the replication issues (Figure 9-4 on page 251).



```
[iscdcora@cdc-redscript bin] & dmshowevents –I cdcdemo –a I grep –i error
2011-04-14 05:31:06.ISIREDSCRIPTI1465IErrorII– Subscription REDSCRIPT is terminating abnormally.
2011-04-14 05:31:06.ISIREDSCRIPTI1713IErrorIIIBM InfoSphere Change Data Capture to REDSCRIPT is initiating shutdown due to failure on the
local system. See the previous messages for additional information.
2011-04-14 05:30:39.ISIREDSCRIPTI647IErrorIIIBM InfoSphere Change Data Capture heartbeat timeout has occurred. Replication will end. The
timeout is currently 10 minutes. There may be a network problem or a problem on the remote system. Resolve the problem and then restart
replication.
[iscdcora@cdc-redscript bin] $
```

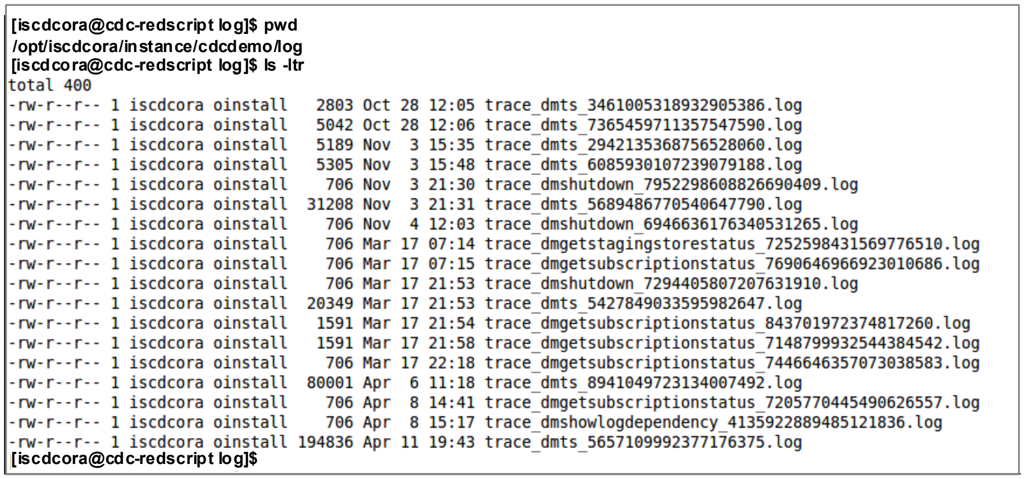
Figure 9-4 Event log

The output can also be directed to a file that is monitored by your external monitoring solution so that replication errors are also captured. Some monitoring solutions expect only new errors to be logged and do not filter out the ones that have already been processed, or they track one file that holds the error messages and only view it when it has been extended. To avoid having errors reported more than once, you might need to build some scripting logic to process the output of the **dmshowevents** command (Example 9-9).

Example 9-9 Output of the dmshowevents command

```
#!/bin/bash

# Environment setup
CDC_HOME="/opt/InfoSphere CDCdb2" # CDC Home directory
CDCInstance="cdcdemo" # CDC Instance configured
LogFile="/monitor/cdc_reported_errors.log" # Log file that is monitored
TmpLogFile="/tmp/cdc_errors_$$.tmp" # Temporary file holding all dmshowevents
                                    # log entries

# List the events to a temporary output file
$CDC_HOME/bin/dmshowevents -I $CDCInstance -a | grep -i error > $TmpLogFile

# Find differences between temporary and permanent output file
touch $LogFile # Create log file if it does not exist
# Output new lines to log file
diff $TmpLogFile $LogFile | tail -n+2 >> $LogFile
```

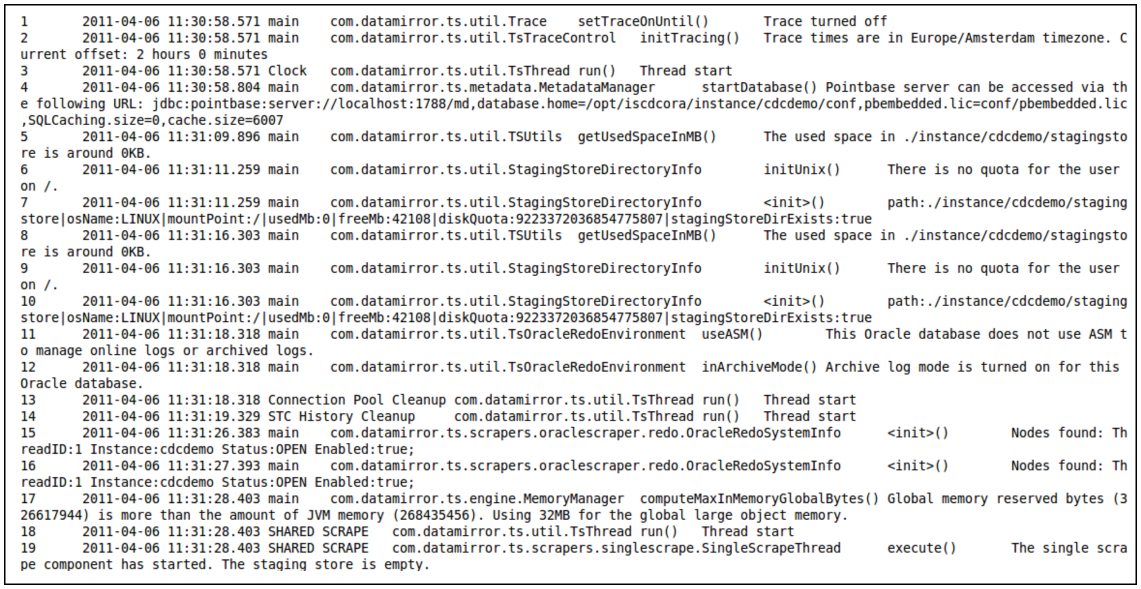
In addition to the **dmshowevents** messages, the Linux, UNIX, and Windows engine also logs more detailed messages in the `<cdc_home>/instance/<instance>/log` directory (Figure 9-5).



```
[iscdcora@cdc-redscript log]$ pwd
/opt/iscdcora/instance/cdcdemo/log
[iscdcora@cdc-redscript log]$ ls -ltr
total 400
-rw-r--r-- 1 iscdcora oinstall   2803 Oct 28 12:05 trace_dmts_3461005318932905386.log
-rw-r--r-- 1 iscdcora oinstall   5042 Oct 28 12:06 trace_dmts_7365459711357547590.log
-rw-r--r-- 1 iscdcora oinstall   5189 Nov  3 15:35 trace_dmts_2942135368756528060.log
-rw-r--r-- 1 iscdcora oinstall   5305 Nov  3 15:48 trace_dmts_6085930107239079188.log
-rw-r--r-- 1 iscdcora oinstall    706 Nov  3 21:30 trace_dmshutdown_7952298608826690409.log
-rw-r--r-- 1 iscdcora oinstall  31208 Nov  3 21:31 trace_dmts_5689486770540647790.log
-rw-r--r-- 1 iscdcora oinstall    706 Nov  4 12:03 trace_dmshutdown_6946636176340531265.log
-rw-r--r-- 1 iscdcora oinstall    706 Mar 17 07:14 trace_dmgetstagingstorestatus_7252598431569776510.log
-rw-r--r-- 1 iscdcora oinstall    706 Mar 17 07:15 trace_dmgetsubscriptionstatus_7690646966923010686.log
-rw-r--r-- 1 iscdcora oinstall    706 Mar 17 21:53 trace_dmshutdown_7294405807207631910.log
-rw-r--r-- 1 iscdcora oinstall  20349 Mar 17 21:53 trace_dmts_5427849033595982647.log
-rw-r--r-- 1 iscdcora oinstall   1591 Mar 17 21:54 trace_dmgetsubscriptionstatus_843701972374817260.log
-rw-r--r-- 1 iscdcora oinstall   1591 Mar 17 21:58 trace_dmgetsubscriptionstatus_7148799932544384542.log
-rw-r--r-- 1 iscdcora oinstall    706 Mar 17 22:18 trace_dmgetsubscriptionstatus_7446646357073038583.log
-rw-r--r-- 1 iscdcora oinstall  80001 Apr  6 11:18 trace_dmts_8941049723134007492.log
-rw-r--r-- 1 iscdcora oinstall    706 Apr  8 14:41 trace_dmgetsubscriptionstatus_7205770445490626557.log
-rw-r--r-- 1 iscdcora oinstall    706 Apr  8 15:17 trace_dmshowlogdependency_4135922889485121836.log
-rw-r--r-- 1 iscdcora oinstall 194836 Apr 11 19:43 trace_dmts_5657109992377176375.log
[iscdcora@cdc-redscript log]$
```

Figure 9-5 InfoSphere CDC events trace

Also, if you have switched on debugging for the engines or if you are using the `com.datamirror.ts.util.Trace` class from your Java user exit program, these messages are also logged in the `trace_dmts*` files in this directory (Figure 9-6).



```
1       2011-04-06 11:30:58.571 main    com.datamirror.ts.util.Trace    setTraceOnUntil()       Trace turned off
2       2011-04-06 11:30:58.571 main    com.datamirror.ts.util.TsTraceControl   initTracing()   Trace times are in Europe/Amsterdam timezone. C
urrent offset: 2 hours 0 minutes
3       2011-04-06 11:30:58.571 Clock   com.datamirror.ts.util.TsThread run()   Thread start
4       2011-04-06 11:30:58.804 main    com.datamirror.ts.metadata.MetadataManager      startDatabase() Pointbase server can be accessed via th
e following URL: jdbc:pointbase:server://localhost:1788/md,database.home=/opt/iscdcora/instance/cdcdemo/conf,pbembedded.lic=conf/pbembedded.lic
,SQLCaching.size=0,cache.size=6007
5       2011-04-06 11:31:09.896 main    com.datamirror.ts.util.TSUtils  getUsedSpaceInMB()      The used space in ./instance/cdcdemo/stagingsto
re is around 0KB.
6       2011-04-06 11:31:11.259 main    com.datamirror.ts.util.StagingStoreDirectoryInfo        initUnix()      There is no quota for the user
on /.
7       2011-04-06 11:31:11.259 main    com.datamirror.ts.util.StagingStoreDirectoryInfo        <init>()        path:./instance/cdcdemo/staging
store|osName:LINUX|mountPoint:/|usedMb:0|freeMb:42108|diskQuota:9223372036854775807|stagingStoreDirExists:true
8       2011-04-06 11:31:16.303 main    com.datamirror.ts.util.TSUtils  getUsedSpaceInMB()      The used space in ./instance/cdcdemo/stagingsto
re is around 0KB.
9       2011-04-06 11:31:16.303 main    com.datamirror.ts.util.StagingStoreDirectoryInfo        initUnix()      There is no quota for the user
on /.
10      2011-04-06 11:31:16.303 main    com.datamirror.ts.util.StagingStoreDirectoryInfo        <init>()        path:./instance/cdcdemo/staging
store|osName:LINUX|mountPoint:/|usedMb:0|freeMb:42108|diskQuota:9223372036854775807|stagingStoreDirExists:true
11      2011-04-06 11:31:18.318 main    com.datamirror.ts.util.TsOracleRedoEnvironment  useASM()        This Oracle database does not use ASM t
o manage online logs or archived logs.
12      2011-04-06 11:31:18.318 main    com.datamirror.ts.util.TsOracleRedoEnvironment  inArchiveMode() Archive log mode is turned on for this
Oracle database.
13      2011-04-06 11:31:18.318 Connection Pool Cleanup com.datamirror.ts.util.TsThread run()   Thread start
14      2011-04-06 11:31:19.329 STC History Cleanup     com.datamirror.ts.util.TsThread run()   Thread start
15      2011-04-06 11:31:26.383 main    com.datamirror.ts.scrapers.oraclescraper.redo.OracleRedoSystemInfo      <init>()        Nodes found: Th
readID:1 Instance:cdcdemo Status:OPEN Enabled:true;
16      2011-04-06 11:31:27.393 main    com.datamirror.ts.scrapers.oraclescraper.redo.OracleRedoSystemInfo      <init>()        Nodes found: Th
readID:1 Instance:cdcdemo Status:OPEN Enabled:true;
17      2011-04-06 11:31:28.403 main    com.datamirror.ts.engine.MemoryManager  computeMaxInMemoryGlobalBytes() Global memory reserved bytes (3
26617944) is more than the amount of JVM memory (268435456). Using 32MB for the global large object memory.
18      2011-04-06 11:31:28.403 SHARED SCRAPE   com.datamirror.ts.util.TsThread run()   Thread start
19      2011-04-06 11:31:28.403 SHARED SCRAPE   com.datamirror.ts.scrapers.singlescrape.SingleScrapeThread      execute()       The single scra
pe component has started. The staging store is empty.
```

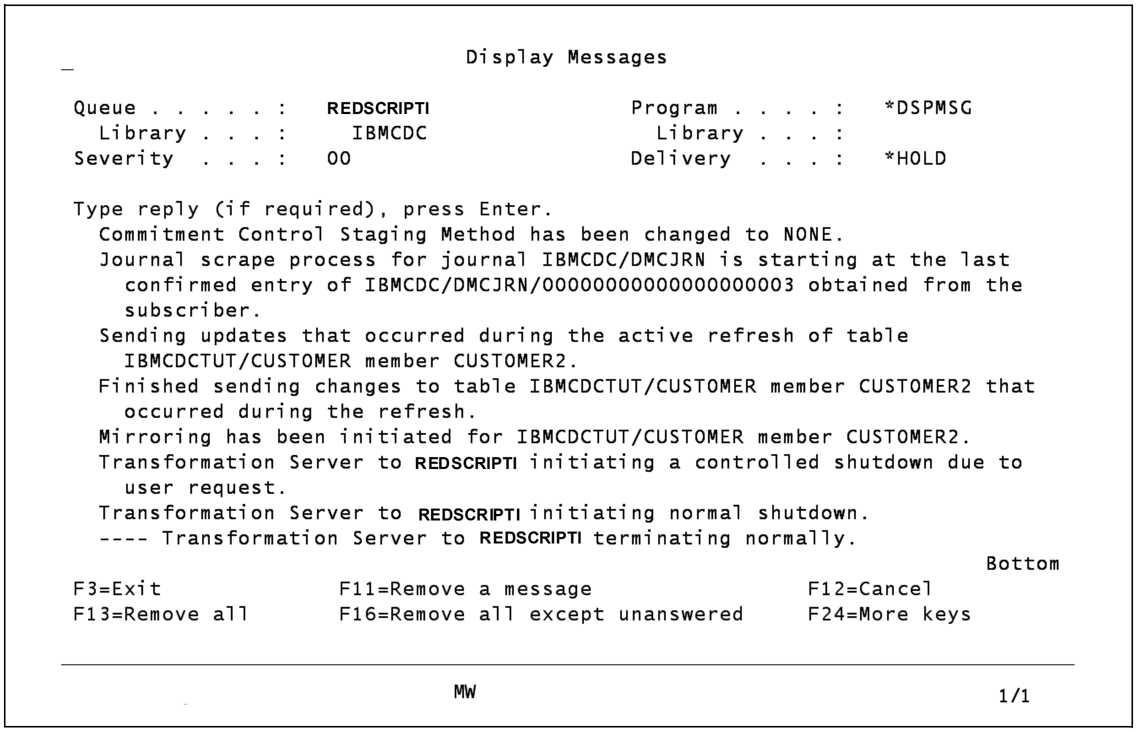
Figure 9-6 InfoSphere CDC events trace - details

System i event log monitoring

InfoSphere CDC for DB2 on System i logs its engine and subscription events in message queue objects (*MSGQ) in the InfoSphere CDC product library. The following message queue objects can be distinguished:

- COMMEVENT: InfoSphere CDC data store events (source and target).
- <subscription>: Subscription source events, for example, REDSCRIPTI for a subscription that is called REDSCRIPTI.
- <publisher ID>_T: Subscription target events, for example, REDSCRIPTI_T for a subscription that is called REDSCRIPTI and whose publisher ID is REDSCRIPTI.

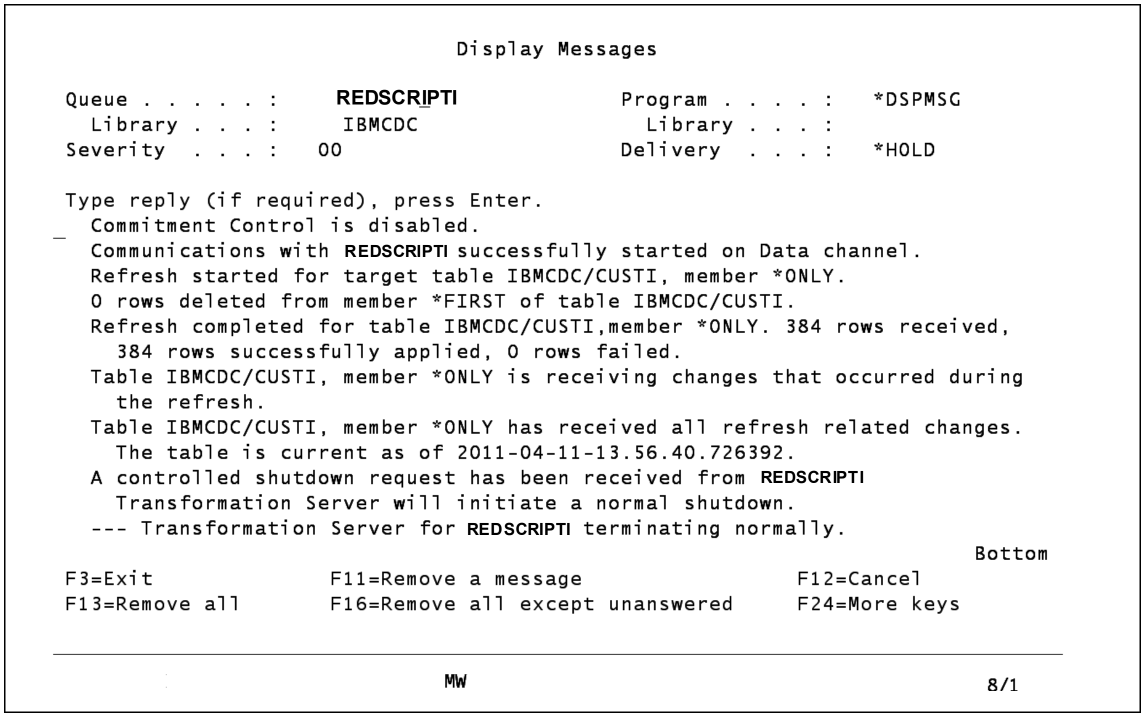
Figure 9-7 shows the output of the **DSPMSG MSGQ(REDSCRIPTI)** command to provide an example of source subscription messages.



```
_                              Display Messages

 Queue . . . . . :   REDSCRIPTI                Program . . . . :   *DSPMSG
   Library . . . :     IBMCDC                    Library . . . :
 Severity  . . . :   00                        Delivery  . . . :   *HOLD

 Type reply (if required), press Enter.
   Commitment Control Staging Method has been changed to NONE.
   Journal scrape process for journal IBMCDC/DMCJRN is starting at the last
     confirmed entry of IBMCDC/DMCJRN/00000000000000000003 obtained from the
     subscriber.
   Sending updates that occurred during the active refresh of table
     IBMCDCTUT/CUSTOMER member CUSTOMER2.
   Finished sending changes to table IBMCDCTUT/CUSTOMER member CUSTOMER2 that
     occurred during the refresh.
   Mirroring has been initiated for IBMCDCTUT/CUSTOMER member CUSTOMER2.
   Transformation Server to REDSCRIPTI initiating a controlled shutdown due to
     user request.
   Transformation Server to REDSCRIPTI initiating normal shutdown.
   ---- Transformation Server to REDSCRIPTI terminating normally.
                                                                      Bottom
 F3=Exit              F11=Remove a message                 F12=Cancel
 F13=Remove all       F16=Remove all except unanswered     F24=More keys

                                   MW                                  1/1
```

Figure 9-7 InfoSphere CDC subscription source events

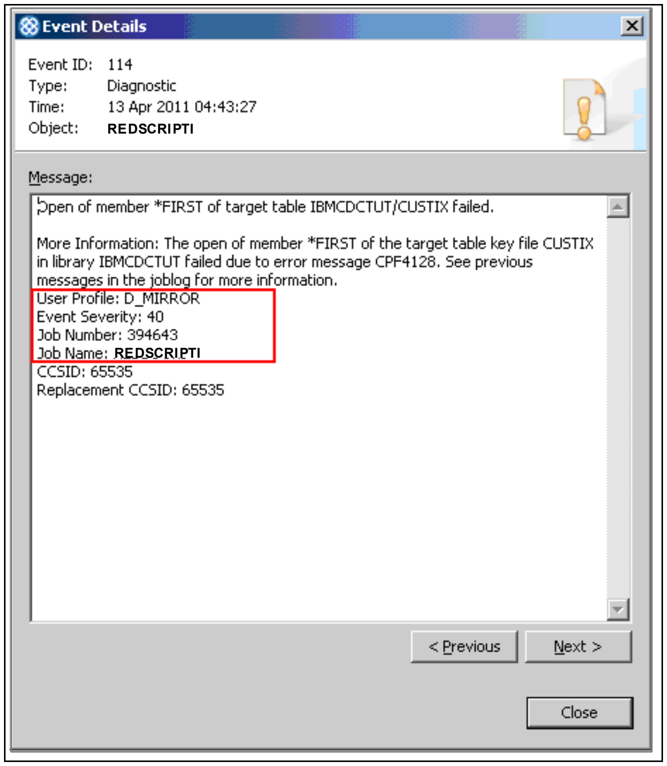
Figure 9-8 shows the target output of the **DSPMSG MSGQ(REDSCRIPTI_T)** command.



```
                              Display Messages

 Queue . . . . . :    REDSCRIPTI            Program . . . . :   *DSPMSG
   Library . . . :     IBMCDC                 Library . . . :
 Severity  . . . :   00                     Delivery  . . . :   *HOLD

 Type reply (if required), press Enter.
_  Commitment Control is disabled.
   Communications with REDSCRIPTI successfully started on Data channel.
   Refresh started for target table IBMCDC/CUSTI, member *ONLY.
   0 rows deleted from member *FIRST of table IBMCDC/CUSTI.
   Refresh completed for table IBMCDC/CUSTI,member *ONLY. 384 rows received,
     384 rows successfully applied, 0 rows failed.
   Table IBMCDC/CUSTI, member *ONLY is receiving changes that occurred during
     the refresh.
   Table IBMCDC/CUSTI, member *ONLY has received all refresh related changes.
     The table is current as of 2011-04-11-13.56.40.726392.
   A controlled shutdown request has been received from REDSCRIPTI
     Transformation Server will initiate a normal shutdown.
   --- Transformation Server for REDSCRIPTI terminating normally.
                                                                        Bottom
 F3=Exit              F11=Remove a message                 F12=Cancel
 F13=Remove all       F16=Remove all except unanswered     F24=More keys

                             MW                                      8/1
```

Figure 9-8 InfoSphere CDC subscription target events

You can find more detailed messages in the job logs of the InfoSphere CDC jobs. For example, Figure 9-9 shows a target event for a subscription (CPF4128).



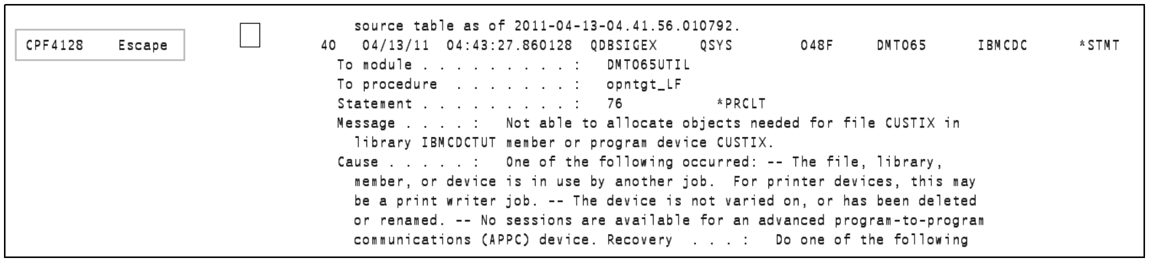
Event ID: 114
Type: Diagnostic
Time: 13 Apr 2011 04:43:27
Object: REDSCRIPTI

Message:

```
Open of member *FIRST of target table IBMCDCTUT/CUSTIX failed.

More Information: The open of member *FIRST of the target table key file CUSTIX
in library IBMCDCTUT failed due to error message CPF4128. See previous
messages in the joblog for more information.
User Profile: D_MIRROR
Event Severity: 40
Job Number: 394643
Job Name: REDSCRIPTI
CCSID: 65535
Replacement CCSID: 65535
```

Figure 9-9 Subscription event

The error entry in the event log indicates the InfoSphere CDC job on the system in which the error occurred. You can find the job log of the associated job by running the **`WRKJOB JOB(394643/D_MIRROR/REDSCRIPTI)`** command. In this case, the job had already ended, so it left a spooled file job log (`QPJOBLOG`) on the system (Figure 9-10).



```
CPF4128    Escape           source table as of 2011-04-13-04.41.56.010792.
                     40   04/13/11  04:43:27.860128  QDBSIGEX     QSYS        048F     DMT065      IBMCDC      *STMT
                       To module . . . . . . . . . :   DMT065UTIL
                       To procedure  . . . . . . . :   opntgt_LF
                       Statement . . . . . . . . . :   76          *PRCLT
                       Message . . . . :   Not able to allocate objects needed for file CUSTIX in
                         library IBMCDCTUT member or program device CUSTIX.
                       Cause . . . . . :   One of the following occurred: -- The file, library,
                         member, or device is in use by another job.  For printer devices, this may
                         be a print writer job. -- The device is not varied on, or has been deleted
                         or renamed. -- No sessions are available for an advanced program-to-program
                         communications (APPC) device. Recovery  . . . :   Do one of the following
```

Figure 9-10 InfoSphere CDC event error

System z event log monitoring

As with most System z applications, job logs are accessed through the System Display and Search Facility (SDSF). Standard data sets are JESMSGLG, JESJCL, and JESYSMSG.

There are also a few InfoSphere CDC for System z specific data sets, CHCPRINT, CHCAUDIT, and CHCREPRT:

- CHCPRINT contains a log of all event log messages issued since the instance was started.
- CHCAUDIT contains information about z/OS, DB2, and InfoSphere CDC for System z, including versions, maintenance levels, and configuration parameters, and significant events, such as users connecting to the data store from the Management Console, subscriptions starting and ending, and tables being parked.
- CHCREPRT contains reports written by InfoSphere CDC for System z during its processing, either by request or to report errors. An example of a report produced by request is the Staging Space Report. A target subscription also produces a report when a target error is encountered. For example, if an SQL error is encountered during the application, this report contains details about what was being applied and to which table.

9.4.6 Removing obsolete database logs

InfoSphere CDC depends on the availability of database transaction logs, and these logs must not be removed from the system before all subscriptions have finished processing them.

Most clients have an automated procedure for cleaning up archive transaction logs from the system to avoid flooding the disk space. When implementing these purging procedures, you must take the dependencies of InfoSphere CDC into account.

Linux, UNIX, and Windows engine log maintenance

Example 9-10 shows a sample script to demonstrate how an Oracle archive log cleanup procedure could be aligned with the log dependencies of InfoSphere CDC. Although there are multiple methods that can be used, this method looks at the archive logs registered in the Oracle catalog (`v$archived_log` view) and compares them with the log dependencies reported by InfoSphere CDC.

Example 9-10 Archive log clean-up procedure for Oracle

```
#!/usr/bin/ksh
# set -x                              # Uncomment to debug
# ******************************************************************** #
# Description    : Remove Oracle Archive Log Oracle already processed  #
#                  and applied by InfoSphere CDC (Transformation       #
#                  Server)                                             #
#                                                                      #
#                  Designed for InfoSphere CDC 6.3 and higher          #
#                  (metadata stored outside Oracle database)           #
#                                                                      #
# Language       : ksh                                                 #
# ******************************************************************** #

prg=${0##*(*/|-)}                     # Get script name

# Initialization of variables (to be customized for implementation)
CDC_HOME="/opt/InfoSphere CDCora" # CDC Home directory
SrcOraSID="cdcdemo" # SID (TNS name) of source database
SrcOraUser="InfoSphere CDCora" # User to log on to Oracle
SrcOraPwd="passw0rd" # Password of user
SrcCDCInstance="cdcdemo" # CDC Instance configured

TmpDepLogFile="/tmp/CDCDepLog$$.tmp"  # Temporary file holding the archive
                                      # files CDC is still dependent on
TmpArcLogFile="/tmp/CDCArcLog$$.tmp"  # Temporary file holding archive files
                                      # to be deleted
LogFile="${HOME}/CDCDltArcLog.log"    # Logging of all runs, change to
                                      # "/dev/null" if no logging desired

MaxLogDays=9999                       # Archive list threshold in days (to avoid
                                      # unnecessary processing of archives)

# Function to throw an error and exit with a return code
function throwError
{
  exitStatus=$?
  (( exitStatus == 0 )) && exitStatus=1

  if (( $# > 0 )); then
    print "${prg}: $*" >&2
  fi

  exit ${exitStatus}
}

# Function to write an entry in the log
function writeLog
```

```
{
  echo $1
  print $1 >> ${LogFile}
}

# --------------------------------------------------------------------------- #
# Main line                                                                   #
# --------------------------------------------------------------------------- #

# Define variables
typeset -i i=0
typeset -i nDlt=0
typeset -i nRtn=0
typeset -i nDltErr=0

# Check if log file exists or can be created and accessed
cat /dev/null >> ${LogFile}
(( $? == 0 )) || throwError "Cannot create or access log file: ${LogFile}"
# Check if temporary files can be created
cat /dev/null >> ${TmpDepLogFile}
(( $? == 0 )) ||
  throwError "Cannot create temporary log dependency file: ${TmpDepLogFile}"
cat /dev/null >> ${TmpArcLogFile}
(( $? == 0 )) ||
  throwError "Cannot create temporary archive log file: ${TmpArcLogFile}"

writeLog "## Archive log deletion for database ${SrcOraSID} started at `date`."

# First, obtain which log files are still needed by CDC
$CDC_HOME/bin/dmshowlogdependency -I ${SrcCDCInstance} -i -A > ${TmpDepLogFile}

# If there was an error retrieving the log dependencies, throw it
(( $? == 0 )) ||
  throwError "Error running dmshowlogdependency command for CDC instance
  ${SrcCDCInstance}"

# Report back the archive logs still required by the product
writeLog "Archive logs still required by CDC instance ${SrcCDCInstance}:"
cat ${TmpDepLogFile} | while read DepLog
do
  writeLog "${DepLog}"
done
nRtn=$(wc -l ${TmpDepLogFile} | awk '{print $1}')

# Now, list all the archive logs
```

```
sqlplus -S ${SrcOraUser}/${SrcOraPwd}@${SrcOraSID} << EOF > /dev/null
set lines 1000
set termout off
set echo off
set term off
set pages 0
set verify off
set feedback off
set trimspool on
spool ${TmpArcLogFile}
select name from v\$archived_log
       where next_time > (sysdate-${MaxLogDays})
order by dest_id,thread#,sequence#;
spool off
exit;
EOF

# If there was an error running the SQL script, throw it
(( $? == 0 )) ||
  throwError "Error running SQL script to list archive logs for database
  ${SrcOraSID}"

# Process list of archives which may be deleted
grep -v -f ${TmpDepLogFile} ${TmpArcLogFile} | while read ArcLine
do
  # Skip archive file if it doesn't exist anymore
  if [[ -e ${ArcLine} ]]; then
    rm ${ArcLine} 2>> ${LogFile}
    if (( $? != 0 )); then
      writeLog "Error deleting archive file ${ArcLine}"
      ((nDltErr += 1))
    else
      writeLog "Archive log ${ArcLine} deleted"
      ((nDlt += 1))
    fi
  fi
done

# Report how many archives deleted and retained
writeLog "Number of archives deleted: ${nDlt}"
if (( ${nDltErr} != 0 )) ; then
  writeLog "Number of errors when deleting archives: ${nDltErr}"
fi
writeLog "Number of archives retained: ${nRtn}"
```

```
# Remove temporary files
rm ${TmpDepLogFile}
rm ${TmpArcLogFile}

writeLog "## Archive log deletion for database ${SrcOraSID} finished at `date`."
writeLog " "
```

When running the script, it automatically removes all archive logs from the file system that are no longer needed by any of the subscriptions. The output of the script is shown in Example 9-11.

Example 9-11 Deleting the archive files

```
[InfoSphere CDCora@cdc-redscript bin]$ ./CDCDltArcLog.sh
## Archive log deletion for database cdcdemo started at Fri Apr 8
15:17:45 CEST 2011.
Archive logs still required by CDC instance cdcdemo:
IBM InfoSphere Change Data Capture Show Log Dependency Utility
Archive log /oradata/cdcdemo/archive/1_19_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_20_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_21_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_22_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_23_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_24_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_25_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_26_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_27_698772245.dbf deleted
Archive log /oradata/cdcdemo/archive/1_28_698772245.dbf deleted
Number of archives deleted: 10
Number of archives retained: 1
## Archive log deletion for database cdcdemo finished at Fri Apr 8
15:17:50 CEST 2011.
```

If the archive logs are kept in an Oracle ASM file system, they would not be accessible using operating system commands. In most Oracle implementations, clients probably already have a log management process in place using RMAN. The script in Example 9-11 must be adjusted to retrieve the log sequence number from the `v$archived_log` view and generate the RMAN commands (**`RMAN> delete noprompt archivelog logseq=nn`**) to remove the archive logs.

System i journal receiver maintenance

On System i, InfoSphere CDC subscription process journal receiver entries are stored as journal receiver (*JRNRCV) objects. After all subscriptions reading a certain journal have finished processing a journal receiver, this journal receiver can be removed from the system from a InfoSphere CDC perspective.

InfoSphere CDC for System i provides the **`CHGJRNDM`** command to clean up journal receivers from the system after the subscriptions no longer need them.

If there are other co-existing applications using the same journals that InfoSphere CDC does, management of the journal receivers should be coordinated between these applications. The oldest journal receiver that is still in use by any of the applications, and the next receivers in the chain, must be retained on the system.

For these situations, the **`RTVDMJENT`** command can be used to retrieve the oldest journal receiver needed by InfoSphere CDC. If you already have a journal management solution, see if it can check of the InfoSphere CDC journal receiver dependencies.

System z database log maintenance

InfoSphere CDC for System z contains no log management features. Active DB2 log data sets are automatically archived as they grow. When the DB2 IFI must read the log entries from the archived DB2 log data sets, this action might result in slow processing of the logs.

9.5 InfoSphere CDC API

By providing an API for accessing InfoSphere CDC Management Console functions within a Java application or program, you are able to create specialized software to satisfy needs and requirements unique to your InfoSphere CDC implementation. In particular, customized Java interfaces that include necessary enhancements and the ability to access some or all of InfoSphere CDC Management Console functions can be developed by your organization.

This section provides a background on how the API classes and methods are organized and how that relates to the functionality of the Management Console. You learn how to use the data stores, configure subscriptions and table mappings, and operate and monitor the environment.

9.5.1 Development environment setup

Development using the InfoSphere CDC Java APIs requires that you have a specific compilation and building environment configured. Although most Java Development Kits at release 1.5 or higher can be used to compile classes that use the APIs, use the IBM JDK for your development environment to be in line with the required running environment.

Running any custom programs that use the InfoSphere CDC Java API classes requires that you use IBM Java Runtime Engine V1.5 or later. If you attempt to instantiate an object from the InfoSphere CDC API using a Java Runtime Environment (JRE) from a different provider, exceptions are thrown.

The class path for compiling your self-developed classes must include the `api.jar` file. This file can be found in the `lib` directory of the Access Server or Management Console installation. Do not use the `api.jar` file that ships with the Java CDC replication engines, as it is for engine-only use. To avoid errors when Access Server or Management Console installations are done on your development server, consider copying the `api.jar` file to your development environment. The `api.jar` file works together with the version of Access Server that is running in your environment. Should a new version of the Access Server be installed, it might be necessary to replace the `api.jar` file with the latest version and review your code.

9.5.2 Contents of the api.jar file

The API shipped with InfoSphere CDC Management Console and Access Server contains the following three main packages:

- `com.datamirror.ea.api`
- `com.datamirror.ea.api.publisher`
- `com.datamirror.ea.api.subscriber`

The `com.datamirror.common.util.*` package is also useful, especially when automatic management of the data store is required. This package contains the methods for encryption and decryption of passwords.

Interfaces and classes that are contained in the API and expose InfoSphere CDC functions are in the API Javadocs. The index for the API Javadocs is found in the following directory:

```
<MC_install_directory>\api\index.html
```

Furthermore, a high-level description of the API and the architecture can be found in the IBM Information Center at the following address:

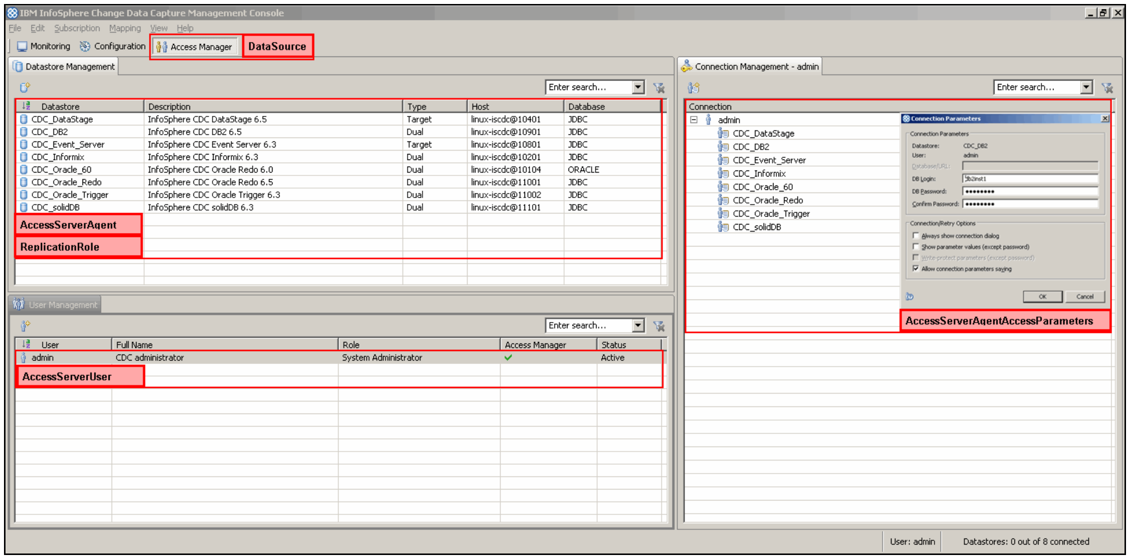
http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.mcapiandcommands.doc/concepts/aboutthisguide.html

Mapping the api.jar components to the Management Console

There are three main areas (packages) in the `api.jar` file, each encompassing a different part of the entire InfoSphere CDC configuration:

- `com.datamirror.ea.ap`: Classes related to Access Server and its configured users and data stores.
- `com.datamirror.ea.api.publisher`: Classes for controlling the source metadata, including the catalog tables and subscriptions (source side).
- `com.datamirror.ea.api.subscriber`: Classes for controlling the target metadata, mainly the publications (target component of subscriptions).

Figure 9-11 shows the Access Manager perspective of the InfoSphere CDC Management Console with the most important API classes mapped to screen items. All classes in this perspective are contained in the `com.datamirror.ea.api` package.



Datastore	Description	Type	Host	Database
CDC_DataStage	InfoSphere CDC DataStage 6.5	Target	linux-iscdc@10401	JDBC
CDC_DB2	InfoSphere CDC DB2 6.5	Dual	linux-iscdc@10901	JDBC
CDC_Event_Server	InfoSphere CDC Event Server 6.3	Target	linux-iscdc@10801	JDBC
CDC_Informix	InfoSphere CDC Informix 6.3	Dual	linux-iscdc@10201	JDBC
CDC_Oracle_60	InfoSphere CDC Oracle Redo 6.0	Dual	linux-iscdc@10104	ORACLE
CDC_Oracle_Redo	InfoSphere CDC Oracle Redo 6.5	Dual	linux-iscdc@11001	JDBC
CDC_Oracle_Trigger	InfoSphere CDC Oracle Trigger 6.3	Dual	linux-iscdc@11002	JDBC
CDC_solidDB	InfoSphere CDC solidDB 6.3	Dual	linux-iscdc@11101	JDBC

User	Full Name	Role	Access Manager	Status
admin	CDC administrator	System Administrator	✓	Active

Connection
- admin
  - CDC_DataStage
  - CDC_DB2
  - CDC_Event_Server
  - CDC_Informix
  - CDC_Oracle_60
  - CDC_Oracle_Redo
  - CDC_Oracle_Trigger
  - CDC_solidDB

Connection Parameters

Connection Parameters
Datastore: CDC_DB2
User: admin
Database/URL:
DB Login: db2inst1
DB Password: ●●●●●●●●●
Confirm Password: ●●●●●●●●●

Connection/Retry Options
- [ ] Always show connection dialog
- [ ] Show parameter values (except password)
- [ ] Write-protect parameters (except password)
- [x] Allow connection parameters saying

OK Cancel

User: admin   Datastores: 0 out of 8 connected

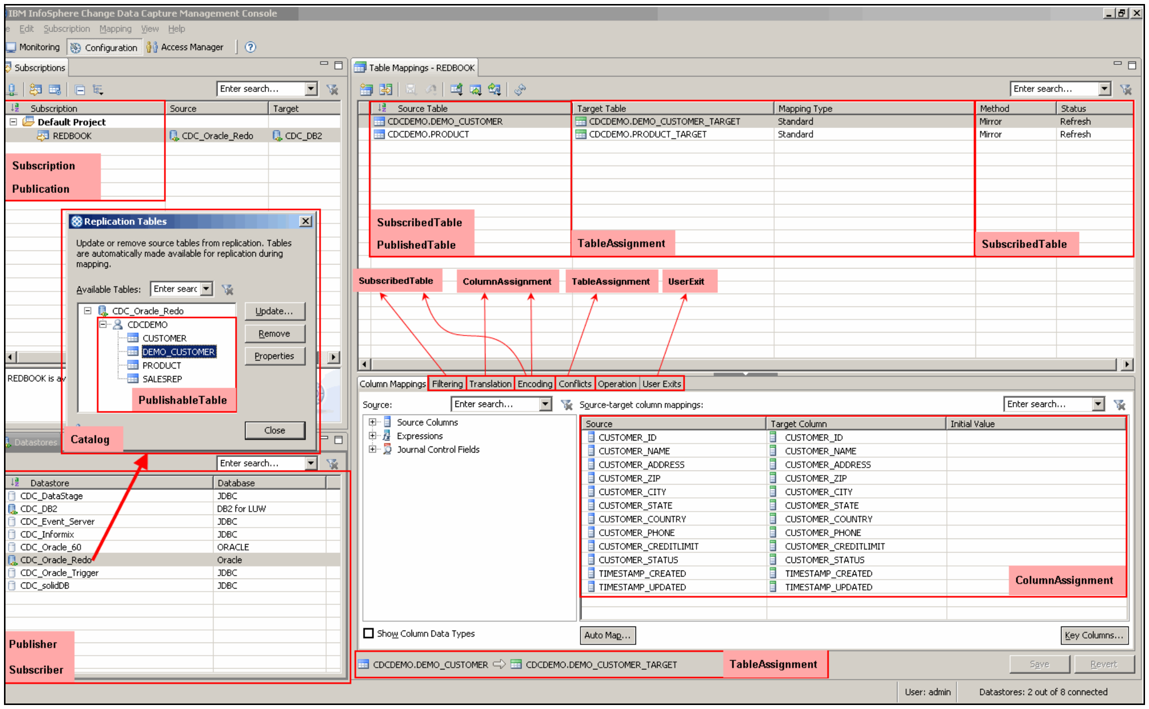
Figure 9-11 MC configuration mapping of API classes

Table 9-1 provides a list of the most commonly used classes contained in the `com.datamirror.ea.api package`.

Table 9-1 Commonly used classes in the com.datamirror.ea.api package

Class	Description
DataSource	Used to establish and keep the connection to the Access Server. Equivalent to connecting the Management Console to the Access Server and specifying the user and password.
AccessServerAgent	Represents an Access Server data store. After establishing a connection to the Access Server, you can obtain a list of data stores that have been registered.
AccessServerUser	Represents a user that is registered for the Access Server; the user can connect to the Access Server.
AccessServerAgentAccessParameters	Holds the link between an Access Server user and a data store. After establishing a connection to the Access Server, the user can only connect to the data stores for which a connection exists. Also, this object stores the database user and password that are used to connect to the database represented by the data store.
ReplicationRole	The interface that is implemented by the Publisher and Subscriber classes. An AccessServerAgent can be connected through one of the classes that implement the ReplicationRole interface.

Figure 9-12 shows the Configuration perspective of the Management Console and a map of the most commonly used API classes mapped to screen items. All classes in this perspective are contained in the `com.datamirror.ea.api.publisher` (source) and `com.datamirror.ea.api.subscriber` (target) packages.



Subscription	Source	Target
Default Project		
REDBOOK	CDC_Oracle_Redo	CDC_DB2

REDBOOK is a

Update or remove source tables from replication. Tables are automatically made available for replication during mapping.

Available Tables: Enter searc

- CDC_Oracle_Redo
  - CDCDEMO
    - CUSTOMER
    - DEMO_CUSTOMER
    - PRODUCT
    - SALESREP

Datastore	Database
CDC_DataStage	JDBC
CDC_DB2	DB2 for LUW
CDC_Event_Server	JDBC
CDC_Informix	JDBC
CDC_Oracle_60	ORACLE
CDC_Oracle_Redo	Oracle
CDC_Oracle_Trigger	JDBC
CDC_solidDB	JDBC

Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.DEMO_CUSTOMER	CDCDEMO.DEMO_CUSTOMER_TARGET	Standard	Mirror	Refresh
CDCDEMO.PRODUCT	CDCDEMO.PRODUCT_TARGET	Standard	Mirror	Refresh

Source: Enter search...

- Source Columns
- Expressions
- Journal Control Fields

Source-target column mappings:

Source	Target Column	Initial Value
CUSTOMER_ID	CUSTOMER_ID	
CUSTOMER_NAME	CUSTOMER_NAME	
CUSTOMER_ADDRESS	CUSTOMER_ADDRESS	
CUSTOMER_ZIP	CUSTOMER_ZIP	
CUSTOMER_CITY	CUSTOMER_CITY	
CUSTOMER_STATE	CUSTOMER_STATE	
CUSTOMER_COUNTRY	CUSTOMER_COUNTRY	
CUSTOMER_PHONE	CUSTOMER_PHONE	
CUSTOMER_CREDITLIMIT	CUSTOMER_CREDITLIMIT	
CUSTOMER_STATUS	CUSTOMER_STATUS	
TIMESTAMP_CREATED	TIMESTAMP_CREATED	
TIMESTAMP_UPDATED	TIMESTAMP_UPDATED	

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_TARGET

User: admin   Datastores: 2 out of 8 connected

Figure 9-12 Management Console configuration of commonly used API classes

Table 9-2 provides a list of the most commonly used classes for the configuration of subscriptions, which are contained in the `com.datamirror.ea.api.publisher` (source) and `com.datamirror.ea.api.subscriber` (target) packages.

Table 9-2 Commonly used subscription configuration packages

Class	Location	Description
Publisher	Source	The object representing a data store that has been connected to be used as a source. When connecting to a data store using the Publisher class, the source metadata is revealed to the Java API methods. Most InfoSphere CDC engines have a dual role and can be connected either as a source or as a target. If the engine must be used in both roles, you must instantiate a Publisher object and a Subscriber object.
Subscriber	Target	An object representing a data store that has been connected to be used as a target. When connecting to a data store using the Subscriber class, the target metadata is revealed to the Java API methods. Most InfoSphere CDC engines have a dual role and can be connected either as a source or as a target. If the engine must be used in both roles, you must instantiate a Publisher object and a Subscriber object.
Catalog	Source	A repository of tables that may be used to replicate from. Every database table from which you want to replicate data or changes must first be registered in the catalog. The catalog keeps a record of table names and table structure, together with some technical information, such as supplemental logging identification (journal and log group).
PublishableTable	Source	This class describes the source database table, including table level information, such as logging and the columns that make up the table, regardless whether they have been selected for replication or not. Among others, InfoSphere CDC keeps the table description to detect discrepancies between the real database table structure and the algorithm that InfoSphere CDC uses to process the database transaction log entries.

Class	Location	Description
Subscription	Source	The connection between the source and the target data store that references the source side of the table mappings. From the InfoSphere CDC metadata point of view, a subscription is the container of the source side of the table mapping definitions. A subscription is identified by a Subscription name, which is an uppercase string made up of a maximum of 30 characters. A subscription has a Publisher ID attribute (a string that has a maximum of eight uppercase characters) that is the unique identification of a subscription on the target side (Publication).
Publication	Target	The target side of a Subscription object. A Publication references the source tables that are selected for replication (SubscribedTable) as PublishedTable objects. A publication is identified by a Publisher ID, which is an uppercase string that has a maximum eight characters that are registered for the subscription.
SubscribedTable	Source	These tables are the Catalog tables (PublishableTable) that have been selected for replication by a subscription. When mapping tables in a subscription, the catalog tables definitions are copied to SubscribedTable objects under a Subscription. A SubscribedTable object can also hold derived columns (virtual columns) as DerivedColumn objects.
PublishedTable	Target	The target side of a SubscribedTable object. It holds the table and columns definitions of tables that are replicated by a subscription. When finding the PublishedTable object on the target side, the schema name and table name are converted to uppercase during the description. If your source schema / table is in lowercase, they show in uppercase in the PublishedTable object. A PublishedTable object is owned by a Publication (target side of a Subscription).

Class	Location	Description
TableAssignment	Target	Defines the link between PublishedTable (table coming from the source) and destination database table (or application object). TableAssignment is referenced by PublishedTable (one published table can be assigned to one destination object). A TableAssignment holds all information about how the source table is mapped to the target table, including operations and user exits.
ColumnAssignment	Target	Defines the mapping of source columns (PublishedDBColumn) to target columns, including derived expressions and journal control columns.

Figure 9-13 shows the relationship between the source and target metadata and how the various classes fit into this relationship. To keep the picture simple, we only included the main objects that make up subscriptions and table mappings. The relationship between source and target metadata is important to understand the table mapping process, which is covered in 9.5.7, “Procedure for mapping tables” on page 287.

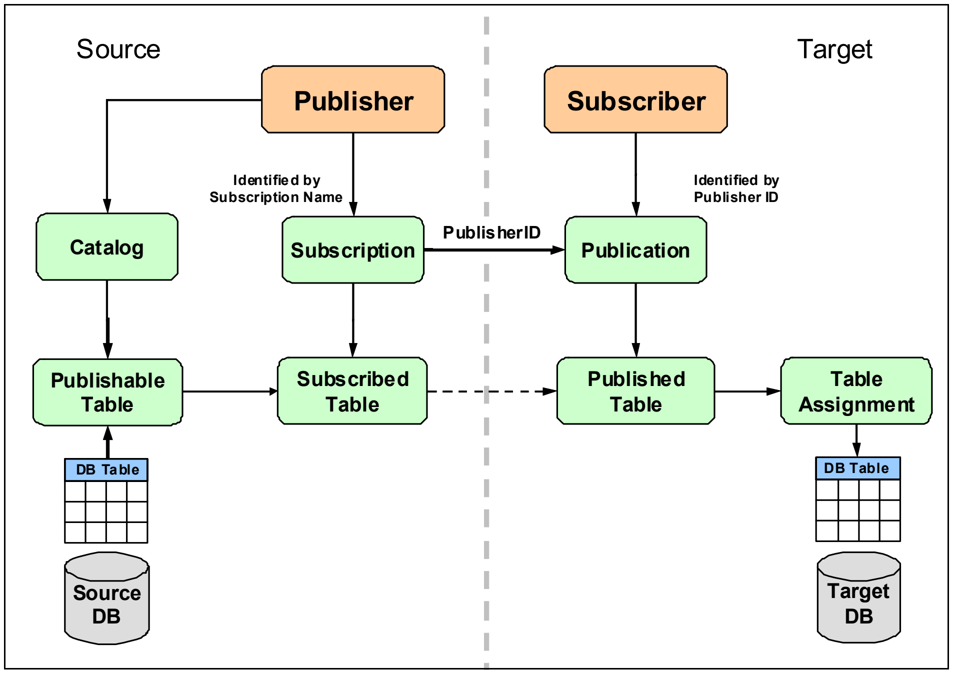


Figure 9-13 InfoSphere CDC source to target relationship

9.5.3 Connecting to and managing the Access Server

The first task in controlling InfoSphere CDC using the Java API is to connect to the Access Server. This action is the equivalent of starting the InfoSphere CDC Management Console and typing the user, password, host, and port of the Access Server. When connecting to the Access Server using the API, the exact same arguments must be passed to the method that connects to the DataSource object. The DataSource object is the object representing the Access Server.

The code first prepares the context of connecting to the Access Server and then tries to create the DataSource object and connect to it using the context. As with all API methods, the method throws an ApiException if it encounters a failure. When running the **connectAccessServer()** method, for example, that error is received.

Example 9-12 shows the sample code.

Example 9-12 Sample code to connect to Access Server

```
/**
 * Connect to an Access Server
 *
 * @param hostName
 *            - Access Server host name
 * @param port
 *            - Access Server port
 * @param userName
 *            - Access Server user
 * @param password
 *            - Access Server password
 * @return The connected Access Server
 * @throws ApiException
 */
public DataSource connectAccessServer(String hostName, Integer port,
      String userName, String password) throws ApiException {
   DataSource accessServer = null;
   DefaultContext asCtx = new DefaultContext();
   asCtx.setString(DataSource.Hostname, hostName);
   asCtx.setInt(DataSource.Port, port);
   asCtx.setString(DataSource.User, userName);
   asCtx.setString(DataSource.Password, password);
   accessServer = Toolkit.getDefaultToolkit().createDataSource();
   accessServer.connect(asCtx);
   System.out.println("Connected to Access Server on host " + hostName
      + " and port " + port + ".");
   return accessServer;
}
```

After you are finished with the custom processing, disconnect from the Access Server to release the connections. You could build in the disconnection in the general exception handling code of your custom program, as shown in Example 9-13.

Example 9-13 Sample code to disconnect from Access Server

```
/**
 * Close the existing connection to the Access Server
 *
 * @param accessServer
 *            - The Access Server from which you will be disconnected
 */
public void disconnectAccessServer(DataSource accessServer) {
   if (accessServer != null) {
      accessServer.close();
   }
   System.out.println("Disconnected from Access Server.");
}
```

Two attempts are made to connect to the Access Server. The first attempt fails with an invalid password and the second one is successful. The sample code is shown in Example 9-14.

Example 9-14 Example of connecting to Access Server

```
// Failed attempt
DataSource failedAccessServer = null;
System.out.println("Failed attempt to connect to Access Server:");
try {
   failedAccessServer = connectAccessServer("172.16.74.191", 10101,
         "admin", "invalid_passw0rd");
} catch (ApiException e) {
   System.err.println(e.getMessage());
}
// Successful attempt
DataSource accessServer = null;
System.out.println("Successful attempt to connect to Access Server:");
try {
   accessServer = connectAccessServer("172.16.74.191", 10101, "admin",
         "passw0rd");
} catch (ApiException e) {
   System.err.println(e.getMessage());
}
```

The output of Example 9-14 on page 271 is shown in Example 9-15.

Example 9-15 Successful connection to Access Server

```
Failed attempt to connect to Access Server:
Invalid user name or password.
Successful attempt to connect to Access Server:
Connected to Access Server on host 172.16.74.191 and port 10101.
```

After you have connected to the Access Server, the API does not automatically connect all data stores that the user is entitled to use. If you have kept the default Management Console settings, the Access Server attempts to connect to all data stores when started. The API allows you to selectively connect and disconnect to data stores as needed.

However, now that the connection has been established, you can list all data stores (agents) that have been defined in the Access Manager and attempt to connect to them. Creating connections is described in 9.5.4, “Connecting to the data stores” on page 277. Before elaborating on this subject, here are a few examples for retrieving information from the data store definitions. The first example is shown in Example 9-16.

Example 9-16 Retrieving information from data store definitions

```
/**
    * List the accessible data stores for the user connected to the Access
    * Server
    *
    * @param accessServer
    *            - The Access Server to which you're connected
    * @throws ApiException
    */
   public void listDataStores(DataSource accessServer) throws ApiException {
      SortedMap<String, ReplicationRole> agentMap = new TreeMap<String,
ReplicationRole>();
      ReplicationRole[] dataStores = accessServer.getAgents();
      accessServer.getAgentList();
      // First put all data stores in a map to remove duplicates. A data store
      // that has a dual role is listed twice when using the getAgentList()
      // method
      for (ReplicationRole dataStore : dataStores) {
         agentMap.put(dataStore.getName(), dataStore);
      }
      // Now iterate through the map entries to list
      Iterator<String> dataStoreList = agentMap.keySet().iterator();
      while (dataStoreList.hasNext()) {
         ReplicationRole dataStore = agentMap.get(dataStoreList.next());
         System.out.println("Datastore name   : " + dataStore.getName());
         System.out.println("Description      : "
               + dataStore.getDescription());
         System.out.print("Type             : ");
         if (accessServer.getAgentProperties(dataStore.getName())
               .getSourceOrTarget() == AccessServerAgent.SOURCE_OR_TARGET_DUAL)
            System.out.println("Dual");
         else if (accessServer.getAgentProperties(dataStore.getName())
               .getSourceOrTarget() == AccessServerAgent.SOURCE_OR_TARGET_TARGET)
            System.out.println("Target");
         else
            System.out.println("Source");
         System.out.println("Version          : " + dataStore.getVersion());
         System.out.println("Host             : " + dataStore.getHostName());
         System.out.println("Port             : " + dataStore.getPort());
         AccessServerAgentAccessParameters[] dataStoreParms = accessServer
               .getUserAgentProperties(accessServer.getUserProfile()
                     .getUserID(), dataStore.getName());
         System.out.println("Database user    : "
               + dataStoreParms[0].getDbLogin());
         System.out.println("Database password: "
               + dataStoreParms[0].getDbPassword());
         System.out.println();
      }
   }
```

You are already connected to the Access Server. The code sample in Example 9-16 on page 272 shows how to list all data stores to which the user has access. Use this approach if you want to connect to the data stores, subscription definitions, and table mappings. If the intention is to create additional data stores, and you want to ensure that you are not using an existing name, you could use the **`getAgentList()`** method. This method lists all data stores, regardless whether the user has access to them.

If a data store has a dual role (it can serve both as a source and as a target), the **`getAgents()`** method returns two entries of the data store, once for the source role and once for the target role. The **`listDataStores()`** method avoids displaying multiple entries of the same data store by first adding the entries to a sorted map and then listing the map entries.

For each data store, the name registered in the Access Server, the host name and port number on which the engine is running, and the database user and password that is used to connect to the target database / application is shown. Normally, you would not show the users and passwords in clear text, but you do so here to create a subscription later.

The output of the listing of accessible data stores looks similar to Example 9-17.

Example 9-17 List of accessible data stores

```
Data store name : CDC_DB2
Description : InfoSphere CDC DB2 6.5
Type : Dual
Version : V6R5M0T0BCDC_HOHDLPJU_164
Host : linux-iscdc
Port : 10901
Database user : db2inst1
Database password: passw0rd

eData store name : CDC_Oracle_Redo
Description : InfoSphere CDC Oracle Redo 6.5
Type : Dual
Version : V6R5M0T0BCDC_HOHDLPJU_165_38
Host : linux-InfoSphere CDC
Port : 11001
Database user : InfoSphere CDCora
Database password: passw0rd
```

Some of the methods defined for the data stores, such as retrieving the agent database extended ID (engine type) and version information, can only be called after the connection to the data store has been established. If you attempt to call these methods before connecting to the data store, the methods throw an ApiException.

In environments where many InfoSphere CDC engines are installed on various servers, you might want to automate the creation of the Access Server components, such as new data stores and users, and make the data stores available by assigning them to the entitled users. For a fully automated configuration of InfoSphere CDC in large environments, this step is the first step in the definition process, as shown in Example 9-18.

Example 9-18 Creating Access Server components

```
/**
 * Create a new Access Server user, datastore and assign the user to the
 * newly created data store
 *
 * @param accessServer
 *            - The Access Server to which you are connected
 * @throws ApiException
 */
public void createDataStoreAndUser(DataSource accessServer)
      throws ApiException {
   // Create the data store
   String dataStoreHost = "linux-iscdc";
   int dataStorePort = 11001;
   String dataStoreDefaultDB = "";
   String dataStoreDefaultDBUser = "iscdcora";
   String dataStoreDefaultDBPassword = “Passw0rd”;
   byte dataStoreMultiUser = (byte) 0;
   // First ping the CDC engine to check existence and retrieve
   // type/version, then create the data store
   AccessServerAgent dataStore = accessServer.pingAgent(dataStoreHost,
         dataStorePort);
   Result createDatastoreResult = accessServer.createNewAgent(
         "New_Datastore", "New Datastore Description",
         dataStore.getAgentType(), TransformationServer.COMM_TCPIP,
         dataStoreHost, dataStorePort, dataStore.getDatabaseType(),
         dataStore.getDmVersion(), dataStore.getSourceOrTarget(),
         dataStoreDefaultDB, dataStoreDefaultDBUser,
         Encryptor.encryptAndEncode(dataStoreDefaultDBPassword),
         dataStoreMultiUser);
   // Now create the new user
   String userRole = "TS System Administrator";
   String userPassword = "password";
   byte isAccountDisabled = (byte) 0;
   byte isAccountLocked = (byte) 0;
   byte isPasswordChangeRequired = (byte) 0;
   byte isPasswordNeverExpires = (byte) 1;
   byte enableUserDataStoreAdmin = (byte) 1;
   byte isForceSavePassword = (byte) 1;
   Result createUserResult = accessServer.createNewUser("New_User",
         "New User Full Name", userRole, "New User Description",
         Encryptor.encryptAndEncode(userPassword), isAccountDisabled,
         isAccountLocked, isPasswordChangeRequired,
         isPasswordNeverExpires, enableUserDataStoreAdmin,
         isForceSavePassword);
   // Assign the data store to the user
   if (createDatastoreResult.isSucceed() && createUserResult.isSucceed()) {
      byte showConnectionDialog = (byte) 0;
      byte showParamsValues = (byte) 1;
      byte writeProtectParams = (byte) 1;
      byte allowConnectionParamsSaving = (byte) 1;
      accessServer.addAgentAccessInUser("New_User", "New_Datastore",
         dataStoreDefaultDB, dataStoreDefaultDBUser,
         dataStoreDefaultDBPassword, showConnectionDialog,
         showParamsValues, writeProtectParams,
         allowConnectionParamsSaving);
   }
}
```

Before creating a data store for the Access Server, first run **`ping`** against the InfoSphere CDC engine in question to validate its existence and retrieve attributes such as the engine type, database type, and engine version. The **`pingAgent`** method establishes a TCP/IP connection between the Access Server and the InfoSphere CDC engine using the host and port you pass to it and retrieves those engine attributes.

Although you could create a data store and specify all the attributes, you might end up with a data store you cannot connect to. The Access Manager in the Management Console interface uses the same technique to validate a data store before allowing creation. When creating a data store, you can also store the default database connection parameters (database, database user, and password). If you choose to store the default connection parameters, assigning a new user to the data store takes these parameters as the database connection settings for that user. The password in the data store default database connection must be encrypted when passed.

For the new Access Server user, you must also pass the user's password in an encrypted manner. The password is validated to meet the password rules configured for the Access Server. If you want to create the user even if the password does pass the validation, you can force saving the password when calling the method.

For the user's role, the following string values are allowed (not case-sensitive):

- `"TS System Administrator"`
- `"TS Administrator"`
- `"TS Operator"`
- `"TS Monitor"`

After the data store and user have been created, you can assign the data store to the user to make it available when the user is connected. Again, the database, database user, and password can be specified for the user connected to the data store. The password must be passed in an unencrypted manner (however, the information is stored as encrypted). The **`getUserAgentProperties`** command also returns the decrypted password.

9.5.4 Connecting to the data stores

In accordance with the Management Console, you must connect to at least one data store before you can do anything with the replication definitions (subscriptions). Typically, you connect to a source and a target data store.

A data store can have a source, target, or dual (can serve both as source or target) role. All source configurations are accessible by classes in the `com.datamirror.ea.publisher` package, the main one being Publisher. The target configuration is accessible by classes in the `com.datamirror.ea.subscriber` package, the main one being Subscriber. Both classes implement the ReplicationRole interface and a number of common methods are inherited from this superclass.

Publisher is a replication role responsible for configuring the source side of the replication definitions and table mappings. It publishes source table definitions and database transactions that the source business applications generate to the target side. A Publisher refers to a Catalog, which maintains a list of source database tables that are available for subscriptions. Additionally, a Publisher object holds the source side of subscriptions, which define what and how data is being replicated.

Subscriber is a replication role and maintains a list of subscription targets (called Publications) and mappings from source tables to target (destination) tables or application objects. It subscribes to the information that the Publisher distributes (table definitions and database transactions) and uses it to perform activities on the target.

All data stores can be connected using the **`connect()`** method in the ReplicationRole class. Before establishing a connection to a data store, you need to determine the role of the data store. This role can either be Publisher or Subscriber. Before trying to establish a connection to a data store, run **`ping`** against the InfoSphere CDC engine to see if it is listening on the port and host configured for the data store. Use the **`pingAgent()`** method if you need to check InfoSphere CDC instance activity from an external monitoring solution.

In some environments, it might take some time before the connection to the data stores is established. This delay could be because of poor network connections, slow servers, or a combination of these and other factors. To avoid lengthy waits when trying to ping or connect to the data stores, set the communications timeout value. The maximum time the Access Server waits to establish a connection to the data store can then be controlled.

The **connectPublisherDataStore()** method connects to a data store and establishes a Publisher object that provides access to the source engine metadata. When connecting to the data store, the attempt times out if the connection takes more than 10 seconds (10,000 milliseconds). The sample code for the connection is shown in Example 9-19.

Example 9-19 Connect to a publisher (source) data store

```
/**
 * Connect to a publisher data store defined for an Access Server
 *
 * @param accessServer
 *            - The Access Server to which you're connected
 * @param dataStoreName
 *            - The name of the publisher data store
 * @return The Publisher, null if the data store is not found or could
 * not be connected
 * @throws ApiException
 */
public Publisher connectPublisherDataStore(DataSource accessServer,
      String dataStoreName) throws ApiException {
   int dataStoreTimeout = 10000;
   Publisher dataStore = null;
   dataStore = accessServer.getPublisher(dataStoreName);
   if (dataStore != null) {
      try {
         // First ping the data store to see if it can be reached
         accessServer.pingAgent(dataStore.getHostName(),
               dataStore.getPort());
      } catch (ApiException e) {
         System.out.println("No reponse from publisher datastore "
               + dataStoreName + " on " + dataStore.getHostName()
               + ":" + dataStore.getPort());
         return null;
      }
      // Now try to connect
      dataStore.setTimeOut(dataStoreTimeout);
      dataStore.connect();
      System.out.println("Connected to publisher datastore "
            + dataStoreName);
   }
   return dataStore;
}
```

In the **connectSubscriberDataStore()** method, connect to a data store and establish a Subscriber object that allows you to access the target metadata. The sample code for this connection is shown in Example 9-20.

Example 9-20 Connect to a subscriber (target) data store

```
/**
 * Connect to a subscriber data store defined for an Access Server
 *
 * @param accessServer
 *            - The Access Server to which you're connected
 * @param dataStoreName
 *            - The name of the subscriber data store
 * @return The Subscriber, null if the data store is not found or could
 * not be connected
 * @throws ApiException
 */
public Subscriber connectSubscriberDataStore(DataSource accessServer,
      String dataStoreName) throws ApiException {
   int dataStoreTimeout = 10000;
   Subscriber dataStore = null;
   dataStore = accessServer.getSubscriber(dataStoreName);
   if (dataStore != null) {
      try {
         // First ping the data store to see if it can be reached
         accessServer.pingAgent(dataStore.getHostName(),
               dataStore.getPort());
      } catch (ApiException e) {
         System.out.println("No reponse from subscriber datastore "
               + dataStoreName + " on " + dataStore.getHostName()
               + ":" + dataStore.getPort());
         return null;
      }
      // Now try to connect
      dataStore.setTimeOut(dataStoreTimeout);
      dataStore.connect();
      System.out.println("Connected to subscriber datastore "
            + dataStoreName);
   }
   return dataStore;
}
```

After a data store is connected, you can retrieve additional properties, such as engine type, and work with system parameters, subscriptions, and so on. The **listDataStoreDetail()** method determines, among other details, the InfoSphere CDC engine type, which an attribute that can only be retrieved after the connection to the engine has been established. Also, the version returned by the connected data store (**getAgentVersion**) is always accurate, but the version that is returned by the **getVersion()** method reflects the last version that was obtained when the Access Manager **ping** command was sent. The sample code for this method is shown in Example 9-21.

Example 9-21 List data store details

```
/**
 * List the details of a data store defined for an Access Server once it is connected
*
 * @param accessServer
 *            - The Access Server to which you're connected
 * @param dataStoreName
 *            - The name of the data store
 * @throws ApiException
 */
public void listDataStoreDetail(ReplicationRole dataStore)
      throws ApiException {
   System.out.println("Datastore name   : " + dataStore.getName());
   System.out.println("Description      : " + dataStore.getDescription());
   System.out.println("Engine type      : "
         + getDataStoreTypeString(dataStore.getExtendedDatabaseId()));
   System.out.println("Version          : " + dataStore.getAgentVersion());
   System.out.println();
}
```

If an additional method is defined to list the InfoSphere CDC engine type as a string, this definition could be useful if you want to build customized configuration and monitoring solutions across many different types of InfoSphere CDC engines. Suppose that you are writing custom code to import subscriptions from an XML file. You want to validate that both the source and target data stores you are importing to are of the same type as the data stores used when the XML file was generated. You want to accomplish this task to avoid compatibility problems. The sample code for this process is shown in Example 9-22.

Example 9-22 Retrieve engine type as a string

```
/**
 * Find and return the engine type of a data store
 *
 * @param dataStoreType
 *            - Type of the data store
 * @return the type name of the data store
 */
private String getDataStoreTypeString(int dataStoreType) {
   switch (dataStoreType) {
   case DataTypeUtilities.PRODUCT_DATASTAGE:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_DATASTAGE2;
   case DataTypeUtilities.PRODUCT_INFORMIX:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_INFORMIX;
   case DataTypeUtilities.PRODUCT_MSSQL:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_MSSQL2;
   case DataTypeUtilities.PRODUCT_ORACLE_REDO:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_ORACLE_REDO2;
   case DataTypeUtilities.PRODUCT_ORACLE_TRIGGER:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_ORACLE_TRIGGER2;
   case DataTypeUtilities.PRODUCT_SOLIDDB:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_SOLIDDB;
   case DataTypeUtilities.PRODUCT_SYBASE:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_SYBASE2;
   case DataTypeUtilities.PRODUCT_TERADATA:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_TERADATA;
   case DataTypeUtilities.PRODUCT_TSES:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_TSES;
   case DataTypeUtilities.PRODUCT_UDB:
      return DataTypeUtilities.EXTENDED_PRODUCT_NAME_UDB2;
   case DataTypeUtilities.DB_DB2_400:
      return DataTypeUtilities.DB_DB2_400_NAME;
   case DataTypeUtilities.DB_DB2_MVS:
      return DataTypeUtilities.DB_DB2_MVS_NAME;
   case DataTypeUtilities.DB_CLASSIC:
      return DataTypeUtilities.DB_CLASSIC_NAME;
   default:
      return "Datastore type " + dataStoreType + " not found.";
   }
}
```

If you connect to two of the data stores and then run the **`listDataStoreDetail()`** method, you can find additional information. The sample code to accomplish this task is shown in Example 9-23.

Example 9-23 Connecting to source and target data stores

```
Publisher db2DataStore = connectPublisherDataStore(accessServer,
      "CDC_DB2");
if (db2DataStore != null) {
   listDataStoreDetail(db2DataStore);
   db2DataStore.disconnect();
}
Subscriber dataStageDataStore = connectSubscriberDataStore(
      accessServer, "CDC_DataStage");
if (dataStageDataStore != null) {
   listDataStoreDetail(dataStageDataStore);
   dataStageDataStore.disconnect();
}
```

The output from Example 9-23 on page 281 is shown in Example 9-24.

Example 9-24 Output of data store details

```
Connected to publisher data store CDC_DB2
Data store name   : CDC_DB2
Description      : InfoSphere CDC DB2 6.5
Engine type      : IBM DB2
Version          : V6R5M0T0BCDC_HOHDLPJU_164

Connected to subscriber data store CDC_DataStage
Data store name   : CDC_DataStage
Description      : InfoSphere CDC DataStage 6.5
Engine type      : IBM InfoSphere DataStage
Version          : V6R5M0T0BCDC_HOHDLPJU_165_31
```

Now that you have seen how to connect to the Access Server and data stores, you can continue with managing subscriptions and table mappings, and start using the various classes related to Publisher and Subscriber.

There is one more technique that you can use: Locating the target data store for a subscription. A subscription is always created on a Publisher data store, and you do not find the name of the target data store in the subscription's properties. Instead, you find the host name (or IP address) and the port number to which connection must be established. Data store names can be flexibly assigned and are only known to the Access Server, not to the InfoSphere CDC replication engines.

Example 9-25 shows the **`getTargetDataStoreForSubscription()`** method, which obtains the target data store for the passed subscription.

Example 9-25 Sample getTargetDataStoreForSubscription() method

```
/**
 * Obtains the target data store for a subscription. Uses the subscriber host
 * name (or IP address) and port number to locate the target data store of
 * the subscription.
 *
 * @param accessServer
 *            - The Access Server to which you're connected
 * @param subscription
 *            - Subscription for which the target data store must be found
 * @return The Subscriber, null if the data store was not found
 * @throws ApiException
 */
public Subscriber getTargetDataStoreForSubscription(
      DataSource accessServer, Subscription subscription)
      throws ApiException {
   // Get the Publication for the Subscription
   Context subCtx = subscription.getProperty();
   String targetHost = subCtx.getString(Subscription.SubscriberLocation);
   int targetPort = subCtx.getInt(Subscription.SubscriberPort);
   ReplicationRole[] dataStores = accessServer.getAgents();
   for (ReplicationRole dataStore : dataStores) {
      if (dataStore.getHostName().equalsIgnoreCase(targetHost)
            && dataStore.getPort() == targetPort) {
         System.out.println("Target datastore for subscription "
               + subscription.getName() + " is " + dataStore.getName()
               + ".");
         return accessServer.getSubscriber(dataStore.getName());
      }
   }
   return null;
}
```

9.5.5 Configuring InfoSphere CDC replication

The configuration of InfoSphere CDC is kept in a set of internal metadata tables that are associated with a InfoSphere CDC instance. The operation of subscriptions within the InfoSphere CDC instance entirely depends on the metadata. In most common implementations, you do not need to be aware of the metadata and where it is located because the Management Console GUI provides a subscription-centric view of the configuration and conceals the complexity of the underlying metadata database. When working with the InfoSphere CDC Linux, UNIX, and Windows engine, the configuration metadata tables are kept in a proprietary database. Only the InfoSphere CDC for DB2 on System i and InfoSphere CDC for DB2 on System z engines keep the metadata in database tables.

Important: When designing your automatic configuration process, follow the steps InfoSphere CDC uses to create the individual items in the metadata. When creating subscriptions, you must work from the source towards the target or you will not be able to link the source and the target tables and start the subscriptions. More importantly, when removing subscriptions from your configuration, the steps must be done (almost) in the reverse order or you end up with orphaned metadata in the target InfoSphere CDC instance. You will not be able to remove this metadata unless you delete and recreate the InfoSphere CDC instance.

The following sections describe the various steps that are needed to create a subscription and map tables. Try to align the steps with the actions that the InfoSphere CDC Management Console performs.

9.5.6 Creating a subscription

Creating a subscription is always done against the source InfoSphere CDC data store (Publisher). To create a subscription using the API, you must provide a number of obvious context variables, such as subscription name and subscription description. Other context variables require additional explanation. The example code for creating a subscription only provides basic attributes, such as the source data store, target data store, subscription name, and subscription description. The remaining parameters are derived from the data store definition or kept at their defaults. The sample code for creating a subscription is shown in Example 9-26.

Example 9-26 Creating a subscription

```
/**
 * Create a subscription for a specific source data store and target data store
*
 * @param accessServer
 *            - The Access Server to which you are connected
 * @param sourceDataStore
 *            - The source data store of the subscription
 * @param targetDataStore
 *            - The target data store of the subscription
 * @param subscriptionName
 *            - Name of the subscription
 * @param subscriptionDescription
 *            - Description of the subscription
 * @return subscription
 * @throws ApiException
 */
public Subscription createSubscription(DataSource accessServer,
      Publisher sourceDataStore, Subscriber targetDataStore,
      String subscriptionName, String subscriptionDescription)
      throws ApiException {
   Subscription subscription = null;
   // Obtain the database login information of the target data store
   // using the currently logged in Access Server user
   AccessServerAgentAccessParameters[] targetAgentParms;
   targetAgentParms = accessServer.getUserAgentProperties(accessServer
         .getUserProfile().getUserID(), targetDataStore.getName());
   String databaseUser = targetAgentParms[0].getDbLogin();
   ;
   String databasePassword = targetAgentParms[0].getDbPassword();
   // Prepare context for creating subscription
   Context subCtx = new DefaultContext();
   subCtx.setString(Subscription.SubscriptionDescription,
         subscriptionDescription);
   subCtx.setString(
         Subscription.PublisherID,
         ((subscriptionName.length() > 8) ? subscriptionName.substring(
               0, 8).toUpperCase() : subscriptionName.toUpperCase()));
   subCtx.setString(
         Subscription.PublisherIDPending,
         ((subscriptionName.length() > 8) ? subscriptionName.substring(
               0, 8).toUpperCase() : subscriptionName.toUpperCase()));
   subCtx.setString(Subscription.PublisherDescription,
         subscriptionDescription);
   subCtx.setString(Subscription.PublisherDescriptionPending,
         subscriptionDescription);
   subCtx.setString(Subscription.CommunicationProtocol,
         TransformationServer
               .getProtocolByID(TransformationServer.COMM_TCPIP));
   subCtx.setString(Subscription.SubscriberOS,
         targetDataStore.getOSPlatform());
   subCtx.setString(Subscription.SubscriberDBPlatform,
         targetDataStore.getDBPlatform());
   subCtx.setString(Subscription.SubscriberVersion, targetDataStore
         .getAgentVersion().substring(0, 4));
   subCtx.setString(Subscription.SubscriberLocation,
         targetDataStore.getHostName());
   subCtx.setInt(Subscription.SubscriberPort, targetDataStore.getPort());
   subCtx.setString(Subscription.DatabaseUser, databaseUser);
   subCtx.setString(Subscription.DatabasePassword, databasePassword);
   // Source IP address or host name
   subCtx.setString(Subscription.PublisherLocation, "");
   subCtx.setInt(Subscription.PublisherPort, 0);
   // Source firewall port
   // Define whether the source and target engines support MBCS
   // automapping (CDC 6.5+)
   if (sourceDataStore.isMBCSAutomappingSupported()
         && targetDataStore.isMBCSAutomappingSupported())
      subCtx.setByte(Subscription.SubscriptionMBCSState, (byte) 2);
   else
      subCtx.setByte(Subscription.SubscriptionMBCSState, (byte) 1);
   // Define whether the target engine can accept transferable work
   // from the source engine
   if (sourceDataStore.isSubscriptionTransferableWorkSupported())
      subCtx.setByte(Subscription.SubscriptionTransferableWork, (byte) 1);
   else
      subCtx.setByte(Subscription.SubscriptionTransferableWork, (byte) 0);
   // Now create subscription in source data store
   subscription = sourceDataStore
         .addSubscription(subscriptionName, subCtx);
   System.out.println("Subscription " + subscriptionName + " created.");
   return subscription;
}
```

The publisher ID is the identification of the subscription on the target data store (Subscriber) and has a maximum length of eight. In most cases, using the uppercase left eight characters of the subscription name establishes the publisher ID. However, if you have multiple subscriptions with almost matching names, such as NEW_SUBSCRIPTION_1 and NEW_SUBSCRIPTION_2, this action leads to duplicate publisher IDs (NEW_SUBS) and the Subscriber does not create the Publication when you describe the second subscription.

Ensure that you create unique names for the publisher ID for each target data store. You can use the **`getPublicationNames()`** method against the target data store to get a list of existing publisher IDs.

The **`PublisherIDPending`** argument must be populated with the publisher ID you want to assign to the subscription (identification of a subscription in the target metadata). After the subscription source metadata has been sent to the target and the uniqueness of the publisher ID has been confirmed, **`PublisherIDPending`** is blanked out and **`PublisherID`** is populated with the specified value. The same situation applies to **`PublisherDescriptionPending`**.

To have the subscription successfully find the target InfoSphere CDC engine, you must specify the host / IP address and port number to establish the connection. Additionally, the source engine needs to understand the format of the information that must be sent across to the target. For that reason, the operating system platform and database platform parameters must be specified.

During subscription creation, you can determine whether the source and target engines support multibyte character set (MBCS) automapping. MBCS is useful when specifying the encoding conversions for the columns in 9.5.13, “Encoding conversions (before and after Version 6.5)” on page 315.

After the subscription has connected to the target engine, it connects to the target database using the database user and password specified in the subscription parameters. For security reasons, the database credentials must be specified in the subscription to confirm the source system is allowed to send data to the target database.

9.5.7 Procedure for mapping tables

Figure 9-14 shows the successive steps that must be performed for table mappings in InfoSphere CDC.

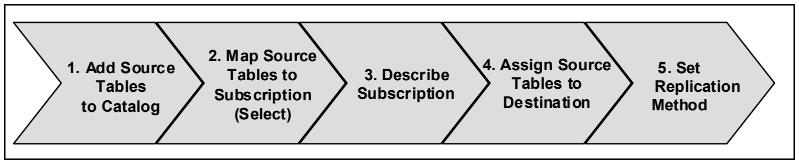

1. Add Source Tables to Catalog
2. Map Source Tables to Subscription (Select)
3. Describe Subscription
4. Assign Source Tables to Destination
5. Set Replication Method

Figure 9-14 Table mapping process

The process includes the functions of add catalog, select, describe, assign, and set method, as described in the following list:

1. Add Source Tables to Catalog.

   Before mapping any source table in a subscription, the table must exist in the InfoSphere CDC catalog. Among other things, this separate store is needed to determine if the real table, as defined in the source database, matches with what InfoSphere CDC expects to find when parsing entries in the database log. When you add a table to the catalog, InfoSphere CDC retrieves the table structure from the database and stores information about the table location (path), internal database object ID, column names, data types, and length in its Publisher metadata. A table that is located in the catalog does not necessarily have to be used in any subscription, but can be used by one or more subscriptions. Only tables that are used as a source for replication must be added to the catalog; tables that are only used as a target do not have to be present in the catalog.

2. Map Source Tables to Subscription (Select).

   This process takes the tables from the catalog and selects them for replication in the subscription. When selecting the tables for replication, you can also configure the row and column filtering and code page conversions. This step is the final step on the source side to configure tables to be replicated.

3. Describe Subscription.

   To inform the target InfoSphere CDC engine that tables will be published, the subscription definition and the definitions for selected tables must be sent to the target engine. In InfoSphere CDC, this activity is called describing the subscription. The subscription definition (Subscription) is sent to the target and creates a publication (Publication) object. Also, all definitions of tables that have been selected for replication in the subscription are sent to the target. The SubscribedTable objects on the source are turned into PublishedTable objects on the target. When describing a subscription, the source InfoSphere CDC engine makes a connection to the target InfoSphere CDC engine through the same channel that is normally used for replicating the data changes. If the path to the target server (host or port) or connection to the database (database name, user and password) cannot be established, the describe process fails, and the target metadata is not populated.

4. Assign Source Tables to Destination.

   After the source tables are known on the target InfoSphere CDC engine, they can be linked to their destination, for example, the target database tables. Through this linkage, the target InfoSphere CDC engine knows how to direct incoming database operations to the designated destination and which column mapping to apply.

5. Set Replication Method.

   The final step in the mapping process is to set the replication method (Mirror or Refresh). Replication methods are kept with the source table, as they tell the source engine whether to refresh the table contents (Refresh) or the changes from the database log (Mirror). Even though the replication method could be set in an earlier step, you should set it as the final step. Depending on the engine, logging (that is, DATA CAPTURE CHANGES flag for DB2 for Linux, UNIX, and Windows, table full supplemental logging for Oracle, publication for SQL Server, and journaling for DB2/400) for the source table in question must be activated if a table replication method is Mirror. Most database management systems require a short exclusive lock on the table to activate / deactivate the logging. By setting the replication method as the final step, the table mapping has already been accomplished. Should the method not be able to acquire an exclusive lock on the table in question, only this step must be repeated instead of the entire mapping process.

9.5.8 Table mapping example

This example creates a simple table mapping from Oracle to DB2. The example first addresses the individual steps and, at the end of this section, ties them together.

The sample code for the method shown in Example 9-27 adds a table to the Publisher catalog if it does not exist yet.

Example 9-27 Add a table to the catalog

```
/**
 * Add a table to the source data store catalog
 *
 * @param sourceDataStore
 * @param tablePath
 * @param tableName
 * @return publisableTable
 * @throws ApiException
 */
public PublishableTable addTableCatalog(Publisher sourceDataStore,
      String tablePath, String tableName) throws ApiException {
   PublishableTable publishableTable = null;
   // Build DBTable object to represent source table
   DBTable newTable = sourceDataStore.getTable(
         sourceDataStore.getDBPath(tablePath), tableName);
   Catalog sourceCatalog = sourceDataStore.getCatalog();
   // Only add table to the catalog if not already existing
   if (!sourceCatalog.isTablePublishable(newTable)) {
```

```
            sourceCatalog.addTable(newTable);
            System.out.println("Table " + tablePath + "." + tableName
                  + " added to catalog of datastore "
                  + sourceDataStore.getName() + ".");
         } else {
            System.out.println("Table " + tablePath + "." + tableName
                  + " already existed in catalog of datastore "
                  + sourceDataStore.getName() + ".");
         }
         publishableTable = sourceCatalog.getPublishableTable(tablePath,
               tableName);
         return publishableTable;
      }
```

A table can be replicated by multiple subscriptions, so the table could have already been added to the catalog. To determine if the table has already been added, you must create a DBTable object that is composed of a DBPath object that represents the schema and a string representing the table name. In some databases, such as DB2 on System z, the table is not qualified by a schema name, but a hierarchy of qualifiers.

For historical reasons within the InfoSphere CDC describe processing, you need to specify the pending publisher ID and pending publisher description when creating a subscription. When doing a description for the first time, InfoSphere CDC validates the publisher ID on the target side for uniqueness and, if accepted, the pending publisher ID and pending publisher description are blanked out. This action is not apparent to the user and probably is of no concern for definitions using the Java APIs, but is mentioned for completeness if you choose to analyze the subscription attributes.

After the subscription exists, you can select the tables to be replicated in the subscription. However, only tables that have been previously added to the catalog can be added to the subscription. The sample code to perform this action is shown in Example 9-28.

Example 9-28 Select source table for replication

```
/**
 * Select a source table to the subscription and set its method-status
 * to Refresh-Idle
 *
 * @param sourceDataStore
 *            - The source data store of the subscription
 * @param subscription
 *            - Subscription
 * @param publishableTable
 *            - Table from the catalog to be selected
 * @return SubscribedTable
 * @throws ApiException
 */
public SubscribedTable selectTable(Publisher sourceDataStore,
      Subscription subscription, PublishableTable publishableTable)
      throws ApiException {
   // Select the table to the subscription
   subscription.addTable(publishableTable);
   SubscribedTable subscribedTable = subscription.getSubscribedTable(
         publishableTable.getUpperPath(), publishableTable.getName());
   // Set method to Refresh and status to Parked
   try {
      subscribedTable.setReplicationMethod(
            SubscribedTable.METHOD_REFRESH, null, (byte) 0);
      subscribedTable.setReplicationStatus(SubscribedTable.STATUS_IDLE);
   } catch (ApiException e) {
      e.printStackTrace();
   }
   System.out.println("Table " + publishableTable.getUpperPath().getName()
         + "." + publishableTable.getName()
         + " selected to subscription " + subscription.getName() + ".");
   return subscribedTable;
}
```

Although you want to set the eventual replication method and status immediately after having added the table to the subscription, set this setting in a separate step. This separate step avoids the wait for table locks if supplemental logging must be activated on the database table.

The source side of the subscription is now finished and you can describe the Publisher side subscription metadata to the target (Subscriber) side. When starting the description, the source InfoSphere CDC engine establishes a connection to the target InfoSphere CDC engine using the host name and port used when creating the subscription. If the target engine responds, it tries to connect to the target database / application using the database user and password specified when the subscription was created. The sample code for describing the subscription is shown in Example 9-29.

Example 9-29 Describing the subscription

```
/**
 * Describe the subscription to the target data store
 *
 * @param subscription
 * @param targetDataStore
 * @throws ApiException
 * @throws InterruptedException
 */
public void describeSubscription(Subscription subscription,
      Subscriber targetDataStore) throws ApiException,
      InterruptedException {
   System.out.println("Describing subscription " +
subscription.getName()
         + ".");
   subscription.describe();
   waitForDescribe(subscription, targetDataStore);
}
```

After the **describe()** method has been issued for the subscription, it could take a few seconds or longer before the metadata has arrived on the Subscriber side. Your custom code needs to wait long enough to allow the description to complete and yet not wait too long if the description has failed. Unfortunately, there is no built-in method available to verify whether the description has finished and succeeded. The procedure that is most successful is a combination of a small wait time after the description has been submitted and then checking the existence of the target metadata. The sample code for this action is shown in Example 9-30.

Example 9-30 Waiting for the description to complete

```
/**
 * Waiting for the subscription to be described
 *
 * @param subscription
 *            - Subscription to be described
 * @param targetDataStore
 *            - The target data store of the subscription
 * @throws UnsupportedFeatureException
 * @throws ApiException
 * @throws InterruptedException
 */
private void waitForDescribe(Subscription subscription,
   Subscriber targetDataStore) throws ApiException,
   InterruptedException {
int minWaitDescribeStart = 2000;
// Minimum time the describe will take int maxWaitDescribeStop = 1800000;
// Maximum wait time for describe to be finished
int maxWaitPublication = 30000;
```

```
// Maximum time to wait for publication created/deleted
int maxWaitTableDescribed = 10000;
// Maximum time to wait for tables describe (per table)
int waitInterval = 300; // Wait interval
// First give the describe some time to start
Thread.sleep(minWaitDescribeStart);
// Wait until the subscription is no longer active
int waitedTimeDescribe = 0;
   byte[] status;
   do {
      System.out
            .println("Waiting until describe activity of subscription "
                  + subscription.getName() + " finished.");
         Thread.sleep(waitInterval);
         status = subscription.getLiveActivityStatus();
         waitedTimeDescribe += waitInterval;
         if (status[1] != Subscription.LIVE_STATUS_IDLE
               && waitedTimeDescribe > maxWaitDescribeStop) {
            throw new ApiException(
               "Timeout while waiting for describe for subscription "
                     + subscription.getName() + " to finish.");
         }
      } while (status[1] != Subscription.LIVE_STATUS_IDLE);
// Retrieve all selected tables for the subscription
      ArrayList<SubscribedTable> subscribedTables = new
ArrayList<SubscribedTable>();
      for (DBPath dbPath : subscription.getSubscribedTableDBPaths()) {
         for (SubscribedTable subscribedTable : subscription
               .getSubscribedTables(dbPath)) {
            subscribedTables.add(subscribedTable);
         }
      }
      subscribedTables.trimToSize();
      if (subscribedTables.size() != 0) {
// Wait for publication to be created
         int waitedTimePublication = 0;
         Publication publication = null;
         do {
            System.out
                  .println("Waiting for creation of the publication for subscription "
                        + subscription.getName() + ".");
            Thread.sleep(waitInterval);
            waitedTimePublication += waitInterval;
            publication = getSubscriptionPublication(subscription,
                  targetDataStore);
            if (publication == null
                  && waitedTimePublication > maxWaitPublication) {
               throw new ApiException(
                     "Timeout while waiting for publication to be created for "
                           + subscription.getName() + ".");
            }
         } while (publication == null);
// Wait for all source tables to be described
         int waitedTimeTablesDescribed = 0;
         int numberSubscribedTables = subscribedTables.size();
         int numberTablesDescribed = 0;
         while (numberTablesDescribed < numberSubscribedTables) {
            publication.refresh();
            numberTablesDescribed = 0;
            for (SubscribedTable subscribedTable : subscribedTables) {
               PublishedTable publishedTable = getPublishedTableForSubscribedTable(
                     subscription, subscribedTable, targetDataStore);
               if (publishedTable != null)
                  numberTablesDescribed += 1;
            }
            System.out.println("Waiting for describe of "
                  + numberSubscribedTables
                  + " tables. Remaining number of tables: "
                  + (numberSubscribedTables - numberTablesDescribed)
                  + ".");
            Thread.sleep(waitInterval);
            waitedTimeTablesDescribed += waitInterval;
            if (subscribedTables.size() > 0
                  && waitedTimeTablesDescribed > (numberSubscribedTables *
                     maxWaitTableDescribed))
               throw new ApiException(
               "Timeout while waiting for all tables to be described.");
         }
      } else {
// Wait until Publication no longer exists (maximum 30 seconds)
         int waitedTime = 0;
         Publication publication;
         do {
            System.out
                  .println("Waiting for deletion of publication for subscription "
                        + subscription.getName() + ".");
            targetDataStore.refresh();
            publication = getSubscriptionPublication(subscription,
                  targetDataStore);
            Thread.sleep(waitInterval);
            if (publication == null && waitedTime > maxWaitPublication)
               throw new ApiException(
                     "Timeout while waiting for publication to be removed for "
                           + subscription.getName() + ".");
         } while (publication != null);
// No more tables in the subscription
   }
```

It is best to define a method to retrieve the Publication object for a subscription, as the link between source and target metadata must be made when doing automated mappings. The sample code for this action is shown in Example 9-31.

Example 9-31 Retrieving the Publication object for a subscription

```
/**
 * Get the Publication object for a subscription through the Publisher ID.
 * As long as the subscription has not yet been described successfully, the
 * Pending Publisher ID must be used to identify the Publication on the
 * target data store.
 *
 * @param subscription
 *            - Subscription for which you want to obtain the Publication
 * @param targetDataStore
 *            - The target data store of the subscription
 * @return The Publication that was found
 * @throws ApiException
 */
public Publication getSubscriptionPublication(Subscription subscription,
      Subscriber targetDataStore) throws ApiException {
   // Get the Publication for the Subscription
   Context subCtx = subscription.getProperty();
   String publisherID = (subCtx.getString(Subscription.PublisherID)
         .isEmpty() ? subCtx.getString(Subscription.PublisherIDPending)
         : subCtx.getString(Subscription.PublisherID));
   Publication publication = targetDataStore.getPublication(publisherID);
   return publication;
}
```

The first part of the sample code codes an initial wait of two seconds. This wait time allows a subscription to be started. An alternative to using the wait is to poll for subscription describe activity. However, on some fast servers, the description process could last a short time and you run the chance of missing the description process activity.

The second part of the sample code builds the list of SubscribedTables (selected tables) for the subscription. This action is primarily done to determine whether you expect the Publication to be created and exist after the description has completed or whether it should be removed. Removal of the publication is done by the description process when all table mappings have been removed. More details about this action can be found in 9.5.9, "Procedure for removing mapped tables" on page 303.

In the next phase, you wait for the subscription to become inactive (`status[1]!=Subscription.LIVE_STATUS_IDLE`). Typically, the description process takes a few seconds, but in some environments with slow or unstable networks, and if there are many tables to describe, it could take longer, so set the timeout value to 1800 seconds.

After the description is complete, the Publication has been created in the target metadata. If it has not yet been created, any action on the Publication could result in a null pointer exception.

The code iterates through the list of subscribed (selected) tables for the subscription and checks if the PublishedTable object has already been created on the target side. The PublishedTable object identification (path and table name) is not always an exact match with the SubscribedTable object identification. Sometimes aliasing is applied to the source tables to reduce the length of schema and table names, and any lowercase source table names are translated to uppercase (for example, when the source is IBM Informix®). Use a special method, **`getPublishedTableForSubscribedTable()`**, to retrieve the PublishedTable object for a SubscribedTable. The code for **`getPublishedTableForSubscribedTable`** can be found in Example 9-32.

Example 9-32 Retrieving the PublishedTable for a SubscribedTable

```
/**
 * Retrieves the PublishedTable (target) object for a SubscribedTable
 * (source) object. First, the alias of the SubscribedTable name and path is
 * retrieved through its PublishableTable object that is part of the
 * Catalog. Subsequently, the method tries to retrieve the PublishedTable
 * object through the alias. If not found, an attempt is made to retrieve
 * through the full table identification.
 *
 * @param subscription
 *           The subscription that holds the SubscribedTable object.
 * @param subscribedTable
 *           SubscribedTable object for which the PublishedTable object
 *           must be retrieved.
 * @param targetDataStore
 *           The target data store which holds the target metadata for the
 *            passed subscription.
 * @return The PublishedTable object that was found, else null.
 * @throws ApiException
 */
private PublishedTable getPublishedTableForSubscribedTable(
      Subscription subscription, SubscribedTable subscribedTable,
      Subscriber targetDataStore) throws ApiException {
   Publisher sourceDataStore = subscription.getPublisher();
   Catalog catalog = sourceDataStore.getCatalog();
   String sourceTablePath = subscribedTable.getUpperPath().getFullName();
   String sourceTableName = subscribedTable.getName();
   PublishableTable publishableTable = catalog.getPublishableTable(
         sourceTablePath, sourceTableName);
   String sourceTablePathAlias = publishableTable.getPathAlias();
   String sourceTableNameAlias = publishableTable.getTableAlias();
   Publication publication = getSubscriptionPublication(subscription,
         targetDataStore);
   // First check if the PublishedTable can be retrieved using the alias
   PublishedTable publishedTable = publication.getPublishedTable(
         targetDataStore.createDBPath(sourceTablePathAlias),
         sourceTableNameAlias);
   if (publishedTable == null) {
      try {
         publishedTable = publication.getPublishedTable(
               targetDataStore.createDBPath(sourceTablePath),
               sourceTableName);
      } catch (Exception ignore) {
      }
   }
   return publishedTable;
}
```

The maximum wait time depends on the number of tables to be described (10 seconds per table).

The final piece of code of the **`waitForDescribe()`** method is for the situation where there are no tables selected for the subscription. If that is the case, the description process removes the Publication from the target metadata. This condition is the condition that the loop waits for.

Now that the PublishedTable objects have been created on the target side, you can link them to the target tables (assign activity), as shown in Example 9-33.

Example 9-33 Linking published table objects to target tables (assign)

```
/**
 * @param subscription
 *            - Subscription for which the table must be assigned
 * @param targetDataStore
 *            - Target data store of the subscription
 * @param subscribedTable
 *            - Subscribed (selected) table
 * @param targetTablePath
 *            - Schema (path) of the target table
 * @param targetTableName
 *            - Name of the target table
 * @throws ApiException
 */
public void assignTable(Subscription subscription,
      Subscriber targetDataStore, SubscribedTable subscribedTable,
      String targetTablePath, String targetTableName) throws ApiException {
   // Get the Publication for the Subscription
   Context subCtx = subscription.getProperty();
   String publisherID = (subCtx.getString(Subscription.PublisherID)
         .isEmpty() ? subCtx.getString(Subscription.PublisherIDPending)
         : subCtx.getString(Subscription.PublisherID));
   Publication publication = targetDataStore.getPublication(publisherID);
   // Get the PublishedTable associated with the SubscribedTable object
   DBPath publishedTableDBPath = targetDataStore
         .createDBPath(subscribedTable.getUpperPath().getFullName()
               .toUpperCase());
   String publishedTableName = subscribedTable.getName().toUpperCase();
   PublishedTable publishedTable = publication.getPublishedTable(
         publishedTableDBPath, publishedTableName);
   // Map the table to the target table using standard mapping type
   String targetDatabase = "";
   String destinedMember = null;
   String indexLibrary = null;
   String indexName = null;
   publishedTable.assign(targetTableName, targetTablePath, targetDatabase,
         destinedMember, indexLibrary, indexName,
         PublishedTable.STANDARD);
   System.out.println("Table " + subscribedTable.getUpperPath().getName()
         + "." + subscribedTable.getName() + " assigned to "
         + targetTablePath + "." + targetTableName + ".");
   TableAssignment tableAssignment = publishedTable.getTableAssignment();
   // Loop through all target columns and map values depending on name
   String[] destinedColumnNames = tableAssignment.getDestinedColumnNames();
   for (String targetColumnName : destinedColumnNames) {
      ColumnAssignment columnAssignment = tableAssignment
            .getColumnAssignment(targetColumnName);
      if (targetColumnName.equals("AUD_TIMESTAMP")) {
         columnAssignment.mapTo(ColumnAssignment.MAP_JOURNAL_CONTROL,
               "&TIMSTAMP");
         System.out
               .println("Journal control column &TIMSTAMP assigned to column "
                     + targetColumnName + ".");
      } else if ((targetColumnName.equals("APPLY_TIMESTAMP"))) {
         columnAssignment.mapTo(ColumnAssignment.MAP_CURRENT_DATE);
         System.out
               .println("Default value CURRENT DATE assigned to column "
                     + targetColumnName + ".");
      }
   }
}
```

The example has been kept straightforward to accommodate most of the mapping scenarios you want to accomplish. **PublishedTable.assign()** has a number of overloaded methods suited for different mapping types. For Standard, Adaptive Apply, and LiveAudit mapping types, the simplest method to use is the one used in the **assignTable()** example. When leaving the index name and library blank, the InfoSphere CDC engine automatically chooses the most appropriate unique key index when starting the apply process.

A number of parameters, such as **targetDatabase**, **destinedMember**, **indexLibrary**, and **indexName**, are passed as null. The **destinedMember** parameter is only applicable to DB2 on System i, and the **indexLibrary** and **indexName** parameters are only applicable if you want to choose the index that InfoSphere CDC should use to determine the unique keys of the target table. The **targetDatabase** parameter is applicable if the target engine is InfoSphere CDC for DB2 on z/OS. This engine is the only engine that requires the target database and uses a composite table path (database.schema). The last argument (**PublishedTable.STANDARD**) indicates the apply method to be used for this mapped table.

When the table is assigned, InfoSphere CDC automatically maps any source and target columns that have the same names and compatible data types. This behavior is the same that the Management Console exposes. In the example, after the table has been successfully assigned, you also iterate through the columns and map a journal control column and constant value to the target column.

If the target InfoSphere CDC engine is associated with a database (as in this example), use the target columns as a basis to configure the column mappings. For a few engines, such as InfoSphere CDC for DataStage and InfoSphere CDC Event Server, the PublishedTable object is the basis for all columns being mapped to the target engine. Target columns are not available in these engines.

Now that the source table has been assigned to a target, the table mapping is completed. However, during the table selection (selectTable) processing, set the replication method to Refresh and status to Parked to avoid potential failure when activating supplemental logging for the source tables.

Similar to what the Management Console does, set the replication method as the final step. If the method is set to mirror and supplemental logging cannot be started (for example, because of locks on the table), at least you have completed the table mapping. You can then change the replication methods through the Management Console or use the API without having to worry about incomplete table mappings. The sample code to accomplish this task is shown in Example 9-34.

Example 9-34 Setting the replication method

```
/**
 * @param subscribedTable
 *          - Source table selected for the subscription
 * @throws ApiException
 */
public void setReplicationMethod(
      SubscribedTable subscribedTable) throws ApiException {
   DBTable journalTable = null;
   subscribedTable.setReplicationMethod(
         SubscribedTable.METHOD_MIRROR, journalTable,
         SubscribedTable.MEMBER_SELECTION_SINGLE);
   subscribedTable
         .setReplicationStatus(SubscribedTable.STATUS_REFRESH);
   System.out.println("Replication method and status set"
         + " to Mirror-Refresh for table "
         + subscribedTable.getUpperPath().getName() + "."
```

```
            + subscribedTable.getName() + ".");
   }
```

If the replication method is set to mirror (METHOD_MIRROR), InfoSphere CDC determines if supplemental logging (that is, journaling and data capture changes) has already been enabled for the source table in question. If not, InfoSphere CDC activates logging automatically depending on the source engine you are running. The **journalTable** parameter is only applicable to InfoSphere CDC for Oracle Trigger-based, and the member selection final parameter is applicable to DB2 for System i source tables that have multiple members.

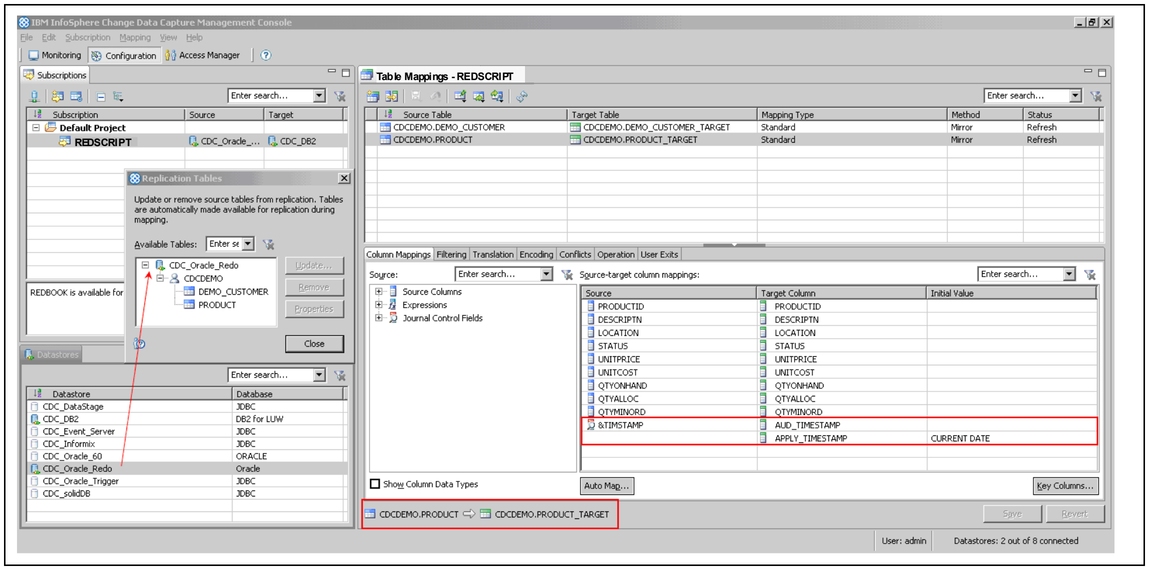
Now that you have all the individual methods to map tables, place them in sequence to create a subscription and map two tables. The sample code for that task is shown in Example 9-35.

Example 9-35 Create subscription and map two tables

```
Publisher oracleDataStore = connectPublisherDataStore(accessServer,
      "CDC_Oracle_Redo");
Subscriber db2DataStore = connectSubscriberDataStore(accessServer,
      "CDC_DB2");
if (oracleDataStore != null && db2DataStore != null) {
   Subscription subscription = null;
   // Create the subscription
   subscription = createSubscription(accessServer, oracleDataStore,
         db2DataStore, "REDSCRIPT", "Redscript example");
   // Add the tables to the catalog
   PublishableTable[] publishableTables = new PublishableTable[2];
   publishableTables[0] = addTableCatalog(oracleDataStore, "CDCDEMO",
         "DEMO_CUSTOMER");
   publishableTables[1] = addTableCatalog(oracleDataStore, "CDCDEMO",
         "PRODUCT");
   System.out.println("Schema alias: "
         + publishableTables[0].getPathAlias());
   System.out.println("Table alias: "
         + publishableTables[0].getTableAlias());
   // Select tables to subscription
   SubscribedTable[] subscribedTables = new SubscribedTable[2];
   subscribedTables[0] = selectTable(oracleDataStore, subscription,
         publishableTables[0]);
   subscribedTables[1] = selectTable(oracleDataStore, subscription,
         publishableTables[1]);
   // Describe the subscription
   describeSubscription(subscription, db2DataStore);
   // Assign the tables
```

```
        assignTable(subscription, db2DataStore, subscribedTables[0],
              "CDCDEMO", "DEMO_CUSTOMER_TARGET");
        assignTable(subscription, db2DataStore, subscribedTables[1],
              "CDCDEMO", "PRODUCT_TARGET");
        // Set the replication method for the tables
        setReplicationMethod(subscribedTables[0]);
        setReplicationMethod(subscribedTables[1]);
     }
     // Disconnect from the data stores
     if (oracleDataStore != null)
        oracleDataStore.disconnect();
     if (db2DataStore != null)
        db2DataStore.disconnect();
```

After the snippet has finished running, you can open the subscription in the Management Console and show the table mappings. If you select the source data store and open the replication tables, you see the tables. Figure 9-15 shows the table mapping after running the code. The CDCDEMO.DEMO_CUSTOMER and CDCDEMO.PRODUCT source tables have been added to the catalog (Replication Tables) and are also mapped in subscription REDSCRIPT. The columns AUD_TIMESTAMP and APPLY_TIMESTAMP have been mapped to journal control column &TIMSTAMP and initial value CURRENT DATE.



Subscription	Source	Target
Default Project		
REDSCRIPT	CDC_Oracle_...	CDC_DB2

REDBOOK is available for

Update or remove source tables from replication. Tables are automatically made available for replication during mapping.

CDC_Oracle_Redo
CDCDEMO
DEMO_CUSTOMER
PRODUCT

Datastore	Database
CDC_DataStage	JDBC
CDC_DB2	DB2 for LUW
CDC_Event_Server	JDBC
CDC_Informix	JDBC
CDC_Oracle_60	ORACLE
CDC_Oracle_Redo	Oracle
CDC_Oracle_Trigger	JDBC
CDC_solidDB	JDBC

Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.DEMO_CUSTOMER	CDCDEMO.DEMO_CUSTOMER_TARGET	Standard	Mirror	Refresh
CDCDEMO.PRODUCT	CDCDEMO.PRODUCT_TARGET	Standard	Mirror	Refresh

Source	Target Column	Initial Value
PRODUCTID	PRODUCTID	
DESCRIPTN	DESCRIPTN	
LOCATION	LOCATION	
STATUS	STATUS	
UNITPRICE	UNITPRICE	
UNITCOST	UNITCOST	
QTYONHAND	QTYONHAND	
QTYALLOC	QTYALLOC	
QTYMINORD	QTYMINORD	
&TIMSTAMP	AUD_TIMESTAMP	
	APPLY_TIMESTAMP	CURRENT DATE

CDCDEMO.PRODUCT ⇨ CDCDEMO.PRODUCT_TARGET

User: admin   Datastores: 2 out of 8 connected

Figure 9-15 Management Console -Table Mapping

9.5.9 Procedure for removing mapped tables

You must follow the procedure for removing mapped tables (unassign, deselect, describe, and remove the catalog) exactly to avoid orphaned metadata in the source or target agent's metadata. The Java API cannot remove orphaned metadata; only the Management Console can do so. Figure 9-16 shows the successive steps that must be performed to unmap tables in a subscription and remove the table from the catalog.

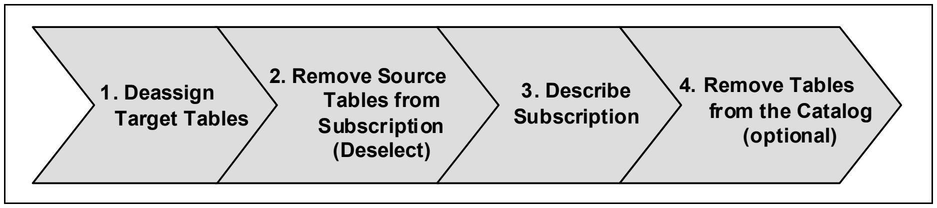

1. Deassign Target Tables
2. Remove Source Tables from Subscription (Deselect)
3. Describe Subscription
4. Remove Tables from the Catalog (optional)

Figure 9-16 InfoSphere CDC - table mapping delete process

The steps in Figure 9-16 are briefly described in the following list:

1. Deassign Target Tables.

   This step removes the link between the source and target table and removes any information about mapped columns, operations, and user exits that have been configured for the table mapping. At this stage, it does not remove the source table entry in the target engine's metadata, and the bookmark of the subscription and marked table capture points are kept.

2. Remove Source Table from Subscription (deselect).

   When removing the source table from the subscription, the subscription no longer replicates the changes for this table. Also, any source side configuration of the previously mapped table, such as column filtering, row filtering, and code page conversions, is removed from the source metadata. When deselecting tables at the source, the marked table capture point is lost. When selecting the same table to the subscription again, it assumes a new capture point.

3. Describe

   Describing the subscription causes the InfoSphere CDC source engine to communicate with the InfoSphere CDC target engine. The target InfoSphere CDC engine is informed about the tables that are still replicated according to the source engine. Any table that is no longer selected for replication in the subscription (source side) and has been unassigned on the target side is removed from the target metadata. Additionally, if there are no more tables replicated, it removes the publication entry of the subscription.

It is important to stress that any table that is still assigned on the target InfoSphere CDC engine is not removed from the target metadata and is orphaned. Additionally, you could delete the subscription from the source metadata and even leave the publication orphaned on the target side. After a publication has been orphaned, you can no longer remove it from the target engine's metadata using the API. You either must remove it through the Management Console or recreate the entire InfoSphere CDC instance.

4. Remove Tables from Catalog (optional).

   Database tables that are no longer mapped by any subscription can be removed from the source engine's catalog. InfoSphere CDC does not allow you to remove catalog entries of any tables that are still referenced in a subscription, and throws an exception if you try to remove it.

9.5.10 Table mapping removal example

This example removes the previously created subscription and its table mappings. Also, the tables re removed from the source data store catalog. The individual steps are being addressed and then they are tied together in consecutive calls.

First, remove tables is to unlink the target tables from the source tables (the deassign process). Remember that it is important to perform the deassignment to allow the description process to remove the published table eventually. The sample code for this task is shown in Example 9-36.

Example 9-36 Deassign a table

```
/**
 * @param sourceDataStore
 *            - Source data store of the subscription
 * @param targetDataStore
 *            - Target data store of the subscription
 * @param subscription
 *            - Subscription for which the table must be deassigned
 * @param sourceTablePath
 *            - Schema (path) of the source table
 * @param sourceTableName
 *            - Name of the source table
 * @throws ApiException
 */
public void deassignTable(Publisher sourceDataStore,
      Subscriber targetDataStore, Subscription subscription,
      String sourceTablePath, String sourceTableName) throws ApiException {
   // Get the Publication for the Subscription
```

```
    Publication publication = getSubscriptionPublication(subscription,
          targetDataStore);
    // Get the PublishedTable object
    PublishedTable publishedTable = publication.getPublishedTable(
          sourceDataStore.getDBPath(sourceTablePath), sourceTableName);
    publishedTable.deassign();
    System.out.println("Table " + sourceTablePath + "." + sourceTableName
          + " deassigned from publication "
          + publication.getPublisherID() + ".");
}
```

We chose to only provide limited information for deassigning the tables, that is, the source data store, target data store, subscription, source table path, and source table name. Using these keys, the PublishedTable object can be retrieved from the target data store (through the Publication).

It might seem counter-intuitive to provide the source table path and source table name, but the table assignment is kept under the PublishedTable object, which is effectively the equivalent of the SubscribedTable (selected table) on the source side. The table's schema (table path) can be established by the source data store **`getDBPath()`** method.

Now deselect the source table from the subscription. The sample code for this action is shown in Example 9-37.

Example 9-37 Deselect the source table from the subscription

```
/**
 * Deselect a source table from the subscription
 *
 * @param sourceDataStore
 *            - The source data store of the subscription
 * @param subscription
 *            - Subscription
 * @param tablePath
 * @param tableName
 * @throws ApiException
 */
public void deselectTable(Publisher sourceDataStore,
      Subscription subscription, String tablePath, String tableName)
      throws ApiException {
   SubscribedTable subscribedTable = subscription.getSubscribedTable(
         sourceDataStore.getDBPath(tablePath), tableName);
   subscription.removeTable(subscribedTable);
   System.out.println("Table " + tablePath + "." + tableName
         + " deselected from subscription " + subscription.getName()
         + ".");
}
```

After the table has been deselected from the subscription, the description process must be run to remove the PublishedTable object from the target Publication. The sample code for this action is shown in Example 9-38.

Example 9-38 Remove a table from source data store catalog

```
/**
 * Remove a table from the source data store catalog
 *
 * @param sourceDataStore
 * @param tablePath
 * @param tableName
 * @throws ApiException
 */
public void deleteTableCatalog(Publisher sourceDataStore, String 
tablePath,
      String tableName) throws ApiException {
   Catalog sourceCatalog = sourceDataStore.getCatalog();
   PublishableTable publishableTable = 
sourceCatalog.getPublishableTable(
         tablePath, tableName);
   if (publishableTable != null) {
      sourceCatalog.removeTable(publishableTable, true);
      System.out.println("Table " + tablePath + "." + tableName
            + " removed from catalog of datastore "
            + sourceDataStore.getName() + ".");
   } else {
      System.out.println("Table " + tablePath + "." + tableName
            + " does not exist in catalog of datastore "
            + sourceDataStore.getName() + ".");
   }
}
```

When setting the description, any PublishedTable object that is no longer assigned and has no equivalent selected table (SubscribedTable) in the subscription is removed from the target metadata. The description process is the same as described in 9.5.8, “Table mapping example” on page 289, but if there are no more selected tables for the subscription, the **`waitDescribe()`** method waits for the removal of the Publication object from the target metadata.

If the source table is no longer used in any subscription, it can also be removed from the source data store catalog.

Removal of the table from the catalog is optional. However, if the structure of the source table changes over time, for example, when columns are added or removed, the information stored in the InfoSphere CDC catalog would no longer apply. Therefore, always remove tables from the catalog after they are no longer needed.

The method shown in Example 9-39 removes the subscription from the source data store.

Example 9-39 Removal of the subscription from the source data store

```
/**
 * Delete a subscription from the source data store. The method first checks
 * that the publication has already been removed from the target data store
 * before allowing to delete the subscription.
 *
 * @param sourceDataStore
 *            - Source data store from which the subscription must be removed
 * @param subscriptionName
 *            - The name of the subscription you want to delete
 * @param targetDataStore
 *            - The target data store of the subscription
 * @throws ApiException
 */
public void deleteSubscription(Publisher sourceDataStore,
      Subscription subscription, Subscriber targetDataStore)
      throws ApiException {
   // Verify that the publication no longer exists
   Publication publication = getSubscriptionPublication(subscription,
         targetDataStore);
   if (publication != null) {
      throw new ApiException("Publication with publisher ID "
            + publication.getPublisherID()
            + " still exists, cannot delete subscription.");
   } else {
      sourceDataStore.removeSubscription(subscription);
   }
}
```

In the method, check that the Publication in the target data store has been removed, which prevents orphaned metadata.

Now that you have the individual methods to remove table mappings and delete a subscription, you can run them in sequence and remove the table mappings and subscriptions that had been created. The sample code for this action is shown in Example 9-40.

Example 9-40 Running the methods in sequence

```
Publisher oracleDataStore = connectPublisherDataStore(accessServer,
      "CDC_Oracle_Redo");
Subscriber db2DataStore = connectSubscriberDataStore(accessServer,
      "CDC_DB2");
if (oracleDataStore != null && db2DataStore != null) {
/*
* Delete the subscription and catalog tables
*/
Subscription subscription = oracleDataStore
      .getSubscription("REDSCRIPT");
// Deassign the tables from the subscription
      deassignTable(oracleDataStore, db2DataStore, subscription,
         "CDCDEMO", "DEMO_CUSTOMER");
      deassignTable(oracleDataStore, db2DataStore, subscription,
         "CDCDEMO", "PRODUCT");
// Deselect the tables from the subscription
   deselectTable(oracleDataStore, subscription, "CDCDEMO",
         "DEMO_CUSTOMER");
   deselectTable(oracleDataStore, subscription, "CDCDEMO", "PRODUCT");
// Describe the subscription
   describeSubscription(subscription, db2DataStore);
// Delete the tables from the catalog
   deleteTableCatalog(oracleDataStore, "CDCDEMO", "DEMO_CUSTOMER");
   deleteTableCatalog(oracleDataStore, "CDCDEMO", "PRODUCT");
// Delete the subscription
   deleteSubscription(oracleDataStore, subscription, db2DataStore);
}
// Disconnect from the data stores
if (oracleDataStore != null)
   oracleDataStore.disconnect();
if (db2DataStore != null)
   db2DataStore.disconnect();

Example 9-39a Result of deleting the subscription
Connected to publisher data store CDC_Oracle_Redo
Connected to subscriber data store CDC_DB2
Table CDCDEMO.DEMO_CUSTOMER deassigned from publication REDSCRIPT.
Table CDCDEMO.PRODUCT deassigned from publication REDSCRIPT.
Table CDCDEMO.DEMO_CUSTOMER deselected from subscription REDSCRIPT.
Table CDCDEMO.PRODUCT deselected from subscription REDSCRIPT.
Describing subscription REDSCRIPT.
Waiting until describe activity of subscription REDSCRIPT finished.
Waiting for deletion of publication REDSCRIPT for subscription REDSCRIPT.
Table CDCDEMO.DEMO_CUSTOMER removed from catalog of data store CDC_Oracle_Redo.
Table CDCDEMO.PRODUCT removed from catalog of data store CDC_Oracle_Redo.
Disconnected from Access Server.
```

9.5.11 Row and column filtering

Many clients use InfoSphere CDC to exclude certain rows and columns from the replication process, which is called row filtering and column filtering. By filtering rows or columns, the volume of changes sent to the target can be dramatically reduced. Filtering columns that you do not need on the target side has an additional positive side effect: If an update only affects non-replicated columns, the entire change is not sent to the target.

A good example is the CRM application that keeps a record of users accessing customer opportunity information. Every table has a column that indicates when a record was last retrieved and the user who retrieved it, and this column is updated on every read operation (the reading of a record automatically becomes an update). If the information about when a user retrieves the record is not important for target processing, you can clear the last retrieve time stamp and the user columns (column filtering) and reduce the number of operations that are sent to the target.

Row and column filtering is configured at the source side, in the SubscribedTable object. Configure the row and column selection (especially the column selection) when the table is selected to the subscription. If columns are cleared later in the process, this action affects the PublishedTable object on the target, which requires that the subscription be described again. If you forget to redescribe the subscription after having cleared columns from replication, the subscription is not run until it has been described again.

Example 9-41 shows the **`setFilter()`** method that accepts a SubscribedTable object as an argument. If the table has a STATUS column, this column is used to set the row filtering for the table in question. If the table has a TIMESTAMP_UPDATED column, it is cleared from the replicated columns.

Example 9-41 Setting the row and column filtering

```
/**
 * Sets the row filter to STATUS='A' if the table has a STATUS column.
 * Clears the TIMESTAMP_UPDATED column from replication
 *
 * @param subscribedTable
 *            - The table that is selected to the subscription
 * @throws ApiException
 */
public void setFilter(SubscribedTable subscribedTable) throws
ApiException {
   // Set the row filter if the replicated table has a STATUS column
   if (subscribedTable.getColumn("STATUS") != null) {
      subscribedTable.setRowSelection("STATUS='A'",
            SubscribedTable.ROW_SELECTION_SELECT);
      System.out.println("Row selection STATUS='A' set for table "
            + subscribedTable.getFullName() + ".");
   }
   // Clear the TIMESTAMP_UPDATED column from replication if it exists
   DBColumn timestampUpdatedColumn = subscribedTable
         .getColumn("TIMESTAMP_UPDATED");
   if (timestampUpdatedColumn != null) {

subscribedTable.setColumnSelected(timestampUpdatedColumn.getName(),
            false, false);
      System.out.println("Column " + timestampUpdatedColumn.getName()
            + " deselected from replication for table "
            + subscribedTable.getFullName() + ".");
   }
}
```

The **setRowSelected()** method sets the row filter. The sample code for this action is shown in Example 9-42.

Example 9-42 Implementing row and column filtering

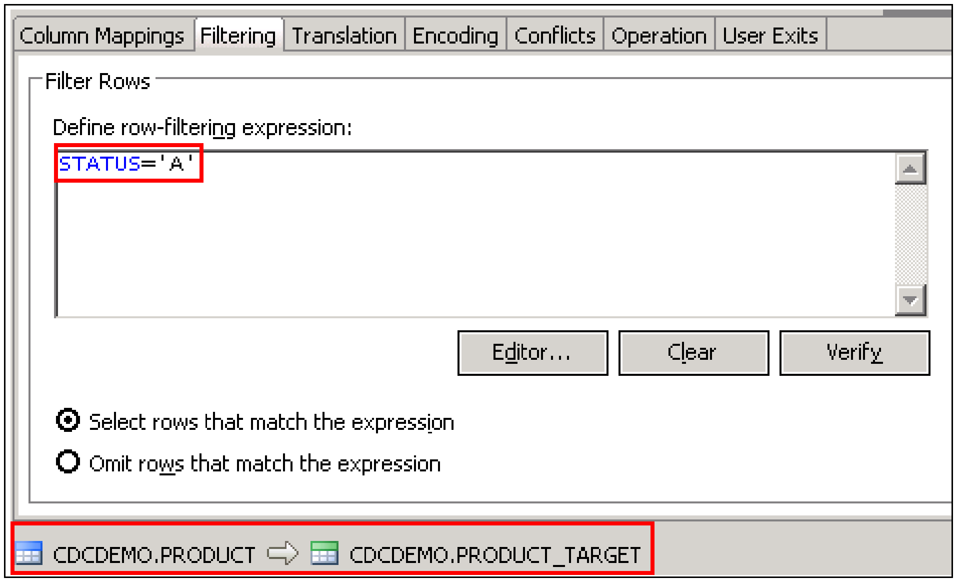
```
...
// Select tables to subscription
SubscribedTable[] subscribedTables = new SubscribedTable[2];
subscribedTables[0] = selectTable(oracleDataStore, subscription,
      publishableTables[0]);
subscribedTables[1] = selectTable(oracleDataStore, subscription,
      publishableTables[1]);
// Set the row and column filtering
setFilter(subscribedTables[0]);
setFilter(subscribedTables[1]);
// Describe the subscription
```

```
describeSubscription(subscription, db2DataStore);
...
```

Similar to the Management Console, you can specify whether the row selection condition causes rows to be selected for or omitted from replication (they are each other's counterparts). For the **`setColumnSelected()`** method, you must specify the selected and the critical attributes. When setting the selected attribute to false, the column is cleared from replication. When you select a column as critical, InfoSphere CDC only replicates update operations when any critical column has changed value. A critical column cannot be cleared from replication, so there are three possible combinations of selected / critical.

This example includes the **`setFilter()`** method shown in Example 9-41 on page 309 in the table mapping flow.

After recreating the subscription and mapping the tables, Figure 9-17 shows that row filtering was applied to the CDCDEMO.PRODUCT table because it has a STATUS column.



Define row-filtering expression:

```
STATUS='A'
```

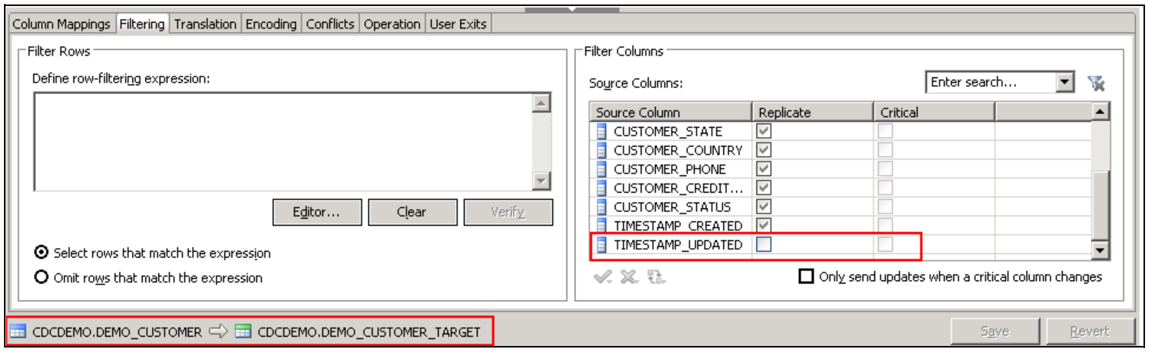
[✓ Select rows that match the expression]

Omit rows that match the expression

CDCDEMO.PRODUCT ⇨ CDCDEMO.PRODUCT_TARGET

Figure 9-17 Row filtering

For the CDCDEMO.DEMO_CUSTOMER table, column filtering has been applied by running the **setFilter()** method (Figure 9-18).



[✓ Select rows that match the expression] Omit rows that match the expression

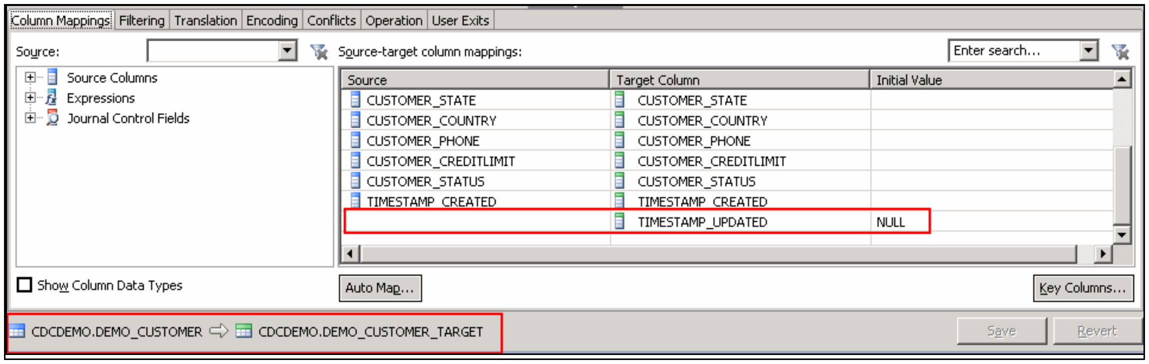
Source Column	Replicate	Critical
CUSTOMER_STATE	✓	☐
CUSTOMER_COUNTRY	✓	☐
CUSTOMER_PHONE	✓	☐
CUSTOMER_CREDIT...	✓	☐
CUSTOMER_STATUS	✓	☐
TIMESTAMP_CREATED	✓	☐
TIMESTAMP_UPDATED	☐	☐

☐ Only send updates when a critical column changes

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_TARGET

Figure 9-18 InfoSphere CDC - API column filter

Figure 9-18 shows that the TIMESTAMP_UPDATED column has been cleared from replication. At the same time, the automated column mapping was affected because the TIMESTAMP_UPDATED column was no longer sent to the target during the description process. Therefore, the **assign()** method could not map the source column to the target column of the same name (Figure 9-19).



Source	Target Column	Initial Value
CUSTOMER_STATE	CUSTOMER_STATE	
CUSTOMER_COUNTRY	CUSTOMER_COUNTRY	
CUSTOMER_PHONE	CUSTOMER_PHONE	
CUSTOMER_CREDITLIMIT	CUSTOMER_CREDITLIMIT	
CUSTOMER_STATUS	CUSTOMER_STATUS	
TIMESTAMP_CREATED	TIMESTAMP_CREATED	
	TIMESTAMP_UPDATED	NULL

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_TARGET

Figure 9-19 Column filter

9.5.12 Derived columns

InfoSphere CDC allows you to move the processing of a derived expression from the target to the source. This action might be useful in situations where the information needed to calculate a column cannot be found on the target, for example, a %GETCOL retrieval of a column of a table that is only on the source). This action also might be useful if you need to call a user function (%USERFUNC) that is not available on the target engine (for example, when replicating from Oracle to DB2 on z/OS and you need to start a Java method to perform a calculation).

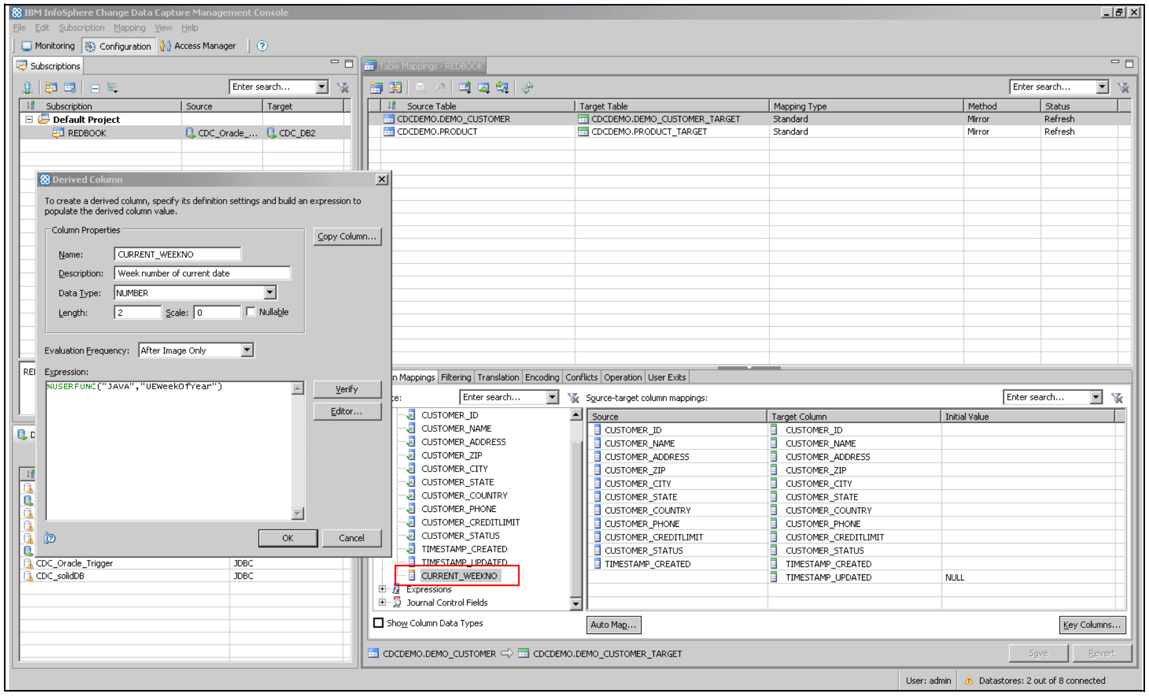
In Example 9-43, the **`addDerivedWeekNo()`** method adds the CURRENT_WEEKNO derived column that calls a Java user function to a subscribed table (selected table). You can find the source for the UEWeekOfYear user exit program in “Java user exit programs” on page 371.

Example 9-43 Add a derived column

```
/**
 * Adds a derived column to the mapped table which will calculate the week
 * number of the current date
 *
 * @param subscribedTable
 *            - The table that is selected to the subscription
 * @throws ApiException
 */
public void addDerivedWeekNo(SubscribedTable subscribedTable)
      throws ApiException {
   DefaultContext derivedColumnContext = new DefaultContext();
   derivedColumnContext.setString(DerivedColumn.Name, "CURRENT_WEEKNO");
   derivedColumnContext.setString(DerivedColumn.BasedOnColumn, "");
   derivedColumnContext.setInt(DerivedColumn.ColumnLength, 2);
   derivedColumnContext.setInt(DerivedColumn.ColumnPrecision, 2);
   derivedColumnContext.setInt(DerivedColumn.ColumnScale, 0);
   derivedColumnContext.setString(DerivedColumn.DataTypeName, "NUMBER");
   derivedColumnContext.setString(DerivedColumn.Description,
         "Week number of current date");
   derivedColumnContext.setString(DerivedColumn.EvaluationFrequency,
         DerivedColumn.EF_AFTER);
// *AFT
// Evalutation
// frequency
   derivedColumnContext.setByte(DerivedColumn.Nullable, (byte) 0);
// False
   derivedColumnContext.setString(DerivedColumn.Expression,
        "%USERFUNC(\"JAVA\",\"UEWeekOfYear\")");
   subscribedTable.addDerivedColumn(derivedColumnContext);
   System.out.println("Derived column "
        + derivedColumnContext.getString(DerivedColumn.Name)
        + " added to table " + subscribedTable.getFullName() + ".");
}
```

As with other classes in the InfoSphere CDC API, DerivedColumn works with a context object to specify the attributes of the derived column. All attributes are mandatory; leaving out any one of them causes the addDerivedColumn to fail.

In Figure 9-20, the CURRENT_WEEKNO column has been added to the list of columns at the source, as a derived column. This example does not map the derived column to a target column; the column mapping for derived columns is the same as regular database columns.


IBM InfoSphere Change Data Capture Management Console
File Edit Subscription Mapping View Help
Monitoring Configuration Access Manager
Subscriptions
Enter search...
Table Mappings - REDBOOK
Enter search...

Subscription	Source	Target
Default Project		
REDBOOK	CDC_Oracle_...	CDC_DB2

Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.DEMO_CUSTOMER	CDCDEMO.DEMO_CUSTOMER_TARGET	Standard	Mirror	Refresh
CDCDEMO.PRODUCT	CDCDEMO.PRODUCT_TARGET	Standard	Mirror	Refresh

Derived Column

To create a derived column, specify its definition settings and build an expression to populate the derived column value.

Column Properties

Name: CURRENT_WEEKNO
Description: Week number of current date
Data Type: NUMBER
Length: 2 Scale: 0 ☐ Nullable

Evaluation Frequency: After Image Only

Expression:

```
%USERFUNC("JAVA","UEWeekOfYear")
```

CUSTOMER_ID
CUSTOMER_NAME
CUSTOMER_ADDRESS
CUSTOMER_ZIP
CUSTOMER_CITY
CUSTOMER_STATE
CUSTOMER_COUNTRY
CUSTOMER_PHONE
CUSTOMER_CREDITLIMIT
CUSTOMER_STATUS
TIMESTAMP_CREATED
TIMESTAMP_UPDATED
CURRENT_WEEKNO
Expressions
Journal Control Fields

Source	Target Column	Initial Value
CUSTOMER_ID	CUSTOMER_ID	
CUSTOMER_NAME	CUSTOMER_NAME	
CUSTOMER_ADDRESS	CUSTOMER_ADDRESS	
CUSTOMER_ZIP	CUSTOMER_ZIP	
CUSTOMER_CITY	CUSTOMER_CITY	
CUSTOMER_STATE	CUSTOMER_STATE	
CUSTOMER_COUNTRY	CUSTOMER_COUNTRY	
CUSTOMER_PHONE	CUSTOMER_PHONE	
CUSTOMER_CREDITLIMIT	CUSTOMER_CREDITLIMIT	
CUSTOMER_STATUS	CUSTOMER_STATUS	
TIMESTAMP_CREATED	TIMESTAMP_CREATED	
	TIMESTAMP_UPDATED	NULL

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_TARGET

User: admin   Datastores: 2 out of 8 connected

Figure 9-20 Derived column

9.5.13 Encoding conversions (before and after Version 6.5)

InfoSphere CDC replicates character data between a wide variety of encodings and automatically converts the data from the column encoding detected on the source to the column encoding detected on the target. For example, you can replicate multibyte character data such as Japanese, Chinese, or Korean, or single-byte character data, such as French, Turkish, Arabic, or Russian.

By default, InfoSphere CDC assumes that the data stored in a character capable column is using the encoding associated with that column type. For example, if your database is set to use Shift-JIS (Japanese), then data stored in the CHAR and VARCHAR columns is assumed to be in Shift-JIS by default. However, InfoSphere CDC is only concerned with the encoding of the data, not the encoding of the column storage type. This flexibility allows the product to deal with situations where the actual contents of the column do not match the encoding specified for the column in the database, commonly seen in many applications. Overriding the detected column encoding allows you to specify the actual encoding of the data as known by you.

In all cases, if the database column encoding defined for source and target matches the contents of the columns, InfoSphere CDC picks up this configuration automatically and there is no need to override it in the InfoSphere CDC configuration.

All InfoSphere CDC engines before InfoSphere CDC V6.5 perform encoding conversions on the source side. Starting from Version 6.5, you can transfer the work of encoding conversions to the target side to reduce processor resource utilization on the source server.

The difference in execution of the encoding conversion is reflected in the metadata and the Java API. Different methods are available to set encoding conversion depending on the version of the InfoSphere CDC engine.

If either the source or target InfoSphere CDC engine is version 6.3 or earlier, encoding settings are configured on the source, for the SubscribedTables columns. Both source and target encoding is configured in the source metadata using the **`setColumnUnicodeHandling()`** and **`setEncoding()`** methods, which are part of the SubscribedTable class, and conversion takes place at the source. If both source and target engines are at Version 6.5 or later, the source encoding is configured at the source and the target encoding is configured at the target. Source and target encodings must then be set using the **`setMBCSColumnEncoding()`** method, for the SubscribedTable class (source) and the ColumnAssignment class (target).

Define a method that retrieves the capability of the subscription to do multi-byte character set (MBCS) automapping and use this capability to determine the encoding conversion type that must be configured. Whether the subscription can automap is defined when the subscription is created; at that time, the source and target data store properties are analyzed to set the SubscriptionMBCSState property. The sample code for this action is shown in Example 9-44.

Example 9-44 Determine if a subscription has MBCS automapping enabled

```
/**
 * Determines if the source and target datastores both support MBCS
 * automatic mapping
 *
 * @param subscription
 *    - The subscription for which MBCS automatic mapping must be
 *            determined
 * @return
 */
private boolean isSubscriptionMbcsAutoMappingEnabled(
      Subscription subscription) {
   boolean enabled = false;
   try {
      Context properties = subscription.getProperty();
      byte state =
properties.getByte(Subscription.SubscriptionMBCSState);
      // 0 - unknown, 1 - 6.3 level (disabled), 2 - 6.5 level (enabled)
      if (state == 2)
         enabled = true;
   } catch (Exception e) {
// ignore the error, consider it disabled anyway
   }
   return enabled;
}
```

When using InfoSphere CDC V6.5 for both the source and target engines, MBCS automapping is supported and you can configure encoding overrides on source and target columns:

- MBCS Encoding type: You can specify whether encoding conversion should be done. For some data types, such as numbers or dates, encoding conversion is not applicable, so be aware of this limitation when specifying the encoding types. When encoding is possible, the default encoding that was picked up from the database during the table mapping can be overridden by selecting either COLUMN_MBCS_ENCODING_USE_AS_IS (no conversion, pass as binary) or COLUMN_MBCS_ENCODING_USE_SPECIFIED. In the latter case, the data encoding must be specified for the column in question.
- Data encoding: The Internet Assigned Numbers Authority (IANA) encoding name is the identification used by the InfoSphere CDC V6.5 or later engines. For example, the IANA name for Western European character sets is windows-1252. If you specify an IANA encoding name that matches the value that was picked up by InfoSphere CDC from the source database, the data encoding is not stored as an overridden encoding conversion.

The **`setSourceColumnEncoding65()`** method sets the encoding of a source column to an IANA name, as shown in Example 9-45.

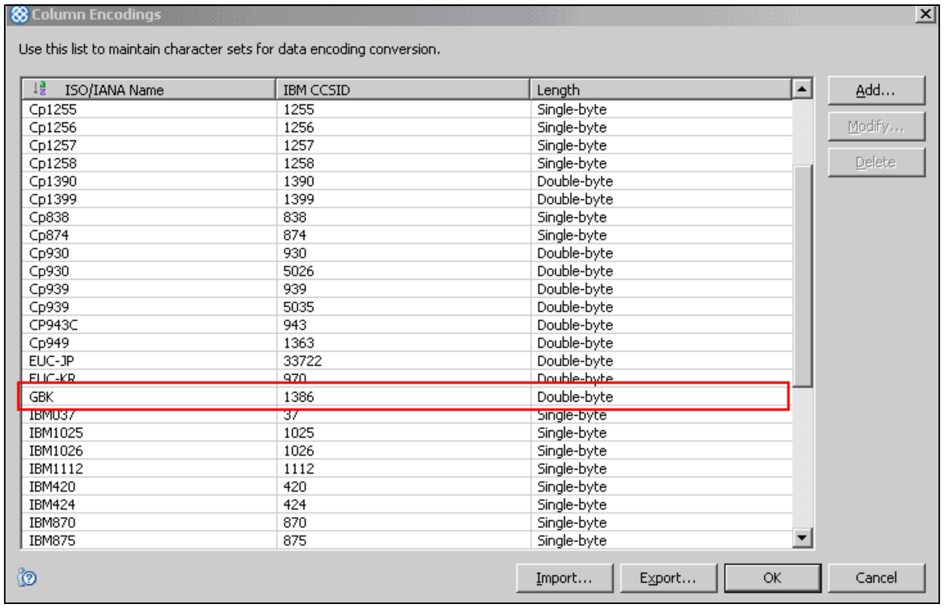
Example 9-45 Set column encoding for InfoSphere CDC V6.5

```
/**
 * Set the encoding for a source column in a SubscribedTable object, CDC
 * 6.5+
 *
 * @param subscription
 *     - Subscription
 * @param subscribedTable
 *     - The table that is selected to the subscription
 * @param columnName
 *     - The column of the selected table for which the source * encoding must be
          set
 * @param encodingName
 *            - IANA encoding name of the encoding
 * @throws ApiException
 */
private void setSourceColumnEncoding65(Subscription subscription,
      SubscribedTable subscribedTable, String columnName,
      String encodingName) throws ApiException {
   subscribedTable.setMBCSColumnEncoding(columnName,
         (byte) MBCSColumnEncoding.COLUMN_MBCS_ENCODING_USE_SPECIFIED,
         encodingName);
   System.out.println("Encoding set to " + encodingName + " for column "
         + columnName + " in table " + subscribedTable.getFullName()
         + ".");
}
```

The attributes of InfoSphere CDC V6.3 and earlier encoding conversions are:

- Unicode handling: Whether Unicode handling is done for the column in question. InfoSphere CDC tries to convert the source columns as though they contain Unicode data (UCS-2, UTF-8, UTF-16, or UTF-32).
- Encoding method: Specifies whether encoding conversion must be done, and if so, whether the database default, overridden conversion, or Unicode conversion must be done.
- CCSID for column: The Coded Character Set ID for the column. When specified for the source column, InfoSphere CDC expects the data in the source column to be encoded with this CCSID. When specified for the target column, InfoSphere CDC converts the data to this CCSID.
- Encoding name: The ISO or IANA encoding name. For example, the IANA name for Western European character sets is windows-1252. This encoding name is used by the InfoSphere CDC V6.5 and later engines.
- Encoding length: Specifies whether the characters must be processed (source) or generated (target) as single bytes (SBCS), double bytes (DBCS), or multiple bytes (MBCS).

The CCSID, encoding name, and encoding length depend on each other. You must specify all three values to ensure that the InfoSphere CDC source engine runs the correct encoding based on the target engine. For a map of IANA/CCSID and encoding length, go to the Management Console preferences (click **Edit**) and click the **Encoding** button. Figure 9-21 shows the IANA names together with the CCSID and encoding length for your reference.



Use this list to maintain character sets for data encoding conversion.

ISO/IANA Name	IBM CCSID	Length
Cp1255	1255	Single-byte
Cp1256	1256	Single-byte
Cp1257	1257	Single-byte
Cp1258	1258	Single-byte
Cp1390	1390	Double-byte
Cp1399	1399	Double-byte
Cp838	838	Single-byte
Cp874	874	Single-byte
Cp930	930	Double-byte
Cp930	5026	Double-byte
Cp939	939	Double-byte
Cp939	5035	Double-byte
CP943C	943	Double-byte
Cp949	1363	Double-byte
EUC-JP	33722	Double-byte
EUC-KR	970	Double-byte
GBK	1386	Double-byte
IBM037	37	Single-byte
IBM1025	1025	Single-byte
IBM1026	1026	Single-byte
IBM1112	1112	Single-byte
IBM420	420	Single-byte
IBM424	424	Single-byte
IBM870	870	Single-byte
IBM875	875	Single-byte

Figure 9-21 IANA information

The **`setSourceColumnEncoding63()`** method, in Example 9-46, shows how to set the encoding for a column if either the source or target InfoSphere CDC engine has a version earlier than Version 6.5. This situation means that the encoding must be specified entirely at the source, that is, both the source and target column encodings.

Example 9-46 Set column encoding for InfoSphere CDC V6.3

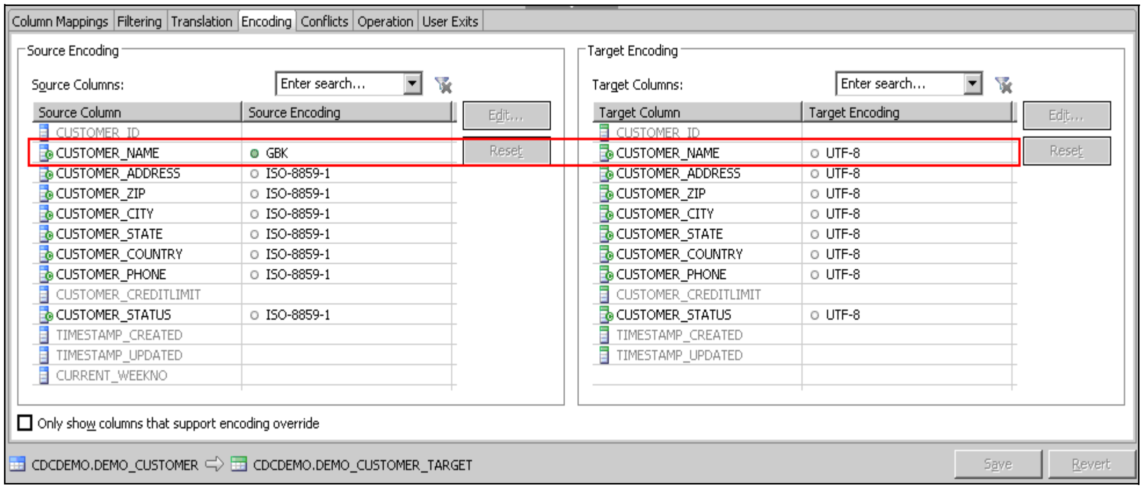
```
/**
 * Set the encoding for a source column in a SubscribedTable object, CDC 6.3
 * and earlier releases
 *
 * @param subscription
 *       - Subscription
 * @param subscribedTable
 *       - The table that is selected to the subscription
 * @param columnName
 *       - The column of the selected table for which the source encoding must be set
 * @param ccsid
 *       - Coded Character Set ID of the encoding
 * @param encodingName
 *       - IANA encoding name of the encoding
 * @param encodingLength
 *       - Single, double or multiple byte
 * @throws ApiException
 */
private void setSourceColumnEncoding63(Subscription subscription,
      SubscribedTable subscribedTable, String columnName, String ccsid,
      String encodingName, int encodingLength) throws ApiException {
   int sourceEncodingMethod = subscribedTable
         .getSourceEncodingMethod(columnName);
   String sourceEncodingCcsid = subscribedTable
         .getSourceEncodingCcsid(columnName);
   String sourceEncodingName = subscribedTable
         .getSourceEncodingName(columnName);
   int sourceEncodingLength = subscribedTable
         .getSourceEncodingLength(columnName);
   int targetEncodingMethod = subscribedTable
         .getTargetEncodingMethod(columnName);
   String targetEncodingCcsid = subscribedTable
         .getTargetEncodingCcsid(columnName);
   String targetEncodingName = subscribedTable
         .getTargetEncodingName(columnName);
   int targetEncodingLength = subscribedTable
         .getTargetEncodingLength(columnName);
   subscribedTable.setEncoding(columnName,
         SubscribedTable.COLUMN_ENCODING_USE_SPECIFIED, ccsid,
         encodingName, encodingLength, targetEncodingMethod,
         targetEncodingCcsid, targetEncodingName, targetEncodingLength);
   System.out.println("Encoding set to CCSID " + ccsid + " and IANA name "
         + encodingName + " for column " + columnName + " in table "
         + subscribedTable.getFullName() + ".");
}
```

The method shown in Example 9-47 retrieves the column encoding that has been set during the mapping process and then overrides it with the CCSID, IANA encoding name, and encoding length that were passed as a parameter. This method is, as an example, how to set the source column encoding to GBK (a Chinese national standard extension) for both InfoSphere CDC V6.5 and InfoSphere CDC V6.3.

Example 9-47   Implementation of setting the source column encoding

```
// Set encoding for CUSTOMER_NAME column
if (isSubscriptionMbcsAutoMappingEnabled(subscription))
   setSourceColumnEncoding65(subscription, subscribedTables[0],
         "CUSTOMER_NAME", "GBK");
else
   setSourceColumnEncoding63(subscription, subscribedTables[0],
         "CUSTOMER_NAME", "1384", "GBK",
         SubscribedTable.COLUMN_ENCODING_DOUBLE_BYTE);
```

Running the example code results in the configuration shown in Figure 9-22. The source encoding for CUSTOMER_NAME has been overridden to GBK while the target encoding has been left unchanged.



Source Column	Source Encoding
CUSTOMER_ID	
CUSTOMER_NAME	● GBK
CUSTOMER_ADDRESS	○ ISO-8859-1
CUSTOMER_ZIP	○ ISO-8859-1
CUSTOMER_CITY	○ ISO-8859-1
CUSTOMER_STATE	○ ISO-8859-1
CUSTOMER_COUNTRY	○ ISO-8859-1
CUSTOMER_PHONE	○ ISO-8859-1
CUSTOMER_CREDITLIMIT	
CUSTOMER_STATUS	○ ISO-8859-1
TIMESTAMP_CREATED	
TIMESTAMP_UPDATED	
CURRENT_WEEKNO	

Target Column	Target Encoding
CUSTOMER_ID	
CUSTOMER_NAME	○ UTF-8
CUSTOMER_ADDRESS	○ UTF-8
CUSTOMER_ZIP	○ UTF-8
CUSTOMER_CITY	○ UTF-8
CUSTOMER_STATE	○ UTF-8
CUSTOMER_COUNTRY	○ UTF-8
CUSTOMER_PHONE	○ UTF-8
CUSTOMER_CREDITLIMIT	
CUSTOMER_STATUS	○ UTF-8
TIMESTAMP_CREATED	
TIMESTAMP_UPDATED	

☐ Only show columns that support encoding override

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_TARGET

Figure 9-22   Management Console table mapping encoding

9.5.14 Operations and user exits

InfoSphere CDC allows you to suppress the standard operations and include custom code in the processing of table-level and row-level operations. For example, you might not want any table clear operations or row delete operations to be applied onto the target side. Typically, this situation is the case when you target a data warehouse or operational data store. Another example is that you might want to process user-defined actions instead of a standard operation, such as applying a soft delete instead of the row delete that InfoSphere CDC would normally run with a user exit.

In addition to the row-level user exits, the InfoSphere CDC Linux, UNIX, and Windows engine now supports subscription-level user exits, which let you define a set of actions that InfoSphere CDC can run before or after a commit event occurs on a specified subscription. Subscription-level user exits can work alone or in tandem with row-level user exits, for example, to complete and send an XML message that has been built using the row-level user exits.

The Management Console has split the standard operations and row-level user exits across two tabs. In the API, these items are controlled through the same class, UserExit.

As an example, you can configure a table mapping for soft delete. Basically, this situation means that you map a table with an adaptive apply mapping type, and then disable the delete operation and specify a pre-delete user exit that calls the UESoftDelete Java class. There is a `SoftDelete.class` file in the target InfoSphere CDC engine's `lib` directory, so it is available (this setting is validated by InfoSphere CDC when trying to configure), as shown in Example 9-48.

Example 9-48 Table mapping for soft delete

```
/**
 * Assigns the source table to a target table, configuring it for soft
 * delete.
 *
 * @param subscription
 *            - Subscription for which the table must be assigned
 * @param targetDataStore
 *            - Target datastore of the subscription
 * @param subscribedTable
 *            - Subscribed (selected) table
 * @param targetTablePath
 *            - Schema (path) of the target table
 * @param targetTableName
 *            - Name of the target table
 * @param mappingType
 *            - The mapping type that must be used
 * @throws ApiException
 */
public void assignSoftDelete(Subscription subscription,
      Subscriber targetDataStore, SubscribedTable subscribedTable,
      String targetTablePath, String targetTableName) throws ApiException {
// Get the publication for the subscription
   Publication publication = getSubscriptionPublication(subscription,
         targetDataStore);
// Get the PublishedTable associated with the SubscribedTable object
   DBPath publishedTableDBPath = targetDataStore
         .createDBPath(subscribedTable.getUpperPath().getFullName()
               .toUpperCase());
   String publishedTableName = subscribedTable.getName().toUpperCase();
   PublishedTable publishedTable = publication.getPublishedTable(
         publishedTableDBPath, publishedTableName);
// Map the table to the target table using specified mapping type
   String targetDatabase = "";
   String destinedMember = null;
   String indexLibrary = null;
   String indexName = null;
   publishedTable.assign(targetTableName, targetTablePath, targetDatabase,
         destinedMember, indexLibrary, indexName,
         PublishedTable.ADAPTIVE_APPLY);
   System.out.println("Table " + subscribedTable.getFullName()
         + " assigned to " + targetTablePath + "." + targetTableName
         + " using soft delete mapping type.");
   TableAssignment tableAssignment = publishedTable.getTableAssignment();
// Loop through all target columns and map values depending on name
   String[] destinedColumnNames = tableAssignment.getDestinedColumnNames();
   for (String targetColumnName : destinedColumnNames) {
      ColumnAssignment columnAssignment = tableAssignment
            .getColumnAssignment(targetColumnName);
      if (targetColumnName.equals("AUD_TIMESTAMP")) {
         columnAssignment.mapTo(ColumnAssignment.MAP_JOURNAL_CONTROL,
               "&TIMSTAMP");
         System.out
               .println("Journal control column &TIMSTAMP assigned to column "
                     + targetColumnName + ".");
      } else if ((targetColumnName.equals("APPLY_TIMESTAMP"))) {
         columnAssignment.mapTo(ColumnAssignment.MAP_CURRENT_DATE);
         System.out
               .println("Default value CURRENT DATE assigned to column "
                     + targetColumnName + ".");
      } else if ((targetColumnName.equals("LAST_OPERATION"))) {
         columnAssignment.mapTo(ColumnAssignment.MAP_JOURNAL_CONTROL,
               "&ENTTYP");
         System.out
               .println("Journal control column &ENTTYP assigned to column "
                     + targetColumnName + ".");
      }
   }
// Set table level operations and user exits
   UserExit tableUserExit = tableAssignment
         .createNewUserExit(UserExit.JAVA_CLASS);
   tableUserExit.setStdOperation(UserExit.STANDARD_INSERT,
         UserExit.INSERT_UPDATE);
   tableUserExit.setStdOperation(UserExit.STANDARD_UPDATE,
         UserExit.INSERT_UPDATE);
   tableUserExit.setStdOperation(UserExit.STANDARD_DELETE,
         UserExit.DISABLE);
   System.out
         .println("Standard delete operation has been disabled for operations on 
table "
               + targetTablePath + "." + targetTableName);
   tableUserExit.setFunctionType(UserExit.JAVA_CLASS);
   tableUserExit.setJavaClass("UESoftDelete");
   tableUserExit.addOperation(UserExit.BEFORE_DELETE, "Y");
   System.out
         .println("Before delete operation has been set to UESoftDelete Java class 
for table "
               + targetTablePath + "." + targetTableName);
   // Set subscription-level user exit
   tableAssignment.setUserExit(tableUserExit);
   System.out.println("Subscription-level user exit for subscription "
         + subscription.getName() + " has been set to UESoftDelete");
}
```

The initial part of the table assignment is the same as before, with just a small difference in the mapping type (PublishedTable.ADAPTIVE_APPLY instead of PublishedTable.STANDARD). Iterate through the columns to map the &ENTTYP (operation type) to the LAST_OPERATION column. If the row has been soft-deleted, this column contains "DL".

During the second part of the method, a new UserExit object is created and the standard operations are assigned. The standard delete (STANDARD_DELETE) operation is disabled. Then the user exit type is set to use Java class UESoftDelete and it is marked to be run before the delete operation occurs. InfoSphere CDC still runs the before and after user exits, even if the standard operation is disabled.

The UESoftDelete class is described in "Java user exit for row-level user exits" on page 383.

For subscription-level user exits, there is not much you must configure. After you have created the subscription, and the Publication object exists in the target metadata, you can get the subscription-level user exit configuration by running the **`getSubscriptionUserExits()`** method from the Publication object. The function type that is currently supported is UserExit.JAVA_CLASS, and you can set the Java class and parameters by using the methods in the SubscriptionUserExits class.

The subscription-level user exit must implement the SubscriptionUserExitIF interface. The UESoftDelete class implements both UserExitIF and SubscriptionUserExitIF, so one class can be used both at the table-level and subscription-level.

9.5.15 Common procedures (updating table definitions)

After you change the structure of a mapped source or target table in your database, you must update the definition of the table in the InfoSphere CDC metadata so that subscriptions are aware of the new structure and can adjust the log reader or apply process.

If you change the definition of a source table (add a column, or change column length or data type) in your database, then you must update the definition of the table in the source data store catalog. InfoSphere CDC requires you to update the source table so that the new structure is available for configuration when editing your table mapping details. For example, if you have added a column on the source table, then you might want to map this new column to a target column. In the Management Console, you can update the source table definition by choosing the table and updating the source definition. When you need to update the source table definition through the API, the Catalog class provides the **`reAddTable()`** method. Both actions retrieve the table structure from the database and update the PublishableTable object in the source data store's catalog.

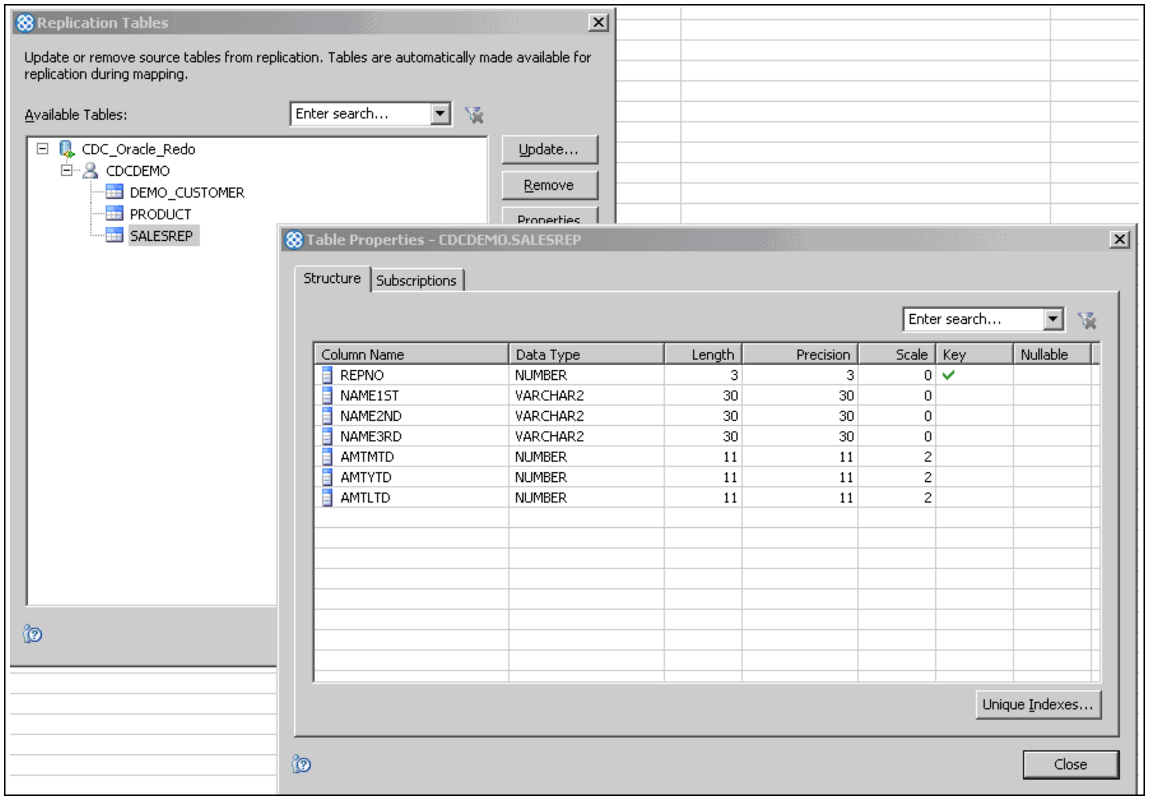
When the definition of a target table is altered (add a column, or set column constraints or different primary key constraint) in your database, you need to update the definition of the target table's metadata. This action ensures that the column mappings can be updated according to the new target table structure. For example, if you have added a column on the target table, then you might want to map a source column to the new target column or populate it with a constant value. The Management Console provides a table menu option to update the target table definition. If you need to update the target table definition through the API, run the **`reassign()`** method from the PublishedTable object that links the source table to the destination (target) table.

Typically, you perform the updates of the InfoSphere CDC source and target tables only if the structure has changed.

Example 9-49 shows whether the CDCDEMO.SALESREP catalog table has changed; if it has, update the table definition in the catalog. Before running the snippet, check the structure of the table registered in the source data store catalog. The table structure before applying the DDL changes is shown in Figure 9-23 on page 329.

Example 9-49 Checking if the table structure has changed

```
/**
 * Check if a table in the source datastore catalog has changed. Returns
 * true if there is a difference between: - Number of columns in database
 * table and catalog table - Data type of one of the columns in the database
 * table and catalog table - Length of one of the columns in the database
 * table and catalog table - Precision of one of the columns in the database
 * table and catalog table - Scale of one of the columns in the database
 * table and catalog table
 *
 * @param sourceDataStore
 *            - The source datastore that has the catalog in which the table
 *            must be checked
 * @param tablePath
 *            - Schema of the table to be checked
 * @param tableName
 *            - Name of the table to be checked
 * @return – true if the table has changed, false if there is no difference
 *         between database table and catalog table
 * @throws ApiException
 */
public boolean isCatalogTableChanged(Publisher sourceDataStore,
      String tablePath, String tableName) throws ApiException {
   boolean isChanged = false;
   DBTable databaseTable = sourceDataStore.getTable(
         sourceDataStore.getDBPath(tablePath), tableName);
   Catalog sourceCatalog = sourceDataStore.getCatalog();
   PublishableTable publishableTable = sourceCatalog.getPublishableTable(
         tablePath, tableName);
   int numberDatabaseColumns = databaseTable.getColumnNames().length;
   int numberCatalogColumns = publishableTable.getColumnNames().length;
   if (numberDatabaseColumns != numberCatalogColumns) {
      System.out
            .println("Number of columns in database does not correspond to number of 
columns in catalog for table "
                  + tablePath + "." + tableName + ".");
      isChanged = true;
   } else {
      for (String databaseColumnName : databaseTable.getColumnNames()) {
         DBColumn databaseColumn = databaseTable
               .getColumn(databaseColumnName);
         try {
            DBColumn catalogColumn = publishableTable
                  .getColumn(databaseColumnName);
            if (!catalogColumn.getDataType().equalsIgnoreCase(
                  databaseColumn.getDataType())) {
               System.out.println("Database column "
                     + databaseColumnName + " of table " + tablePath
                     + "." + tableName
                     + " is of a different data type ("
                     + databaseColumn.getDataType()
                     + ") than catalog column ("
                     + catalogColumn.getDataType() + ").");
               isChanged = true;
            }
            if (catalogColumn.getLength() != databaseColumn.getLength()) {
               System.out.println("Database column "
                     + databaseColumnName + " of table " + tablePath
                     + "." + tableName + " has a different length ("
                     + databaseColumn.getLength()
                     + ") than catalog column ("
                     + catalogColumn.getLength() + ").");
               isChanged = true;
            }
            if (catalogColumn.getPrecision() != databaseColumn
                  .getPrecision()) {
               System.out.println("Database column "
                     + databaseColumnName + " of table " + tablePath
                     + "." + tableName
                     + " has a different precision ("
                     + databaseColumn.getPrecision()
                     + ") than catalog column ("
                     + catalogColumn.getPrecision() + ").");
               isChanged = true;
            }
            if (catalogColumn.getScale() != databaseColumn.getScale()) {
               System.out.println("Database column "
                     + databaseColumnName + " of table " + tablePath
                     + "." + tableName + " has a different scale ("
                     + databaseColumn.getScale()
                     + ") than catalog column ("
                     + catalogColumn.getScale() + ").");
               isChanged = true;
            }
         } catch (ApiException e) {
            System.out.println("Database column " + databaseColumnName
                  + " not found in catalog table " + tablePath + "."
                  + tableName + ".");
            isChanged = true;
         }
      }
   }
   return isChanged;
}
```


Replication Tables

Update or remove source tables from replication. Tables are automatically made available for replication during mapping.

Available Tables:

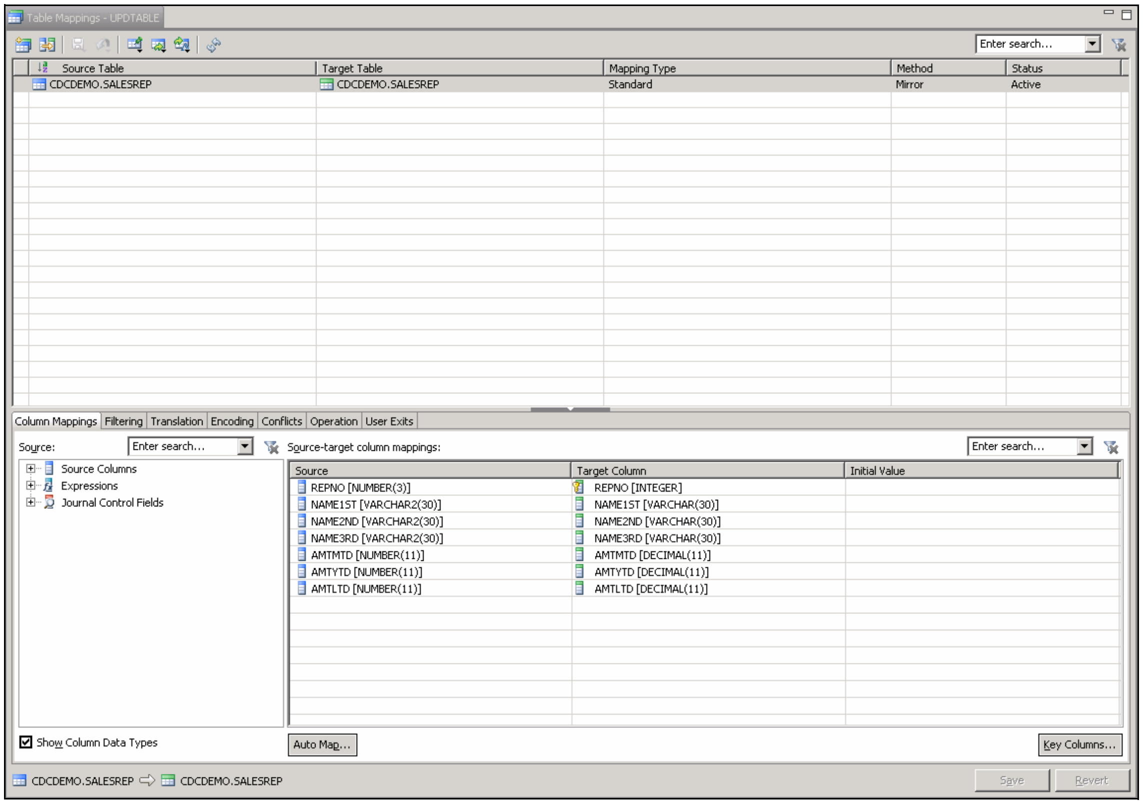
- CDC_Oracle_Redo
  - CDCDEMO
    - DEMO_CUSTOMER
    - PRODUCT
    - SALESREP

Table Properties - CDCDEMO.SALESREP

Column Name	Data Type	Length	Precision	Scale	Key	Nullable
REPNO	NUMBER	3	3	0	✓	
NAME1ST	VARCHAR2	30	30	0		
NAME2ND	VARCHAR2	30	30	0		
NAME3RD	VARCHAR2	30	30	0		
AMTMTD	NUMBER	11	11	2		
AMTYTD	NUMBER	11	11	2		
AMTLTD	NUMBER	11	11	2		

Figure 9-23 Table structure before applying DDL

The table is replicated by UPDTABLE and its status is active (Figure 9-24).



Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.SALESREP	CDCDEMO.SALESREP	Standard	Mirror	Active

Source	Target Column	Initial Value
REPNO [NUMBER(3)]	REPNO [INTEGER]	
NAME1ST [VARCHAR2(30)]	NAME1ST [VARCHAR(30)]	
NAME2ND [VARCHAR2(30)]	NAME2ND [VARCHAR(30)]	
NAME3RD [VARCHAR2(30)]	NAME3RD [VARCHAR(30)]	
AMTMTD [NUMBER(11)]	AMTMTD [DECIMAL(11)]	
AMTYTD [NUMBER(11)]	AMTYTD [DECIMAL(11)]	
AMTLTD [NUMBER(11)]	AMTLTD [DECIMAL(11)]	

Figure 9-24 Replicated table

Now the table is altered. A new column (NEWCOL) is added to the table and the length of an existing column (NAME1ST) is changed from 30 to 50.

The **`isCatalogTableChanged()`** method, shown in Example 9-49 on page 326, retrieves the table structure from the database using the **`getTable()`** method that is part of the Publisher class. It then checks whether the number of columns in the database table corresponds with the number of columns in the catalog table and iterates through the database table columns to compare the attributes of all the database columns with the ones registered in the PublishableTable object. The primary reason for comparing the number of columns is to detect when a column has been dropped from the database table.

The exact code as presented might not work for all InfoSphere CDC engine types. InfoSphere CDC for DB2 on System i, for example, always reports precision 0 for database table columns that are retrieved with the **getTable()** method, while InfoSphere CDC stores the correct precision in its configuration.

If a difference is found between database and catalog table, the table is updated in the catalog using the **reAddTable()** method (Example 9-50) that is part of the Catalog class.

Example 9-50 Method to readd a catalog table

```
/**
 * Update the source table definition in the catalog
 *
 * @param sourceDataStore
 *            - The source datastore that has the catalog in which the table
 *            must be updated
 * @param tablePath
 *            - Schema of the table to be updated
 * @param tableName
 *            - Name of the table to be updated
 * @throws ApiException
 */
public void reAddCatalogTable(Publisher sourceDataStore, String tablePath,
      String tableName) throws ApiException {
   Catalog sourceCatalog = sourceDataStore.getCatalog();
   PublishableTable publishableTable = sourceCatalog.getPublishableTable(
         tablePath, tableName);
   sourceCatalog.reAddTable(publishableTable);
   System.out.println("Updated catalog table definition " + tablePath
         + "." + tableName + ".");
}
```

When these two methods are brought together, the code in Example 9-51 first checks whether the structure of the CDCDEMO.SALESREP has changed; if so, it updates the table definition in the catalog.

Example 9-51 Check to see whether to update the table definition in the catalog

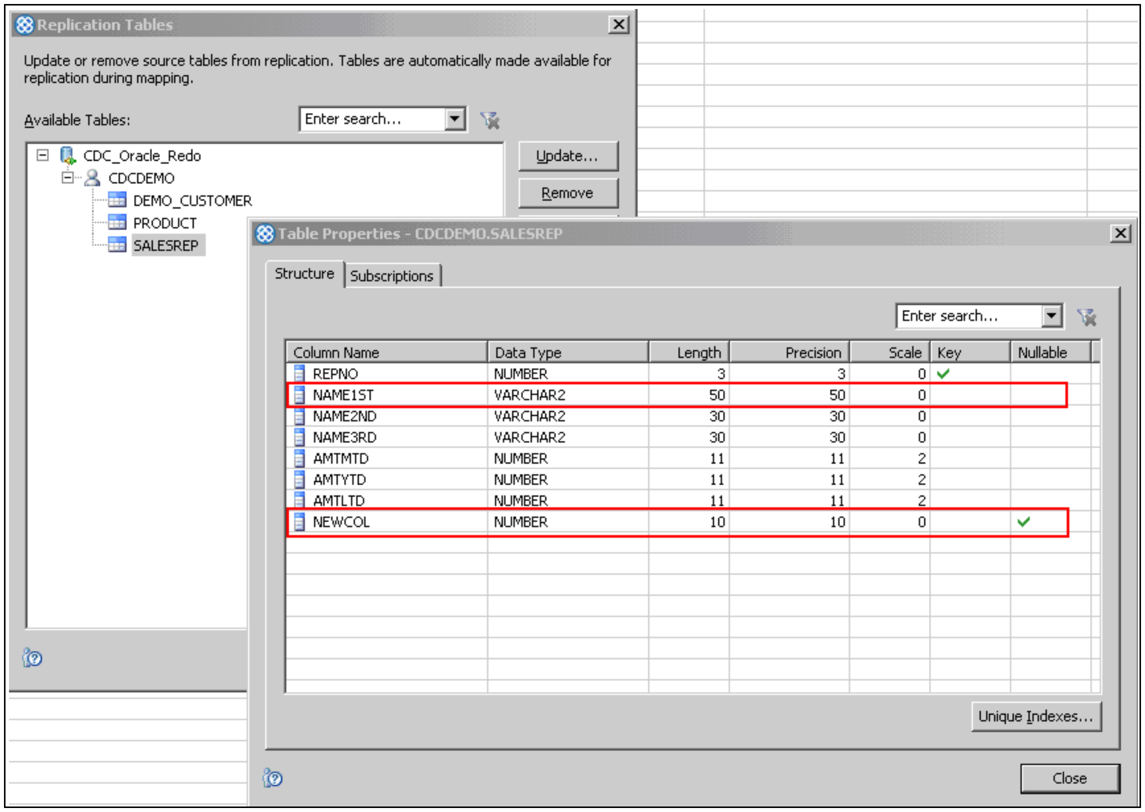
```
if (isCatalogTableChanged(oracleDataStore, "CDCDEMO", "SALESREP")) {
   reAddCatalogTable(oracleDataStore, "CDCDEMO", "SALESREP");
```

The output of the code when running it after the table's structure has changed is shown in Example 9-52.

Example 9-52 Output from Example 9-51 on page 331

```
Number of columns in database does not correspond to number of columns
in catalog for table CDCDEMO.SALESREP.
Updating source table definition CDCDEMO.SALESREP affects subscription
UPDTABLE.
Updated catalog table definition CDCDEMO.SALESREP.
```

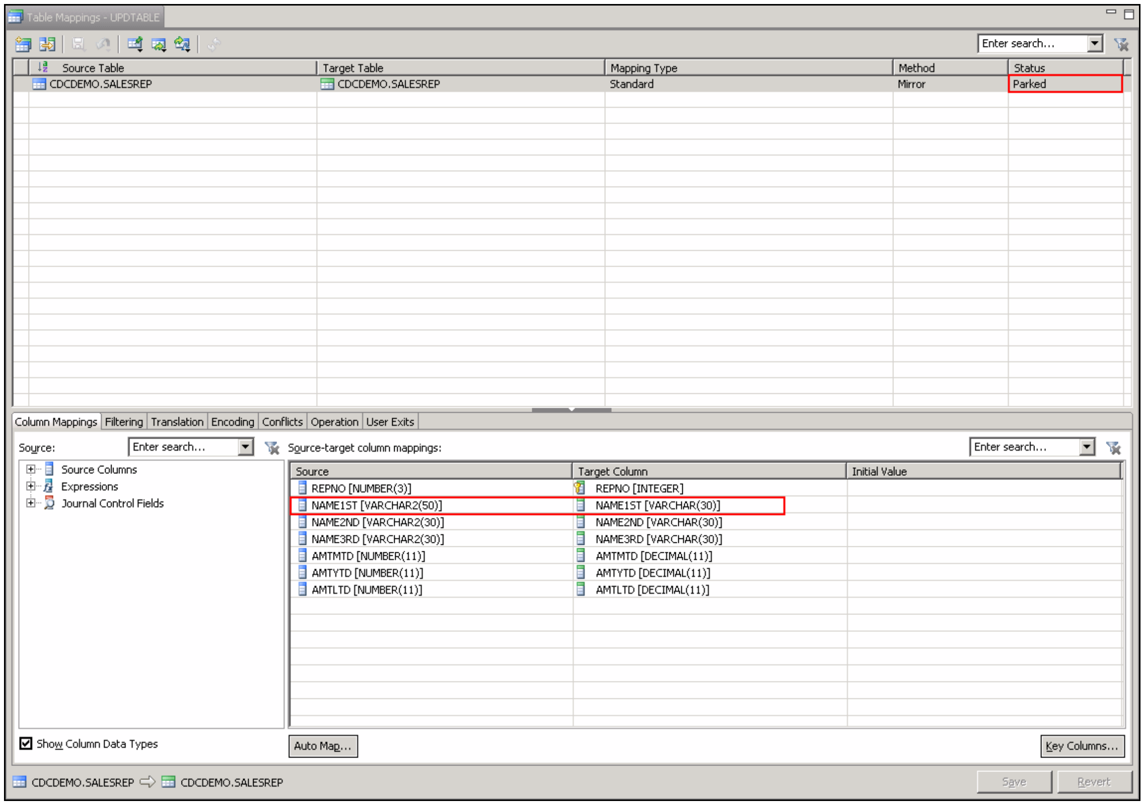
Look at the Management Console (Figure 9-25) and see that the table's structure has been updated.



Column Name	Data Type	Length	Precision	Scale	Key	Nullable
REPNO	NUMBER	3	3	0	✓	
NAME1ST	VARCHAR2	50	50	0		
NAME2ND	VARCHAR2	30	30	0		
NAME3RD	VARCHAR2	30	30	0		
AMTMTD	NUMBER	11	11	2		
AMTYTD	NUMBER	11	11	2		
AMTLTD	NUMBER	11	11	2		
NEWCOL	NUMBER	10	10	0		✓

Figure 9-25 Table structure is updated

The status of the replicated table has been set to parked (Figure 9-26).



Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.SALESREP	CDCDEMO.SALESREP	Standard	Mirror	Parked

Source	Target Column	Initial Value
REPNO [NUMBER(3)]	REPNO [INTEGER]	
NAME1ST [VARCHAR2(50)]	NAME1ST [VARCHAR(30)]	
NAME2ND [VARCHAR2(30)]	NAME2ND [VARCHAR(30)]	
NAME3RD [VARCHAR2(30)]	NAME3RD [VARCHAR(30)]	
AMTMTD [NUMBER(11)]	AMTMTD [DECIMAL(11)]	
AMTYTD [NUMBER(11)]	AMTYTD [DECIMAL(11)]	
AMTLTD [NUMBER(11)]	AMTLTD [DECIMAL(11)]	

Figure 9-26 Replicated table status set to parked

The work to make the subscription runnable is not yet complete. The source table structure has changed and the target side is not aware yet that it will receive data different from before. The PublishedTable must be updated with the new source table structure by describing the subscription.

To discover which subscriptions are affected by the update of the source table definition, that is, getting a list of subscriptions that replicate the table, you must analyze the subscriptions individually (Example 9-53). There is no single method that retrieves the subscriptions that depend on a table.

Example 9-53 Getting the list of subscriptions that replicate a certain source table

```
/**
 * Get the list of subscriptions which replicate a certain source table.
 *
 * @param sourceDataStore
 *            - The source datastore that has the table it its catalog
 * @param tablePath
 *            - Schema of the table
 * @param tableName
 *            - Name of the table
 * @throws ApiException
 */
public ArrayList<Subscription> getSourceTableSubscriptions(
      Publisher sourceDataStore, String tablePath, String tableName)
      throws ApiException {
   Catalog sourceCatalog = sourceDataStore.getCatalog();
   PublishableTable publishableTable = sourceCatalog.getPublishableTable(
         tablePath, tableName);
   ArrayList<Subscription> sourceTableSubscriptions = new ArrayList<Subscription>();
   for (String subscriptionName : sourceDataStore.getSubscriptionNames()) {
      Subscription subscription = sourceDataStore
            .getSubscription(subscriptionName);
      try {
         SubscribedTable subscribedTable = subscription
               .getSubscribedTable(publishableTable.getUpperPath(),
                     publishableTable.getName());
         if (subscribedTable != null) {
            System.out.println("Subscription " + subscription.getName()
                  + " replicates table "
                  + subscribedTable.getFullName() + ".");
            sourceTableSubscriptions.add(subscription);
         }
      } catch (ApiException e) {
         // Do nothing, subscription does not map the table
      }
   }
   return sourceTableSubscriptions;
}
```

If you want to make the process complete, that is, making the subscription runnable again and replicating the changes of the table that has been updated in the catalog, the snippet would have to be extended (Example 9-54).

Example 9-54 Making the subscription runnable and replicating table changes

```
if (isCatalogTableChanged(oracleDataStore, "CDCDEMO", "SALESREP")) {
   // Get the subscriptions which replicate this table
   ArrayList<Subscription> sourceTableSubscriptions = getSourceTableSubscriptions(
         oracleDataStore, "CDCDEMO", "CDCDEMO");
   // Update the source table definition in the catalog
   reAddCatalogTable(oracleDataStore, "CDCDEMO", "SALESREP");
   // Change replication status in all subscriptions to Active and
   // describe
   for (Subscription sourceTableSubscription : sourceTableSubscriptions) {
      // Change table replication status to Active (only if Mirror
      // method)
      SubscribedTable changedSubscribedTable = sourceTableSubscription
            .getSubscribedTable(
                  oracleDataStore.getDBPath("CDCDEMO"),
                  "SALESREP");
      if (changedSubscribedTable.getReplicationMethod() ==
SubscribedTable.METHOD_MIRROR) {
         System.out.println("Replication status of table "
               + changedSubscribedTable.getFullName()
               + " is changed to Active.");
         changedSubscribedTable
               .setReplicationStatus(SubscribedTable.STATUS_ACTIVE);
      }
      // Now describe the subscription
      describeSubscription(sourceTableSubscription, db2DataStore);
   }

}
```

When having to update the table definitions for target tables, the procedure is different. The procedure for retrieval of the database table structure is equivalent to the one done for the source, using the **`getTable()`** method. However, you must now trace back to the published table starting from the destined table. Again, there is no simple method in the InfoSphere CDC Java API that allows direct access to the source table starting from the destination table (Example 9-55).

Example 9-55 Updating the target table definition for a subscription

```
/**
 * Update the target table definition for a subscription
 *
 * @param subscription
 *            - The subscription that targets the table to be
reassigned
 * @param targetDataStore
 *            - The datastore that is targeted by the subscription
 * @param targetTablePath
 *            - Schema of the target table to be updated
 * @param targetTableName
 *            - Name of the target table to be updated
 * @throws ApiException
 */
public void reAssignTable(Subscription subscription,
      Subscriber targetDataStore, String targetTablePath,
      String targetTableName) throws ApiException {
   // Get publication for subscription
   Publication publication = getSubscriptionPublication(subscription,
         targetDataStore);
   // Process all database paths (schemas for publication)
   for (DBPath publicationDBPath : publication.getDBPaths()) {
      // Process all tables for database path
      for (PublishedTable publishedTable : publication
            .getPublishedTables(publicationDBPath)) {
         TableAssignment tableAssignment = publishedTable
               .getTableAssignment();
         DBTable destinedTable = tableAssignment.getDestinedTable();
         DBPath destinedTableUniqueIndex = tableAssignment
               .getDestinedTableUniqueIndex();
         // If destined table is table to reassign, do so
         if (destinedTable.getUpperPath().getFullName()
               .equalsIgnoreCase(targetTablePath)
               && destinedTable.getName().equalsIgnoreCase(
                     targetTableName)) {
            publishedTable.reassign(destinedTableUniqueIndex);
            System.out.println("Updated target table definition "
                  + targetTablePath + "." + targetTableName + ".");
         }
      }
   }
}
```

Check that the subscription is not active when reassigning tables, or the underlying metadata changes while the running subscription might not be aware of this change. Subscription statuses can only be obtained from the source data store. You should trace even further back until you have found the subscription that is connected to the Publication that holds the PublishedTable that is assigned to the target table for which the table structure has changed.

In most cases, you are aware of the table changes that are being deployed and which subscriptions they affect. Therefore, you could include the reassignment of the target table in the deployment of the subscription. When using the **`reAssignTable()`** method, assume that you are deploying a target table change for a known subscription and that the subscription is inactive.

Add a column to the CDCDEMO.SALESREP_TARGET table and run the code snippet shown in Example 9-56 to update the definition of the target table.

Example 9-56 Snippet to update the target table definition

```
// Get the subscription object
Subscription subscription=oracleDataStore.getSubscription("UPDTABLE");
reAssignTable(subscription, db2DataStore, "CDCDEMO",
"SALESREP_TARGET");
```

When refreshing the view in the Management Console, the new column appears with a default mapping (Figure 9-27). If there is a source column with the same name and equivalent data type, the source column is automatically mapped to the new target column.

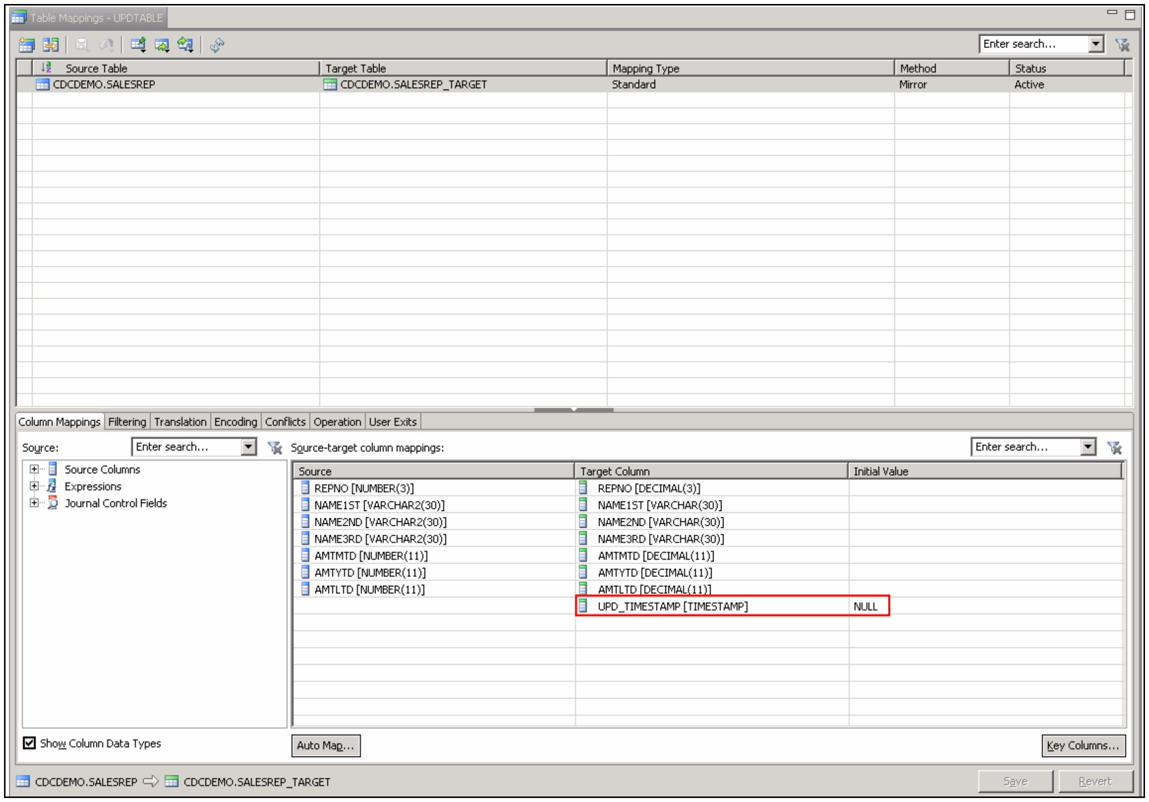


Table Mappings - UPDTABLE

Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.SALESREP	CDCDEMO.SALESREP_TARGET	Standard	Mirror	Active

Column Mappings | Filtering | Translation | Encoding | Conflicts | Operation | User Exits

Source: Enter search...

Source Columns
Expressions
Journal Control Fields

Source-target column mappings: Enter search...

Source	Target Column	Initial Value
REPNO [NUMBER(3)]	REPNO [DECIMAL(3)]	
NAME1ST [VARCHAR2(30)]	NAME1ST [VARCHAR(30)]	
NAME2ND [VARCHAR2(30)]	NAME2ND [VARCHAR(30)]	
NAME3RD [VARCHAR2(30)]	NAME3RD [VARCHAR(30)]	
AMTMTD [NUMBER(11)]	AMTMTD [DECIMAL(11)]	
AMTYTD [NUMBER(11)]	AMTYTD [DECIMAL(11)]	
AMTLTD [NUMBER(11)]	AMTLTD [DECIMAL(11)]	
	UPD_TIMESTAMP [TIMESTAMP]	NULL

☑ Show Column Data Types

Auto Map... | Key Columns...

CDCDEMO.SALESREP ⇨ CDCDEMO.SALESREP_TARGET

Save | Revert

Figure 9-27 New column with default mapping

9.5.16 Deploying subscription changes and considerations

Using the API, you can deploy subscription changes to production environments without manual intervention through the Management Console. You want to make the deployment as smooth as possible and avoid having to do a refresh of the replicated tables or run the risk of losing changes.

This section assumes that you do not want to perform a refresh of the replicated tables when making subscription changes. This situation is the case for most clients, and in environments where very large tables are replicated, refreshing those tables may not be an option.

Before we continue describing how to read how table structure changes pertain to the InfoSphere CDC API, you need to understand the bookmarks of subscriptions and replicated tables. More information about this subject can be found in 7.6.1, “Understanding InfoSphere CDC bookmarks” on page 191.

As long as you do not delete a subscription (or the Publication in the target metadata), the subscription bookmark is kept in the target bookmark table. This situation means that restarting the subscription after making changes to the production subscription causes the bookmark to be retrieved from the bookmark table and start reading the database logs from this bookmark forward. The risk of inadvertently losing the subscription bookmark is not high because you must deassign all target tables (PublishedTable), remove the selected tables (SubscribedTable), and perform a description to remove the Publication object. Only in that situation will InfoSphere CDC drop the bookmark on the target side.

A greater risk is inadvertently marking the table capture point after changing the mapped tables. If you set the status of a selected table (SubscribedTable) to Active, you are effectively marking the table capture point for that table. Any transaction for that table since the last applied bookmark is skipped when the subscription is restarted.

Starting with Version 6.5, the InfoSphere CDC engines offer a CLI command option for **`dmsetbookmark`** to set the bookmark and mark the table capture points of all tables according to the bookmark. This option is useful in situations where subscriptions must be recreated and you want to restart from the last known position without skipping any transactions for any of the tables.

Unfortunately, the Java API does not have an interface for getting or setting the bookmarks for the engine, and you must start engine commands from the custom Java class to perform this function. Starting engine commands from the deployment Java programs might pose a challenge because the API is run against the Access Server, which might be installed on a different server than the source and target engines. You then must run remote CLI commands for the InfoSphere CDC servers.

If you want to keep the deployment processing within the scope of the Java API and not lose any subscription or table bookmarks, the selected tables (SubscribedTable) must be kept in the subscription and the replication method and status for these tables must not be changed. This situation means that if you need to change the mapping attributes of the source tables, such as row filtering, column filtering, or derived columns, you must modify the existing SubscribedTable object instead of recreating it. For example, if the table mapping change encompasses the removal of a derived column, but other derived columns must be kept for the selected table, you could remove all the derived columns from the SubscribedTable objects. Then, add them all (except the removed one) again based on the definition of the subscription to be deployed.

For table assignment processing, you do not have to take the same precautions. It is safe to remove table assignments (deassign) and assign them again with new attributes. Deassigning tables does not touch the subscription bookmark or the marked table capture point for the tables. For example, if you want to change the mapping of tables that have been mapped with a standard mapping type to adaptive apply, you can start by deassigning the mapped tables and then assigning them again using the adaptive apply mapping type.

There are situations in which the table capture point cannot be kept. One of the common situations is when the format of the source table has changed and the table must be readded to source data store catalog. When readding a table to the catalog to replace the table's recorded structure, InfoSphere CDC automatically changes the table's replication status in all subscriptions to idle (parked). To resume replication for the table in question, you must change the replication status to active. This action is the same action as marking the capture point for the table (the operations between the last stopping of the subscription and the marked table capture point are skipped).

9.5.17 Starting, stopping, and monitoring subscriptions

Subscriptions are always started and stopped from the source side. Also, if you want to monitor for subscription activity, this action can only be done on the source server. Some companies have strict policies regarding access to source (production) servers and may not allow command-line access to these servers. In those scenarios, the API might be a good alternative for starting, stopping, and monitoring subscriptions.

If you do have access to the source server, you can start and stop subscriptions using the engine command interface. The Linux, UNIX, and Windows engine also allows you to check the activity of subscriptions using InfoSphere CDC commands. If you want to provide more sophisticated functionality, the API provides more control over specific features within the InfoSphere CDC engine.

Continuous mirroring replicates changes to the target on a continuous basis. Use this type of mirroring when business requirements dictate that you need replication to be running continuously and you do not have a clearly defined reason to end replication now.

Scheduled end (Net Change) mirroring replicates changes (to the target) up to a user-specified point in the source database log and then ends replication. Use this type of mirroring when business requirements dictate that you only replicate your data periodically and you have a clearly defined endpoint for the state of your target database when replication ends. Scheduled end mirroring allows you to end replication at the following points in your source database log:

- Current time or “now”
- User-specified date and time
- User-specified log position

The **startSubscription()** method shown in Example 9-57 starts a subscription in Continuous mirroring or Net Change with a “Now” ending specification (replication stops when the current log position has been reached).

Example 9-57 Start subscriptions (Continuous mirroring or Net Change)

```
/**
 * Start a subscription in continuous mirror or net change mode
 *
 * @param subscription
 *       - Subscription you want to start
 * @param continuousMirror
 *       - Start the subscription in continuous mirror mode (true) or net change
 *           (false)
 * @return
 * @throws ApiException
 */
public void startSubscription(Subscription subscription,
      boolean continuousMirror) throws ApiException {
// Check if the subscription is already active before attempting to
// start it
   byte[] liveActivityStatus = subscription.getLiveActivityStatus();
   if (liveActivityStatus[1] == Subscription.LIVE_STATUS_IDLE) {
      subscription.startMirror(continuousMirror);
      if (continuousMirror)
         System.out.println("Started subscription "
               + subscription.getName()
               + " in Continuous Mirror mode.");
      else
         System.out.println("Started subscription "
            + subscription.getName() + " in Net change mode.");
   } else {
      System.out.println("Subscription " + subscription.getName()
            + " is already active.");
   }
}
```

In **`startSubscription()`**, use the **`Subscription.startMirror`** method to start the replication using the two basic techniques that are available for all versions of InfoSphere CDC. Starting with Version 6.5, InfoSphere CDC supports starting mirroring with a user-specified end date and time or log position when the replication should stop. If you want to use this functionality, use the **`startReplicationSpecifiable`** method, which is part of the Subscription class.

In Example 9-58, a subscription is started to replicate yesterday's changes. After the log position reaches the specified end time (23:59:59.999), it stops normally.

Example 9-58 Specifiable start of replication

```
/**
 * Start a subscription and replicate the changes that were made yesterday
 *    (relative to today).
 *
 * @param subscription
 *            - Subscription you want to start
 * @return
 * @throws ApiException
 */
   public void startReplicationYesterdaysChanges(Subscription subscription)
         throws ApiException {
// Calculate the ending time (yesterday, 23:59:59.999)
      SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmssSSS");
      Calendar subscriptionEndTime = Calendar.getInstance();
      subscriptionEndTime.add(Calendar.DATE, -1);
      subscriptionEndTime.set(Calendar.HOUR_OF_DAY, 23);
      subscriptionEndTime.set(Calendar.MINUTE, 59);
      subscriptionEndTime.set(Calendar.SECOND, 59);
      subscriptionEndTime.set(Calendar.MILLISECOND, 999);
// Check if the subscription is already active before attempting to start it
      byte[] liveActivityStatus = subscription.getLiveActivityStatus();
      if (liveActivityStatus[1] == Subscription.LIVE_STATUS_IDLE) {
         byte flag = 5; // Scheduled end
         byte tag = 1; // Ignored for mirror
         byte endValueType = 2; // Source local time (log time)
         String endValue = sdf.format(subscriptionEndTime.getTime());
         subscription.startReplicationSpecifiable(flag, tag, endValueType,
               endValue);
         System.out.println("Started subscription " + subscription.getName()
               + " with an ending log time of " + endValue);
      } else {
         System.out.println("Subscription " + subscription.getName()
               + " is already active.");
      }
   }
```

To end a subscription, you can use the **endSubscription()** method. Similar to the starting of the replication, ending replication can now also be made "specifiable" by using the **endReplicationSpecifiable()** method (Example 9-59).

Example 9-59 Stopping replication

```
/**
 * Stop a mirroring subscription controlled or immediately
 *
 * @param subscription
 *            - Subscription you want to stop
 * @param continuousMirror
 *            - Start the subscription in continuous mirror mode (true) or
 *            net change (false)
 * @return
 * @throws ApiException
 */
public void stopSubscription(Subscription subscription,
      boolean immediateStop) throws ApiException {
// Check if the subscription is already active before attempting to
// start it
   byte[] liveActivityStatus = subscription.getLiveActivityStatus();
   if (liveActivityStatus[1] == Subscription.LIVE_STATUS_ACTIVE) {
      subscription.stopMirror(immediateStop);
      if (immediateStop)
         System.out.println("Stopping subscription "
               + subscription.getName() + " immediately.");
      else
         System.out
               .println("Stopping subscription "
                     + subscription.getName()
                     + " in a controlled manner. "
                     + "Subscription will stop when current log position "
                     + "has been reached.");
   } else {
      System.out.println("Subscription " + subscription.getName()
            + " is not active.");
   }
}
```

Obtaining the state of a subscription is a bit more elaborate, as the state depends on the activity (Refresh, Continuous Mirror, or Net Change) and status (Starting, Active, or Inactive). The current version of the Java API has relatively limited capabilities for retrieving the actual state of a subscription. The Management Console has a greater variety of states it can retrieve from the InfoSphere CDC engines, such as Refresh before mirror, and Ending. In the **getSubscriptionStateAsString()** method (Example 9-60), attempt to copy the Management Console states as accurately as possible, providing a return string that resembles the activity (what is the subscription doing) and status (is it active or not) of a subscription.

Example 9-60 Returns the activity of a subscription as a string

```
/**
 * Returns the activity of a subscription as a string for display. Trying to
 * copy the activity state from the Management Console as best as possible.
 *
 * @param subscription
 *            - The subscription for which the state must be returned
 * @return State of the subscription
 * @throws ApiException
 * @throws UnsupportedFeatureException
 */
private String getSubscriptionStateAsString(Subscription subscription)
      throws UnsupportedFeatureException, ApiException {
   byte liveActivity = 0;
   byte liveStatus = 0;

   byte[] liveActivityStatus = subscription.getLiveActivityStatus();
   liveActivity = liveActivityStatus[0];
   liveStatus = liveActivityStatus[1];

   String stateString = "Unknown";
   if (liveStatus == Subscription.LIVE_STATUS_IDLE)
      stateString = "Inactive";
   else if (liveStatus == Subscription.LIVE_STATUS_START)
      stateString = "Starting";
   else if (liveStatus == Subscription.LIVE_STATUS_ACTIVE) {
      if (liveActivity == Subscription.LIVE_ACTIVITY_MIRROR)
```

```
         stateString = "Mirror Continuous";
      else if (liveActivity == Subscription.LIVE_ACTIVITY_NET_CHANGE)
         stateString = "Mirror Scheduled End";
      if (liveActivity == Subscription.LIVE_ACTIVITY_DESCRIBE)
         stateString = "Describe";
      if (liveActivity == Subscription.LIVE_ACTIVITY_REFRESH)
         stateString = "Refresh";
   } else if (liveStatus == Subscription.LIVE_STATUS_DS_STARTING_JOB)
      stateString = "Starting DataStage Job";
   else if (liveStatus == Subscription.LIVE_STATUS_DS_WAITING_FOR_JOB_TO_START)
      stateString = "Waiting for DataStage Job to be Started";
   else if (liveStatus == Subscription.LIVE_STATUS_DS_CONNECTING_WITH_TARGET)
      stateString = "Waiting for DataStage Job to Connect";
   else if (liveStatus == Subscription.LIVE_STATUS_DS_JOB_ENDING)
      stateString = "Ending DataStage Job";
   return stateString;
}
```

The code shown in Example 9-61 demonstrates the asynchronous nature of starting and stopping subscriptions. For a duration of 15 seconds, the subscription is monitored, and every half second, the state of the subscription is sent to the standard output. After one second of running, the subscription is started in Continuous mirror mode, and after 10 seconds, the subscription is stopped immediately. An immediate stop of the subscription does not mean terminating the activity. The subscription still stops in a normal fashion, but does not process all the log entries until the log time of the attempted stop is reached (a controlled stop).

Example 9-61 Asynchronous starting and stopping a subscription

```
// Show the subscription state for 15 seconds, interval at 500 ms
// Start the subscription after 1 second, stop after 10 seconds
int maxTime = 15000;
int statusInterval = 500;
int subscriptionStart = 1000;
int subscriptionStop = 10000;
int timeConsumed = 0;
while (timeConsumed < maxTime) {
   System.out.println("Current state of subscription "
         + subscription.getName() + ": "
         + getSubscriptionStateAsString(subscription));
   Thread.sleep(statusInterval);
   timeConsumed += statusInterval;
   if (timeConsumed == subscriptionStart) {
// Start the subscription in continuous mirror mode
```

```
      try {
         startSubscription(subscription, true);
      } catch (ApiException e) {
         e.printStackTrace();
      }
   } else if (timeConsumed == subscriptionStop) {
// Stop the subscription immediate
      try {
         stopSubscription(subscription, true);
      } catch (ApiException e) {
         e.printStackTrace();
      }
   }
}
```

Running the code produces the output shown in Example 9-62.

Example 9-62 Output from Example 9-61 on page 344

```
Current state of subscription REDSCRIPT: Inactive
Current state of subscription REDSCRIPT: Inactive
Started subscription REDSCRIPT in Continuous Mirror mode.
Current state of subscription REDSCRIPT: Starting
Current state of subscription REDSCRIPT: Starting
Current state of subscription REDSCRIPT: Refresh
Current state of subscription REDSCRIPT: Refresh
Current state of subscription REDSCRIPT: Refresh
Current state of subscription REDSCRIPT: Refresh
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Stopping subscription REDSCRIPT immediately.
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Mirror Continuous
Current state of subscription REDSCRIPT: Inactive
Current state of subscription REDSCRIPT: Inactive
Current state of subscription REDSCRIPT: Inactive
Current state of subscription REDSCRIPT: Inactive
Current state of subscription REDSCRIPT: Inactive
```

In Example 9-62 on page 345, you can see that the subscription is inactive during the first second (two messages). After that period, the subscription starts its processing and then commences refreshing the tables that have been marked for refresh, which takes a total of approximately 2.5 seconds. Then the subscription goes into Mirror Continuous mode until the loop reaches the 10 second limit you set and stops the subscription immediately (that is, it takes about 2.5 seconds to make the subscription inactive).

The code could have been coded using threads, but for simplicity, we chose to use a loop.

9.5.18 Monitoring latency

Management Console provides a convenient method for viewing performance and statistics data generated by the InfoSphere CDC engines. But sometimes you need to automate the retrieval of this information, for example, to build an offline history of performance or perform automated application monitoring through third-party solutions. The InfoSphere CDC Java API provides classes to access this information.

Figure 9-28 shows an overview of the process to access live performance and statistics data on the data store.

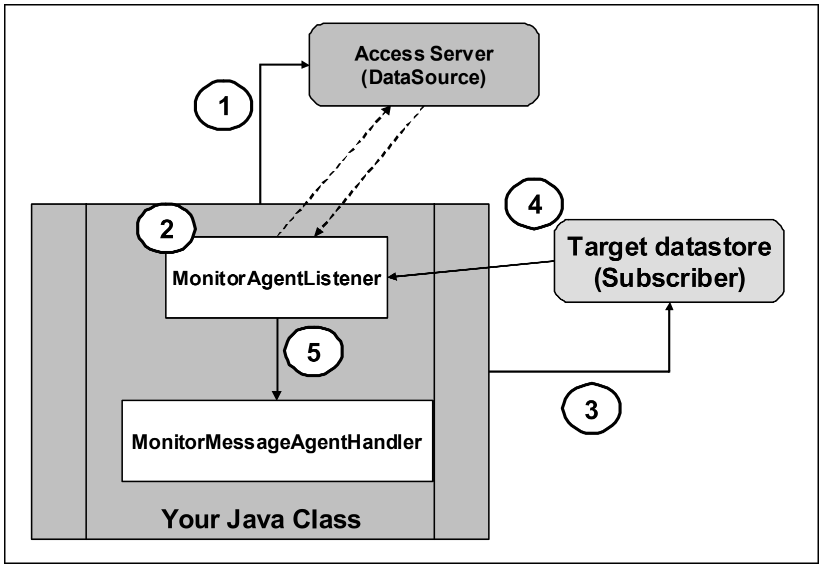

Access Server (DataSource)
1
2
MonitorAgentListener
4
Target datastore (Subscriber)
5
3
MonitorMessageAgentHandler
Your Java Class

Figure 9-28 Monitoring latency

Here is a brief description of the processes shown in Figure 9-28:

1. Connect to the Access Server (create a DataSource object).
2. Create the MonitorAgentListener, a monitor that listens to all incoming messages from data stores and directs them to a MonitorAgentMessageHandler object.
3. Send the performance statistics request to the data store.
4. The data store returns (asynchronously) the response to the MonitorAgentListener object (through the Access Server).
5. MonitorAgentListener accepts the message and starts methods of the MonitorAgentMessageHandler interface.

The dashed lines between MonitorAgentLister and the Access Server represent communication between them, after the listener has been established.

After the connection to the Access Server (DataSource) is established, set up a listener that accepts incoming messages from the data stores. The sample code to perform this task is shown in Example 9-63.

Example 9-63 Sample code to set up a listener

```
/**
 * Configures and starts the monitor agent listener for incoming statistics
 * messages
 *
 * @param accessServerHost
 *            - host name of the access server
 * @param accessServerPort
 *            - port of the access server
 * @return No return value
 * @throws Exception
 */
private CDCMonitorHandler setMonitorAgentListener(DataSource accessServer,
      String accessServerHost, int accessServerPort) throws Exception {
   CDCMonitorHandler monitorHandler = null;
   MonitorAgentListener monitorAgentListener = null;

   monitorHandler = new CDCMonitorHandler();
   monitorAgentListener = new MonitorAgentListener(accessServerHost,
         accessServerPort, 0, 0, monitorHandler);
   monitorAgentListener.setDaemon(true);

   if (monitorAgentListener.init()) {
      monitorAgentListener.start();
      if (monitorAgentListener.isAlive()) {
         accessServer.setMonitorListeningPort(
               monitorAgentListener.getHostName(),
               monitorAgentListener.getPort());
         System.out.println("MonitorAgentListener listens on host "
               + monitorAgentListener.getHostName() + " and port "
               + monitorAgentListener.getPort() + ".");
      }
   }
   return monitorHandler;
}
```

In Example 9-63 on page 348, a MonitorAgentListener object is created, with host name, port, and MyMonitorHandler object, and is initialized and started as a daemon (background) thread. This action starts a listener process on a new port that is assigned by the TCP/IP stack. Then check if the monitor is alive and let the Access Server know that any monitor information must be passed to the monitor agent listener.

After the listener has been started, you must request the data from the data stores. Information about throughput and statistics is requested from the subscribers (target data stores) in the context of a specific subscription. The **`requestPerformanceData()`** method requests statistics from data stores. Depending on the version of the InfoSphere CDC engines, you must determine how to obtain the statistics from the data stores. Starting with InfoSphere CDC V6.5, enhanced monitoring is supported and many statistics figures can be retrieved from those data stores. To understand if the InfoSphere CDC engine represented by the Subscribed object supports enhanced statistics, request the FID_MONITORING_LEVEL feature from the engine. The sample code to perform this action is shown in Example 9-64.

Example 9-64 Requesting performance statistics

```
/**
 * Issue statistics requests to target datastore for a subscription
 *
 * @param subscription
 *            - The subscription for which the statistics are requested
 * @param targetDataStore
 *            - Target datastore of the subscription
 * @param monitorHandler
 *            - The object that will handle the statistics responses
 *
 * @return No return value
 */
private void requestPerformanceData(Subscription subscription,
      Subscriber targetDataStore, CDCMonitorHandler monitorHandler)
      throws ApiException, InterruptedException {
// Check if the subscription is active
   byte[] subscriptionStatus = subscription.getLiveActivityStatus();
   Publication publication = getSubscriptionPublication(subscription,
         targetDataStore);
// If subscription is active, request statistics
   if (subscriptionStatus[1] == Subscription.LIVE_STATUS_ACTIVE) {
// Issue 10 requests
      for (int i = 1; i <= 10; i++) {
         System.out.println("Request " + i);
// Check if datastore supports enhanced statistics
```

```
         if (targetDataStore.getFeature(TsFeatureSet.GID_GENERIC,
               TsFeatureSet.FID_MONITORING_LEVEL).equals("1")) {
// Issue InfoSphere CDC 6.5 request
            monitorHandler.requestTargetOperationalStatistics(
                  targetDataStore, publication.getPublisherID());
         } else {
// Issue InfoSphere CDC 6.3 requests
            targetDataStore.sendGetPerformanceData(publication
                  .getPublisherID());
            targetDataStore.sendGetStatisticsData(
                  publication.getPublisherID(), "");
         }
         Thread.sleep(1000);
      }
   } else {
      System.err.println("Subscription " + subscription.getName()
            + " is not active, cannot retrieve statistics.");
   }
}
```

After checking that the subscription is active, the code issues 10 performance requests to the target data store. Depending on whether the data store supports enhanced statistics (which were introduced in InfoSphere CDC V6.5), statistics requests are sent to the target data store. The method for obtaining operational performance statistics has changed dramatically with the introduction of InfoSphere CDC V6.5. If the InfoSphere CDC engine is Version 6.5 or later, you can use the CDCMonitorHandler class **`requestTargetOperationalStatistics()`** method. For Version 6.3 and earlier, enhanced statistics are not supported and you use the **`sendGetPerformanceData()`** and **`sendGetStatisticsData()`** methods from the Subscriber class. Requests are issued with a three second pause between them.

Responses from data stores arrive asynchronously and they start methods in the MonitorAgentMessageHandler interface, which is implemented by the CDCMonitorHandler class. InfoSphere CDC V6.5 data stores start different methods in this interface than previous versions when the statistics messages arrive. InfoSphere CDC V6.5 data stores can be handled in the same way as InfoSphere CDC V6.3 data stores, but only a limited set of data is available. All the methods are implemented in the CDCMonitorHandler class, which was supplied as an argument to MonitorAgentListener object.

To handle latency data that arrives from InfoSphere CDC V6.3 data stores, the **handleLatencyUpdate()** method in the MonitorAgentMessageHandler interface should be implemented. The sample code to perform this action is shown in Example 9-65.

Example 9-65 Handling latency data for InfoSphere CDC V6.3

```
/**
 * Handle latency message that arrives from InfoSphere CDC 6.3 datastore
 *
 * @param inDataStream
 *            - stream of data that arrives from datastore
 * @return No return value
 */
public void handleLatencyUpdate(DataInputStream inDataStream) throws IOException,
DataNotFoundException {

// Populate query object with the data arrived
   AbstractQuery query = new Query_5002(null);
   Result result = query.receiveResult(inDataStream);

// Get latency records from the stream
   Context[] latencyRecords = result.getContexts(Query.LatencyRecord);

// Iterate over the records and print, if there are more than one
   for (int i = 0; i < latencyRecords.length; i++) {
      long latencyValue = latencyRecords[i].getLong(Query.LatencyValue) / 1000;
      System.out.println("Latency: " + latencyValue + " seconds");
   }
}
```

Your main thread could be busy while the statistics messages arrive. So, the code shown in Example 9-65 demonstrates the processing of an array of latency records.

For every **sendGetPerformanceData()** invocation, a message from the data store arrives and triggers invocation of this method. Subscription latency, expressed in milliseconds, is extracted from the input data stream and printed.

To handle statistics data (number of inserts, updates, and deletes), the **handleStatisticsUpdate()** method, shown in Example 9-66, should be implemented.

Example 9-66 Handling operational statistics for InfoSphere CDC 6.3

```
/**
 * Handle operation statistics message that arrives from InfoSphere CDC 6.3 datastore
 *
 * @param inDataStream
 *            - stream of data that arrives from datastore
 * @return No return value
 */
public void handleStatisticsUpdate(DataInputStream inDataStream) throws IOException,
DataNotFoundException {

// Populate query object with the data arrived
   AbstractQuery query = new Query_5003(null);
   Result result = query.receiveResult(inDataStream);

   if (result.hasData(Query.StatisticsRecord)) {
      Context[] statisticsRecords = result.getContexts(Query.StatisticsRecord);

// Go over the messages received and based on the type of the message print the data
   for (int i = 0; i < statisticsRecords.length; i++) {
         String name = statisticsRecords[i].getString(Query.FieldName);
         String value = statisticsRecords[i].getString(Query.FieldValue);

         if (name.compareToIgnoreCase("APPLY_UPDATE_COUNT") == 0) {
            System.out.println("Updates: " + value);
         } else if (name.compareToIgnoreCase("APPLY_DELETE_COUNT") == 0) {
            System.out.println("Deletes: " + value);
         } else if (name.compareToIgnoreCase("APPLY_INSERT_COUNT") == 0) {
            System.out.println("Inserts: " + value);
         }
      }
   }
}
```

This method is run for every **sendGetStatisticsData()** invocation in the program. It is important to remember that data stores begin to collect data from the first time it was requested. So for the first run of the program, all statistics begin from 0 (even if thousands of operations were performed before) and every other execution shows values updated since the first request.

InfoSphere CDC V6.5 data stores have enhanced monitoring capabilities, and can provide a significant amount of statistical information that was not available in prior versions of InfoSphere CDC. Therefore, accessing that data requires different methods to be used. Because there is much more information that can be requested, not all data arrives by default, and specific categories should be defined in the request. Requests to the target data store is defined in the **requestTargetOperationalStatistics()** method (Example 9-67).

Example 9-67 Requesting statistical information for InfoSphere CDC V6.5

```
/**
 * Request operational statistics from InfoSphere CDC 6.5 datastore
 *
 * @param subscriber
 *            - subscriber that will send messages
 * @param subscriptionName
 *            - subscription name
 * @return No return value
 */
public void requestTargetOperationalStatistics(Subscriber subscriber, String
subscriptionName)
      throws ApiException {

   targetStatisticData = new PerformanceStatisticData();

   StatisticDefinition definition;

// Get the supported statistics from the replication subscriber.
   StatisticCategory[] categories = subscriber.getStatisticDefinitions();
   for (StatisticCategory category : categories) {
      if (category.getId() == StatisticConstants.CATEGORY_SUBSCRIPTION_TARGET) {
         definition = category
               .getDefinition(StatisticConstants.TARGET_APPLY_INSERT_OPERATIONS);
         if (definition != null) {
            targetStatisticData.addDefinition(definition);
         }

         definition = category
               .getDefinition(StatisticConstants.TARGET_APPLY_UPDATE_OPERATIONS);
         if (definition != null) {
            targetStatisticData.addDefinition(definition);
         }

         definition = category
               .getDefinition(StatisticConstants.TARGET_APPLY_DELETE_OPERATIONS);
         if (definition != null) {
            targetStatisticData.addDefinition(definition);
         }

         definition = category.getDefinition(StatisticConstants.TARGET_LATENCY);
         if (definition != null) {
            targetStatisticData.addDefinition(definition);
         }
      }
   }

// Perform the request
   requestStatistics(subscriber, targetStatisticData, false, subscriptionName);
}
```

The targetStatisticsData object of class PerformanceStatisticData is defined as a data member in the CDCMonitorHandler class. From the subscriber, obtain all statistic definitions you are interested in and add them to the PerformanceStatisticData object, which is sent as part of the request to the data store. The **requestStatistics()** method performs the request and is used because it can send requests both to source and target data stores, because only the part of building the statistics definitions is different (Example 9-68).

Example 9-68   Request statistics from both source and target

```
/**
 * Requests the statistics from an agent as identified by the statistic data
 * object. The object must only contain supported statistics and must not
 * mix source and target statistics in the same request.
 *
 * @param replicationRole
 *            - the publisher or subscriber agent.
 * @param statisticData
 *            - the statistic data object that identifies which statistics
 *            to request.
 * @param isSource
 *            - true if the request is made to the source agent, false if
 *            made to the target agent.
 * @param subscriptionName
 *            - the name of the subscription if the request is for the
 *            source, or the published subscription ID if the request is for
 *            the target.
 * @throws ApiException
 *             - if any exception occurs.
 */
```

```
private void requestStatistics(ReplicationRole replicationRole, StatisticData
statisticData,
      boolean isSource, String subscriptionName) throws ApiException {
   Query query = replicationRole.getQuery(Query.MSG_GET_STATISTIC_VALUES);

   ArrayList<StatisticDefinition> statisticDefinitions =
      statisticData.getDefinitions();

   query.setInt(Query.Correlator, statisticData.getCorrelatorId());

// Initialize agentType according to the type of the datastore byte agentType = 0;
   if (!isSource)
      agentType = 1;

// Build the request object
   query.setByte(Query.AgentType, agentType);
   query.setByte(Query.ContextObjectType,
         StatisticContextObjectType.SUBSCRIPTION.getMessageValue());
   query.setString(Query.SubscriptionName, subscriptionName);
   query.setString(Query.TableOwner, ""); //$NON-NLS-1$
   query.setString(Query.TableName, ""); //$NON-NLS-1$

// Add all statistics definitions
   for (StatisticDefinition definition : statisticDefinitions) {
      query.addInt(Query.StatIds, definition.getId());
   }

   query.setLong(Query.BeginTime, 0);
   query.setLong(Query.EndTime, 0);
   query.setInt(Query.SampleCount, 1);

// Send the request to the datastore
   query.request();
}
```

In this method, the Query object is built, based on all statistics definitions that are added and the type of the data store, and it is sent to the data store. Unlike a InfoSphere CDC V6.3 data store, all response messages arrive at the same (**`handleStatisticValues()`**) method, regardless whether the latency or operations was requested (Example 9-69).

Example 9-69 Handling statistics for InfoSphere CDC V6.5

```
/**
 * Handles the returned statistics from an agent as identified by the
 * statistic data object. The object must only contain supported statistics
 * and must not mix source and target statistics in the same request.
 *
 * @param dataStoreHost
 *            - host name of the datastore
 * @param dataStorePort
 *            - port of the datastore
 * @param messageCorrelator
 *            - correlation id of the response (to match request)
 * @param maMessageBodyLength
 *            - length of the message body
 * @param maMessageBody
 *            - message body
 * @return No return value
 * @throws ApiException
 *             - if any exception occurs.
 */
public void handleStatisticValues(String dataStoreHost, int dataStorePort,
      int messageCorrelator, int maMessageBodyLength, byte[] maMessageBody)
      throws IOException, DataNotFoundException {

   PerformanceStatisticData statisticData = null;

// Obtain statistics received
   if (targetStatisticData != null
         && targetStatisticData.getCorrelatorId() == messageCorrelator) {
      statisticData = targetStatisticData;
   }

// If it's not null
   if (statisticData != null) {
      statisticData.loadData(maMessageBody);

      long value;

// Extract all the data that was sent and print it
      value = statisticData
            .getCurrentValueById(StatisticConstants.TARGET_APPLY_UPDATE_OPERATIONS);
      System.out.println("Updates: " + value);

      value = statisticData
            .getCurrentValueById(StatisticConstants.TARGET_APPLY_DELETE_OPERATIONS);
      System.out.println("Deletes: " + value);

      value = statisticData
```

```
            .getCurrentValueById(StatisticConstants.TARGET_APPLY_INSERT_OPERATIONS);
      System.out.println("Inserts: " + value);

      value = statisticData
            .getCurrentValueById(StatisticConstants.TARGET_LATENCY);
      System.out.println("Latency: " + value + " seconds");
   }
}
```

When this method is called, upon message arrival, the data is extracted from the message and printed out. In this example, only latency and operations are requested and received, but in principle everything that appears in Management Console could be extracted.

The code shown in Example 9-70 starts the **requestPerformanceData()** method, which sends 10 requests to the target InfoSphere CDC data store.

Example 9-70 Request performance data

```
// Set up the monitor handler
CDCMonitorHandler monitorHandler = setMonitorAgentListener(
      accessServer, asHost, asPort);
// Send the requests to get subscription statistics
requestPerformanceData(subscription, db2DataStore, monitorHandler);
```

When the code is run, the output could look something like the output shown in Example 9-71. We deliberately included some delays in the apply process to be able to show some latency.

Example 9-71 Sample output from performance data request

```
MonitorAgentListener listens on host 127.0.1.1 and port 50738.
Request 1
Updates: 200
Deletes: 92
Inserts: 1262
Latency: 17 seconds
Request 2
Updates: 200
Deletes: 92
Inserts: 1309
Latency: 22 seconds
Request 3
Updates: 200
Deletes: 92
Inserts: 1309
```

```
Latency: 22 seconds
Request 4
Updates: 200
Deletes: 92
Inserts: 1309
Latency: 22 seconds
Request 5
Updates: 200
Deletes: 92
Inserts: 1309
Latency: 22 seconds
Request 6
Updates: 200
Deletes: 92
Inserts: 1309
Latency: 22 seconds
Request 7
Updates: 200
Deletes: 92
Inserts: 1358
Latency: 27 seconds
Request 8
Updates: 200
Deletes: 92
Inserts: 1358
Latency: 27 seconds
Request 9
Updates: 200
Deletes: 92
Inserts: 1358
Latency: 27 seconds
Request 10
Updates: 200
Deletes: 92
Inserts: 1358
Latency: 27 seconds
```

9.5.19 Monitoring event logs using the API

Event logs are kept in every InfoSphere CDC engine's metadata repository or in operating system objects (System i and System z). In InfoSphere CDC, every event is represented by an object of type TsEvent. A TsEvent object contains all the information about the event, such as date and time, ID, type (Information and Error as examples), originator module of the event, and the text itself. If an event is an object of type TsEvent, the sample content could be as shown in Example 9-72.

Example 9-72 Event sample content

```
event.getTime() : 035247
event.getDate() : 20110330
event.getID() : 1463
event.getSubscription(): REDSCRIPT
event.getType() : 4 (means Information)
event.getOriginator() :
com.datamirror.ts.source.subscription.SubscriberProxy
event.getInformation() : +++ Subscription REDSCRIPT is starting in
Continuous
Mirroring mode.
```

The **getType()** method returns a byte that indicates the severity of the message. The method that translates this code into a meaningful description is shown in Example 9-73.

Example 9-73 Method to translate the message event type

```
/**
 * Return description of the event type code
 *
 * @param type
 *            - byte, returned by event.getType() method
 * @return Description of the type
 */
private String translateEventType(byte type) {
   switch (type) {
   case TsEvent.TYPE_ALL:
      return "All";
   case TsEvent.TYPE_DIAGNOSTIC:
      return "Diagnostic";
   case TsEvent.TYPE_ERROR:
      return "Error";
   case TsEvent.TYPE_ESCAPE:
      return "Escape";
   case TsEvent.TYPE_INFORMATION:
      return "Information";
   case TsEvent.TYPE_NOTICE:
      return "Notice";
   case TsEvent.TYPE_WARNING:
      return "Warning";
   }
   return "Unknown";
}
```

The event log is retrieved as Enumeration of TsEvent objects by the **`getEvents()`** method, which is starts on Subscription or Publication (subscription level events) objects, or Publisher or Subscriber (data store level events). We have created a single method that shows the events (Example 9-74).

Example 9-74 Method to show log events

```
/**
 * Show events that were collected in the <TsEvent> enumeration
 *
 */
private void showEvents(Enumeration<TsEvent> events) {
   int numberEvents = 0;
   while (events != null && events.hasMoreElements()) {
      TsEvent event = events.nextElement();
      System.out.println(event.getTime() + "|" + event.getDate() + "|"
            + event.getID() + "|" + translateEventType(event.getType())
            + "|" + event.getOriginator() + "|"
            + event.getInformation());
      numberEvents++;
   }
   System.out.println(numberEvents + " events displayed.");
}
```

Subscription event log

As previously mentioned, information about subscription is contained in both source and target engines, so events that are related to a specific subscription is generated on them both, depending on where the event took place. To retrieve all subscription events, you need the Subscription and Publication object. The sample code to perform this task is shown in Example 9-75.

Example 9-75 Show subscription events

```
/**
 * Show subscription events stored on the source and target
 *
 * @param Subscription
 *            - The subscription for which to show source and target events
 * @param targetDataStore
 *            - Target engine of the subscription
 * @param subscriptionName
 *            - Name of the subscription
 * @return No return value
 */
private void showSubscriptionEventLog(Subscription subscription,
      Subscriber targetDataStore) throws ApiException {
// Object to hold event log
   Enumeration<TsEvent> events = null;
// Get events of the subscription from the source
   events = subscription.getEvents(true);
   System.out.println("SOURCE EVENTS FOR SUBSCRIPTION "
         + subscription.getName() + ":");
   showEvents(events);
// Get publication for the subscription
   Publication publication = getSubscriptionPublication(subscription,
         targetDataStore);
   if (publication != null) {
// Get events of the subscription from the target
      events = publication.getEvents(true, Publication.REPLICATION_LOG);
      System.out.println("TARGET EVENTS FOR SUBSCRIPTION "
            + subscription.getName() + ":");
      showEvents(events);
   }
}
```

For the passed Subscription object, **`getEvents()`** is starts and the events populate variable events of type Enumeration<TsEvent>. The method has one Boolean parameter, recentFirst. When set to true, the events are sorted from the most recent to the oldest, and when set to false, the oldest events appear first and the most recent last.

Then, the Publication object for the subscription is retrieved from the target data store and the events are retrieved from that object as well. In both cases, the **`showEvents()`** method is called to show the events.

Data store event log

The event log of every data store can be retrieved by starts the **`getEvents()`** method on the ReplicationRole object or any of its subclasses (Publisher or Subscriber). Assuming you have the ReplicationRole object, Example 9-76 shows the sample code to retrieve the event log:

Example 9-76 Show data store events

```
/**
 * Show datastore events
 *
 * @param dataStore
 *            - The source or target datastore to show the events for
 * @return No return value
 */
private void showDatastoreEventLog(ReplicationRole dataStore)
      throws ApiException {

// Object to hold event log
   Enumeration<TsEvent> events = null;
// Get events of the datastore
   events = dataStore.getEvents(true);
   System.out.println("EVENTS FOR DATASTORE " +
      dataStore.getDescription() + ":");
   showEvents(events);
}
```

After the connection to the data store is established, events are retrieved and printed similar to the way it was done from the subscription. The output of the method is similar to the output of subscription events, but with the events that are related to the data store and not a specific subscription.

Single scrape event log

Retrieving the events of a single scrape is different. The sample method that retrieves and prints events is shown in Example 9-77.

Example 9-77 Retrieving single scrape events

```
/**
 * Show single scrape events
 *
 * @param sourceDataStore
 *            - Publisher that holds the single scrape
 * @return No return value
 */
```

```
private void showSingleScrapeEventLog(Publisher sourceDataStore)
      throws ApiException {
// Object to hold event log
   Enumeration<TsEvent> events = null;
// Get events of the datastore
   events = new EventLog(sourceDataStore, (byte) 2,
null).getEvents(true);
   System.out.println("SINGLE SCRAPE EVENTS FOR DATASTORE "
         + sourceDataStore.getDescription() + ":");
   showEvents(events);
}
```

The difference from the previous sample is the way the EventLog object is instantiated by passing the source data store and type of events as the arguments. The single scrape event log is of type (byte)2. Then, on this object starts the **`getEvents()`** method. As you can see in the method's parameters, expect a Publisher object to be passed to it. Because single scrape only exists in source data stores, prevent sending data stores of type target (such as InfoSphere CDC DataStage) to the method and encounter errors in trying to retrieve an event log that does not exist. The output is similar to other methods, with different events.

Showing the event logs

We now combine the three methods in the code shown in Example 9-78.

Example 9-78 Event log display methods

```
showDatastoreEventLog(oracleDataStore);
showSingleScrapeEventLog(oracleDataStore);
showSubscriptionEventLog(subscription, db2DataStore);
showDatastoreEventLog(db2DataStore);
```

When you run this sample, the output shown in Example 9-79 is produced (not all events are shown).

Example 9-79 Output from showing event logs

```
EVENTS FOR DATASTORE InfoSphere CDC Oracle Redo 6.5:
0 events displayed.

SINGLE SCRAPE EVENTS FOR DATASTORE InfoSphere CDC Oracle Redo 6.5:
143324|20110415|2917|Information|com.datamirror.ts.util.oracle.OracleRedoNativeApi|IB
M InfoSphere Change Data Capture daemon has reported an informational message. Redo
log scraping started at position '1855797.79260.332.16786944' timestamp 'FriApr 15
14:32:40 2011'.
```

```
143324|20110415|2917|Information|com.datamirror.ts.util.oracle.OracleRedoNativeApi|IB
M InfoSphere Change Data Capture daemon has reported an informational message.
Started on-line redo log file '/oradata/cdcdemo/redo03.log'. Redo log processing has
been initiated on the on-line file '/oradata/cdcdemo/redo03.log'. The current
sequence is 37. The low scn is 1811661. The low timestamp is Thu Apr 14 22:19:35
2011. The next scn is -. The next timestamp is -.
143324|20110415|2917|Information|com.datamirror.ts.util.oracle.OracleRedoNativeApi|IB
M InfoSphere Change Data Capture daemon has reported an informational message. New
scrape point specified by redo log position '1855797.0.0.0'. The previously recorded
redo log position is '0.0.0.0'. The corresponding previous redo timestamp is 'Thu Jan
1 01:00:00 1970'. A user command has specified the new starting redo log position
'1855797.0.0.0'.
3 events displayed.

SOURCE EVENTS FOR SUBSCRIPTION REDSCRIPT:
143324|20110415|2922|Information|com.datamirror.ts.scrapers.singlescrape.SingleScrape
Thread|Subscription REDSCRIPT has started using the single scrape staging store.
143320|20110415|44|Information|com.datamirror.ts.source.replication.MirrorModerator|M
irroring has been initiated for table CDCDEMO.PRODUCT.
143320|20110415|44|Information|com.datamirror.ts.source.replication.MirrorModerator|M
irroring has been initiated for table CDCDEMO.DEMO_CUSTOMER.
143320|20110415|1437|Information|com.datamirror.ts.source.replication.MirrorModerator
|Table CDCDEMO.PRODUCT refresh to REDSCRIPT is complete. 231 rows were sent.
143320|20110415|225|Information|com.datamirror.ts.source.replication.MirrorModerator|
Table CDCDEMO.PRODUCT refresh to REDSCRIPT has been confirmed by the target system.
231 rows were received, 231 rows were successfully applied, 0 rows failed.
143318|20110415|9703|Information|com.datamirror.ts.source.tablereader.TableReader|Met
hod used for Refresh on source: JDBC
143318|20110415|223|Information|com.datamirror.ts.source.replication.MirrorModerator|
Table CDCDEMO.PRODUCT will be refreshed to REDSCRIPT .
…
26 events displayed.

TARGET EVENTS FOR SUBSCRIPTION REDSCRIPT:
143329|20110415|6673|Information|com.datamirror.ts.target.publication.TargetMirrorApp
lyJob|IBM InfoSphere Change Data Capture will commit on source transaction
boundaries.
143320|20110415|227|Information|com.datamirror.ts.target.publication.TargetRefreshApp
lyJob|Refresh was completed for table CDCDEMO.PRODUCT_TARGET. 231 rows were received,
231 rows were successfully applied, 0 rows failed.
143319|20110415|226|Information|com.datamirror.ts.target.publication.TargetRefreshHan
dler|Refresh was started for table CDCDEMO.PRODUCT_TARGET.
143319|20110415|322|Information|com.datamirror.ts.target.apply.udb.fastload.UDBFastlo
adApply|The table CDCDEMO.PRODUCT_TARGET was cleared.
```

```
143318|20110415|6693|Information|com.datamirror.ts.target.publication.TargetRefreshAp
plyJob|DB2 Fastload apply mode is used for refresh.
143318|20110415|227|Information|com.datamirror.ts.target.publication.TargetRefreshApp
lyJob|Refresh was completed for table CDCDEMO.DEMO_CUSTOMER_TARGET. 228,117 rows were
received, 228,117 rows were successfully applied, 0 rows failed.
143236|20110415|226|Information|com.datamirror.ts.target.publication.TargetRefreshHan
dler|Refresh was started for table CDCDEMO.DEMO_CUSTOMER_TARGET.
143235|20110415|322|Information|com.datamirror.ts.target.apply.udb.fastload.UDBFastlo
adApply|The table CDCDEMO.DEMO_CUSTOMER_TARGET was cleared.
143234|20110415|6693|Information|com.datamirror.ts.target.publication.TargetRefreshAp
plyJob|DB2 Fastload apply mode is used for refresh.
…
54 events displayed.

EVENTS FOR DATASTORE InfoSphere CDC DB2 6.5:
0 events displayed.
```

9.6 Monitoring and integration with external monitoring solutions

The previous sections described configuring InfoSphere CDC so that it is optimal for your environment. The sections demonstrated the usage of several special implementation topologies, such as large distributions and consolidations, and optimizing InfoSphere CDC for high volumes.

For many clients, InfoSphere CDC is a business critical application or a component of another business process upon which users depend. After you have set up your environment and deployed the configuration in your production environment, ensure that the replication functions as expected, that transactions are delivered to the intended destinations, and that the latency meets your requirements.

The Management Console offers functionality to configure, operate, and monitor your replication environment and can show an integral overview of your entire replication landscape. The Management Console has a InfoSphere CDC focus and is interactive, being a graphical user interface, and does not lend itself to integration with systems or business monitoring solutions.

For example, you could have chosen InfoSphere CDC to do the transportation of orders from your website to your order processing application, and at the same time provide feedback about current stock levels to your web customers. You expect that orders entered on the web are instantly made available to the back-end application to be processed for order picking and shipment and that there is maximum latency for the delivery of the order data (for example, less than 5 minutes). At the same time, you want to provide your customers with stock information that is up to date in real time so that they know that certain goods are readily available for shipment.

This section describes the various InfoSphere CDC components that can be monitored and provides ideas about how this monitoring can be integrated with your systems and business monitoring solution.

9.6.1 Components to monitor

The key InfoSphere CDC items you want to monitor are the replication latency of any subscriptions going from the web server to the order processing application (placed orders) and the subscriptions in the reverse direction (stock information).

There are several methods for monitoring the latency of the replication, which are described in Chapter 8, “Performance analysis and design considerations” on page 211. If the lag time between generation of transactions on the source and arrival on the target is too long, there could be several reaons for this lag. Some reasons include the subscription being slow due to congestion on the intermediate network or even the target InfoSphere CDC instance being down, rendering the subscriptions inactive as a result.

9.6.2 InfoSphere CDC instance activity

Subscription activity is directly dependent on the InfoSphere CDC source and target instances running well. Should either instance not be active, subscriptions cannot be kept active and stop. On the server itself, the activity of the InfoSphere CDC instance can be monitored by interrogating the operating system processes. If you want to consolidate the monitoring of InfoSphere CDC and other processes to a single location, use the Java API to ping or connect to the data store to determine its activity. If the data stores cannot be reached, you can then notify the systems monitor of this condition.

Examples for checking the InfoSphere CDC instance activity from the server have been provided in 9.4.5, “Checking an InfoSphere CDC engine and subscriptions activity” on page 246. If you want to implement the instance activity checking using the Java API, see 9.5.4, “Connecting to the data stores” on page 277.

9.6.3 Subscription activity

Starting with InfoSphere CDC V6.5 (and earlier releases for InfoSphere CDC for System z or DB2), subscriptions can be marked persistent, meaning that replication initiates a normal shutdown under certain circumstances, such as interruptions in the network communications. InfoSphere CDC attempts to automatically restart continuous mirroring for persistent subscriptions at regular intervals. This persistence keeps the replication environment active at almost all times and the need for continuously checking the replication activity might no longer be a strong requirement. There might still be reasons why subscriptions stop, such as apply failures due to data inconsistency or even an operator error.

Subscription activity must be monitored on the source server of the subscription. The source InfoSphere CDC engine provides the only valid entry point for checking this activity. Even though it is possible on some operating systems to check the activity of InfoSphere CDC target processes, do not use this information as the only source for determining activity. You can assume that if the subscription is reported active by the InfoSphere CDC source engine that the end-to-end process is running fine. If there is a condition that causes the target replication processes to terminate, the InfoSphere CDC heartbeat feature ensures that the source process also stops, and vice versa.

In some environments, customers have a mix of subscriptions that should be active at all times, and other subscriptions that are only started occasionally (for example, for refreshing certain tables on a daily basis). When you design your activity monitoring, implement a naming convention that allows the monitoring process to distinguish between these classes of subscriptions. If you consider using the Java API to monitor subscription activity, you could mark the subscription activity service levels in the description of the subscription and analyze it through the API.

Examples for checking subscription activity from the server have been provided in 9.4.5, “Checking an InfoSphere CDC engine and subscriptions activity” on page 246. If you want to implement the subscription activity checking using the Java API, see 9.5.17, “Starting, stopping, and monitoring subscriptions” on page 339.

9.6.4 Events

InfoSphere CDC distinguishes the areas where events can occur and are logged in the data store event logs (source and target), subscription event logs (source and target), or single scrape event log (at the source side; it is available for the InfoSphere CDC V6.5 Linux, UNIX, and Windows engine only).

If, for example, the replication stops because of a duplicate key in the target table, this error is logged in the target subscription event log. As a result of stopping the subscription in an abnormal fashion, a number of errors are also logged in the source subscription event log.

InfoSphere CDC offers a number of possibilities to monitor the events that are logged by the InfoSphere CDC engines:

- Show the individual event logs from the Management Console Monitoring perspective.
- Use the Management Console to export the event logs in a text or comma-separated values file (events in all areas can be brought together).
- Collect the events on the source and target servers running InfoSphere CDC, using either the InfoSphere CDC engine or operating system commands.
- Configure notifications for the InfoSphere CDC engines or subscriptions to conditional action events that occur and forward these events to email (available for the Linux, UNIX, and Windows engine), message queues (System i), SYSLOG (System z), CHCPRINT spooled member (System z), or start a notification user exit program (all engines).
- Create a custom program employing the Java API to gather the event logs from source and target engines / subscriptions.

If you want to integrate the InfoSphere CDC events with a systems monitoring solution (such as IBM Tivoli Monitoring), the collection of event logs from the Management Console is not suitable. When gathering events on the servers running InfoSphere CDC, the source and target subscription events must be collected on the source and target server. To determine the cause of a replication failure, you typically need to explore both source and target events to make a correct assessment of what exactly went wrong.

Event logs can be collected on source and target InfoSphere CDC servers using the techniques described in “Monitoring the event logs” on page 250. Your external monitoring solution should then be configured to collect the event log output or monitor the objects on these servers (use the output of **`dmshowevents`** for Linux, UNIX, and Windows engine, a message queue object for System i, and SYSLOG for System z), and take act accordingly.

If your external monitoring solution does not support all the InfoSphere CDC platforms or if you have specific requirements for feeding the monitoring solution, the Java API and writing notification user exits provide the most flexible options. Using the API, you can retrieve the event logs of all InfoSphere CDC engines that have been registered using the techniques described in 9.5.19, "Monitoring event logs using the API" on page 359. Alternatively, if you want the InfoSphere CDC engine to send notifications to your external monitoring solution, you could use the notifications user exits described in 9.7.7, "Notifications" on page 425.

9.6.5 Latency

If subscriptions have a backlog of transactions to process, the subscriptions are latent. InfoSphere CDC provides functionality in the Management Console to view the latency of a subscription and monitor it over time. Many customer environments have critical business processes that depend on the timeliness of the data that is transported and applied through InfoSphere CDC, so these items can be an important one to monitor.

For example, because of production systems overloading, you might decide that reporting is primarily done from a reporting server. Business users make their decisions based on the information that they obtain through the reports, which creates a dependency on the accuracy and timeliness of the data replicated to this reporting server.

Management Console allows you to set latency notification thresholds (in number of minutes) for subscriptions, specifying a threshold for warning and one for error. When the latency of the subscription exceeds the configured latency, the replication process starts sending events to the target subscription event log (the target side is where latency is measured).

Monitoring the latency then becomes a matter of monitoring the event log, which is described in "Monitoring the event logs" on page 250.

9.7 User exits

InfoSphere CDC lets you define subroutines that are starts when a predefined event occurs. You can use these user exits to customize the replication and related functionality to fit your business requirements.

Among other uses, user exits can be starts before or after database operations are being applied to the target or to calculate a column value to be assigned to a column on the target side.

This section describes the most commonly used user exit points and provides guidelines about how to implement them in the InfoSphere CDC environment. Additionally, a number of examples are provided that help you design your own user exit programs.

9.7.1 Common uses for user exits

The most common use case for InfoSphere CDC user exit programs is execution of additional actions on the apply side of the replication process. Your subscriptions replicate changes from a source to a target database and, based on the type of operation, the user exit program is starts to perform custom routines.

For example, in an application integration scenario, you could exchange the data from your source application with your target application using database tables. Then, as application transactions are applied to the target database, you might want to notify the receiving application that there is a transaction to be processed by using a message on a queue. The target application monitors the queue for new incoming messages, and when a message arrives, picks up the transaction details from the tables that have been populated by the subscription application process.

As an alternative, the process could build an XML document based on the incoming operations and then post this document on an Enterprise Service Bus when the transaction is committed.

Another frequent use of custom code is for row filtering. Assume that you want to filter rows based on a lookup in a master table that only has a few rows. InfoSphere CDC can use a %GETCOL function in the row filtering to retrieve the values of this master table for every operation that was read from the log. If the filtering is then defined for a table that has high volumes, this situation could potentially lead to an additional impact on the source. One of the solutions to reduce the workload on the source is to write a user exit program that reads the entries from the master table into a cache when it is called, and then use the cached entries from that moment onwards.

Row filtering and user exits are also sometimes used for scaling the replication if the target database can only handle a certain volume throughput of operations in a single database session. By using a row filter on the key columns, you can then share the work for this single table across multiple subscriptions, and then initiate multiple database sessions on the target side and increase the throughput. Using a derived expression, a user exit provides additional flexibility when designing the row filter. For example, if the replicated table has a numeric key column, a user exit could calculate the modulo value of this numeric key with a certain divisor and allow you to use the remainder in your row selection.

Coding and using the modulo function is described in 9.7.3, “Derived expression user exits” on page 373.

9.7.2 User exit programs

Keep your code for the user exit as efficient as possible. When you start a user-written program at a defined exit point, it is important to realize that a call is issued each time a clear, insert, update, or delete operation is applied to a target table. Therefore, when data replication activity is high, overall throughput and performance impact is affected by the actions that are implemented through the code in the user-written programs.

Java user exit programs

When developing Java user exit programs to interface with InfoSphere CDC, you should be aware that a specific environment must be configured. Although most Java Development Kits at Version 1.6 or later can be used to compile classes that use the JAR files that come with the InfoSphere CDC engines, use the IBM JDK for your development environment to be synchronized with the required running environment. All user exit programs that are called from an InfoSphere CDC V6.5 engine run using IBM Java Runtime Engine V1.6.

The class path for compiling your self-developed classes must include the `ts.jar` file, which can be found in the `lib` directory of the InfoSphere CDC engine product. Consider copying the `ts.jar` file to your development environment if your development environment does not have direct access to the directories that contain the InfoSphere CDC product.

ts.jar file: Keep in mind that the `ts.jar` file integrates with the version of InfoSphere CDC that is running in your environment. Should a new version of InfoSphere CDC be installed, it might be necessary to replace the copy of the `ts.jar` file with the latest version.

To let the InfoSphere CDC engine discover the user exit classes, the classes must be placed in the class path of the engine. A good practice is to store the custom classes in the `cdc_home/lib` directory. If you change the user exit programs, you must restart the InfoSphere CDC instance to ensure that the JVM picks up the new version of the class.

System i user exit programs

System i user exit programs for InfoSphere CDC can be written in any original program model (OPM) or integrated language environment (ILE) compliant language, such as RPG, COBOL, C, or CL.

In order for the InfoSphere CDC subscription jobs to find your user exit programs, the programs must be in the library list of the jobs running the InfoSphere CDC subscription. This library could be the product library or any library that is in the library list of the DMCJOBD job description. To avoid overwriting your custom programs when installing new releases of InfoSphere CDC, place the programs in a separate library.

When you design your user exit programs, remember that loading a program into memory for execution and opening tables (files) are expensive operations. If you intend to use a row-level user exit program to apply changes to a different table, ensure that the program does not open and close files on entry and exit.

Here are some considerations when coding and compiling your user exit programs:

- In RPG, do not set the Last Record (LR) indicator when returning from your program, as this setting causes any open files to be closed and the program removed from memory.
- Compile ILE user exit programs with DFTACTGRP(*NO) ACTGRP(*CALLER) so that your user exit program uses the same activation group as InfoSphere CDC and does not have to reinitialize itself on every call.

More information about user exits for InfoSphere CDC on System i can be found at the following address:

`http://publib.boulder.ibm.com/infocenter/cdc/v6r3m3/topic/com.ibm.cdcdoc.cdcfori.doc/concepts/userexitsfortransformationserver.html`

System z user exit programs

Programs can be written in any high-level language (including C, COBOL, and Assembler) that supports reentrant coding techniques and standard OS linkage, and uses the z/OS IBM Language Environment®. The user exit programs must be compiled and link-edited to use AMODE 31 (31-bit addressing mode).

The link-edited load modules that represent the user exit must be made accessible to InfoSphere CDC using a `CHCUXLIB DD` statement in the execution JCL. This load library must not be APF authorized. User exit programs must be successfully link-edited before proceeding to configure them in the Management Console.

For more information about z/OS user exit programs, go to the following address:

`http://publib.boulder.ibm.com/infocenter/cdc/v6r5m1/topic/com.ibm.cdcdoc.cdcforzos.doc/concepts/infospherechangedatacaptureuserexits.html`

Stored procedure user exits

Besides running user exits using a programming language that is callable from the InfoSphere CDC replication engine, you can also configure stored procedures to be called based on predefined events. Stored procedures are compiled programs that are physically stored in a database and, when called, are run by the database engine. This action could provide a handle for delivering additional scalability when you must run complex operations or calculations.

A typical use of a stored procedure user exit is when you want to replicate the contents of a single table to two target tables. In a standard InfoSphere CDC configuration, this action requires you to create two subscriptions, each mapping the same table but having a different destination table. As subscriptions run independently of each other, the transaction consistency between the target tables can no longer be guaranteed. If you call a stored procedure to operate on the second table (or both tables), information in the target tables is still transaction consistent.

When used in a table-level or row-level exit point, the stored procedures are run in line with the operations that InfoSphere CDC applies into the target tables. Both share a database connection, which ensures that when a stored procedure user exit is called from an after-operation exit point, the changes to the table that were made by InfoSphere CDC are visible to the stored procedure user exit program.

You can retrieve the columns that were replicated from the source table, journal control columns, and system values by specifying parameters in your stored procedure user exit program. InfoSphere CDC analyzes these parameters and passes the values when it calls the stored procedure.

9.7.3 Derived expression user exits

This section describes derived expression user exits.

Java user exit for derived expression

The UEModuloFilter65 class is an implementation of the InfoSphere CDC derived expression interface and can be called using the %USERFUNC("JAVA") function from within InfoSphere CDC, either on the source or the target. In Example 9-80, the user function is suitable for performing row filtering at the source.

Example 9-80 Modulo function derived expression user exit

```
import java.math.BigDecimal;

import com.datamirror.ts.derivedexpressionmanager.DEUserExitIF;
import com.datamirror.ts.derivedexpressionmanager.UserExitInvalidArgumentException;
import com.datamirror.ts.derivedexpressionmanager.UserExitInvokeException;
import com.datamirror.ts.util.Trace;

/*
 Overview:
 The user exit performs a modulo operation against a specified column and
 determine if it equals the specified remainder value. The parameters passed
 into the user exit are:
 a) Dividend  -> Numeric value to perform the modulo function against
 b) Divisor   -> Numeric value by which the Dividend will be divided
 c) Remainder -> Remainder value to be tested

 The comparison logic is as follows:

 <Dividend> % <Divisor> == <Remainder>

 Instructions:

 1) Copy the UEModuloFilter65.class file to the <cdc install>/lib directory
 2) Go to the mapping details for the table
 3) Under Filtering, specify the following

 %USERFUNC("JAVA","UEModuloFilter65",<Dividend column>, <Divisor value>,
           <Remainder>)

 Return value is boolean, true or false.
 */

public class UEModuloFilter65 implements DEUserExitIF {

   public Object starts(Object[] aobjList)
         throws UserExitInvalidArgumentException, UserExitInvokeException {
      try {
         long dividendColumn = ((BigDecimal) aobjList[0]).longValue();
         long divisorValue = ((BigDecimal) aobjList[1]).longValue();
         long remainderValue = ((BigDecimal) aobjList[2]).longValue();
         return new Boolean(
               (dividendColumn % divisorValue) == remainderValue);
      } catch (ClassCastException e) {
         // Piggyback on the CDC logging facility
         Trace.traceAlways(e);
         throw new UserExitInvalidArgumentException(
               "Invalid number parameter passed "
                     + "to the user function, arguments passed: [0]="
```

```
                    + aobjList[0] + ", [1]=" + aobjList[1] + ", [2]="
                    + aobjList[2] + ", Message: " + e.getMessage());
        }
    }
}
```

When specified in the row filter for a replicated table, InfoSphere CDC calls the **`starts()`** method of the class for every log entry of that table. Should invalid numerics be passed as parameters, the exception is caught and the InfoSphere CDC tracing facility starts. The exception that is then thrown is logged under the `<cdc_home>/instance/<instance>/log` directory in the current `trace_dmts*` file.

You can also use the logging facility if you are in the process of developing your user exit and you want to debug it.

For example, suppose that you have a high volume table that has an integer primary key column (KEYCOLUMN). You want to divide the workload of replicating this table across three subscriptions to have multiple database sessions apply the transactions on this table in parallel. You create three subscriptions, with each subscription mapping the same table. However, in the row filter condition, you specify different values, as shown in the following list:

- `RF_SUB1: %USERFUNC("JAVA","UEModuloFilter65",KEYCOLUMN,3,0)`
- `RF_SUB2: %USERFUNC("JAVA","UEModuloFilter65",KEYCOLUMN,3,1)`
- `RF_SUB3: %USERFUNC("JAVA","UEModuloFilter65",KEYCOLUMN,3,2)`

When processing the operations for this table, each subscription does the calculation of KEYCOLUMN%3 and determines if the remainder value is the same as the remainder value specified as the last parameter. The first subscription replicates all rows where KEYCOLUMN%3 can be divided by 3. The second subscription replicates the rows where there is a remainder value of 1. The third second subscription replicates the rows where there is a remainder value of 2. If the least significant digit of the KEYCOLUMN values is distributed evenly, the three subscriptions complement each other and take one-third of the workload.

Figure 9-29 defines a row filter according to the outcome of the modulo function.

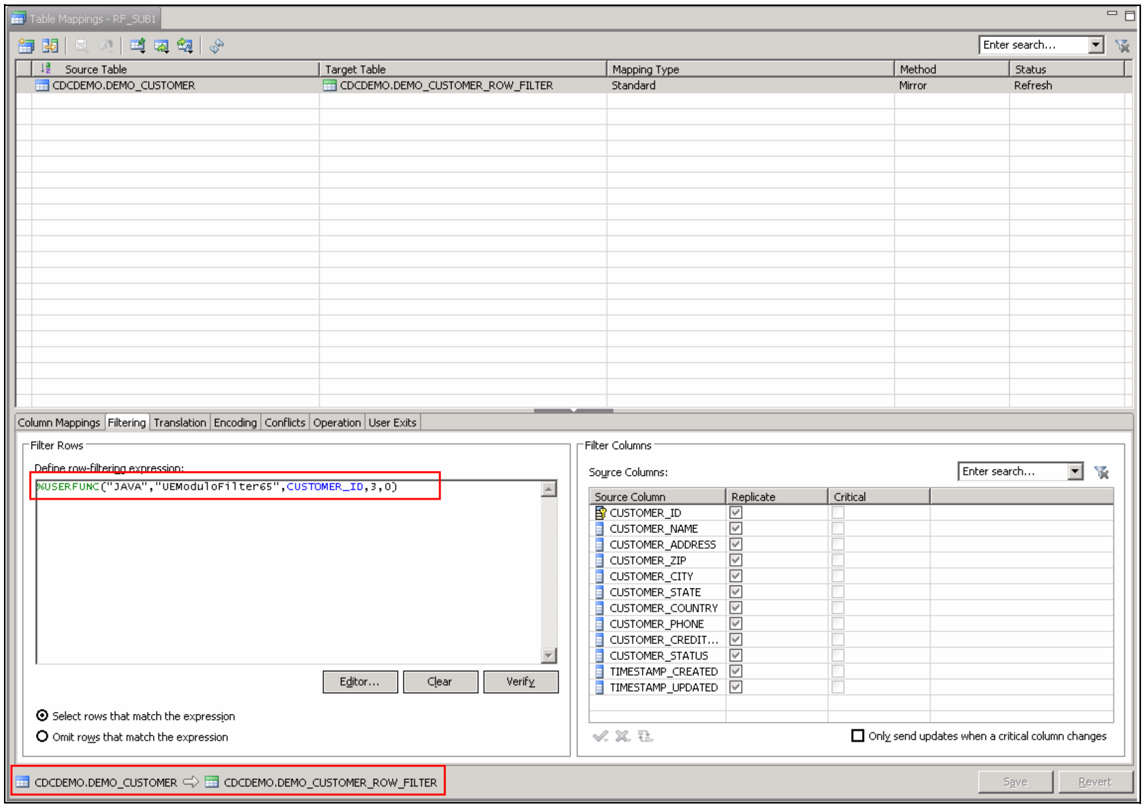


Table Mappings - RF_SUB1

Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.DEMO_CUSTOMER	CDCDEMO.DEMO_CUSTOMER_ROW_FILTER	Standard	Mirror	Refresh

Column Mappings | Filtering | Translation | Encoding | Conflicts | Operation | User Exits

Filter Rows

Define row-filtering expression:

```
%USERFUNC("JAVA","UEModuloFilter65",CUSTOMER_ID,3,0)
```

[✓ Select rows that match the expression]

Omit rows that match the expression

Filter Columns

Source Columns:

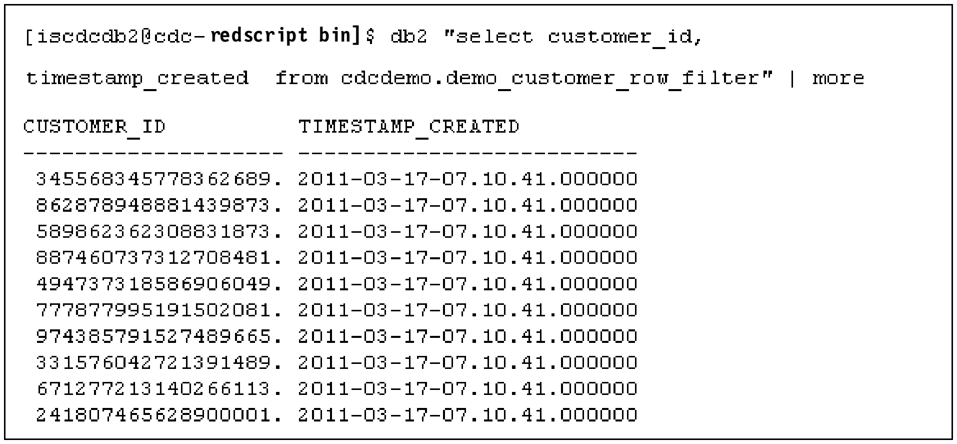
Source Column	Replicate	Critical
CUSTOMER_ID	✓	
CUSTOMER_NAME	✓	
CUSTOMER_ADDRESS	✓	
CUSTOMER_ZIP	✓	
CUSTOMER_CITY	✓	
CUSTOMER_STATE	✓	
CUSTOMER_COUNTRY	✓	
CUSTOMER_PHONE	✓	
CUSTOMER_CREDIT...	✓	
CUSTOMER_STATUS	✓	
TIMESTAMP_CREATED	✓	
TIMESTAMP_UPDATED	✓	

☐ Only send updates when a critical column changes

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_ROW_FILTER

Figure 9-29 InfoSphere CDC user exit DE-ModuloFilter

After the subscription has finished refreshing the table, the target table appears as shown in Figure 9-30. All the CUSTOMER_IDs shown in the output are divisible by 3 (CUSTOMER_ID%3==0).



```
[iscdcdb2@cdc-redscript bin]$ db2 "select customer_id,
timestamp_created  from cdcdemo.demo_customer_row_filter" | more

CUSTOMER_ID            TIMESTAMP_CREATED
--------------------- --------------------------
 3455683457783 62689. 2011-03-17-07.10.41.000000
 862878948881439873. 2011-03-17-07.10.41.000000
 589862362308831873. 2011-03-17-07.10.41.000000
 887460737312708481. 2011-03-17-07.10.41.000000
 494737318586906049. 2011-03-17-07.10.41.000000
 777877995191502081. 2011-03-17-07.10.41.000000
 974385791527489665. 2011-03-17-07.10.41.000000
 331576042721391489. 2011-03-17-07.10.41.000000
 671277213140266113. 2011-03-17-07.10.41.000000
 241807465628900001. 2011-03-17-07.10.41.000000
```

Figure 9-30 InfoSphere CDC user exit DE-ModuloFilter output

Another example is a derived expression user exit to calculate the week of the year. For completeness, we provide the source for that class in Example 9-81.

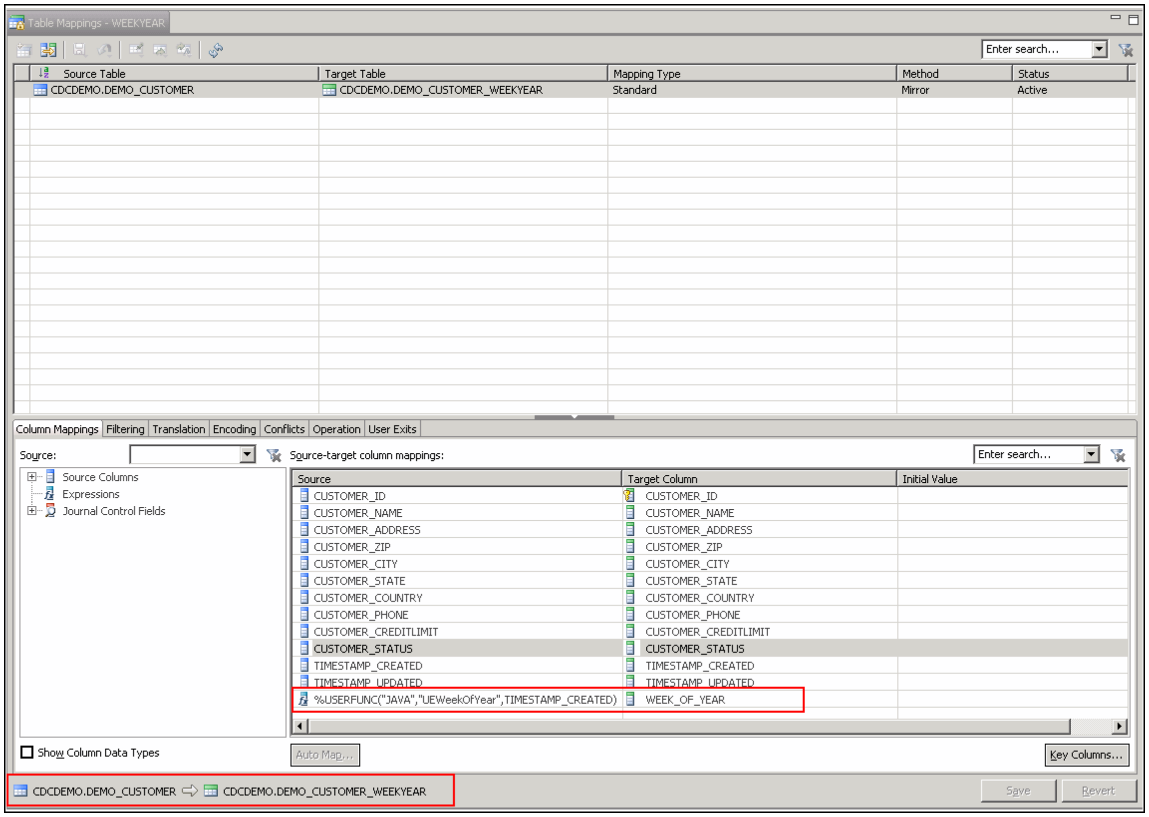
Example 9-81 Week of the year calculation derived expression user exit

```
import com.datamirror.ts.derivedexpressionmanager.DEUserExitIF;
import
com.datamirror.ts.derivedexpressionmanager.UserExitInvalidArgumentException;
import com.datamirror.ts.derivedexpressionmanager.UserExitInvokeException;
import com.datamirror.ts.util.Trace;

import java.text.SimpleDateFormat;

/*
 Overview:
 The user exit returns the week of the year for a date. Week number is returned
 as an integer. The parameters passed into the user exit are:
 a) Date -> Date value for which the week number must be calculated

 Instructions:

 1) Copy the UEWeekOfYear.class to the <cdc install>/lib directory
 2) Go to the mapping details for the table
 3) For the column that must contain the week number, specify the following:

 %USERFUNC("JAVA", "UEWeekOfYear", <date_column>)
 */
```

```
public class UEWeekOfYear implements DEUserExitIF {

   public Object starts(Object[] aobjList)
         throws UserExitInvalidArgumentException, UserExitInvokeException {
      SimpleDateFormat outWeek = new SimpleDateFormat("ww");
      String strWeek = outWeek.format(aobjList[0]);
      int intWeek = Integer.parseInt(strWeek);

      try {
         return new Integer(intWeek);
      } catch (ClassCastException e) {
         Trace.traceAlways(e);
         throw new UserExitInvalidArgumentException(
               "Invalid date parameter passed to " + "the user exit: "
                     + aobjList[0]);
      }
   }
}
```

You could use this function as a derived expression in the column mapping so that it is run on the target side. In this case, the **`starts()`** method expects to be passed a date or time stamp value and returns the number of the week as defined by the SimpleDateFormat class. The example class does not cater to null values in the passed date value. If you write a similar function for a column that contains a null value, you would need to plan for this situation in your code.

When you configure a table mapping using this function, it appears as shown in Figure 9-31.



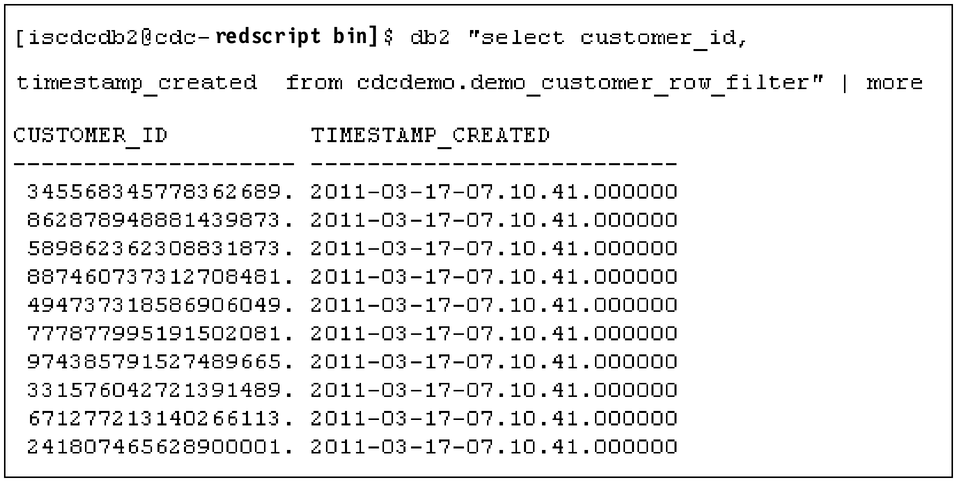
Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.DEMO_CUSTOMER	CDCDEMO.DEMO_CUSTOMER_WEEKYEAR	Standard	Mirror	Active

Source	Target Column	Initial Value
CUSTOMER_ID	CUSTOMER_ID	
CUSTOMER_NAME	CUSTOMER_NAME	
CUSTOMER_ADDRESS	CUSTOMER_ADDRESS	
CUSTOMER_ZIP	CUSTOMER_ZIP	
CUSTOMER_CITY	CUSTOMER_CITY	
CUSTOMER_STATE	CUSTOMER_STATE	
CUSTOMER_COUNTRY	CUSTOMER_COUNTRY	
CUSTOMER_PHONE	CUSTOMER_PHONE	
CUSTOMER_CREDITLIMIT	CUSTOMER_CREDITLIMIT	
CUSTOMER_STATUS	CUSTOMER_STATUS	
TIMESTAMP_CREATED	TIMESTAMP_CREATED	
TIMESTAMP_UPDATED	TIMESTAMP_UPDATED	
%USERFUNC("JAVA","UEWeekOfYear",TIMESTAMP_CREATED)	WEEK_OF_YEAR	

CDCDEMO.DEMO_CUSTOMER ⇨ CDCDEMO.DEMO_CUSTOMER_WEEKYEAR

Figure 9-31 InfoSphere CDC user exit for week of the year

When you look at the target table, the function has generated the values shown in Figure 9-32.



```
[iscdcdb2@cdc-redscript bin]$ db2 "select customer_id,
timestamp_created  from cdcdemo.demo_customer_row_filter" | more

CUSTOMER_ID          TIMESTAMP_CREATED
-------------------- --------------------------
 345568345778362689. 2011-03-17-07.10.41.000000
 862878948881439873. 2011-03-17-07.10.41.000000
 589862362308831873. 2011-03-17-07.10.41.000000
 887460737312708481. 2011-03-17-07.10.41.000000
 494737318586906049. 2011-03-17-07.10.41.000000
 777877995191502081. 2011-03-17-07.10.41.000000
 974385791527489665. 2011-03-17-07.10.41.000000
 331576042721391489. 2011-03-17-07.10.41.000000
 671277213140266113. 2011-03-17-07.10.41.000000
 241807465628900001. 2011-03-17-07.10.41.000000
```

Figure 9-32 InfoSphere CDC user exit week of the year output

System i user exit for derived expressions

When defining a user exit program for System i, create an ILE program and have it called in the same activation group of InfoSphere CDC. The sample program, shown in Example 9-82, uses two input parameters. The first one is packed (5,0) and the second is packed (7,0). It is important that the two parameters that are passed are indeed packed values of the specified length and precision. If you try to pass a zoned number or any other type, the exit program might fail with a decimal data error.

Example 9-82 Sample user exit for derived expressions

```
* =========================================================== *
 * Program name: TDRVFLD                                       *
 *                                                             *
 * Synopsis    : This InfoSphere CDC for System i user exit program splits *
 *               out the input parameter(s) and returns the sum *
 *               of the passed values.                         *
 *                                                             *
 * Parameters  : The %USER function can have 1 output parameter *
 *               and a variable number of input parameters.    *
 *               For each input parameter, a data structure    *
 *               must be created and referred to in the        *
 *               entry parameter list.                         *
 *                                                             *
 * Create remarks: CRTBNDRPG DFTACTGRP(*NO) ACTGRP(*CALLER)     *
 *                 CHGPGM USEADPAUT(*NO)                       *
 *                                                             *
 * ----------------------------------------------------------- *
 *                   Changes made to this source               *
 *                                                             *
 * Date     Who Description                                    *
 * -------- --- ---------------------------------------------- *
 * 20051018 FK  Initial delivery
 * =========================================================== *
HDATFMT(*ISO)
 * ----------------------------------------------------------- *
 * File definitions                                            *
 * ----------------------------------------------------------- *
 * ----------------------------------------------------------- *
 * Arrays and tables                                           *
 * ----------------------------------------------------------- *
 * ----------------------------------------------------------- *
 * Data structures and field definitions                       *
 * ----------------------------------------------------------- *
 * Output parameter
DpOut            DS           256
D OutType                       4b 0                                  * Data type
D OutLen                        4b 0                                  * Length
D OutDigits                     4b 0                                  * Digits
D OutDec                        4b 0                                  * Decimal positions
D OutNull                       4b 0                                  * Null indicator
D OutDatFmt                     4                                     * Date format
D OutValue                      9p 0                                  * Parameter value
 * Input parameter 1 - Value 1
DpIn01           DS           256
D I01Type                       4b 0                                  * Data type
D I01Len                        4b 0                                  * Length
D I01Digits                     4b 0                                  * Digits
D I01Dec                        4b 0                                  * Decimal positions
D I01Null                       4b 0                                  * Null indicator
D I01DatFmt                     4                                     * Date format
D I01Value                      5p 0                                  * Parameter value
 * Input parameter 2 - Value 2
DpIn02           DS           256
D I02Type                       4b 0                                  * Data type
D I02Len                        4b 0                                  * Length
D I02Digits                     4b 0                                  * Digits
D I02Dec                        4b 0                                  * Decimal positions
D I02Null                       4b 0                                  * Null indicator
D I02DatFmt                     4                                     * Date format
D I02Value                      7p 0                                  * Parameter value
 * ----------------------------------------------------------- *
 * Constants                                                   *
 * ----------------------------------------------------------- *
 * Data type of value to be returned
```

```
D#TypChar        C                   CONST(1)                             * Character
D#TypDate        C                   CONST(2)                             * Date
D#TypFloat       C                   CONST(3)                             * Floating point
D#TypInt         C                   CONST(4)                             * Integer
D#TypPacked      C                   CONST(5)                             * Packed
D#TypTime        C                   CONST(6)                             * Time
D#TypZoned       C                   CONST(7)                             * Zoned decimal
 * ---------------------------------------------------------------- *
 * Key lists                                                        *
 * ---------------------------------------------------------------- *
 * ---------------------------------------------------------------- *
 * Parameter lists                                                  *
 * ---------------------------------------------------------------- *
C     *Entry        PList
C                   Parm                    pOut                      * Output parameter
C                   Parm                    pIn01                     * Input parameter 01
C                   Parm                    pIn02                     * Input parameter 02
 *                  ....                    ......                    * Input parameter 03
 * ---------------------------------------------------------------- *
 * Main line                                                        *
 * ---------------------------------------------------------------- *
 * Do user actions ...
 * Prepare output parameter
C                   Eval      OutType = #TypPacked                    * Packed
C                   Eval      OutLen = 5                              * Packed length
C                   Eval      OutDigits = 9                           * Digits
C                   Eval      OutDec = 0                              * Decimal places
C                   Eval      OutNull = 0                             * Null indicator
C                   Eval      OutDatFmt = *Blanks                     * Date format
C                   Eval      OutValue = I01Value * I02Value          * Calculate value
C                   Return
 * ---------------------------------------------------------------- *
 * *INZSR - Initialisation subroutine                               *
 * ---------------------------------------------------------------- *
C     *INZSR        BegSR
C                   EndSR
```

Derived expression user exits always return a value to InfoSphere CDC that is then used for row filtering or to populate a target column. Which type of return value is passed back to InfoSphere CDC is determined by completing the OutType data structure field with the correct data type constant value listed under the Constants section.

9.7.4 Table and row-level user exits

Important: User exit programs and stored procedures must not use COMMIT or ROLLBACK SQL statements; let InfoSphere CDC to manage the units of work. If you commit or roll back transactions in the InfoSphere CDC database session, you interfere with its bookmark processing, which could result in duplicated or missed transactions when restarting after failure.

Java user exit for row-level user exits

The UESoftDelete user exit class implements the UserExitIF interface and must be specified in the User Exits tab under the table mapping. Also, the before-delete exit point check box must be checked so that the user exit is starts (checking any of the check boxes also starts the user exit).

This section clarifies a couple of points that help you design your own user exits. During the initialization (using the **`init()`** method), the user exit evaluates the parameters that were passed, if any. An optimization has been built into the user exit to force it to update only a few columns in the row. This action optimizes the SQL statement and execution on the target.

Also, the user exit explicitly unsubscribes from all possible events and then subscribes to BEFORE_DELETE_EVENT. By specifying this event in the code, you prevent the situation occurring where the user exit is called on an exit point that it cannot handle. The SoftDelete user exit must only be used in the event of a delete operation, hence the registration for this event. The sample code for a soft delete user exit is shown in Example 9-83.

Example 9-83 Sample soft delete user exit

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.util.ArrayList;

// InfoSphere CDC specific imports
import com.datamirror.ts.target.publication.userexit.DataTypeConversionException;
import com.datamirror.ts.target.publication.userexit.ReplicationEventIF;
import com.datamirror.ts.target.publication.userexit.DataRecordIF;
import com.datamirror.ts.target.publication.userexit.ReplicationEventPublisherIF;
import com.datamirror.ts.target.publication.userexit.ReplicationEventTypes;
import com.datamirror.ts.target.publication.userexit.UserExitException;
import com.datamirror.ts.target.publication.userexit.UserExitIF;

/*
```

```
 Overview:
 This soft delete user exit will mark a row as inactive instead of performing a
physical
 delete. If the target row does not exist, a row will be inserted into the target
table.

 Assumptions:
 - Delete operations have been disabled for this table.
 - The &ENTTYP journal control field has been used to determine the value of the
   "deleted" column
 - Table has been configured for Adaptive Apply or Conflict Detection/Resolution
   (Source wins)
 - Table does not possess LOB/LONG columns.

 Instructions
 1. Copy the class files to the <cdc install directory>/lib directory
 2. In Mapping Details->Operation, set the "On Delete" setting to "Do Not Delete".
 3. In Mapping Details->User Exits,
 Set "User Exit Type" to "Java Class"
 Set "Class Name" to "UESoftDelete"  (omit quotes)
 In the "Events and Actions" section, tick the "Delete Before" checkbox

 NOTE:
 By default, when the user exit updates the target row, it will update all
 target columns. For an optimized update operation you can specify the columns
 you want updated in the "Parameter" field. Example: Set "Parameter" to
 "ENTRY_TYPE,AUD_TIME,SRC_AUD_TIME"  (omit quotes)
 */
public class UESoftDelete implements UserExitIF {

   protected static final int[] eventsToSubscribe =
      { ReplicationEventTypes.BEFORE_DELETE_EVENT };

   protected Boolean firstTime = true;
   protected UETrace trace;
   protected Connection dbConnection; // Connection to the target database

   private String qualifiedTableName;
   private String updateStatementString;
   private String[] colNameList = null;

   private ArrayList<Integer> keyColIndexes;
   private ArrayList<Integer> updateColIndexes;

   private PreparedStatement updateStatement;
   private PreparedStatement insertStatement;

/**
 * Executes table level initialization
 *
 * @param publisher
 *            - Handle to engine environment information
 */
   public void init(ReplicationEventPublisherIF publisher)
         throws UserExitException {
// Tracing is always switched on, can be disabled or parameterized
      trace = new UETrace();
      trace.init(true);

// Process parameter (list of columns that will be updated on soft
// delete)
      if (publisher.getParameter() != null) {
         colNameList = publisher.getParameter().split(",");
      }
      if (colNameList[0].equalsIgnoreCase("") || colNameList[0] == null) {
         colNameList = null;
         trace.write(this.getClass().getName()
               + ": Parameter list for UESoftDelete is empty, all columns "
               + "will be updated.");
      } else {
         trace.write(this.getClass().getName()
               + ": Columns that will be updated: "
               + publisher.getParameter());
      }

// For the user exit program to subscribe only to the events which it
// can handle (before-delete).
// Even if the configuration in MC specifies that the user exit will be
// called on other
// exit points, the subscribeEvent overrides this.
      publisher.unsubscribeEvent(ReplicationEventTypes.ALL_EVENTS);
      for (int i = 0; i < eventsToSubscribe.length; i++) {
         publisher.subscribeEvent(eventsToSubscribe[i]);
      }
   }

/**
 * Method: processReplicationEvent -> Table event processing Purpose:
 * handles the change events at the table level
 *
 * @param event
 *            - Handle to the change record
 */
   public boolean processReplicationEvent(ReplicationEventIF event)
         throws UserExitException {
      DataRecordIF image = event.getBeforeData();
      if (firstTime) {
         /*
          * This is the first time the user exit is being starts for this
          * table mapping. Initialize all the SQL statements that will be
          * executed
          */
         qualifiedTableName = event.getTablePath() + "."
               + event.getTableName();
         if (dbConnection == null) {
// Get the shared JDBC database connection with the target
// database (CDC 6.5)
            dbConnection = event.getSharedConnection();
         }
         getUpdateColumnIndexes(image);
         getKeyList(image);
         prepareUpdateStatement(image);
         firstTime = false;
      }

      /*
       * When the event is BEFORE-DELETE, update the row
       */
      switch (event.getEventType()) {
      case ReplicationEventTypes.BEFORE_DELETE_EVENT:
         performUpdate(image);
      default:
      }

      return true;
   }

/**
 * Retrieves the indexes for the key columns of the target tables. This
 * information is used to build the WHERE clause of UPDATE statements.
 *
 * @param image
 *            - The row image
 */
   private boolean getKeyList(DataRecordIF image) throws UserExitException {
      keyColIndexes = new ArrayList<Integer>();

      for (int i = 1; i <= image.getColumnCount(); i++) {
         if (image.isKey(i)) {
            keyColIndexes.add(i);
         }
      }
      return true;
   }

/**
 * Retrieves the indexes for columns that need to be updated
 *
 * @param image
 *            - The row image
 */
   private boolean getUpdateColumnIndexes(DataRecordIF image)
         throws UserExitException {
      updateColIndexes = new ArrayList<Integer>();

// Implements the functionality that you can specify the columns to be
// updated
      if (colNameList != null && colNameList.length != 0) {
         // User has specified specific columns they want updated
         for (int j = 0; j < colNameList.length; j++) {
            for (int i = 1; i <= image.getColumnCount(); i++) {
               if (image.getColumnName(i).equalsIgnoreCase(colNameList[j])) {
                  updateColIndexes.add(i);
                  break;
               }
            }
         }
      } else {
         // Add all columns for update if no parameters are specified
         for (int i = 1; i <= image.getColumnCount(); i++) {
            updateColIndexes.add(i);
         }
      }
      return true;
   }

/**
 * Prepares the UPDATE statement that will be used to mark a row as
 * "deleted". This method is the optimized one where none of the key fields
 * are NULL. All key values can be bound to the key fields in the WHERE
 * statement and the user exit does not have to rebuild the UPDATE statement
 * every time.
 *
 * @param image
 *            - The row image
 */
   private void prepareUpdateStatement(DataRecordIF image)
         throws UserExitException {
      updateStatementString = "UPDATE " + qualifiedTableName + " SET ";
// build the SET clause
      for (int i = 0; i < updateColIndexes.size(); i++) {
         if (i > 0) {
            updateStatementString += ", ";
         }
         updateStatementString += "\""
               + image.getColumnName(updateColIndexes.get(i)) + "\" = ? ";
      }
// Build the WHERE clause
      updateStatementString += " WHERE ";
      for (int i = 0; i < keyColIndexes.size(); i++) {
         if (i > 0) {
            updateStatementString += " AND ";
         }
         updateStatementString += "\""
               + image.getColumnName(keyColIndexes.get(i)) + "\""
               + " = ? ";
      }
      try {
         updateStatement = dbConnection
               .prepareStatement(updateStatementString);
      } catch (SQLException e) {
         e.printStackTrace();
         throw new UserExitException(e.getMessage());
      }

      trace.write(updateStatementString);
      return;
   }

/**
 * Prepares the UPDATE statement if columns in the WHERE clause are NULL. In
 * that case SQL requires that you specify "<column> IS NULL" instead of
 * "<column>=?". As the user exit cannot know in advance which key values
 * are null, the UPDATE statement is dynamically built for every update that
 * has to occur.
 *
 * @param image
 *            - The row image
 */
   private void dynamicUpdate(DataRecordIF image) throws UserExitException {
      String dynamicUpdateStatementString = "UPDATE " + qualifiedTableName
            + " SET ";
      PreparedStatement dynamicStatement;
// Build the SET clause
      for (int i = 0; i < updateColIndexes.size(); i++) {
         if (i > 0) {
            dynamicUpdateStatementString += ", ";
         }
         dynamicUpdateStatementString += "\""
               + image.getColumnName(updateColIndexes.get(i)) + "\" = ? ";
      }
// Build the WHERE clause
      dynamicUpdateStatementString += " WHERE ";
      ArrayList<Integer> keyIndexes = new ArrayList<Integer>();
      int keyCount = 0;
      for (int i = 1; i <= image.getColumnCount(); i++) {
         if (image.isKey(i)) {
            if (keyCount > 0) {
               dynamicUpdateStatementString += " AND ";
            }
// Specify the column name <column>=? or <column> IS NULL
            if (!image.isNull(i)) {
               dynamicUpdateStatementString += "\""
                     + image.getColumnName(i) + "\"" + " = ? ";
               keyIndexes.add(i);
            } else {
               dynamicUpdateStatementString += "\""
                     + image.getColumnName(i) + "\"" + " IS NULL ";
            }
         }
      }
// Prepare the dynamic update statement
      try {
         dynamicStatement = dbConnection
               .prepareStatement(dynamicUpdateStatementString);
      } catch (SQLException e) {
         e.printStackTrace();
         throw new UserExitException(e.getMessage());
      }
```

```
      int parameterCount = 1;
      for (int i = 0; i < updateColIndexes.size(); i++) {
         try {
            dynamicStatement.setObject(parameterCount, image
                  .getObject(updateColIndexes.get(i)), dynamicStatement
                  .getParameterMetaData()
                  .getParameterType(parameterCount));
            parameterCount++;
         } catch (Exception e) {
            throw new UserExitException(e.getMessage());
         }
      }

      for (int i = 0; i < keyIndexes.size(); i++) {
         try {
            dynamicStatement.setObject(parameterCount,
                  image.getObject(keyIndexes.get(i)));
            parameterCount++;
         } catch (Exception e) {
            throw new UserExitException(e.getMessage());
         }
      }
// Update the row
      try {
         dynamicStatement.executeUpdate();
      } catch (SQLException e) {
         e.printStackTrace();
         throw new UserExitException(e.getMessage());
      }
// Close the statement
      try {
         dynamicStatement.close();
      } catch (SQLException e) {
         e.printStackTrace();
         throw new UserExitException(e.getMessage());
      }
      return;
   }

/**
 * Performs the actual soft delete of the row (update with the passed
 * values).
 *
 * @param image
 *            - The row image
 * @throws UserExitException
 */
   private void performUpdate(DataRecordIF image) throws UserExitException {
      trace.write(this.getClass().getName() + ": Soft deleting row");
      int parameterCount = 1;

      for (int i = 0; i < updateColIndexes.size(); i++) {
         try {
            updateStatement.setObject(parameterCount, image
                  .getObject(updateColIndexes.get(i)), updateStatement
                  .getParameterMetaData()
                  .getParameterType(parameterCount));
            parameterCount++;
         } catch (Exception e) {
            throw new UserExitException(e.getMessage());
         }
      }
// Prepare the static update statement or starts the dynamic update
// statement
      for (int i = 0; i < keyColIndexes.size(); i++) {
         try {
            if (image.getObject(keyColIndexes.get(i)) != null) {
               updateStatement.setObject(parameterCount,
                     image.getObject(keyColIndexes.get(i)));
               parameterCount++;
            } else {
               dynamicUpdate(image);
               return;
            }
         } catch (SQLException e) {
            e.printStackTrace();
            throw new UserExitException(e.getMessage());
         } catch (DataTypeConversionException e) {
            e.printStackTrace();
            throw new UserExitException(e.getMessage());
         }
      }
// Execute the prepared (static) statement
      try {
         trace.write("Executing Update");
         if (updateStatement.executeUpdate() == 0) {
            return;
         }
      } catch (SQLException e) {
         e.printStackTrace();
         throw new UserExitException(e.getMessage());
      }
      return;
   }

/**
 * Performs any required cleanup processing, starts on subscription
 * shutdown. Also starts on table-level user exit
 */
   public void finish() {
      // perform any table specific cleanup
      if (updateStatement != null) {
         try {
            updateStatement.close();
         } catch (SQLException e) {
            e.printStackTrace();
         }
      }
      return;
   }
}
```

The **`processReplicationEvent()`** method is starts for every event to which the user exit is subscribed. During first time processing, the method obtains a shared connection to the target database from the InfoSphere CDC engine so that it can use the same session InfoSphere CDC already uses to update the target tables. Additionally, it obtains the list of columns that must be populated in the SET clause of the UPDATE statement and the list of key columns for the WHERE clause.

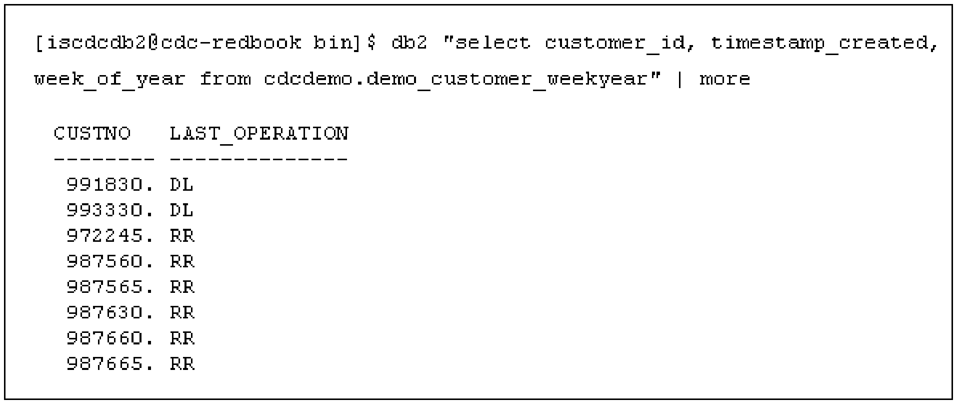
Eventually, when the before-delete event is detected and the **`processReplicationEvent()`** method is called, the method does an update of the existing row instead of an update. There are two versions of the update method, one static, which uses a pre-built PreparedStatement object, and the other one dynamic, which builds the UPDATE statement on every execution. Dynamic building of the prepared statement is less efficient than using a pre-built statement. However, if there are NULL values in the key columns, SQL requires that the WHERE clause includes the IS NULL for the column. As it is impossible to know in advance which key column is null, if at all, the UPDATE statement must be built for every execution. If there are no NULLable key columns, the pre-built PreparedStatement is always used.

In various methods in the UESoftDelete class, trace records are written to the InfoSphere CDC trace files, which are kept in the `<cdc_home>/instance/<instance>/log` directory. This class uses the com.datamirror.ts.util.Trace class to combine the InfoSphere CDC tracing. The sample code for this tracing utility is shown in Example 9-84.

Example 9-84 Sample tracing utility for user exits

```
import com.datamirror.ts.util.Trace;

/**
 * Tracing facility for user exit
 */
public class UETrace {
   boolean enabled = false;

/**
 * Initializes the tracing facility
 */
   public void init(boolean enabled) {
      this.enabled = enabled;
   }

/**
 * Writes a trace message
 *
 * @param message - Messag to write to the trace
 */
   public void write(String message) {
      if (enabled) {
         // Piggyback on the InfoSphere CDC logging facility
         Trace.traceAlways(message);
      }
      return;
   }

/**
 * Cleanup for trace facility -> not used in this implementation
 */
   public void close() {
   }
}
```

After refreshing the table and removing some of the rows on the source, the target table would look something like Figure 9-33.



```
[iscdcdb2@cdc-redbook bin]$ db2 "select customer_id, timestamp_created,
week_of_year from cdcdemo.demo_customer_weekyear" | more

  CUSTNO   LAST_OPERATION
  -------- --------------
   991830. DL
   993330. DL
   972245. RR
   987560. RR
   987565. RR
   987630. RR
   987660. RR
   987665. RR
```

Figure 9-33 Sample soft delete output

Stored procedure user exit for row-level operations

Stored procedure row-level user exits, shown in Example 9-85, are starts just before or after InfoSphere CDC applies the operation to the target table. If you specify a stored procedure user exit and disable the operation that it is attached to, the user exit is still starts and can therefore be run instead of the operation being applied by InfoSphere CDC.

Example 9-85 Sample table for row-level user exit

```
CREATE TABLE CDCDEMO.RISKY_CUSTOMER (
   CUSTNO DECIMAL(6,0), CRLIMIT DECIMAL(7,0), UPD_TIMESTAMP TIMESTAMP)
   IN USERSPACE1
```

Example 9-83 on page 383 adds a row to the CDCDEMO.RISKY_CUSTOMER table, but only if the credit limit of the customer has exceeded 10,000. The inserted row logs the customer number, credit limit, and time stamp of update at the source.

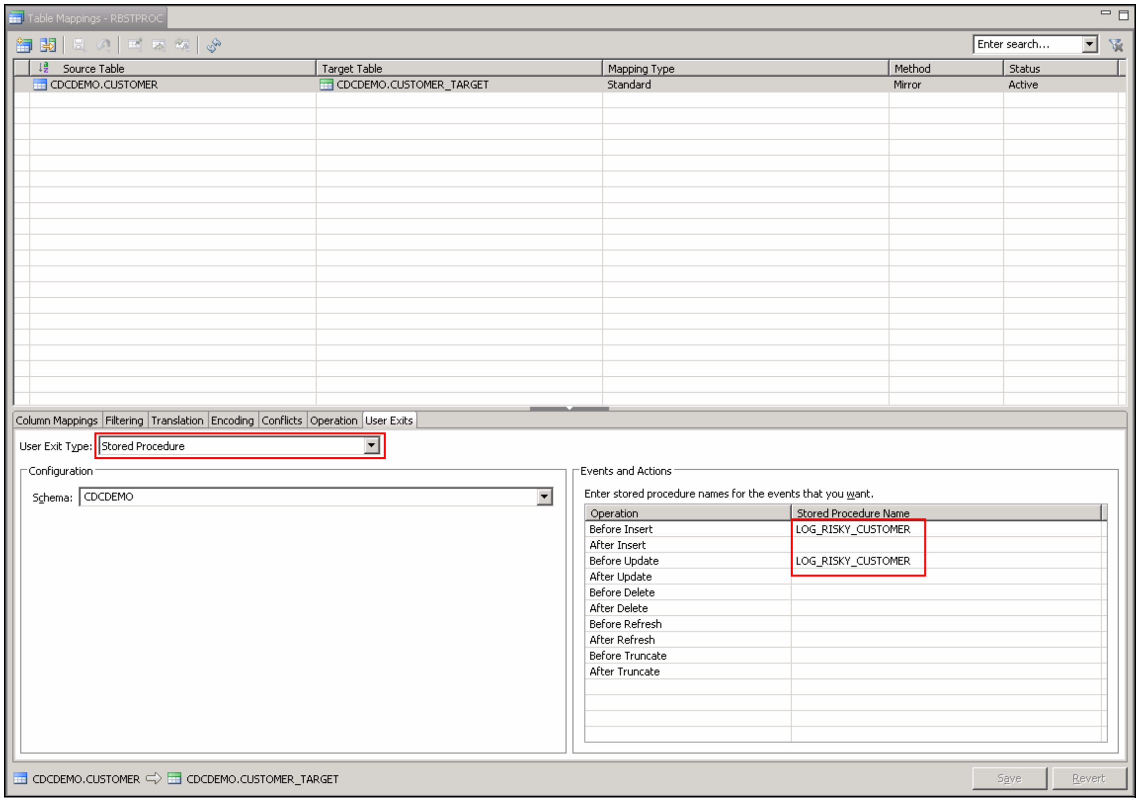
The stored procedure is defined as shown in Example 9-86.

Example 9-86 Sample stored procedure

```
CREATE OR REPLACE PROCEDURE CDCDEMO.LOG_RISKY_CUSTOMER (
OUT result INT, OUT returnMsg CHAR,
IN a$CUSTNO DECIMAL(6,0), IN a$CRLIMIT DECIMAL(7,0), IN j$TIMSTAMP
VARCHAR(26))
BEGIN
if a$CRLIMIT>=10000 then
```

```
   insert into CDCDEMO.RISKY_CUSTOMER values(a$CUSTNO,
   a$CRLIMIT, TIMESTAMP(j$TIMSTAMP));
end if;
set result=0;
set returnMsg='Row inserted';
END@
```

Defining the stored procedure to be called means specifying the schema where it is located and the name of the stored procedure at the exit points that you want it to be (Figure 9-34).



Source Table	Target Table	Mapping Type	Method	Status
CDCDEMO.CUSTOMER	CDCDEMO.CUSTOMER_TARGET	Standard	Mirror	Active

User Exit Type: Stored Procedure

Configuration

Schema: CDCDEMO

Events and Actions

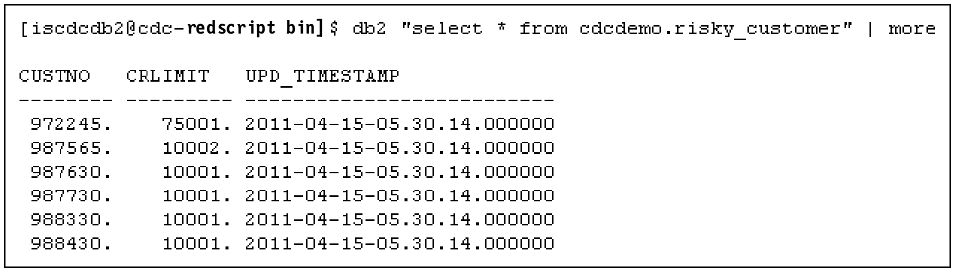
Enter stored procedure names for the events that you want.

Operation	Stored Procedure Name
Before Insert	LOG_RISKY_CUSTOMER
After Insert	
Before Update	LOG_RISKY_CUSTOMER
After Update	
Before Delete	
After Delete	
Before Refresh	
After Refresh	
Before Truncate	
After Truncate	

CDCDEMO.CUSTOMER ⇨ CDCDEMO.CUSTOMER_TARGET

Figure 9-34 User exit stored procedure configuration

When running the subscription and changing a number of source rows that have or attain a credit limit of 10,000 or higher, the stored procedure starts populating the RISKY_CUSTOMER table (Figure 9-35).



```
[iscdcdb2@cdc-redscript bin]$ db2 "select * from cdcdemo.risky_customer" | more

CUSTNO   CRLIMIT   UPD_TIMESTAMP
-------- --------- --------------------------
 972245.    75001. 2011-04-15-05.30.14.000000
 987565.    10002. 2011-04-15-05.30.14.000000
 987630.    10001. 2011-04-15-05.30.14.000000
 987730.    10001. 2011-04-15-05.30.14.000000
 988330.    10001. 2011-04-15-05.30.14.000000
 988430.    10001. 2011-04-15-05.30.14.000000
```

Figure 9-35 Sample row stored procedure output

System i RPG user exit for row-level operations

Row-level and table-level user exits for InfoSphere CDC on System i are handled using a single interface in which the apply process passes a fixed set of arguments to the user exit program, such as the before and after data images, the publisher ID, information about the operation type (entry type), and journal control columns. The data images are sent to the program as fixed-length strings in the format of the target table and the user exit program is responsible for unraveling the individual columns. The easiest way to perform this task depends on the language in which you have developed your user exit. RPG allows you to create an external data structure that maps the flat input to individual fields.

The example user exit must be customized per source and target table for which you want to soft delete rows. However, you can change the program so that a single program accommodates all possible source and target tables. The before image for the delete is mapped onto a source table structure using the BfrImg data structure, and the journal control columns are also populated. By interrogating the journal control column for the file name, you can determine which mapping must take place.

The TGTCUST target table is opened when the program is called for the first time, and because the Last Record indicator is not set on, the file remains opened and the program active. If you want to use one program to handle soft deletes on multiple target tables, dynamically open the files, as soft delete operations must be applied to limit the amount of resources taken by the InfoSphere CDC apply process.

The sample code for performing soft deletes in a table is shown in Example 9-87.

Example 9-87 Sample code for soft deletes in a table

```
 * ============================================================================================ *
 * Program name: TROWSFTDLT *
 *                                                              *
 * Synopsis    : This user exit program is an example of how to handle soft deletes in a table. *
 *                                                              *
 *              We assume that the subscription replicates the CUSTOMER table to the TGTCUST table on
 *              the target system. Inserts and Updates are defined as normal operation. However, for the*
 *              Delete operation "No Action" is specified and this program is specified as the Before
 *              Delete user exit program.                               *
 *                                                              *
 *              In effect, the program checks the row operation it was starts for. If it was for    *
 *              a delete, the target table's fields TXTYP and TXDTS are updated with journal control *
 *              columns &ENTTYP and &TIMSTAMP                   *
 *                                                              *
 * Create parms: CRTBNDRPG DFTACTGRP(*NO) ACTGRP(*CALLER)       *
 * After create: CHGPGM USEADPAUT(*NO)                          *
 *                                                              *
 * ------------------------------------------------------------ *
 *                                                              *
 * ------------------------------------------------------------ *
 *                   Changes made to this source                *
 *                                                              *
 * Date     Who Description                                     *
 * -------- --- ------------------------------------------------ *
 * 20061122 FK  Initial delivery
 * ============================================================ *
HDATFMT(*ISO) DFTACTGRP(*NO) ACTGRP(*CALLER)
 * ------------------------------------------------------------ *
 * File definitions                                             *
 * ------------------------------------------------------------ *
 * Customer file on the target database, equipped with
 * Transaction type and Transaction timestamp
FTGTCUST   UF   E           K DISK
 * ------------------------------------------------------------ *
 * Arrays and tables                                            *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ *
 * Data structures and field definitions                        *
 * ------------------------------------------------------------ *
 * Binary field definitions
DEntTypB         DS            4
D EntTyp                1      4B 0
DBfrNbrNullB     DS            4
D BfrNbrNull            1      4B 0
DAftNbrNullB     DS            4
D AftNbrNull            1      4B 0
DTSBufLenB       DS            4
D TSBufLen              1      4B 0
DTSNbrNullB      DS            4
D TSNbrNull             1      4B 0
```

```
 * Before Image for record (mapped to external structure for original table, CUSTOMER)
DBfrImg         E DS                 ExtName(CUSTOMER) Prefix(Bfr)
 * After Image for record (mapped to external structure for
 *                          original table, CUSTOMER)
DAftImg         E DS                 ExtName(CUSTOMER) Prefix(Aft)
 * Before Image for journal control fields
DBfrJrnCtl        DS
D JBEntLen                      5                                       * Entry length
D JBSeqNbr                     10                                       * J/E Sequence #
D JBJrnNam                     10                                       * Journal name
D JBJrnLib                     10                                       * Journal library
D JBRcvNam                     10                                       * Receiver name
D JBRcvLib                     10                                       * Receiver library
D JBJrnCde                      1                                       * Journal code
D JBEntTyp                      2                                       * Entry type
D JBSysEnt                      2                                       * System J/E
D JBTimStp                     26                                       * Time stamp
D JBTimStpDt                   10    OVERLAY(JBTimStp:1)                * Time stamp date
D JBTimStpHr                    2    OVERLAY(JBTimStp:12)               $ Time stamp hour
D JBJobNam                     10                                       * Job name
D JBJobUsr                     10                                       * Job user
D JBJobNbr                      6                                       * Job number
D JBPgmNam                     10                                       * Program name
D JBFilNam                     10                                       * File name
D JBFilLib                     10                                       * File library
D JBFilMbr                     10                                       * File member
D JBRRN                        10                                       * Relative record
D JBFlg                         1                                       * Flag 1 or 0
D JBCmtID                      10                                       * Commit cycle
D JBUsrPrf                     10                                       * User profile
D JBSysNam                      8                                       * System name
 * After Image for journal control fields
DAftJrnCtl        DS
D JAEntLen                      5                                       * Entry length
D JASeqNbr                     10                                       * J/E Sequence #
D JAJrnNam                     10                                       * Journal name
D JAJrnLib                     10                                       * Journal library
D JARcvNam                     10                                       * Receiver name
D JARcvLib                     10                                       * Receiver library
D JAJrnCde                      1                                       * Journal code
D JAEntTyp                      2                                       * Entry type
D JASysEnt                      2                                       * System J/E
D JATimStp                     26                                       * Time stamp
D JATimStpDt                   10    OVERLAY(JATimStp:1)                * Time stamp date
D JATimStpHr                    2    OVERLAY(JATimStp:12)               $ Time stamp hour
D JAJobNam                     10                                       * Job name
D JAJobUsr                     10                                       * Job user
D JAJobNbr                      6                                       * Job number
D JAPgmNam                     10                                       * Program name
D JAFilNam                     10                                       * File name
D JAFilLib                     10                                       * File library
D JAFilMbr                     10                                       * File member
D JARRN                        10                                       * Relative record
D JAFlg                         1                                       * Flag 1 or 0
```

```
D JACmtID                      10                                         * Commit cycle
D JAUsrPrf                     10                                         * User profile
D JASysNam                      8                                         * System name
 * ------------------------------------------------------------ *
 * Constants                                                    *
 * ------------------------------------------------------------ *
D#BfrClr          C                   CONST(1)
D#AftClr          C                   CONST(2)
D#BfrIns          C                   CONST(3)
D#AftIns          C                   CONST(4)
D#BfrUpd          C                   CONST(5)
D#AftUpd          C                   CONST(6)
D#BfrDlt          C                   CONST(7)
D#AftDlt          C                   CONST(8)
D#BfrRsh          C                   CONST(9)
D#AftRsh          C                   CONST(10)
D#None            C                   CONST('*N')
 * ------------------------------------------------------------ *
 * Key lists                                                    *
 * ------------------------------------------------------------ *
 * Key list target customer table
C     KeyCus        KList
C                   KFld                    BfrCUSTNO                     * Customer number
 * ------------------------------------------------------------ *
 * Parameter lists                                              *
 * ------------------------------------------------------------ *
C     *ENTRY        Plist
C                   Parm                    RtnCde           10           * Return code
C                   Parm                    PgmName          10           * Program name
C                   Parm                    EntTypB                       * Entry type
C                   Parm                    BfrImg                        * Before Image
C                   Parm                    AftImg                        * After Image
C                   Parm                    BfrNbrNullB                   * # of Null in Bfr
C                   Parm                    AftNbrNullB                   * # of Null in Aft
C                   Parm                    BfrNull           1           * Null in Bfr
C                   Parm                    AftNull           1           * Null in Aft
C                   Parm                    BfrJrnCtl                     * Bfr Journal Ctl
C                   Parm                    AftJrnCtl                     * Aft Journal Ctl
C                   Parm                    TSBufLenB                     * TS Buffer length
C                   Parm                    TSBuf           158           * TS Buffer
C                   Parm                    TSNbrNullB                    * # of Null TS Buf
C                   Parm                    TSNull            5           * Null in TS Buf
C                   Parm                    TSSrcID           8           * TS Source system
 * ------------------------------------------------------------ *
 * Main line                                                    *
 * ------------------------------------------------------------ *
 * Determine type of transaction
C                   Select
 * Clear
C     EntTyp        WhenEQ    #BfrClr                                     * Before clear
C     EntTyp        OrEQ      #AftClr                                     * After clear
 * Insert
C     EntTyp        WhenEQ    #BfrIns                                     * Before clear
C     EntTyp        OrEQ      #AftIns                                     * After clear
```

```
 * Update
C     EntTyp        WhenEQ    #BfrUpd                                    * Before clear
C     EntTyp        OrEQ      #AftUpd                                    * After clear
 * Delete
C     EntTyp        WhenEQ    #BfrDlt                                    * Before clear
C     EntTyp        OrEQ      #AftDlt                                    * After clear
 * Find the record in the target table and update Transaction type/Timestamp
C     KeyCus        Chain     TGTCUSTR                          95
C     *In95         IfEQ      *Off
C                   Eval      TXTYP=JBEntTyp                             * &ENTTYP
C                   Eval      TXDTS=JBTimStp                             * &TIMSTAMP
C                   Update    TGTCUSTR
C                   EndIf
C                   EndSL
C                   Return
 * ------------------------------------------------------------ *
 * *INZSR - Initialisation subroutine                           *
 * ------------------------------------------------------------ *
C     *INZSR        BegSR
C                   EndSR
```

Configuration of this user exit program in the Management Console is shown in Figure 9-36. The table has been mapped for Adaptive Apply, which ensures that if a customer that has been soft deleted on the target is reinserted, the row on the target is overwritten. In the column mapping, the &ENTTYP journal control column is mapped to the TXTYP column. A soft deleted row on the target has DL in this column. Finally, the standard delete operation is disabled (Do Not Delete) and the Before Delete user exit is configured to call the TROWSFTDLT user exit program. As stated before, the user exit program must be found in the library list of the DMCJOBD job description or in the InfoSphere CDC installation library.

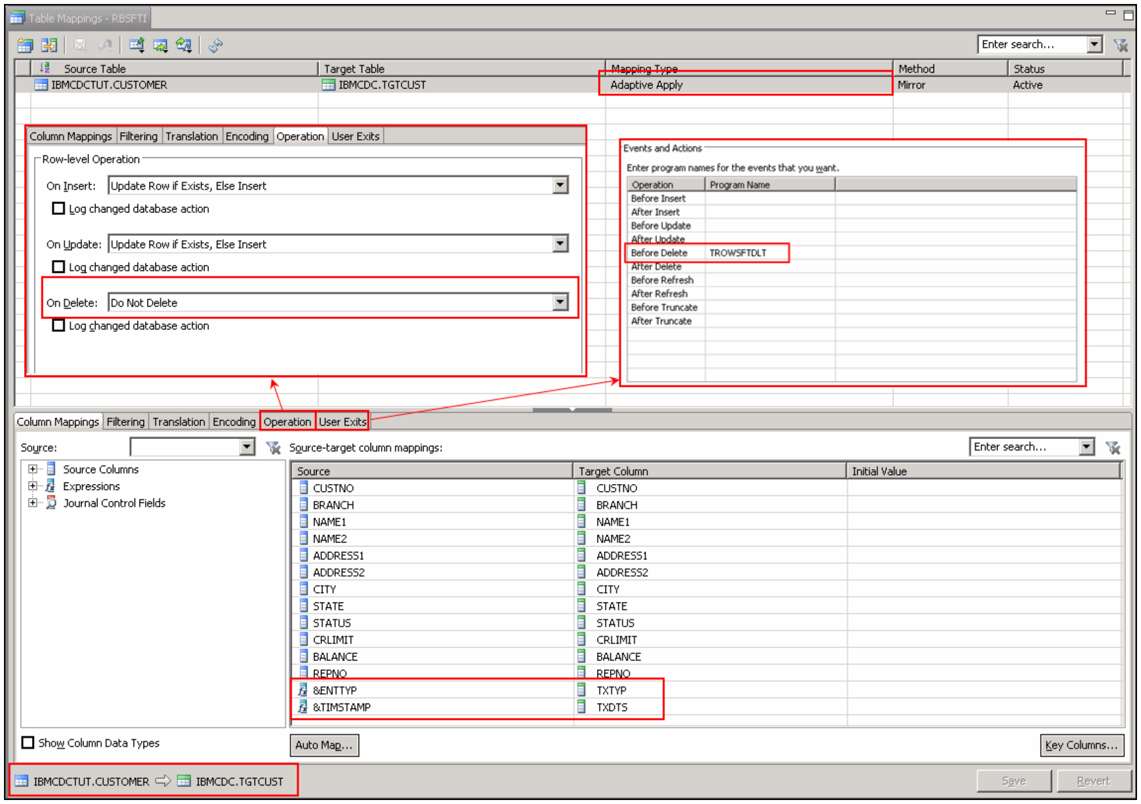


Table Mappings - RBSFTI

Source Table	Target Table	Mapping Type	Method	Status
IBMCDCTUT.CUSTOMER	IBMCDC.TGTCUST	Adaptive Apply	Mirror	Active

Column Mappings | Filtering | Translation | Encoding | Operation | User Exits

Row-level Operation

On Insert: Update Row if Exists, Else Insert
☐ Log changed database action

On Update: Update Row if Exists, Else Insert
☐ Log changed database action

On Delete: Do Not Delete
☐ Log changed database action

Events and Actions

Enter program names for the events that you want.

Operation	Program Name
Before Insert	
After Insert	
Before Update	
After Update	
Before Delete	TROWSFTDLT
After Delete	
Before Refresh	
After Refresh	
Before Truncate	
After Truncate	

Column Mappings | Filtering | Translation | Encoding | Operation | User Exits

Source:

Source-target column mappings:

Source Columns
Expressions
Journal Control Fields

Source	Target Column	Initial Value
CUSTNO	CUSTNO	
BRANCH	BRANCH	
NAME1	NAME1	
NAME2	NAME2	
ADDRESS1	ADDRESS1	
ADDRESS2	ADDRESS2	
CITY	CITY	
STATE	STATE	
STATUS	STATUS	
CRLIMIT	CRLIMIT	
BALANCE	BALANCE	
REPNO	REPNO	
&ENTTYP	TXTYP	
&TIMSTAMP	TXDTS	

☐ Show Column Data Types

Figure 9-36 User exit configuration

9.7.5 Subscription-level (unit of work)

In some cases, your implementation might require committed transactions that are delivered to a non-database target, such as a message queue or web service. Row-level user exits can enhance or replace InfoSphere CDC apply processing and run custom code based on insert, update, delete, or even table level events. The subscription-level user exits take your user exits a step further by letting you to ignite a process based on the unit of work that was read from the source database.

For example, your customers fill their shopping cart on your website. When the first item is added to the shopping cart, an order header row is created in the underlying database. As the customer adds items to the shopping cart, rows are inserted into the order detail table and, after the checkout process is started, your business application commits the transaction into the database and provides a unit of work.

Assume that you want to place the completed transaction as a single message on a message queue (or enterprise service bus). InfoSphere CDC row-level user exits allow you to pick up the individual database operations and run them, but you do not know when the last item has been placed in the shopping cart to complete the transaction.

The InfoSphere CDC subscription-level user exit point provides a method that is starts when the transaction is committed by the subscription. You can implement this method to start an action based on a transaction that was prepared in the row-level user exit points. Referring to the previous example, your row-level user exit points start building the message to be sent (in XML or other format). When the commit is started by InfoSphere CDC, the subscription-level user exit takes the message that was built, appends any closing tags in the case of XML, and places it onto a message queue.

In Example 9-88, the sample code builds an XML document (flat file) for each transaction sent from the source side.

Example 9-88 Sample code to build an XML document for each transaction

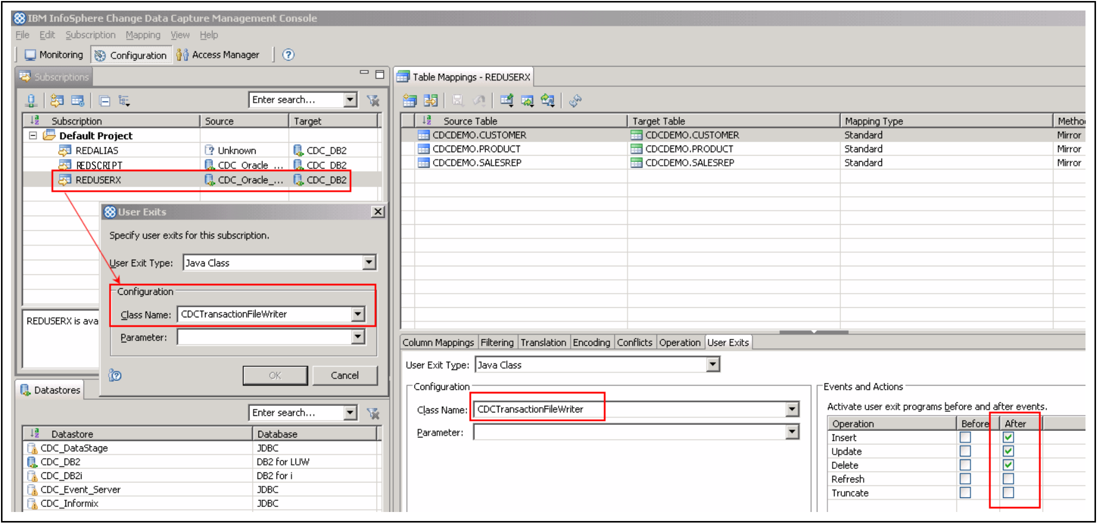
```
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.*;

import com.datamirror.ts.target.publication.userexit.*;

public class CDCTransactionFileWriter implements UserExitIF,
      SubscriptionUserExitIF {
```

```
static final String FILE_TIMESTAMP_FORMAT = "yyyyMMdd_HHmmssSSS";
private boolean calledAtSubscriptionLevel = false;
protected SubscriptionContext subscriptionContext; // shared between all
                                       // instances
private UETrace trace;
private String publisherID;

/**
 * Subscription-level initialization.
 *
 * This method is called once when the subscription is started and
 * initializes the subscription context. Also, it ensures that the
 * processSubscriptionEvent method is starts before every commit.
 *
 * @param publisher
 *            - Handle to the event publisher; this parameter can only be
 *            used during this method to subscribe to certain events.
 */
public void init(SubscriptionEventPublisherIF publisher)
      throws UserExitException {
   // Initialize the subscription context
   subscriptionContext = (SubscriptionContext) publisher
         .getUserExitSubscriptionContext();
   if (subscriptionContext == null) {
      // this is only called once during initialization
      // an object that calls getUserExitSubscriptionContext will be
      // passed this context we are creating
      subscriptionContext = createContext(publisher.getSourceSystemID());
      publisher.setUserExitSubscriptionContext(subscriptionContext);
   }
   calledAtSubscriptionLevel = true;
   trace = subscriptionContext.trace;
   publisherID = subscriptionContext.publisherID;
   trace.write("Subscription-level init() start");
   publisher.unsubscribeEvent(SubscriptionEventTypes.ALL_EVENTS);
   publisher.subscribeEvent(SubscriptionEventTypes.BEFORE_COMMIT_EVENT);
   trace.write("Subscription-level init() end");
}

/**
 * Executed when a subscription-level event is detected (commit). This
 * method writes the ending tag to the XML message and closes the current
 * output file.
 *
 * @param subscriptionEvent
 *            - handle to subscription event
 */
public boolean processSubscriptionEvent(
      SubscriptionEventIF subscriptionEvent) throws UserExitException {
   trace.write("processSubscriptionEvent() started");
   trace.write("Subscription event: "
         + getSubscriptionEventTypeAsString(subscriptionEvent
               .getEventType()));
   trace.write("Commit reason: "
         + getSubscriptionCommitReasonAsString(subscriptionEvent
               .getCommitReason()));
   closeOutputFile();
   return true;
}

/**
 * Table-level initialization.
 *
 * This method is called once for every mapped table at subscription
 * startup. It first retrieves the subscription context and then registers
 * the events it wants to listen to.
 *
 * @param eventPublisher
 *            - Handle to engine environment information
 */
public void init(ReplicationEventPublisherIF eventPublisher) {
   // Retrieve the subscription-level context
   subscriptionContext = (SubscriptionContext) eventPublisher
         .getUserExitSubscriptionContext();
   trace = subscriptionContext.trace;
   publisherID = subscriptionContext.publisherID;
   trace.write("Table-level init() start");

   // Subscribe to After-Insert/Update/Delete events
   eventPublisher.unsubscribeEvent(ReplicationEventTypes.ALL_EVENTS);
   eventPublisher.subscribeEvent(ReplicationEventTypes.AFTER_INSERT_EVENT);
   eventPublisher.subscribeEvent(ReplicationEventTypes.AFTER_UPDATE_EVENT);
   eventPublisher.subscribeEvent(ReplicationEventTypes.AFTER_DELETE_EVENT);

   trace.write("Table-level init() end");
}

/**
 * Executed when table-level event is detected (insert/update/delete). This
 * method writes an XML entry for the table-level operation to the currently
 * open output file. If there is no open file, it creates a new output file
 * on the fly.
 *
 * @param replicationEvent
 *            - Handle to replication event
 * @return true - This flag indicates whether the default operation should
 *         be applied (true) or not (false). As the event is only called
 *         after insert/update/delete, the returned value is of no
 *         importance
 */
public boolean processReplicationEvent(ReplicationEventIF replicationEvent)
      throws UserExitException {
   trace.write("processReplicationEvent() start");
   String tableName = replicationEvent.getTableName();
   String entryType = replicationEvent.getJournalHeader().getEntryType();
   DataRecordIF beforeImage = replicationEvent.getBeforeData();
   DataRecordIF afterImage = replicationEvent.getData();
   trace.write("Table: " + tableName);
   trace.write("Operation type: " + entryType);
   // If there is no open file, create one on the fly to write the XML
   // records to
   if (subscriptionContext.printStream == null) {
      createNewOutputFile();
   }
   // Write the table-level information
   subscriptionContext.printStream.println("\t<table" + tableName + ">");
   subscriptionContext.printStream.println("\t\t<tableOperation>"
         + entryType + "</tableOperation>");
   // Write column-level information for the before-image (update + delete)
   if (beforeImage != null) {
      for (int i = 1; i <= beforeImage.getColumnCount(); i++) {
         try {
            subscriptionContext.printStream.println("\t\t<before"
                  + beforeImage.getColumnName(i)
                  + ">"
                  + formatXmlContent(beforeImage.getObject(i)
                        .toString()) + "</before"
                  + beforeImage.getColumnName(i) + ">");
         } catch (DataTypeConversionException e) {
            trace.write(e.getMessage());
         }
      }
   }
   // Write column-level information for the after-image (insert+update)
   if (afterImage != null) {
      for (int i = 1; i <= afterImage.getColumnCount(); i++) {
         try {
            subscriptionContext.printStream.println("\t\t<after"
                  + afterImage.getColumnName(i)
                  + ">"
                  + formatXmlContent(afterImage.getObject(i)
                        .toString()) + "</after"
                  + afterImage.getColumnName(i) + ">");
         } catch (DataTypeConversionException e) {
            trace.write(e.getMessage());
         }
      }
   }
   // Write the ending tag for the tabel-level information
   subscriptionContext.printStream.println("\t</table"
         + replicationEvent.getTableName() + ">");
   return true;
}

/**
 * This method is called for both subscription-level and table-level
 * clean-up. It will close the current output file.
 */
public void finish() {
   if (calledAtSubscriptionLevel) {
      trace.write("finish() start");
      closeOutputFile();
      trace.write("finish() end");
   }
   return;
}

/**
 * Formats the content of an XML element and substitutes special characters
 * with mark-up replacements.
 *
 * @param content
 *            - Input element content.
 * @return Contents with special characters marked up.
 */
private String formatXmlContent(String content) {
   String markedUpString = null;
   markedUpString = content;
   markedUpString = markedUpString.replaceAll("\"", "&quot;");
   markedUpString = markedUpString.replaceAll("'", "&apos;");
   markedUpString = markedUpString.replaceAll("&", "&amp;");
   markedUpString = markedUpString.replaceAll("<", "&lt;");
   markedUpString = markedUpString.replaceAll(">", "&gt;");
   return markedUpString;
}

/**
 * Translates the subscription event to a readable text string (mainly for
 * debugging).
 *
 * @param eventType
 *            - Type of the subscription event
 * @return Event type description
 */
private String getSubscriptionEventTypeAsString(int eventType) {
   if (eventType == SubscriptionEventTypes.BEFORE_COMMIT_EVENT)
      return "BEFORE_COMMIT_EVENT";
   else if (eventType == SubscriptionEventTypes.AFTER_COMMIT_EVENT)
      return "AFTER_COMMIT_EVENT";
   else if (eventType == SubscriptionEventTypes.AFTER_EVENT_SHIFT)
      return "AFTER_EVENT_SHIFT";
   else
      return "UNKNOWN_SUBSCRIPTION_EVENT_TYPE";
}

/**
 * Translates the commit reason to a readable text string (mainly for
 * debugging).
 *
 * @param commitReason
 *            - Reason for committing the transaction
 * @return Commit reason description
 */
private String getSubscriptionCommitReasonAsString(int commitReason) {
   if (commitReason == CommitReasonTypes.SOURCE_COMMIT)
      return "SOURCE_COMMIT";
   else if (commitReason ==
            CommitReasonTypes.OPERATION_WITHOUT_COMMITMENT_CONTROL)
      return "OPERATION_WITHOUT_COMMITMENT_CONTROL";
   else if (commitReason == CommitReasonTypes.REFRESH)
      return "REFRESH";
   else if (commitReason == CommitReasonTypes.REPORT_POSITION)
      return "REPORT_POSITION";
   else if (commitReason == CommitReasonTypes.INTERIM_COMMIT)
      return "INTERIM_COMMIT";
   else if (commitReason == CommitReasonTypes.SHUTDOWN)
      return "SHUTDOWN";
   else
      return "UNKNOWN_COMMIT_REASON";
}

/**
 * Creates an output file to hold the XML representation of a database
 * transaction. The method creates a file with the name
 * <PublisherID>_yyyyMMdd_HHmmssSSS.xml and prepares the XML heading. Other
 * methods subsequently write the XML records to the output file.
 *
 * @throws UserExitException
 */
private void createNewOutputFile() throws UserExitException {
   Calendar cal = Calendar.getInstance();
   SimpleDateFormat sdf = new SimpleDateFormat(FILE_TIMESTAMP_FORMAT);
   String timestampSuffix = sdf.format(cal.getTime());

   // Compose name of output file
   subscriptionContext.outputFileName = publisherID + "_"
         + timestampSuffix + ".xml";
   trace.write("Name of work file: " + subscriptionContext.outputFileName);

   try {
      subscriptionContext.printStream = new PrintStream(
            subscriptionContext.outputFileName);
   } catch (FileNotFoundException e) {
      trace.write(e.getMessage());
      throw new UserExitException("Error creating output file "
            + subscriptionContext.outputFileName);
   }
   trace.write("Output file " + subscriptionContext.outputFileName
         + " created");
   subscriptionContext.printStream
         .println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
   subscriptionContext.printStream.println("<transaction>");
}

/**
 * Writes the ending tag for the XML transaction and closes the current
 * output file.
 */
private void closeOutputFile() {
   if (subscriptionContext.printStream != null) {
      subscriptionContext.printStream.println("</transaction>");
      trace.write("Closing output file: "
            + subscriptionContext.outputFileName);
      subscriptionContext.printStream.close();
   }
   subscriptionContext.printStream = null;
}

/**
 * This subclass is used to maintain the overall subscription context as
 * this user exit is instantiated at the subscription (target) level and for
 * all tables. As the same class is used at the subscription and table level
 * we have chosen to create a subclass to create a subclass for the
 * subscription context.
 */
protected class SubscriptionContext {
   protected UETrace trace; // trace object
   protected String publisherID; // Publisher ID for subscription
   protected String outputFileName; // Current output file name
   protected PrintStream printStream; // Current output print stream
}

/**
 * Initializes the context for the subscription. This method is only called
 * once at the subscription level. At the table level, the
 * SubscriptionContext object is retrieved only.
 *
 * @param publisherID
 *           - The publisher ID of the subscription
 */
protected SubscriptionContext createContext(String publisherID) {
   SubscriptionContext context = new SubscriptionContext();
   context.publisherID = new String(publisherID);
   context.trace = new UETrace();
   context.trace.init(true);
   context.trace.write("Context created for publisher ID " + publisherID);
   context.outputFileName = null;
   context.printStream = null;
   return context;
}
}
```

In the InfoSphere CDC configuration, the Java user exit class is specified both at the subscription level and the table level. Both the subscription-level and table-level interfaces are implemented by the user exit, SubscriptionUserExitIF and UserExitIF. For the functionality intended by the user exit, it is imperative to register it at both levels. If you do not configure the subscription-level user exit, the table-level entry points do not have all information available to properly write the XML records.

When the subscription is started, the target side instantiates a CDCTransactionFileWriter object and immediately starts the **`init(SubscriptionEventPublisherIF)`** method (subscription-level), which has the primary task of creating a subscription context object to be shared with the table-level objects. Then, for each mapped table that has the CDCTransactionFileWrite specified as the user exit, a table-level object is instantiated and the **`init(ReplicationEventPublisherIF)`** method is starts. This method registers the after-insert, after-update, and after-delete exit points for the table in question. Every insert, update, and delete operation on the table in question causes the **`processReplicationEvent()`** method to be starts. This method first checks whether there already is an open file for this subscription, which is done by checking the subscriptionContext.printStream object. If there is no open file, a new file is created dynamically and opened. Then, the insert, update, or delete operation for the table in question is written as an XML format structure. When a commit is received from the source side, the **`processSubscriptionEvent()`** method is called, which closes the transaction tag and then closes the file.

The configuration of the CDCTransactionFileWriter user exit is shown in Figure 9-37.


IBM InfoSphere Change Data Capture Management Console
File Edit Subscription Mapping View Help
Monitoring Configuration Access Manager
Subscriptions
Enter search...
Table Mappings - REDUSERX
User Exits
Specify user exits for this subscription.
OK Cancel
Datastores
Enter search...
REDUSERX is ava
Column Mappings Filtering Translation Encoding Conflicts Operation User Exits

Subscription	Source	Target
Default Project		
REDALIAS	Unknown	CDC_DB2
REDSCRIPT	CDC_Oracle ...	CDC_DB2
REDUSERX	CDC_Oracle_...	CDC_DB2

User Exit Type: Java Class
Configuration
Class Name: CDCTransactionFileWriter
Parameter:

Datastore	Database
CDC_DataStage	JDBC
CDC_DB2	DB2 for LUW
CDC_DB2i	DB2 for i
CDC_Event_Server	JDBC
CDC_Informix	JDBC

Source Table	Target Table	Mapping Type	Metho
CDCDEMO.CUSTOMER	CDCDEMO.CUSTOMER	Standard	Mirror
CDCDEMO.PRODUCT	CDCDEMO.PRODUCT	Standard	Mirror
CDCDEMO.SALESREP	CDCDEMO.SALESREP	Standard	Mirror

User Exit Type: Java Class
Configuration
Class Name: CDCTransactionFileWriter
Parameter:

Events and Actions
Activate user exit programs before and after events.

Operation	Before	After
Insert	☐	✓
Update	☐	✓
Delete	☐	✓
Refresh	☐	☐
Truncate	☐	☐

Figure 9-37 InfoSphere CDC user exit subscription level

When doing two updates in a single unit of work, the resulting XML appears as shown in Example 9-89.

Example 9-89 Two updates in a single unit of work

```
<?xml version="1.0" encoding="UTF-8"?>
<transaction>
<tableCUSTOMER>
<tableOperation>UP</tableOperation>
<beforeCUSTNO>987560</beforeCUSTNO>
<beforeBRANCH>11</beforeBRANCH>
<beforeNAME1>CALIFORNIA SPA & FITNESS</beforeNAME1>
<beforeNAME2>abc12345444</beforeNAME2>
<beforeADDRESS1>100 SANDYHOOK SQ.</beforeADDRESS1>
<beforeADDRESS2> </beforeADDRESS2>
<beforeCITY>UPLAND</beforeCITY>
<beforeSTATE>CA</beforeSTATE>
<beforeSTATUS>A</beforeSTATUS>
<beforeCRLIMIT>7506</beforeCRLIMIT>
<beforeBALANCE>6500</beforeBALANCE>
<beforeREPNO>251</beforeREPNO>
<afterCUSTNO>987560</afterCUSTNO>
<afterBRANCH>11</afterBRANCH>
<afterNAME1>CALIFORNIA SPA & FITNESS</afterNAME1>
```

```
<afterNAME2>abc12345444</afterNAME2>
<afterADDRESS1>100 SANDYHOOK SQ.</afterADDRESS1>
<afterADDRESS2> </afterADDRESS2>
<afterCITY>UPLAND</afterCITY>
<afterSTATE>CA</afterSTATE>
<afterSTATUS>A</afterSTATUS>
<afterCRLIMIT>7507</afterCRLIMIT>
<afterBALANCE>6500</afterBALANCE>
<afterREPNO>251</afterREPNO>
</tableCUSTOMER>
<tablePRODUCT>
<tableOperation>UP</tableOperation>
<beforePRODUCTID>100</beforePRODUCTID>
<beforeDESCRIPTN>White paper 8.5 by 11</beforeDESCRIPTN>
<beforeLOCATION>Aisle 5</beforeLOCATION>
<beforeSTATUS>O</beforeSTATUS>
<beforeUNITPRICE>7.00</beforeUNITPRICE>
<beforeUNITCOST>2200.00</beforeUNITCOST>
<beforeQTYONHAND>17850</beforeQTYONHAND>
<beforeQTYALLOC>50</beforeQTYALLOC>
<beforeQTYMINORD>5000</beforeQTYMINORD>
<afterPRODUCTID>100</afterPRODUCTID>
<afterDESCRIPTN>White paper 8.5 by 11</afterDESCRIPTN>
<afterLOCATION>Aisle 5</afterLOCATION>
<afterSTATUS>O</afterSTATUS>
<afterUNITPRICE>8.00</afterUNITPRICE>
<afterUNITCOST>2200.00</afterUNITCOST>
<afterQTYONHAND>17850</afterQTYONHAND>
<afterQTYALLOC>50</afterQTYALLOC>
<afterQTYMINORD>5000</afterQTYMINORD>
</tablePRODUCT>
</transaction>
```

9.7.6 Java user exit for flat file custom formatter

When using InfoSphere CDC to deliver flat files for consumption by external applications, the target engine is often InfoSphere CDC for DataStage. This engine can generate flat files and has additional functionality to automatically close and make the flat files available based on time or number of rows.

Flat files that are generated by InfoSphere CDC for DataStage have the following characteristics:

- Journal control information written as the first few columns on every line
- Characters written in UTF-8 encoding
- Columns that are separated by a comma
- Columns that are delimited by a double quotation mark

This fixed output format might not be suitable for the targeted applications. Example 9-90 tailors the standard InfoSphere CDC for DataStage Flat File output to use a different column delimiter ("ª", feminine ordinal indicator, instead of the double quotation mark) and column separator (|, vertical line, instead of the comma). Also, the flat file is generated in ISO8859-1 (Western-European) instead of the default unicode UTF-8 encoding.

Example 9-90 Sample code for tailored flat file output

```
import java.io.UnsupportedEncodingException;
import java.math.BigDecimal;
import java.nio.ByteBuffer;
import java.sql.Time;
import java.sql.Timestamp;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;

import com.datamirror.ts.target.publication.UserExitJournalHeader;
import com.datamirror.ts.target.publication.userexit.DataRecordIF;
import com.datamirror.ts.target.publication.userexit
         .DataTypeConversionException;
import com.datamirror.ts.target.publication.userexit.ReplicationEventIF;
import com.datamirror.ts.target.publication.userexit
         .datastage.DataStageDataFormatIF;

/**
 *
 * Format the data suitable for the target application's sequential file reader.
 *
 */
public class CDCDataStageFormat implements DataStageDataFormatIF {

// Specified encoding for output (formatted string)
   private static final String OUTPUT_STRING_ENCODING = "ISO-8859-1";

// Separator character (between columns) and delimiter (surrounding columns)
   private static String SEPARATOR = "|";
   private static String DELIMITER = "ª";

// Types of record images that can be received by the formatter user exit
   public final char SUB_RLA_AUDIT = 'A';
   public final char SUB_RLA_AUDIT_BEFORE = 'B';
   public final char SUB_RLA_INS_UPD = 'I';
   public final char SUB_RLA_NON_UPD = 'U';
   public final char SUB_RLA_DEL_NONE = 'D';

// Formatting for date, time and timestamp columns
   private SimpleDateFormat outDateFormat = new SimpleDateFormat(
         "yyyy-MM-dd HH:mm:ss");
   private SimpleDateFormat outDateOnlyFormat = new SimpleDateFormat(
         "yyyy-MM-dd");
   private SimpleDateFormat outTimeOnlyFormat = new SimpleDateFormat(
         "HH:mm:ss");

   public static final int BYTE_BUFFER_AUTO_INCREMENT_SIZE = 10000;
   public static final int BYTE_BUFFER_AUTO_INCREMENT_BREATHING_SPACE = 1000;
   public static final int BYTE_BUFFER_SPACE_FOR_FIELD_SEPARATORS = 100;
   public static final int BYTE_BUFFER_SPACE_FOR_JOURNAL_CONTROL = 1000;
   ByteBuffer outBuffer = ByteBuffer.allocate(BYTE_BUFFER_AUTO_INCREMENT_SIZE);

   private int destinationType;
// Is this flat file or direct connect?
   private int clobTruncationPoint;
// Truncation for CLOB columns as specified
// in subscription
   private int blobTruncationPoint;
// Truncation for BLOB columns as specified
// in subscription

   private static String ZERO_STRING = "0";
// Used for boolean FALSE external
// representation
   private static String ONE_STRING = "1";
// Used for boolean TRUE external
// representation

// These are the byte arrays to be appended to the buffer. The content is
// already encoded
// in the specified encoding
   private static byte[] SEPARATOR_AS_BYTE_ARRAY =
                        getAsEncodedByteArray(SEPARATOR);
   private static byte[] DELIMITER_AS_BYTE_ARRAY =
                         getAsEncodedByteArray(DELIMITER);
   private static byte[] SEP_DEL_SEP_AS_BYTE_ARRAY =
                         getAsEncodedByteArray(SEPARATOR + DELIMITER);
   private static byte[] DEL_SEP_DEL_AS_BYTE_ARRAY =
                         getAsEncodedByteArray(DELIMITER+SEPARATOR+DELIMITER);
   private static byte[] DEL_SEP_AS_BYTE_ARRAY =
                         getAsEncodedByteArray(DELIMITER + SEPARATOR);
   private static byte[] ZERO_AS_BYTE_ARRAY =
                         getAsEncodedByteArray(ZERO_STRING);
   private static byte[] ONE_AS_BYTE_ARRAY = getAsEncodedByteArray(ONE_STRING);

   private int numberDataColumns = 0; // Number of data columns in table
   ByteBuffer nullImage = null;
   boolean firstTimeData = true;
   boolean firstTimeJournal = true;
   UETrace trace;

   public CDCDataStageFormat() {
      trace = new UETrace();
      trace.init(true);
   }

/**
 * This method is called when the subscription is started to indicate at
 * which position to truncate CLOB and BLOB columns.
 */
   public void setLobTruncationPoint(int maxClobLengthInChars,
         int maxBlobLengthInBytes) {
      clobTruncationPoint = maxClobLengthInChars;
      blobTruncationPoint = maxBlobLengthInBytes;
   }

/**
 * Create a string containing the data images for the row; this method is
 * starts from the InfoSphere CDC for DataStage engine when a new row has to be
 * formatted.
 *
 * @param image
 *            - The record image that must be processed
 * @return A buffer of bytes with the formatted data image
 */
   public ByteBuffer formatDataImage(DataRecordIF image)
         throws DataTypeConversionException {
      boolean needToCloseQuote = false;
      outBuffer.position(0);
      if (image != null) {
// When called for the first time, determine the number of data
// columns
         if (firstTimeData) {
            numberDataColumns = getNumberDataColumns(image);
            firstTimeData = false;
         }
// End debug
         for (int i = 1; i <= numberDataColumns; i++) {
            Object colObj = image.getObject(i);

// For NULL values, we just leave the field empty
            if (colObj != null) {
// For performance, we have this logic to only do one insert
// of separators and delimiters between columns;
// this reduces the amount of processing the user exit has
// to do
               if (needToCloseQuote) {
                  outBuffer.put(DEL_SEP_DEL_AS_BYTE_ARRAY);
               } else {
                  outBuffer.put(SEP_DEL_SEP_AS_BYTE_ARRAY);
               }
               needToCloseQuote = true;

// Time, timestamp and dates must be processed in this order
// because time and timestamp are also date objects
               if (colObj instanceof Time) {
                  outBuffer = addEncodedStringToByteBuffer(outBuffer,
                        outTimeOnlyFormat.format((Time) colObj));
               } else if (colObj instanceof Timestamp) {
                  outBuffer = addEncodedStringToByteBuffer(outBuffer,
                        outDateFormat.format((Timestamp) colObj));
               } else if (colObj instanceof Date) {
                  outBuffer = addEncodedStringToByteBuffer(outBuffer,
                        outDateOnlyFormat.format((Date) colObj));
               } else if (colObj instanceof byte[]) { // BLOB
                  byte[] val = (byte[]) colObj;
                  if (val.length > blobTruncationPoint) {
                     byte[] truncVal = new byte[blobTruncationPoint];
                     ByteBuffer truncBuffer = ByteBuffer.wrap(truncVal);
                     truncBuffer.put(val, 0, blobTruncationPoint);
                     val = truncVal;
                  }
                  outBuffer = addBytesToByteBuffer(outBuffer, val);
               } else if (colObj instanceof Boolean) { // Boolean
                  if (((Boolean) colObj).booleanValue()) {
                     outBuffer = addBytesToByteBuffer(outBuffer,
                           ONE_AS_BYTE_ARRAY);
                  } else {
                     outBuffer = addBytesToByteBuffer(outBuffer,
                           ZERO_AS_BYTE_ARRAY);
                  }
               } else if (colObj instanceof String) { // CLOB and strings
                  String val = ((String) colObj);
                  if (val.length() > clobTruncationPoint) {
                     val = val.substring(0, clobTruncationPoint);
                  }
                  outBuffer = addEncodedStringToByteBuffer(outBuffer, val);
               } else if (colObj instanceof BigDecimal) { // All numerics
// Use toPlainString for Java 1.5
                  outBuffer = addEncodedStringToByteBuffer(outBuffer,
                        ((BigDecimal) colObj).toString());
               } else { // Any other type
                  outBuffer = addEncodedStringToByteBuffer(outBuffer,
                        colObj.toString());
               }
            } else {
               if (needToCloseQuote) {
                  outBuffer.put(DEL_SEP_AS_BYTE_ARRAY);
                  needToCloseQuote = false;
               } else {
                  outBuffer.put(SEPARATOR_AS_BYTE_ARRAY);
               }
            }
         }
         if (needToCloseQuote) {
            outBuffer.put(DELIMITER_AS_BYTE_ARRAY);
         }
      }
      return outBuffer;
   }

/**
 * Create a string containing the null images for the row. This is the
 * before-image for an insert operation, or the after-image for a delete
 * operation.
 *
 * @param image
 *            - The null record image that must be processed
 * @return A buffer of bytes with the formatted null image
 */
   public ByteBuffer formatNullImage(DataRecordIF image)
         throws DataTypeConversionException {
// There is a separate data formatter for each table, so a null image is
// the same for each row, so just need to create it once
      if (nullImage == null) {
// when called for the first time, determine the number of data
// columns
         if (firstTimeData) {
            numberDataColumns = getNumberDataColumns(image);
            firstTimeData = false;
         }
         String outString = "";
         if (image != null) {
            for (int i = 1; i <= numberDataColumns; i++) {
               outString = outString + SEPARATOR;
            }
         }
         nullImage = ByteBuffer.wrap(getAsEncodedByteArray(outString));
         nullImage.position(nullImage.capacity());
      }
      return nullImage;
   }

/**
 * Create a string holding the journal control columns which must be output
 * in the flat file
 *
 * @param event
 *            - Indication of when the event occurred
 * @param operationType
 *            - The type of the operation for which the image must be
 *            generated
 * @return A buffer of bytes with the formatted journal control image
 */
/**
 * Return a ByteBuffer containing the journal control field values that are
 * of interest.
 *
 */
   public ByteBuffer formatJournalControlFields(ReplicationEventIF event,
         int operationType) throws DataTypeConversionException {
// when called for the first time, determine the number of data columns
      if (firstTimeJournal) {
         trace.write("Table that will be formatted: "
               + event.getJournalHeader().getLibrary() + "."
               + event.getJournalHeader().getObjectName());
         firstTimeJournal = false;
      }
// Determine the character to use to indicate the operation type
      char opChar = ' ';
      switch (operationType) {
      case DataStageDataFormatIF.INSERT_RECORD:
         opChar = SUB_RLA_INS_UPD;
         break;
      case DataStageDataFormatIF.DELETE_RECORD:
         opChar = SUB_RLA_DEL_NONE;
         break;
      case DataStageDataFormatIF.FULL_UPDATE_RECORD:
         opChar = SUB_RLA_NON_UPD;
         break;
      case DataStageDataFormatIF.BEFORE_UPDATE_RECORD:
         opChar = SUB_RLA_AUDIT_BEFORE;
         break;
      case DataStageDataFormatIF.AFTER_UPDATE_RECORD:
         opChar = SUB_RLA_AUDIT;
         break;

      }

      UserExitJournalHeader header = (UserExitJournalHeader) event
            .getJournalHeader();
      String journalControlString = DELIMITER
            + header.getDSOutputTimestampStr() + DELIMITER + SEPARATOR
            + DELIMITER + header.getCommitID() + DELIMITER + SEPARATOR
            + DELIMITER + opChar + DELIMITER + SEPARATOR + DELIMITER
            + header.getUserName() + DELIMITER;

      ByteBuffer retVal = ByteBuffer
            .allocate(BYTE_BUFFER_SPACE_FOR_JOURNAL_CONTROL);
      retVal = addEncodedStringToByteBuffer(retVal, journalControlString);
      return retVal;
   }

/**
 * Indicate whether this table is being delivered to DataStage using flat
 * files or by direct connect.
 *
 * @param destination
 *            indicates the destination type
 */
   public void setDestinationType(int destination) {
      destinationType = destination;
   }

/**
 * Method used when mapping tables with Direct Connect (InfoSphere CDC 6.5+).
 *
 * @param journalHeader
 *            - Information about the journal control columns
 * @param rowDataImage
 *            - Changed row data columns
 * @param changeRecord
 *            - Changed row data columns in Map object
 * @param opType
 *            - Operation type
 */
   public void formatChangedRowFields(UserExitJournalHeader journalHeader,
         DataRecordIF rowDataImage, Map<String, Object> changeRecord,
         int opType) throws DataTypeConversionException {

   }

/**
 * Determines the number of data columns in the image, excluding the journal
 * control columns as these are also passed when the data must be formatted
 *
 * @param image
 *            - The data image to be evaluated
 * @return Number of table (data) columns in the table
 */
   private int getNumberDataColumns(DataRecordIF image) {
      int numberColumns = 0;
      trace.write("Columns in published image:");
      for (int i = 1; i <= image.getColumnCount(); i++) {
         if (!image.getColumnName(i).startsWith("&"))
            numberColumns++;
         trace.write(image.getColumnName(i));
      }
      trace.write("Number of data columns: " + numberColumns);
      return numberColumns;
   }

/**
```

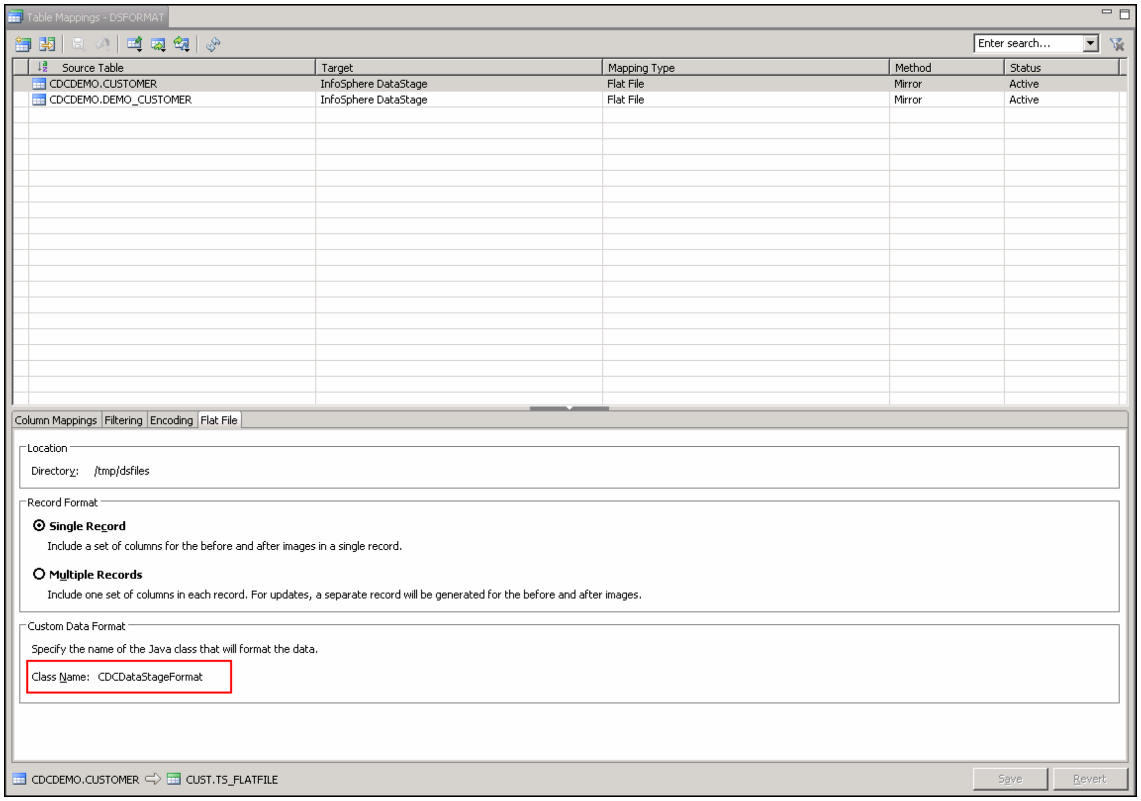
```
 * Returns the passed string as a byte array, encoded in the character set
 * that is specified in the settings
 *
 * @param inString
 *            - Input string to encode and return as byte array
 * @return
 */
   public static byte[] getAsEncodedByteArray(String inString) {
      byte[] retval;

      try {
         retval = inString.getBytes(OUTPUT_STRING_ENCODING);
      } catch (UnsupportedEncodingException e) {
// If the encoding is not supported, the default encoding is used
         retval = inString.getBytes();
      }
      return retval;
   }

/**
 * Append the passed string to the ByteBuffer object, encoded in the
 * character set that is specified in the settings. This method is used to
 * format non-binary objects which all must be encoded (including timestamp
 * and numerics).
  *
  * @param buf
 *            - Byte buffer to which the string will be appended
 * @param inString
 *            - String to be appended to the byte buffer
 * @return Changed byte buffer
 * @throws DataTypeConversionException
 */
   public static ByteBuffer addEncodedStringToByteBuffer(ByteBuffer buf,
         String inString) throws DataTypeConversionException {
      ByteBuffer retVal;
      byte[] asBytes;

      asBytes = getAsEncodedByteArray(inString);

      if (buf.capacity() < buf.position() + asBytes.length
            + BYTE_BUFFER_SPACE_FOR_FIELD_SEPARATORS) {
         int increment = BYTE_BUFFER_AUTO_INCREMENT_SIZE;
         if (increment < asBytes.length) {
            increment = asBytes.length
                  + BYTE_BUFFER_AUTO_INCREMENT_BREATHING_SPACE;
         }
         retVal = ByteBuffer.allocate(buf.capacity() + increment);
         buf.flip();
         retVal.put(buf);
      } else {
         retVal = buf;
      }

      retVal.put(asBytes);

      return retVal;
   }

/**
 * Append the passed byte to the ByteBuffer object, as-is. This method is
 * used to format binary objects which must not be encoded.
 *
 * @param buf
 *            - Byte buffer to which the byte will be appended
 * @param inByte
 *            - Byte to be appended to the byte buffer
 * @return Changed byte buffer
 */
   public static ByteBuffer addByteToByteBuffer(ByteBuffer buf, byte inByte) {
      ByteBuffer retVal;

      if (buf.capacity() < buf.position() + 1
            + BYTE_BUFFER_SPACE_FOR_FIELD_SEPARATORS) {
         retVal = ByteBuffer.allocate(buf.capacity()
               + BYTE_BUFFER_AUTO_INCREMENT_SIZE);
         buf.flip();
         retVal.put(buf);
      } else {
         retVal = buf;
      }
      retVal.put(inByte);

      return retVal;
   }

/**
 * Append the passed byte array to the ByteBuffer object, as-is. This method
 * is used to format binary objects which must not be encoded.
 *
 * @param buf
 *             - Byte buffer to which the byte will be appended
 * @param asBytes
 *             - Byte array to be appended to the byte buffer
 * @return Changed byte buffer
 */
   public static ByteBuffer addBytesToByteBuffer(ByteBuffer buf,
      byte[] asBytes) {
      ByteBuffer retVal;

      if (buf.capacity() < buf.position() + asBytes.length
            + BYTE_BUFFER_SPACE_FOR_FIELD_SEPARATORS) {
         int increment = BYTE_BUFFER_AUTO_INCREMENT_SIZE;
         if (increment < asBytes.length) {
            increment = asBytes.length
                  + BYTE_BUFFER_AUTO_INCREMENT_BREATHING_SPACE;
         }
         retVal = ByteBuffer.allocate(buf.capacity() + increment);
         buf.flip();
         retVal.put(buf);
      } else {
         retVal = buf;
      }
      retVal.put(asBytes);
      return retVal;
   }
}
```

The CDCDataStageFormat class provides an example of how the flat file output can be customized to meet your needs. There are three main methods in the interface that must be implemented to format the data: formatDataImage(), formatNullImage() and formatJournalControlFields(). The other interface methods must be implemented too, but you can choose whether you want to provide any code for methods. When data is formatted, the **`formatJournalControlFields()`** method is called first. This method provides the means to do initialization processing at the table level and formats the journal control columns. For formatting the data image, the **`formatDataImage()`** method is passed the argument, which holds the data record object. When an update operation is processed, this method is starts twice, once for the before image and once for the after image. The **`formatNullImage()`** method is called for the insert and delete operations if the before and after images are in a single record and can return the number of column separators equivalent to the number of columns.

If a character sequence shows the column delimiter, which is common when binary objects are replicated, the example could also fit in that scenario. Change the DELIMITER to the sequence of characters that defines the column delimiter, for example, "?~#".

To activate the custom data formatter, the class must be specified in the Flat File properties of the table mappings. The class cannot accept any parameters.

An example configuration for DataStage is shown in Figure 9-38.



Source Table	Target	Mapping Type	Method	Status
CDCDEMO.CUSTOMER	InfoSphere DataStage	Flat File	Mirror	Active
CDCDEMO.DEMO_CUSTOMER	InfoSphere DataStage	Flat File	Mirror	Active

Location
Directory: /tmp/dsfiles

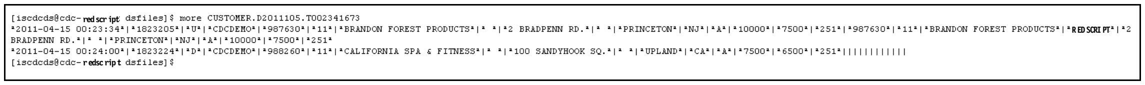
Record Format
(●) Single Record
Include a set of columns for the before and after images in a single record.
( ) Multiple Records
Include one set of columns in each record. For updates, a separate record will be generated for the before and after images.

Custom Data Format
Specify the name of the Java class that will format the data.
Class Name: CDCDataStageFormat

CDCDEMO.CUSTOMER ⇨ CUST.TS_FLATFILE

Figure 9-38 InfoSphere CDC user exit configuration for DataStage

When running the subscription and making a few changes, the flat file output appears as shown in Figure 9-39.



```
[iscdcds@cdc-redscript dsfiles]$ more CUSTOMER.D2011105.T002341673
ª2011-04-15 00:23:34ª|ª1823205ª|ªUª|ªCDCDEMOª|ª987630ª|ª11ª|ªBRANDON FOREST PRODUCTSª|ª ª|ª2 BRADPENN RD.ª|ª ª|ªPRINCETONª|ªNJª|ªAª|ª10000ª|ª7500ª|ª251ª|ª987630ª|ª11ª|ªBRANDON FOREST PRODUCTSª|ªREDSCRIPTª|ª2
BRADPENN RD.ª|ª ª|ªPRINCETONª|ªNJª|ªAª|ª10000ª|ª7500ª|ª251ª
ª2011-04-15 00:24:00ª|ª1823224ª|ªDª|ªCDCDEMOª|ª988260ª|ª11ª|ªCALIFORNIA SPA & FITNESSª|ª ª|ª100 SANDYHOOK SQ.ª|ª ª|ªUPLANDª|ªCAª|ªAª|ª7500ª|ª6500ª|ª251ª|||||||||||
[iscdcds@cdc-redscript dsfiles]$
```

Figure 9-39 InfoSphere CDC user exit DataStage output

9.7.7 Notifications

As explained in 9.6.4, “Events” on page 367, some external monitoring solutions expect that monitored log files be appended to determine if there are any new messages to be acted upon.

The example Java notification user exit writes events that have been selected for user exit handling to the `<cdc_home>/log/cdc_notifications.log` file. When a notification has been configured for a certain category in the InfoSphere CDC process and that category is detected, InfoSphere CDC starts the **`handle()`** method.

Example 9-91 shows sample code for a notification user exit.

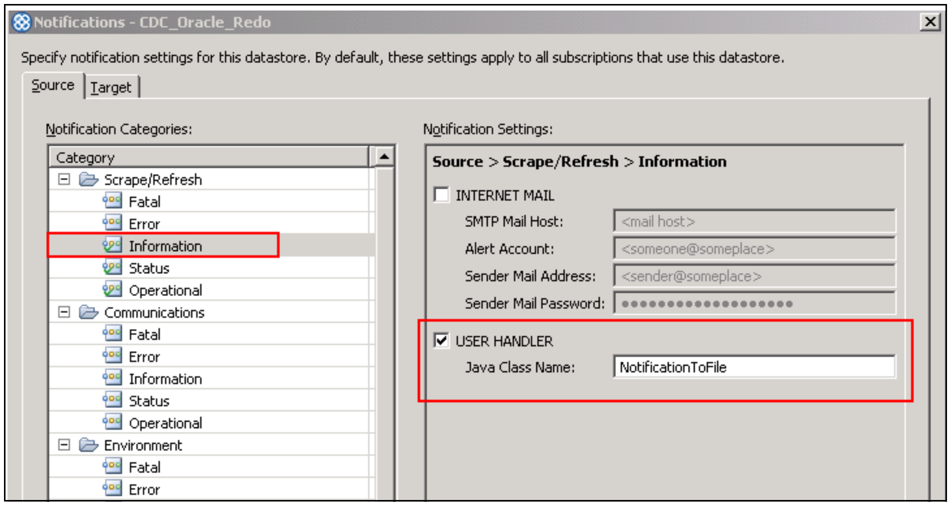
Example 9-91 Notification user exit

```
import java.io.*;
import java.util.*;
import java.text.SimpleDateFormat;
import com.datamirror.ts.api.*;

public class NotificationToFile implements AlertHandlerIF {

// Line separator is dependent on the platform (Linux/Unix/Windows)
   private final static String LINE_SEPARATOR = System
         .getProperty("line.separator");
// Directory separator is dependent on the platform (/ on Unix or Linux, \
// on Windows)
   private final static String FILE_SEPARATOR = System
         .getProperty("file.separator");

/**
 * Constructor, will be called when the object is instantiated. You could
 * include activity such as creating the log file.
 */
   public NotificationToFile() {
   }

/**
 * This method is starts for every event that is defined to be handled by a
 * USER HANDLER at the datastore or subscription level.
 *
 * When the method is called, it opens the cdc_notifications.log file in the
 * <cdc_home>/log directory and writes the event in a format that is
 * equivalent to the output of dmshowevents. The log file is continuously
 * appended to and can be monitored by an external monitoring solution.
 *
 * @param zone
 *            - Zone of the event (not used anymore with InfoSphere CDC 6.5)
 * @param category
 *            - Category of the event (information, error, ...)
 * @param sourceOrTarget
 *            - Did the event happen on the source or the target
 * @param subscriptionName
 *            - Subscription that generated the event
 * @param eventID
 *            - Numeric representation of the event
 * @param eventText
 *            - Message issued by CDC engine
 * @param otherInfo
 *            - Other properties information (not used)
 */
  public void handle(int zone, int category, String sourceOrTarget,
        String subscriptionName, int eventID, String eventText,
        Properties otherInfo) throws Exception {
     BufferedWriter notificationWriter = null;
     try {
// Locate or create the file and open it for output
        notificationWriter = new BufferedWriter(new FileWriter("log"
              + FILE_SEPARATOR + "cdc_notifications.log", true));
// Determine the current time and convert it to ISO representation
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(new Date());
        SimpleDateFormat dateFormat = new SimpleDateFormat(
              "yyyy-MM-dd HH:mm:ss");

// Format the message string and write to the log file
        String message = dateFormat.format(calendar.getTime()) + "|"
              + sourceOrTarget + "|" + subscriptionName + "|" + eventID
              + "|" + getCategoryString(category) + "|"
              + getZoneString(zone) + "|" + eventText + LINE_SEPARATOR;
        notificationWriter.write(message);
     } finally {
// Always executed
        if (notificationWriter != null) {
           try {
              notificationWriter.close();
           } catch (IOException e) {
              e.printStackTrace();
           }
        }
```

```
      }
      return;
   }

/**
 * Converts the event category to a readable string
 *
 * @param category
 *            - Category of the event
 * @return Category string representation
 */
   private String getCategoryString(int category) {
      switch (category) {
      case 1:
         return "Fatal";
      case 2:
         return "Error";
      case 3:
         return "Information";
      case 4:
         return "Status";
      case 5:
         return "Operational";
      default:
         return "Unknown Zone";
      }
   }

/**
 * Converts the event zone to a readable string
 *
 * @param zone
 *            - Zone of the event
 * @return Zone string representation
 */
   private String getZoneString(int zone) {
      switch (zone) {
      case 1:
         return "Communication";
      case 2:
         return "Apply";
      case 3:
         return "Environment";
      default:
         return "";
```

```
            }
        }
    }
```

To implement the notification handling, go to the Notifications section of your data store and select the notifications. If you only want to handle certain events (for example, unrecoverable and error messages), select only the notifications that generate entries in this file. Figure 9-40 shows an example of how the notification user handler can be specified at the data store level.



Specify notification settings for this datastore. By default, these settings apply to all subscriptions that use this datastore.

Source > Scrape/Refresh > Information

☐ INTERNET MAIL

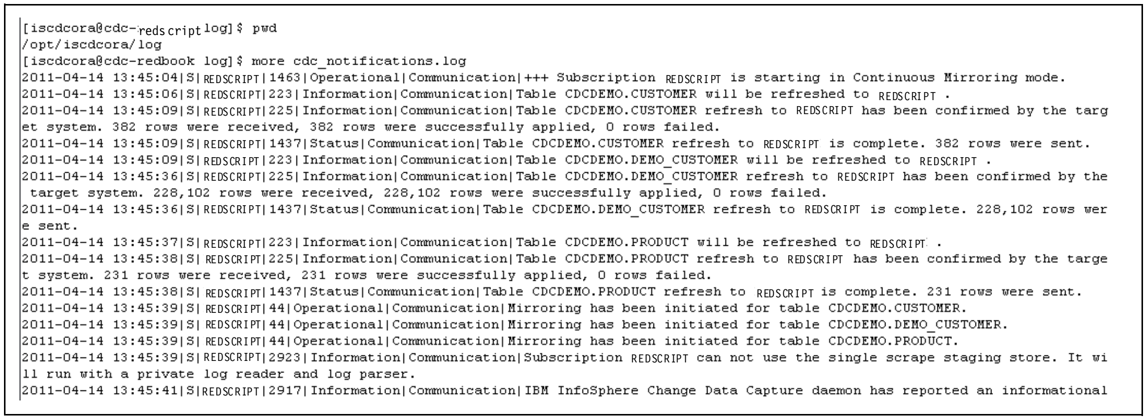
SMTP Mail Host:	<mail host>
Alert Account:	<someone@someplace>
Sender Mail Address:	<sender@someplace>
Sender Mail Password:	●●●●●●●●●●●●●●●●●●

☑ USER HANDLER

Java Class Name: NotificationToFile

Figure 9-40 User exit notification configuration

When you start a subscription and look at the events being generated, you see the event log messages logged in the `<cdc_home>/log/cdc_notifications.log` file (Figure 9-41).



```
[iscdcora@cdc-redscript log]$ pwd
/opt/iscdcora/log
[iscdcora@cdc-redbook log]$ more cdc_notifications.log
2011-04-14 13:45:04|S|REDSCRIPT|1463|Operational|Communication|+++ Subscription REDSCRIPT is starting in Continuous Mirroring mode.
2011-04-14 13:45:06|S|REDSCRIPT|223|Information|Communication|Table CDCDEMO.CUSTOMER will be refreshed to REDSCRIPT .
2011-04-14 13:45:09|S|REDSCRIPT|225|Information|Communication|Table CDCDEMO.CUSTOMER refresh to REDSCRIPT has been confirmed by the targ
et system. 382 rows were received, 382 rows were successfully applied, 0 rows failed.
2011-04-14 13:45:09|S|REDSCRIPT|1437|Status|Communication|Table CDCDEMO.CUSTOMER refresh to REDSCRIPT is complete. 382 rows were sent.
2011-04-14 13:45:09|S|REDSCRIPT|223|Information|Communication|Table CDCDEMO.DEMO_CUSTOMER will be refreshed to REDSCRIPT .
2011-04-14 13:45:36|S|REDSCRIPT|225|Information|Communication|Table CDCDEMO.DEMO_CUSTOMER refresh to REDSCRIPT has been confirmed by the
 target system. 228,102 rows were received, 228,102 rows were successfully applied, 0 rows failed.
2011-04-14 13:45:36|S|REDSCRIPT|1437|Status|Communication|Table CDCDEMO.DEMO_CUSTOMER refresh to REDSCRIPT is complete. 228,102 rows wer
e sent.
2011-04-14 13:45:37|S|REDSCRIPT|223|Information|Communication|Table CDCDEMO.PRODUCT will be refreshed to REDSCRIPT .
2011-04-14 13:45:38|S|REDSCRIPT|225|Information|Communication|Table CDCDEMO.PRODUCT refresh to REDSCRIPT has been confirmed by the targe
t system. 231 rows were received, 231 rows were successfully applied, 0 rows failed.
2011-04-14 13:45:38|S|REDSCRIPT|1437|Status|Communication|Table CDCDEMO.PRODUCT refresh to REDSCRIPT is complete. 231 rows were sent.
2011-04-14 13:45:39|S|REDSCRIPT|44|Operational|Communication|Mirroring has been initiated for table CDCDEMO.CUSTOMER.
2011-04-14 13:45:39|S|REDSCRIPT|44|Operational|Communication|Mirroring has been initiated for table CDCDEMO.DEMO_CUSTOMER.
2011-04-14 13:45:39|S|REDSCRIPT|44|Operational|Communication|Mirroring has been initiated for table CDCDEMO.PRODUCT.
2011-04-14 13:45:39|S|REDSCRIPT|2923|Information|Communication|Subscription REDSCRIPT can not use the single scrape staging store. It wi
ll run with a private log reader and log parser.
2011-04-14 13:45:41|S|REDSCRIPT|2917|Information|Communication|IBM InfoSphere Change Data Capture daemon has reported an informational
```

Figure 9-41 Output of event log messages

A

Single scrape events and errors

The following events are explicitly generated by single scrape and are visible in the IBM InfoSphere Change Data Capture (InfoSphere CDC) Event Log:

- I_SHAREDSCRAPE_COMPONENT_STARTED 2920 EventCategory.INFORMATION, EventSeverity.INFO

  The single scrape component has started. The staging store is {0}% full.
- I_SHAREDSCRAPE_COMPONENT_STARTED_EMPTY 2921, EventCategory.INFORMATION, EventSeverity.INFO

  The single scrape component has started. The staging store is empty.
- I_SHAREDSCRAPE_SUBSCRIPTION_STARTED 2922, EventCategory.INFORMATION, EventSeverity.INFO

  Subscription {0} has started using the single scrape staging store.
- I_SHAREDSCRAPE_SUBSCRIPTION_REJECTED 2923 EventCategory.INFORMATION, EventSeverity.INFO

  Subscription {0} ca not use the single scrape staging store. It runs with a private log reader and log parser.

- I_SHAREDSCRAPE_SUBSCRIPTION_KICKED_OUT 2924 EventCategory.INFORMATION, EventSeverity.INFO

  Subscription {0} has stopped using the single scrape staging store, and is now using a private log reader and log parser.

- W_SINGLESCRAPE_STAGINGSTORE_INCORRECT_VERSION 2927 EventCategory.INFORMATION, EventSeverity.WARNING

  Incorrect single scrape staging store version. This version: {0} Expected version: {1}.

- W_SINGLESCRAPE_STAGINGSTORE_DISK_WRITE_ERROR 2928 EventCategory.INFORMATION, EventSeverity.WARNING

  The single scrape staging store experienced an IOException when writing data to disk. This error is most likely an out-of-disk-space error. No data has been lost, as the data is still in memory. No further operations will be stored until some of the data in the staging store is no longer needed. Consider refreshing subscription tables with stale data, unconfiguring any unneeded subscription tables, or deleting any unneeded subscriptions. Do not stop the instance until this problem is rectified.

- W_SINGLESCRAPE_STAGINGSTORE_DISK_SPACE_QUOTA_EXCEEDED 2929 EventCategory.INFORMATION, EventSeverity.WARNING

  The single scrape staging store disk space usage has exceeded the quota. No further operations will be stored until the disk space usage is below the quota again. This situation could affect source performance, and could be because of poor target apply performance. Consider increasing the quota, refreshing subscription tables with stale data, unconfiguring any unneeded subscription tables, or deleting any unneeded subscriptions.

- W_SINGLESCRAPE_STAGINGSTORE_DISK_SPACE_QUOTA_EXCEEDED_INDEPENDENT 2950, EventCategory.INFORMATION, EventSeverity.WARNING

  The single scrape staging store disk space usage has exceeded the quota. Some of the oldest data will be deleted from the store. Some subscriptions that were using the single scrape staging store might now need to run with a private scraper.

- W_SINGLESCRAPE_STAGINGSTORE_DISK_WRITE_ERROR_INDEPENDENT 2951 EventCategory.INFORMATION, EventSeverity.WARNING

  The single scrape staging store experienced an IOException when writing data to disk. This situation is most likely an out-of-disk-space error. No data has been lost, as the data is still in memory. Some of the oldest data will be deleted from the store. Some subscriptions that were using the single scrape staging store might now need to run with a private scraper.

- E_SINGLESCRAPE_STAGINGSTORE_CORRUPT 2930 EventCategory.FATAL, EventSeverity.ERROR

  The single scrape staging store is corrupted. To resolve this issue, you must clear the staging store using the **`dmclearstagingstore`** command-line utility.
- E_SINGLESCRAPE_STAGINGSTORE_CLEAR_WHILE_RUNNING 2931 EventCategory.FATAL, EventSeverity.ERROR

  The single scrape staging store cannot be cleared while subscriptions are mirroring.
- E_SINGLESCRAPE_ERROR 2933 EventCategory.FATAL, EventSeverity.ERROR

  Exception in single scrape.

Single scrape error events

The following events are explicitly generated by single scrape errors and are visible in the InfoSphere CDC Event Log:

- SingleScrape_StagingStore_TamperedWith =The single scrape staging store block file {0} is corrupted or has been tampered with.
- SingleScrape_StagingStore_FileNotDeleted =The deletion of the single scrape staging store block file {0} failed.
- SingleScrape_StagingStore_PersistAborted=The {0} instance was shut down with the abort option while the single scrape staging store was being written to disk.
- SingleScrape_DisconnectedScrape_NotAllowedWithIndependentSubs=The Single scrape staging store disconnected scrape feature cannot be enabled if subscriptions can run independently. To enable disconnected scrape, you must set the system property `staging_store_can_run_independently` to false, and then restart the engine.

Entries within braces, such as {0}, are placeholders for input values supplied to the event text at run time.

B

Additional material

This book refers to additional material that can be downloaded from the Internet.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247941`

Alternatively, you can go to the IBM Redbooks website at:

`ibm.com``/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247941.

Using the web material

The additional web material that accompanies this book includes the following file:

`SG247941 CDC Metadata Tables.pdf`	This file contains a listing of the CDC configuration metadata tables for the Linux, UNIX, and Windows engine versions with some information about some of the key tables as of the time of publication. There is no commitment to keep this file up to date, and the metadata is subject to change without notice. The current structure of your version's metadata is viewable using the CDC tool **`dmmdconsole`** located in the CDC installation `bin` directory if your system can view Java Swing-based GUI applications, or using **`dmmdcommander`** for command-line viewing using standard SQL commands. Both interfaces are read only, and no direct editing of the CDC metadata is possible, but you might find it helpful for reporting and other purposes.

How to use the web material

Download the file to your workstation and open it to view the metadata information.

Glossary

access control list. The list of principals that have explicit permission (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

Access Manager. A functional area of the Management Console used by the administrator to configure the security model of InfoSphere CDC users and their permissions.

Access Server. A background process on a Windows or UNIX workstation that implements the InfoSphere CDC security model.

ACL. See *access control list*.

adaptive apply. A method of applying data to a target table that allows the target table to have incorrect data. As data changes are replicated from the source table, the target table becomes more accurate.

after image. The updated content of a source-table column after the source operation has completed.

aggregate. Pre-calculated and pre-stored summaries, kept in the data warehouse to improve query performance.

aggregation. An attribute-level transformation that reduces the level of detail of available data, for example, having a Total Quantity by Category of Items rather than the individual quantity of each item in the category.

application programming interface. An interface provided by a software product that enables programs to request services.

asynchronous messaging. A method of communication between programs in which a program places a message on a message queue, and then proceeds with its own processing without waiting for a reply to its message.

attribute. A field in a dimension table.

auditing. The process of tracking table and row-level operations (insert, update, delete, and clear operations) that have been applied to a source table. The tracking occurs in a target table.

before image. The content of a replication source-table column before the source operation.

bidirectional replication. Replication in which 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. You must choose which copy of the table wins if a conflict occurs.

BLOB. Binary large object, a block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted.

bookmark. Information that InfoSphere CDC uses to maintain transactional data consistency.

cascading replication. A replication topology where changes are first replicated to a target table, and then the changes made to that target table are further replicated to one or more additional target tables (often located in another database).

change data capture. Provides rapid and timely capture and delivery of data changes across enterprise systems in real time.

column function. An expression within a derived expression that performs enhanced data manipulation. Typically, column functions calculate statistical aggregations, such as COUNT, SUM, AVG, or MAX, based on the values of existing columns, and then place the results in new columns.

column mapping. The action of choosing which source columns are replicated to which target columns.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

composite key. A key in a fact table that is the concatenation of the foreign keys in the dimension tables.

computer. A device that accepts information (in the form of digitalized data) and manipulates it for some result based on a program or sequence of instructions about how the data is to be processed.

conflict detection. The process of detecting whether the same row was updated by users or application programs in both the source and target tables at essentially the same time.

configuration. The collection of brokers, their execution groups, the message flows, and sets that are assigned to them, and the topics and associated access control specifications.

continuous data replication. See *Enterprise Replication*.

continuous mirroring. A method of mirroring data where a change applied to a source table is immediately replicated to the target table. You should use continuous mirroring when it is important that source and target tables be synchronized at all times.

critical column. A change to a critical column indicates that the source operation is important and should be replicated to the target environment. If a source row is updated but no critical columns are changed, then this change is not replicated to the target table. The changes to the non-critical columns are replicated to the target the next time an important update is done.

data definition language. An SQL statement that creates or modifies the structure of a table or database, for example, CREATE TABLE, DROP TABLE, ALTER TABLE, or CREATE DATABASE.

data manipulation language. An INSERT, UPDATE, DELETE, or SELECT SQL statement.

data append. A data loading technique where new data is added to the database, leaving the existing data unaltered.

data cleansing. A process of data manipulation and transformation to eliminate variations and inconsistencies in data content. This process is typically done to improve the quality, consistency, and usability of the data.

data federation. The process of enabling data from multiple heterogeneous data sources to appear as though it is contained in a single relational database. Can also be referred to as *distributed access*.

data mart. An implementation of a data warehouse, typically with a smaller and more tightly restricted scope, such as for a department or workgroup. It can be independent, or derived from another data warehouse environment.

data mining. A mode of data analysis that has a focus on the discovery of new information, such as unknown facts, data relationships, or data patterns.

data partition. A segment of a database that can be accessed and operated on independently, even though it is part of a larger data structure.

data refresh. A data loading technique where all the data in a database is replaced with a new set of data.

data store. An instance of a InfoSphere CDC engine. Instances are created using the configuration tool provided for each engine.

data type. Defines the type of data stored in a specific database column, such as date, numeric, or character data. Significant differences in data types exist between different platforms' databases.

data warehouse. A specialized data environment developed, structured, and used specifically for decision support and informational applications. It is subject oriented rather than application oriented. Data is integrated, non-volatile, and time variant.

database partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

DB Connect. Enables connection to several relational database systems and the transfer of data from these database systems into the SAP Business Information Warehouse.

DDL. See *data definition language*.

debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

deploy. Makes operational the configuration and topology of the broker domain.

derived column. A column that is calculated from other source column values using a derived expression.

derived expressions. An expression that defines the value placed in a column for each row inserted or updated in a target table.

differential refresh. A process that synchronizes the target table with the current contents of the source table by applying only the differences between the target and the source table.

dimension. Data that further qualifies or describes a measure, or both, such as amounts or durations.

distributed application In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

DML. See *data manipulation language*.

drill-down. Iterative analysis, exploring facts at more detailed levels of the dimension hierarchies.

dynamic SQL. SQL that is interpreted during execution of the statement.

embedded database. A database that works exclusively with a single application or appliance.

engine. A program that performs a core or essential function for other programs. A database engine performs database functions on behalf of the database user programs.

enrichment. The creation of derived data. An attribute-level transformation performed by some type of algorithm to create one or more new (derived) attributes.

Enterprise Replication. An asynchronous, log-based tool for replicating data between IBM Informix database servers.

extenders. These itmes are program modules that provide extended capabilities for DB2 and are tightly integrated with DB2.

FACTS. A collection of measures, and the information to interpret those measures in a given context.

federation. Provides a unified interface to diverse data.

filtering columns. The process of identifying which source columns you want to include or exclude for replication.

gateway. A means to access a heterogeneous data source. It can use native access or ODBC technology.

grain. The fundamental lowest level of data represented in a dimensional fact table.

in-scope objects. Objects that are considered, or included, for replication to the backup system.

instance. A particular realization of a computer process. Relative to the database, the realization of a complete database environment.

Java Database Connectivity. An application programming interface that has the same characteristics as ODBC, but is specifically designed for use by Java database applications.

Java Development Kit. A software package used to write, compile, debug, and run Java applets and applications.

Java Message Service. An application programming interface that provides Java language functions for handling messages.

Java Runtime Environment. A subset of the Java Development Kit that enables you to run Java applets and applications.

journal control fields. A set of fields that contains different information about a journal entry associated with a source table change. Journal control fields can be assigned to target columns, as journal entry information is replicated with source table data. In Management Console, journal control fields are denoted with an ampersand (&) before the name of the field.

latency. The latency of the target table indicates how closely synchronized it is with the source table. For example, if 30 seconds ago on the source system have been applied to the target table, but some changes that were done in the last 30 seconds have not yet been applied, then the table would be 30 seconds latent.

LiveAudit. A feature that maintains an audit trail of table changes in the mapped target table.

Management Console. The administrator applications that provide the necessary support to configure, manage, and monitor an entire replication configuration from a central location.

materialized query table. A table where the results of a query are stored for later reuse.

measure. A data item that measures the performance or behavior of business processes.

member identifiers. An expression identifying the member in a multimember IBM i source table that is the source of replicated data. You require member identifiers to identify the member for each replicated record so that data in the target table is not erased during a refresh.

message domain. The value that determines how the message is interpreted (parsed).

message flow. A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing connectors.

message parser. A program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A parser is also responsible for generating a bit stream for an outgoing message from the internal representation.

metadata. The internal set of tables that maintains the entities, attributes, and characteristics of your replication configuration.

mirroring. The process of continuous replication of changed data from the source system to the target system

MOLAP. See *Multidimensional OLAP.*

multidimensional OLAP. Can be called MD-OLAP. It is OLAP that uses a multidimensional database as the underlying data structure.

multidimensional analysis. Analysis of data along several dimensions, for example, analyzing revenue by product, store, and date.

multidimensional clustering. A technique that allows for rows of data with similar values across multiple dimensions to be physically clustered together on disk.

multitasking. Operating system capability that allows multiple tasks to run concurrently, taking turns using the resources of the computer.

multithreading. Operating system capability that enables multiple concurrent users to use the same program. This situation saves on the impact of initiating the program multiple times.

nickname. An identifier that is used to reference the object at the data source that you want to access.

node group. Group of one or more database partitions.

node. An instance of a database or database partition.

notifications. A mechanism that generates a notification in response to a generated product message. A notification can be conveyed through different methods (email notification, user exit program notification, and so on). This mechanism allows a user to monitor replication activity in their network.

ODBC. See *Open Database Connectivity*.

OLAP. See *online analytical processing*.

online analytical processing. Multidimensional data analysis, performed in real time. Not dependent on an underlying data schema.

Open Database Connectivity. A standard application programming interface for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on various computers, even if each database management system uses a different data storage format and programming interface. ODBC is based on the call-level interface (CLI) specification of the X/Open SQL Access Group.

optimization. The capability to enable a process to run and perform in such a way as to maximize performance, minimize resource utilization, and minimize the process execution response time delivered to the user.

out of scope objects. Objects that are not considered for, or excluded from, replication to the target system.

partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

pass-through. The act of passing the SQL for an operation directly to the data source without being changed by the federation server.

persistent subscription. InfoSphere CDC ensures that a persistent subscription is restarted when the engine is restarted after shutdown or after the subscription ends with a recoverable error.

pivoting. Analysis operation where a user takes a different viewpoint of the results, for example, by changing the way the dimensions are arranged.

primary key. Field in a table that is uniquely different for each record in the table.

process. An instance of a program running in a computer.

program. A specific set of ordered operations for a computer to perform.

propagation control. The process of preventing the replication of data from a particular source. This process is useful if you are using a bidirectional replication configuration, and prevents subscriptions from unnecessarily repeating operations like inserting data.

pushdown. The act of optimizing a data operation by pushing the SQL down to the lowest point in the federated architecture where that operation can be run. More simply, a pushdown operation is one that is run at a remote server.

recursion. The process by which a perpetual series of repetitive changes applied to the same record occurs between two or more tables in a bidirectional replication scenario.

recursion prevention. When performing bidirectional replication, the ability to avoid a change to one table from being repetitively applied to both tables without termination.

refresh. A process that synchronizes the target table with the current contents of the source table.

refresh order. The order in which source tables are refreshed.

replication. The process of maintaining an on-going synchronization between the contents of source tables and target tables. Also the process of sending changes from source tables to the target system. In InfoSphere CDC, this term encompasses all methods of transferring data (refresh and mirroring).

replication method. A property of table mapping that indicates whether it performs mirroring or refresh.

Relational Sequential Access Method. The disk access method and storage manager for the Informix DBMS.

Relational OLAP. Multidimensional analysis using a multidimensional view of relational data. A relational database is used as the underlying data structure.

ROLAP. See *relational OLAP*.

roll-up. Iterative analysis, exploring facts at a higher level of summarization.

row filtering. The process of determining which rows in a source table to replicate. Typically, an expression tests one or more column values in each row and returns a Boolean result that replicates or discards the row.

RSAM. See *Relational Sequential Access Method*.

server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

Schedule End (Net Change) mirroring. Replicates changes (to the target) up to a point consistent with the state of the source database at a given point in time and then ends replication.

shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

source database. A database from which replication transfers captured changes.

static SQL. SQL that has been compiled before execution. Typically provides best performance.

subject area. A logical grouping of data by categories, such as customers or items.

subscription. A group of table mappings.

subscription promotion. The process of copying replication definitions from one environment to another environment without having to recreate the definitions.

subscription state. An indicator of the activity involving a subscription defined in Management Console. The subscription state indicates whether replication operations to a subscription are starting, refresh, mirror continuous, mirror net-change, ending, inactive, or unknown.

subscription status. The subscription status indicates how replication is progressing. The subscription status can be Normal, Error, or Unknown.

summarization. A method of applying data to a target table where numeric data in the table is incremented or decremented based on the type of row-level operation (insert, update, or delete) applied to the table. Summarization allows you to aggregate data without having to define expressions that maintains the equivalent results.

synchronous messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

system parameter. A setting that can be modified to change a certain behavior of InfoSphere CDC.

table mapping. A component of a subscription that connects a source replication object with a target replication object and specifies communication paths, such as queues or TCP/IP connections.

target database. A database to which replication software applies changes captured from a source database.

task. The basic unit of programming that an operating system controls. Also see *Multitasking*.

thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. Also see *Multithreading*.

throughput. A measure of the rate at which data changes are retrieved, sent, and applied on the target system.

transaction consistency. While mirroring data, transaction consistency is the guarantee that if part of a transaction has been committed to the target database, then all other in-scope operations from that source transaction have been committed as well.

transformation. The process of manipulating replicated data from a source to a target.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency.

user mapping. An association made between the federated server user ID and password and the data source (to be accessed) user ID and password.

virtual database. A federation of multiple heterogeneous relational databases.

warehouse catalog. A subsystem that stores and manages all the system metadata.

xtree. A query-tree tool that enables you to monitor the query plan execution of individual queries in a graphical environment.

Abbreviations and acronyms

ACS	access control system
ADK	Archive Development Kit
AIX	Advanced Interactive eXecutive from IBM
ALE	application link enabling
AMI	application messaging interface
API	application programming interface
AQR	automatic query rewrite
AR	access register
ARM	automatic restart manager
ART	access register translation
ASCII	American Standard Code for Information Interchange
ASM	Automatic Storage Manager
AST	application summary table
ATM	asynchronous transfer mode
B2B	business to business
BAM	business activity monitoring
BAPI	business application programming interface
BAS	business application services
BI	business intelligence
BIRA	business integration reference architecture
BIW	Business Information Warehouse (SAP)
BLOB	binary large object
BSM	business service management
BW	Business Information Warehouse (SAP)
CCMS	Computing Center Management System
CDB	customer database
CDC	change data capture
CDD	change data delivery
CL	command line
CLI	call level interface
CLOB	character large object
CLP	command-line processor
CPM	corporate performance management
CPU	central processing unit
CRM	customer relationship management
CSA	common storage area
CS-WS	Conversation Support for Web Services
DADx	document access definition extension
DB	database
DBA	database administrator
DB2	Database 2
DB2II	DB2 Information Integrator
DB2 UDB	IBM DB2 Universal Database™
DB2 II	DB2 Information Integrator
DBMS	database management system
DCE	distributed computing environment
DCM	dynamic coserver management
DCOM	distributed component object model

DDL	data definition language
DLL	dynamically linked library
DIMID	dimension identifier
DML	data manipulation language
DNS	domain name system
IBM DRDA®	IBM Distributed Relational Database Architecture™
DSN	data source name
DSS	decision support system
DTR	database table refresh
EAI	enterprise application integration
EAR	enterprise archive
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDA	enterprise data architecture
EDA	event-driven architecture
EDU	engine dispatchable unit
EDW	enterprise data warehouse
EGM	enterprise gateway manager
EII	enterprise information integration
EIS	enterprise information system
EJB	Enterprise JavaBeans
ER	enterprise replication
ERP	enterprise resource planning
ESB	enterprise service bus
ESE	Enterprise Server Edition
ETL	extract transform and load
FDL	flow definition language
FTP	file transfer protocol
Gb	gigabits
GB	gigabytes
GUI	graphical user interface
HDR	high availability data replication
HPL	high performance loader
HTML	hypertext markup language
HTTP	hypertext transfer protocol
HTTPS	hypertext transfer protocol secure
I/O	input/output
IANA	Internet Assigned Numbers Authority
IBM	International Business Machines Corporation
ID	identifier
IDE	Integrated Development Environment
IDS	Informix database server
II	information integration
IIOP	Internet Inter-ORB Protocol
IBM IMS™	IBM Information Management System
ISAM	Indexed Sequential Access Method
InfoSphere CDC	InfoSphere Change Data Capture
ISV	independent software vendor
IT	information technology
ITR	internal throughput rate
ITSO	International Technical Support Organization
IX	index
J2C	J2EE Connector
J2EE	Java 2 Platform Enterprise Edition
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JE	Java Edition
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment

JSP	JavaServer Pages
JSR	Java Specification Requests
JTA	Java Transaction API
JVM	Java Virtual Machine
KB	kilobyte (1024 bytes)
LDAP	lightweight directory access protocol
LOB	large object
LOB	line of business
LPAR	logical partition
LRSN	logical record sequence number
LV	logical volume
Mb	megabits
MB	Megabytes (1,048,576 bytes)
MC	Management Console
MD	master data
MDC	multidimensional clustering
MDI	master data integration
MDM	master data management
MIS	management information system
MVC	model-view-controller
MQT	materialized query table
MPP	massively parallel processing
MRM	message repository manager
NFS	network file system
NPI	non-partitioning index
ODA	object discovery agent
ODBC	open database connectivity
ODS	operational data store
OLAP	online analytical processing
OLE	object linking and embedding
OLTP	online transaction processing
ORDBMS	object relational database management system
OS	operating system
PDS	partitioned data set
PIB	parallel index build
PSA	persistent staging area
RAC	real application clusters
RBA	relative byte address
RDBMS	relational database management system
RDP	rapid deployment package
RID	record identifier
RMI	remote method invocation
RR	repeatable read
RS	read stability
SAN	storage area network
SAX	Simple API for XML
SCD	slowly changing dimensions
SDK	software developers kit
SMIT	Systems Management Interface Tool
SMP	symmetric multiprocessing
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
SPL	stored procedure language
SSH	Secure Shell
SQL	structured query
TNS	transparent network substrate
TS	transformation server
TS	terminal server
TS	table space
TSO	IBM Time Sharing Option
UDDI	Universal Description, Discovery, and Integration of Web Services
UDF	user-defined function
UI	user interface
UID	user identifier
UDR	user-defined routine

URL	uniform resource locator
UTC	Coordinated Universal Time
VG	volume group (RAID disk terminology).
VLDB	very large database
VPN	virtual private network
VTI	virtual table interface
W3C	World Wide Web Consortium
WAR	web archive
WLM	workload management
WORF	Web Services Object Runtime Framework
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WS-I	Web Services Interoperability Organization
WSIC	Web Services Choreography Interface
WSIF	Web Services Invocation Framework
WSIL	Web Services Inspection Language
WSMF	Web Services Management Framework
WWW	World Wide Web
XBSA	X-Open Backup and Restore APIs
XML	eXtensible Markup Language
XSD	XML Schema Definition

Related publications

The publications listed in this section are considered suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Some publications referenced in this list might be available in softcopy only.

- ► *Implementing IBM InfoSphere Change Data Capture for DB2 z/OS V6.5*, REDP-4726
- ► *TCP/IP Tutorial and Technical Overview*, GG24-3376-06

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, and order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- ► BM InfoSphere Data Replication change data capture

  http://ibm.com/software/data/infosphere/change-data-capture/

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A
Access Manager 104–105, 112, 114, 178, 264, 272, 276, 280
Access Server 14, 106, 110, 119, 149, 151, 153–157, 231, 233–236, 263, 265, 270–274, 276–277, 279–280, 282, 284, 309, 338, 347, 349
activate changed tables 202
Active data warehousing 24
Adaptive Apply 76, 91, 93, 99, 185, 299, 384, 401
Adaptive Apply replication 93
Admin API 14
Analyze performance statistics 217
API classes 264
API Javadocs 263
application consolidation 1
Application Integration 53
Application Synchronization 53
Apply agent 14, 16
apply method 76, 82, 84, 299
Apply methods 91
  Adaptive Apply replication 93
  Consolidation One to Many 94
  Consolidation One to One 94
  LiveAudit replication 92
  Row consolidation 94
  Summarization replication 93
archive transaction logs 257
ASCII 83

B
BAM 64
  Also see business activity monitoring
Batch Framework 43
BEA WebLogic 64
Before Delete user exit 401
before-delete event 392
BI 3, 24, 45–46, 67, 212
  Also see business intelligence
bi-directional mirroring 53
Bi-directional replication 52, 100
bi-directional replication 15, 19, 53, 77
bi-directional topology 53
BIRT
  Business Intelligence and Reporting Tools 43
bookmark 17, 35, 68, 88, 115, 129–131, 134, 182–185, 190, 198, 208, 227–229, 237, 303, 338–339, 383
Bookmark information 122
bottlenecks 215
BPM 64
  Also see business process management
Bulk Data Delivery 11
Bulk Load 12
business activity monitoring 64
business agility 10
business intelligence xii, 3, 25, 45, 67
Business Intelligence and Reporting Tools 43
business process management 4, 64
business use cases 19

C
cascading integration 50
Cascading replication 49, 226
cascading replication 221, 226
CDC
  messages queues 37
CDD 98
Center of Excellence xiii
Change Data Capture 29
change data capture xi
Change Data Delivery 98
change management
  Oracle environment 209
change management automation 198
change management environment 193
change management scenario
  no service window 207
  service window 193
change tables activation 202
change tracking interface xi
CLI 106–107, 111, 120, 237, 246, 250, 338
Coded Character Set Identifier 147
column filtering 15, 83, 288, 303, 309–310, 339
column functions 77–78, 82–83, 89, 100
column level filtering 90
column mapping 83, 288, 312, 314, 378, 401

Command line interface 14
Communication Initialization and Termination 133
Communication Layer 15
compilation command 235
composite events 73
concatenation 78
Configuration changes 85
Configuration metadata 170
configuration metadata tables 283
Configuration Tool 109
conflict and detection resolutions
  Largest Value Wins 100
  Smallest Value Wins 100
  Source Wins 100
  Target Wins 100
  User Exit 100
conflict detection 53, 76, 99, 184
Conflict detection and resolution 99
conflict detection and resolutions
conflict detections 15
conflict resolution 76
Consolidation 28–29, 76, 82, 91, 94
Consolidation One to Many 95
Consolidation One to One 94
Consolidation Replication 54
Consolidation Server 33
Context-rich 73
continuous analysis 15
continuous availability 3
continuous bi-directional synchronization 60
continuous business availability 12
continuous business monitoring 4
continuous capture 227–230
Continuous Capture feature 227
continuous capture process 228
Continuous Mirror 343
continuous mirror mode 238, 344
Continuous mirroring 87, 340
  Abort 88
  Immediate 88
  Normal 88
  Scheduled End 88
continuous mirroring 12, 81, 248, 367
Continuous mode 83
continuous scraping 221, 228
Creating a subscription 284
CRM 37, 309
  process 37
  system 37
CRM system 38
Cross database version replication 48
Custom apply methods 96
Customer Matching Critical Data Rule UI 43

D

data consolidation 30–31
data distribution 3, 57
data distribution replication 57
Data functions 79
data integration xii, 2–3, 7, 9, 39, 53
  optimized 3
data manipulation language 29
data marts 45
data normalization 28
data redundancy 48
Data Replication xi, 1, 75, 103, 137, 211, 231, 431
data replication process 29, 215–216
Data Stewardship UI 43
data synchronization 48, 53
data transformations 78, 82, 214
data warehouse appliance 64
data warehousing xii, xiv, 1–2, 24, 94
database xii
database log 16, 88, 129, 159, 161, 171, 174, 176, 190, 194, 198, 207, 227–228, 230, 288–289, 340
Database Log Scraper 133
database logs 12–13, 15, 44, 126, 132, 159, 162–163, 166, 207, 210, 213–214, 224, 227–230, 338
Database migration 35
Database Staging 21, 68
Database Table Change 134
Database Table Refresh 134
database transaction logs 257
DataMirror xiii–xiv
DataStage 16, 25–26, 43–44, 55, 62, 67–69, 96–97, 104, 107, 184, 282, 300, 344, 412–413, 415, 419, 424
DataStage engine 96
DataStage jobs 16
Datastore 15, 228, 249, 273–275, 280–282, 347, 362
datastore role 277
date and time 78, 87–88, 184, 340–341, 359
DB2 for Linux, UNIX, and Windows xv, 27
DB2 IFI 262
  Also see DB2 instrumentation facility interface

DB2 instrumentation facility interface 13, 160
DB2 stored procedure 28
DB2/400 xiii, xv, 289
DB2/z logs 13
DBPath object 290
DBTable object 290
DDL changes refresh 198
DDL operations
  DB2 on z/OS 209
  InfoSphere CDC for z/OS 210
DDL statements
  Linux, UNIX, and Windows engine 207
Deassign Target Tables 303
default connection parameters 276
delete a subscription 308
Deliver Pillar 7, 9
delta changes 35, 42–43
Denormalization 78
denormalize 81
deploy subscription changes 338
deploying subscription changes 191
derived column 78, 146, 313–314
derived columns 214, 339
derived expression 78–79, 313, 370
derived expression user exit 377
Derived expression user exits 382
derived expression user exits 373
derived expressions 77–78, 80, 84, 87, 130, 214, 269
Differential Refresh 85–87
  Only Log Differences 87
  Refresh and Log Differences 87
  Refresh Only 87
  requirements 87
Differential refresh 86
differential refresh 85–87
Direct Connect 67, 69, 96, 98, 420
discrete data events 72
DML 29, 208–209
  also see data manipulation language
Dynamic data 10
dynamic data 9–10, 12
Dynamic Warehousing xi

E
e-Commerce application 40
EDA 72
EDW 27–28
  Enterprise Data Warehouse 27
electronic point of sale 32
encoding conversion 318
encoding conversions 15, 287, 315, 318
Encoding method 318
Enterprise Data Warehouse 27
Enterprise Service Bus 37
EPOS 32
  electronic point of sale 32
ERP 30–31
ERP systems 30–31
ESB 37
  Also see Enterprise Service Bus
ETL 9, 12, 20, 22–23, 25–26, 41–43, 46, 55, 62, 64, 68–70, 76, 91, 93, 96, 212
ETL engine 96
ETL staging tables 26
Event Log 127
event log 215
event monitoring 367
event server 37, 68
Event-Driven 71
event-driven xi, 71
Event-Driven Architecture 71–72
event-driven architecture 62
external monitoring solution 251
extracting data 55

F
fault tolerant architecture 16
Federation 12
File based integration 23
filtering 89
  column level 89
  row level 89
Firewalls 152
Flat File 23, 62, 96, 98, 413, 424
  considerations 64
Flat Files 64
Flat files
  Configurable options 97
Fraud Prevention 64

H
heterogeneous data 6, 9
heterogeneous data sources 7
Hub-and-Spoke 58

I
IBM Cognos 55
  Data Manager 55
IBM DataMirror® xiii
IBM DB2/z 25
IBM InfoSphere DataStage 16, 282
IBM InfoSphere Information Server 4–5
IBM JDK 371
IBM Resource Access Control Facility (RACF®) 238
IBM WebSphere 64
IBM® InfoSphere® Data Replication xi
IFI 13, 160, 262
  Also see instrumentation facility interface
import subscriptions from XML file 280
Incremental Data Delivery 11–12
Incremental data delivery 13
Information Server 4–5, 9, 31, 42–44, 72
  architecture strategy 4
  Metadata-driven integration 4
  scalable 4
InfoSphere
  Proof of Concepts xiii
InfoSphere CDC 14, 20, 119–120, 153, 170, 213, 264, 366
  Access Server 119
  alternative architectures 221
  architecture 211
  bi-directional functionality 37
  binaries and configuration metadata 171
  change management workflow 190
  code page conversions 147
  Command Line Interface 232
  communications component 134
  configuration metadata 130
  daemon process 239
  DB2 on System i engine 238
  engine commands 237
  engine commands on z/OS 238
  event log 118
  Event Server 300
  firewalled network environment 153
  high availability 168
  high availability solutions 181
  in an IASP environment 180
  instance 241
  large environments 211
  log reader 15
  Log shipping 166
  major components 213
  Management Console 15, 49, 106, 111, 120, 151, 154, 216, 262–263, 270, 284
  Management Console GUI 232
  metadata 325
  N-tier architecture 224
  N-tier data replication 224
  on a shared volume 171
  on separate instances 175
  On separate nodes - database shared 173
  on System i 178
  operational information 171
  Oracle system parameters 167
  Oracle Trigger-based 301
  potential bottlenecks 211
  Remote processing capabilities 162
  remote target environment 164
  silent install 217
  target engine 303
  TCP/IP application protocol 149
  unidirectional functionality 37
  virtual IP 169
  workflow 216
InfoSphere CDC 6.3 318, 321, 350
InfoSphere CDC 6.5 315, 317, 321, 341, 350, 367
InfoSphere CDC 6.5+ engines 317
InfoSphere CDC API 106, 119, 263, 314, 338
InfoSphere CDC bookmarks 191
InfoSphere CDC command tools 109
InfoSphere CDC engine on z/OS 115
InfoSphere CDC Event Server 96, 98, 107
InfoSphere CDC for z/OS
  log parser 132
  log reader 132
  scraper component 133
  Single scrape 132
  source flow 133
InfoSphere CDC instance 15, 104–105, 107, 110, 163, 167, 171–173, 217–218, 221, 223, 237, 239
InfoSphere CDC Management Console GUI 106, 233
InfoSphere CDC metadata 105, 130, 132, 134, 210, 219, 268
InfoSphere CDC Mirror Moderator 123
InfoSphere CDC mirroring operation 123
InfoSphere CDC replication system
  Access control 111
  Command-Line Interface 111
  Configuration 111

Monitoring 111
Operation 111
InfoSphere CDC Server CLI 120
InfoSphere CDC Server command line interface 106
InfoSphere CDC source engine 98, 104, 114, 227, 237, 303, 319, 367
InfoSphere CDC subscriptions 116
InfoSphere CDC target engine 104, 133, 237
InfoSphere CDC types of replication 114
mirroring 114
refresh 114
InfoSphere CDC z/OS
address space 187
failover 188
Resiliency 187
resiliency scenarios 188
subscriptions 188
suspension 189
TCP/IP configuration 188
InfoSphere Change Data Capture xii–xiv, 7, 16, 29, 31–35, 37, 39–41, 43, 45, 53, 62, 71, 135, 235, 240, 243–244, 247, 261, 363–364
InfoSphere Change Data Delivery xii
InfoSphere Data Replication xi, 9
change data capture solution 1, 75, 103, 137, 211, 231, 431
InfoSphere DataStage 16, 96–97
InfoSphere DataStage Direct Connect 24
InfoSphere Foundation Tools xv, 6
InfoSphere Information Server 4–5
InfoSphere® Data Replication xi
instrumentation facility interface 13, 160
intra-system data replication 214
iReflect xiv
iSeries 32–33, 178

J
Java API 234
Java API methods 267
Java Development Kits 371
Java user exit 383
JDBC refresh operations 124
JMS Message Queue 62–64
JMS message queues 98
JMS Messages 16
JMS provider 184
JMS queue 104
journal control fields 27, 29, 80, 87, 89, 100, 398
journal receivers 182

L
Large Object 84, 100
large object 83
Latency 68, 212, 214, 216, 351, 357–358, 369
latency 3, 15, 27–28, 38, 53, 55, 67, 69, 88, 127, 158, 160, 208, 212, 214–216, 221–222, 226, 229, 232, 246, 346–347, 351, 355, 357, 365, 369
latency alerts 118
latency notifcation thresholds 369
latency thresholds for notification 115
Linux, UNIX, and Windows engine activity checking 246
Linux, UNIX, and Windows engine event log monitoring 250
Linux, UNIX, and Windows engine log maintenance 257
Live Audit 29
LiveAudit replication 92
One table mapping 92
LOB 83–84, 100, 384
Also see large object
locating the target datastore 282
log cache 250
log parser 15, 122, 124, 213, 431
log reader 15, 122, 124, 131, 190, 210, 213–214, 223, 227–229, 325, 431
log reader bottlenecks 214
Log retention
DB2 for Linux, UNIX, and Windows 159
Oracle platforms 160
SQL Server 161
Sybase 161
System I 159
z/OS 160
log retention policy 229
log scraper 12, 191, 228, 250
log scraper agent 16
Log shipping 166
log shipping 162

M
mainframe 5, 13, 25, 41, 148
DB2 logs 41
Maintenance operations 85
Management Console 14–15, 49, 77, 90–92, 97,

104–107, 109–111, 114, 116, 130, 132, 147, 149, 151, 153–155, 169–170, 178, 194, 203, 231–233, 242, 250, 257, 262–264, 266, 272, 276–277, 300, 302–303, 311, 319, 322, 325, 332, 337–338, 343, 346, 357, 365, 368–369, 372, 401
    Monitoring Interface 117
    monitoring tool 215
    performance monitoring 215
Management Console Configuration interface 117
Management Console GUI 283
Management Console Monitoring interface 118
Map Tables wizard 49, 99
Mark Table Capture Point 35
Marketing xii
Master Data Management xi, 40
master data management 1, 40
master table 82, 370
MBCS 287
    Also see multi-byte character set
MC commands interface 233
MC GUI 106
MDM 3, 9, 40–43
    Also see Master Data Management
message queue 12, 22, 68, 96, 98, 107, 253, 368, 402
Metadata 15, 132, 174, 177
Metadata-driven integration 4
Mirror 15, 29, 125, 229, 300, 335, 340, 343–344
Mirror Continuous 114
Mirror Continuous mode 346
mirrored mode 76
mirroring 15, 48, 53, 63, 81, 83, 85, 87–88, 93, 97, 125, 128, 131, 134, 159, 169, 179, 182, 184–185, 189, 198–199, 248, 340–342, 433
mirroring continuous 129
mirroring replication 118
MQ 22, 68, 98
    Also see message queue
MS Windows 33
multi-byte character set xii, 287
multiple InfoSphere CDC instances 221
multiple parallel subscriptions 221

N
Net change 343
Net Change mirroring 340
N-tier architecture 224
N-tier data replication 224

O
ODS 45–46, 55, 91
    operational data store 45, 55
OLTP 24–25
    Also see Online Transactional Processing
one-to-one replication 49
Online Gift Registry 40
Online Transactional Processing 24
operational BI 45
Operational business intelligence 45
operational data store 45, 55, 322
Operational information 170
optimization 211
optimized data integration 3
Oracle xv
Oracle change management 209
Oracle Data Guard log 167
Oracle log shipping considerations 168
Oracle logs 163
Oracle Trigger version 124

P
parallel refresh 85
Parked tables 84
Party Maintenance UI 43
performance bottlenecks 214
performance monitoring 215
performance tuning 211
persistent subscriptions 367
PointBase xiii, 130
propagation control 60
publication servers 54
publication table definitions 54
PublishedTable objects 288
Publisher class 267
publisher ID 286
Publisher replication role 277
purging procedures 257

Q
Query and Reporting 31
Queue Targeting Mapping Designer 63

R
R3load export 36
Rapid Deployment Package 41
RDBMS 64

Also see relational database management system
RDP 41
Also see Rapid Deployment Package
real-time xi–xii, 1, 4, 6, 9, 11–12, 27, 31–32, 37, 39–40, 43, 45, 47, 53, 56, 63, 69–72, 159, 166, 215, 217
analytics 9
dashboards 4
data 11
flow of data changes 9
operational BI 45
recursion prevention 52
Redbooks Web site 449
Contact us xvii
redo logs 213
Referential Integrity 86
Refresh 15, 35–36, 85, 87, 114, 125, 216, 290, 300, 343–344, 364
refresh 15, 35, 56, 69–70, 76, 81, 83–87, 93, 97, 101, 132, 134, 176, 198, 207–209, 227, 289, 294, 338, 364–365
Refresh mode 83
refresh mode 76
refresh operation 86, 97, 114, 134
Differential Refresh 114
mirroring 114
snapshot 114
refresh operations with JDBC 124
Refresh Order 85
registering a socket 149
relational database management system 64
Remote apply 165
Remote log reading 163
Remote source and target engine 165
remote source scrape 162
remove subscription 304
remove table mappings 308
remove tables 304
Remove Tables from Catalog 304
removing mapped tables 303
Replication Engine 15
replication method 289
Mirror 289
Refresh 289
replication source tables 290
ReplicationRole interface 277
Reporting server 56
Reporting UI 43
retrieve the Publication object 295
RFID tag xi
RI 86, 115
rollback strategy 53
row filtering 15, 116, 221, 288, 303, 309, 311, 339, 370, 373, 382
row identifier 54
Row level filtering 89
Row-level user exits 396, 402
row-level user exits 402

S
Sample User Interface 43
SAP
heterogeneous system copy 35
migration monitor 36
R3load 35
SAP R3load 35
SAP R3load Utility 36
scalability xi
SCD 26, 28
Also see Slowly Changing Dimensions
Type 2 28
Scheduled End 33
Net Change 83
scheduled end 182
Scheduled End mirroring 88, 340
Current time 88
User specified date and time 88
User specified log position 88
Scheduled End mode 83
Server 64
Service Oriented Architecture 72
service oriented architecture 1, 62, 71
service windows 191
shadow database 20
Shared scrape 128
short message service 37
SIF 42
Also see Standard Interface Format
SIF Parser 43
SIF Sequencer 43
Single Scrape 15, 124–125, 127, 431, 433
single scrape events 362
single scrape optimization 124
Single Scrape staging store 125
single scrape staging store 228
single use systems 47

Slowly Changing Dimensions 26
smarter planet xi
smarter solutions xi
SMS 37
  Also see short message service
SOA 1, 72
  Also see service oriented architecture
Soft Deletes 95
Source Capture Engine 15
source database log 114, 122, 124
Source database logs 15
Source datastore 234
Source transformation engine 15
source-only log reader 15
spoke-to-hub subscription 59
SQL Server xiii–xv, 32–33, 110, 165, 289
staging area table 21
staging store 213, 227
staging store disk quota 125
Standard Interface Format 42
Standard Refresh 85–86
standard refresh 85
standard replication 48, 76, 91, 93–94
  row insert operation 91
start/stop Linux, UNIX, and Windows engine 239
start/stop System i engine 241
start/stop System z 243
state of a subscription 343
stored procedure user exit 394
Stored procedure user exits 373
stored procedures 79, 373, 383
Stream data 91
Streaming 10
SubscribedTable objects 288
Subscriber class 267
Subscriber replication role 277
Subscription 115
subscription changes
  deployment 191
subscription configuration packages 267
subscription import named index 198
Subscriptionbookmark 191
Subscription-level user exits 82
subscription-level user exits 402
Summarization Apply replication 94
supplemental logging 301
synchronize changed data 26
System i 182
  failover 184
  journal receiver 182
  Software-based HA solution 183
System i activity checking 248
  TCPLISTEN job 248
System i event log monitoring 253
System i journal receiver maintenance 262
System i RPG user exit 396
System i user exit derived expressions 380
System i user exit programs 371
system monitoring tool 215
System z activity checking 249
System z database log maintenance 262
System z event log monitoring 257
System z user exit programs 372

T
table assignment processing 339
table capture point bookmark 191
table capture points 338
table mapping example
  Oracle to DB2 289
table mapping process 287
Table-level user exits 82
table-level user exits 396
target apply engine 12
target datastore engine 15
Target Engine 15
target InfoSphere CDC engine 287
target table changes 191
Target transformation engine 15
TCP/IP network socket 12
Tibco 64
Time zone considerations 138
transaction conflicts 53
transaction log 20, 81, 97, 161, 227, 267
transaction logs 213
Transaction Stage 70
transactional integrity 70
transformation engine 15
transformational capabilities 77
transformations 12, 75, 78, 82, 85, 96, 98, 133, 193, 214
trigger xi

U
UEModuloFilter65 373
UESoftDelete class 393
UI 43

Customer Matching Critical Data Rule 43
Data Stewardship 43
Party Maintenance 43
Sample User Interface 43
unattended deployment 219
Uncommitted transactions 213
Unicode handling 318
Uni-directional replication 48
update table definitions 325
User exit 79, 371
user exit 28, 43, 80, 82, 96, 99–100, 130, 222, 226, 253, 313, 322, 324–325, 368, 370–372, 376–379, 383–386, 388, 392–394, 409–412, 414, 416, 424–425
User exit programs 372, 383
User Exits
Full Unit of Work 96
Soft Delete 96
Web Service 96
User exits 82
Subscription-level 82
Table-level 82
user exits 15, 27–28, 78, 96, 133, 222, 231, 244, 269, 303, 322, 324–325, 369–370, 372, 393
user functions 79

V
value translations 15
Virtual Data Delivery 11
virtual IP 169

W
WebSphere ESB 64
WebSphere MQ 22, 64, 69
connector 22, 69
Server 64
Workload Balancing 53
workload balancing 53
workload distribution 31–32, 53

X
XML messages 63

Z
zero downtime migration 53

IBM

Redbooks

Smarter Business: Dynamic Information with IBM InfoSphere Data Replication CDC

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

Smarter Business
Dynamic Information with IBM InfoSphere Data Replication CDC

Log-based for real-time high volume replication and scalability

High throughput replication with integrity and consistency

Programming-free data integration

To make better informed business decisions, better serve clients, and increase operational efficiencies, you must be aware of changes to key data as they occur. In addition, you must enable the immediate delivery of this information to the people and processes that need to act upon it. This ability to sense and respond to data changes is fundamental to dynamic warehousing, master data management, and many other key initiatives. A major challenge in providing this type of environment is determining how to tie all the independent systems together and process the immense data flow requirements. IBM InfoSphere Change Data Capture (InfoSphere CDC) can respond to that challenge, providing programming-free data integration, and eliminating redundant data transfer, to minimize the impact on production systems.

In this IBM Redbooks publication, we show you examples of how InfoSphere CDC can be used to implement integrated systems, to keep those systems updated immediately as changes occur, and to use your existing infrastructure and scale up as your workload grows. InfoSphere CDC can also enhance your investment in other software, such as IBM DataStage and IBM QualityStage, IBM InfoSphere Warehouse, and IBM InfoSphere Master Data Management Server, enabling real-time and event-driven processes. Enable the integration of your critical data and make it immediately available as your business needs it.

SG24-7941-00 ISBN 0738436372