# **Replication Overview**

Microsoft® SQL Server<sup>™</sup> 2000 replication is a set of solutions that allow you to copy, distribute, and potentially modify data across your enterprise. SQL Server 2000 includes several methods and options for replication design, implementation, monitoring, and administration to give you the functionality and flexibility needed for distributing data and maintaining data consistency.

Торіс	Description
Introducing Replication	Describes how replication can be used in various business environments. Explains the SQL Server 2000 replication model, the types of replication, and how replication works.
<u>Planning for Replication</u>	Provides the information needed to make critical decisions when creating a replication plan, including: business considerations, enterprise data needs, network considerations, and designing a replication topology.
Types of Replication	Details the types of replication (snapshot replication, transactional replication, and merge replication). Discusses the components of each type, how each type works, the architecture, and the benefits and strengths of each type.
<u>Replication Tools</u>	Describes the primary methods used to implement replication, including using the replication wizards, using system stored procedures, programming a replication application with Microsoft ActiveX® controls, and using tools such as Windows Synchronization Manager.
Implementing Replication	Describes the steps for implementing replication using the replication tools. Includes configuring Publishers and Distributors, creating publications, creating different types of subscriptions, replicating between instances of

	SQL Server 2000 and instances of SQL Server version 7.0 and earlier.
Replication Options	Explains the options available with each type of replication, including filtering published data, publishing database objects, immediate updating, queued updating, and transforming published data.
Replication Data Considerations	Includes handling <b>identity</b> columns, <b>timestamp</b> data, <b>uniqueidentifiers</b> and data types.
Administering and Monitoring Replication	Details the agents used during replication, replication alerts, validating data at the Subscriber, strategies for monitoring replication, and remote agent activation.
<u>Replication and</u> <u>Heterogeneous Data</u> <u>Sources</u>	Describes how you can replicate data between heterogeneous data sources (such as DB2, Oracle, Microsoft Access, or Microsoft Exchange), how to publish to heterogeneous Subscribers, and how to subscribe to heterogeneous Publishers.
Replication Security	Discusses security access layers and replication, Internet security issues, snapshot folder security, agent login security, role requirements, and security for updatable subscriptions.
Enhancing Replication Performance	Provides techniques for optimizing replication performance including enhancement techniques for each type of replication, effective data partitioning, and Distributor options.
<u>Backing Up and</u> <u>Restoring Replication</u> <u>Databases</u>	Describes strategies for backing up replication databases, how to restore each type of replication, and restoring backups of replicated databases.
<u>Getting Started with</u> <u>Replication</u> <u>Programming</u>	Discusses programming replication including using ActiveX controls, SQL-DMO, and the Replication Distributor Interface.

# **Introducing Replication**

Microsoft® SQL Server<sup>™</sup> 2000 replication is a set of technologies for copying and distributing data and database objects from one database to another and then synchronizing between databases for consistency.

Using <u>replication</u>, you can distribute data to different locations, to remote or mobile users over a local area network, using a dial-up connection, and over the Internet. Replication also allows you to enhance application performance, physically separate data based on how it is used (for example, to separate online transaction processing (OLTP) and decision support systems), or distribute database processing across multiple servers.

## **Benefits of Replication**

Replication offers various benefits depending on the type of replication and the options you choose, but the common benefit of SQL Server 2000 replication is the availability of data when and where it is needed.

Other benefits include:

- Allowing multiple sites to keep copies of the same data. This is useful when multiple sites need to read the same data or need separate servers for reporting applications.
- Separating OLTP applications from read-intensive applications such as online analytical processing (OLAP) databases, data marts, or data warehouses.
- Allowing greater autonomy. Users can work with copies of data while disconnected and then propagate changes they make to other databases when they are connected.
- Scale out of data to be browsed, such as browsing data using Web-based applications.

- Increasing aggregate read performance.
- Bringing data closer to individuals or groups. This helps to reduce conflicts based on multiple user data modifications and queries because data can be distributed throughout the network, and you can partition data based on the needs of different business units or users.
- Using replication as part of a customized standby server strategy. Replication is one choice for standby server strategy. Other choices in SQL Server 2000 include log shipping and failover clustering, which provide copies of data in case of server failure.

#### When to Use Replication

With organizations supporting diverse hardware and software applications in distributed environments, it becomes necessary to store data redundantly. Moreover, different applications have different needs for autonomy and data consistency.

Replication is a solution for a distributed data environment when you need to:

- Copy and distribute data to one or more sites.
- Distribute copies of data on a scheduled basis.
- Distribute data changes to other servers.
- Allow multiple users and sites to make changes then merge the data modifications together, potentially identifying and resolving conflicts.
- Build data applications that need to be used in online and offline environments.
- Build Web applications where users can browse large volumes of data.

• Optionally make changes at subscribing sites that are transparently under transactional control of the <u>Publisher</u>.

# **Replication Model**

Microsoft® SQL Server<sup>™</sup> 2000 replication uses a publishing industry metaphor to represent the components and processes in a <u>replication topology</u>. The model is composed of the following: Publisher, Distributor, Subscribers, Publications, articles, and subscriptions.

There are also several replication processes that are responsible for copying and moving data between the Publisher and Subscriber. These are the Snapshot Agent, Distribution Agent, Log Reader Agent, Queue Reader Agent, and Merge Agent. For more information about the agent processes, see <u>Agents and Monitors</u>.

### Publisher

The Publisher is a server that makes data available for replication to other servers. The Publisher can have one or more publications, each representing a logically related set of data. In addition to being the server where you specify which data is to be replicated, the Publisher also detects which data has changed during transactional replication and maintains information about all publications at that site.

## Distributor

The Distributor is a server that hosts the distribution database and stores history data, and/or transactions and meta data. The role of the Distributor varies depending on which type of replication you implement. For more information, see Types of Replication.

A remote Distributor is a server that is separate from the Publisher and is configured as a Distributor of replication. A local Distributor is a server that is configured to be both a Publisher and a Distributor of replication.

### Subscribers

Subscribers are servers that receive replicated data. Subscribers subscribe to publications, not to individual articles within a publication, and they subscribe

only to the publications that they need, not all of the publications available on a Publisher. Depending on the type of replication and replication options you choose, the Subscriber could also propagate data changes back to the Publisher or republish the data to other Subscribers.

## Publication

A publication is a collection of one or more articles from one database. This grouping of multiple articles makes it easier to specify a logically related set of data and database objects that you want to replicate together.

## Article

An article is a table of data, a partition of data, or a database object that is specified for replication. An article can be an entire table, certain columns (using a vertical filter), certain rows (using a horizontal filter), a stored procedure or view definition, the execution of a stored procedure, a view, an indexed view, or a user-defined function.

## Subscription

A subscription is a request for a copy of data or database objects to be replicated. A subscription defines what publication will be received, where, and when. Synchronization or data distribution of a subscription can be requested either by the Publisher (a <u>push subscription</u>) or by the Subscriber (a <u>pull subscription</u>). A publication can support a mixture of push and pull subscriptions.

## See Also

**Implementing** Replication

Publishers, Distributors, and Subscribers

Subscribing to Publications

# **Introducing the Types of Replication**

There are three types of replication available with Microsoft® SQL Server<sup>™</sup> 2000: snapshot replication, transactional replication and merge replication.

## **Snapshot Replication**

Snapshot replication is the process of copying and distributing data and database objects exactly as they appear at a moment in time. Snapshot replication does not require continuous monitoring of changes because changes made to published data are not propagated to the Subscriber incrementally. Subscribers are updated with a complete refresh of the data set and not individual transactions. Because snapshot replication replicates an entire data set at one time, it may take longer to propagate data modifications to Subscribers. Snapshot <u>publications</u> are typically replicated less frequently than other types of publications.

Options available with snapshot replication allow you to filter published data, allow Subscribers to make modifications to <u>replicated data</u> and propagate those changes to the <u>Publisher</u> and then to other Subscribers, and allow you to transform data as it is published.

Snapshot replication can be helpful in situations when:

- Data is mostly static and does not change often.
- It is acceptable to have copies of data that are out of date for a period of time.
- Replicating small volumes of data.
- Sites are often disconnected and high latency (the amount of time between when data is updated at one site and when it is updated at another) is acceptable.

### **Transactional Replication**

With transactional replication, an initial snapshot of data is propagated to Subscribers, and then when data modifications are made at the Publisher, the individual transactions are captured and propagated to Subscribers.

SQL Server 2000 monitors INSERT, UPDATE, and DELETE statements, and changes to stored procedure executions and indexed views. SQL Server 2000 stores the transactions affecting replicated objects and then it propagates those changes to Subscribers continuously or at scheduled intervals. Transaction boundaries are preserved. If, for example, 100 rows are updated in a transaction, either the entire transaction with all 100 data modifications are accepted and propagated to Subscribers or none of them are. When all changes are propagated, all Subscribers will have the same values as the Publisher.

• Options available with transactional replication allow you to filter published data, allow users at the Subscriber to make modifications to replicated data and propagate those changes to the Publisher and to other Subscribers, and allow you to transform data as it is published.

Transactional replication is typically used when:

- You want data modifications to be propagated to Subscribers, often within seconds of when they occur.
- You need transactions to be atomic (either all or none applied at the Subscriber).
- Subscribers are mostly connected to the Publisher.
- Your application will not tolerate high latency for Subscribers receiving changes.

## **Merge Replication**

Merge replication allows various sites to work autonomously (online or offline) and merge data modifications made at multiple sites into a single, uniform result at a later time. The <u>initial snapshot</u> is applied to Subscribers and then SQL Server 2000 tracks changes to published data at the Publisher and at the

Subscribers. The data is synchronized between servers either at a scheduled time or on demand. Updates are made independently (no commit protocol) at more than one server, so the same data may have been updated by the Publisher or by more than one Subscriber. Therefore, conflicts can occur when data modifications are merged.

Merge replication includes default and custom choices for conflict resolution that you can define when you configure a merge publication. When a conflict occurs, a resolver is invoked by the Merge Agent to determine which data will be accepted and propagated to other sites.

Options available with merge replication include filtering published data horizontally and vertically, including join filters and <u>dynamic filters</u>, using alternate synchronization partners, optimizing synchronization to improve merge performance, validating replicated data to ensure synchronization, and using attachable subscription databases.

Merge replication is helpful when:

- Multiple Subscribers need to update data at various times and propagate those changes to the Publisher and to other Subscribers.
- Subscribers need to receive data, make changes offline, and synchronize changes later with the Publisher and other Subscribers.
- The application <u>latency</u> requirement is either high or low.
- Site autonomy is critical.

### See Also

Designing a Replication Topology Planning for Replication Replication Options

Types of Replication

Validating Replicated Data

# **Introducing Replication Options**

Options available with the types of replication allow you more replication solutions and greater flexibility and control in your applications. Replication options are:

- Filtering published data
- Publishing database objects
- Publishing schema objects
- Updatable subscriptions
- Transforming published data
- Alternate synchronization partners

## **Filtering Published Data**

Filtering data during replication allows you to publish only the data or partitions of data that are needed at the Subscriber. You can filter data to create partitions that include only the columns and/or only the rows that you specify for replication.

With all types of replication, you can choose to copy and distribute complete tables, or data filtered horizontally or vertically with static filters. <u>Merge replication</u> is especially strong in filtering options, and you can use dynamic filters to customize the filter based on a property of the Subscriber receiving the data.

Filtering data horizontally allows you to publish only the data that is needed, partition data to different sites, avoid conflicts (because Subscribers will be viewing and updating different subsets of data), and manage publications based

on user needs or applications.

Additionally, you have the option of employing user-defined functions in your static and dynamic filters and leveraging the power of customized functions.

Merge replication provides the added functionality of join filters and dynamic filters. Join filters enable you to extend filters created on one table to another. For example, if you are publishing customer data based on the state where the customer resides, you may want to extend that filter to the related orders and order details of the customers in a particular state. Dynamic filters allow you to create a merge publication and then filter data from the <u>publishing table</u>.. The filter value can be the user ID or login retrieved based on a Transact-SQL function, such as **SUSER\_SNAME() or HOSTNAME()**.

## **Publishing Database Objects**

You can publish database objects including views, indexed views, user-defined functions, stored procedure definitions, and the execution of stored procedures. You can include data and database objects in the same publication or in different publications. Publishing database objects is available with all types of replication (snapshot replication, transactional replication, and merge replication).

## **Publishing Schema Objects**

In addition to database objects, you can also specify if you want schema objects to be published such as declared referential integrity (primary key constraints, reference constraints, unique constraints), clustered indexes, nonclustered indexes, user triggers, extended properties, and collation. You can also change destination table owner names and data formats to optimize for SQL Server 2000 or heterogeneous Subscribers.

## **Updatable Subscriptions**

Data at the Subscriber can be modified if you use merge replication or if you use snapshot replication or transactional replication with an updatable subscription option.

Updatable subscription options available with snapshot replication and transactional replication allow you to make changes to replicated data at the

Subscriber and propagate those changes to the Publisher and to other Subscribers. Updatable subscription options include immediate updating, queued updating, and immediate updating with queued updating as a failover.

Immediate updating allows Subscribers to update data only if the Publisher will accept them immediately. If the changes are accepted at the Publisher, they are propagated to other Subscribers. The Subscriber must be continuously and reliably connected to the Publisher to make changes at the Subscriber.

Queued updating allows Subscribers to modify data and store those data modifications in a queue while disconnected from the Publisher for a period of time. When the Subscriber reconnects to the Publisher, the changes are propagated to the Publisher. If the Publisher accepts the changes, normal replication processes occur and the changes are propagated to other Subscribers from the Publisher. You can store data modifications in a SQL Server 2000 queue or use Microsoft Message Queuing.

Immediate updating with the queued updating option allows you to use immediate updating and switch to queued updating if a connection cannot be maintained between the Publisher and Subscribers. After switching to queued updating, reconnecting to the Publisher, and emptying the queue, you can switch back to immediate updating mode.

When using merge replication, data at the Subscriber is automatically updatable.

## **Transforming Published Data**

With snapshot replication or transactional replication, you can leverage the transformation mapping and scripting capabilities of Data Transformation Services (DTS) when building a <u>replication topology</u>. Replication integrated with DTS allows you to customize and distribute data based on the requirements of individual Subscribers. For example, a Subscriber might need to have different table names, column names, or compatible data types.

By transforming published data, you can filter data and simulate dynamic partitions of data so that data from one snapshot or transactional publication can be distributed to Subscribers that require different partitions of data. With static partitions, you need to create and filter separate publications for each Subscriber based on the needs of the Subscriber.

### **Alternate Synchronization Partners**

Subscribers to merge publications can synchronize with servers other than the Publisher at which the subscription originated. Synchronizing with alternate partners allows Subscribers to synchronize data even if the primary Publisher is unavailable. This feature is also useful when mobile Subscribers have access to a faster or more reliable network connection with an alternate Publisher.

### See Also

Alternate Synchronization Partners Filtering Published Data Merge Replication or Updatable Subscriptions Publishing Data and Database Objects

# **Typical Uses of Replication**

Microsoft® SQL Server<sup>™</sup> 2000 replication supports the distributed environment of increasingly global and mobile corporate operations. Replication allows you to share information across heterogeneous platforms and databases and then modify and reconcile that information. Replication ensures that correct data will be available when and where it is needed.

Replication is used for a variety of applications:

- Reporting, decision support, and data warehousing applications.
- Online and offline applications.
- Web-based applications with many users browsing data.
- Keeping data close to users (providing more site autonomy and efficient network usage).

# **Reporting, Decision Support, and Data Warehousing Applications**

A data warehouse is a database that contains enterprise data representing the business history of an organization. It is used to consolidate information stored in various business systems and heterogeneous platforms. Data in a data warehouse is often structured and optimized for decision support.

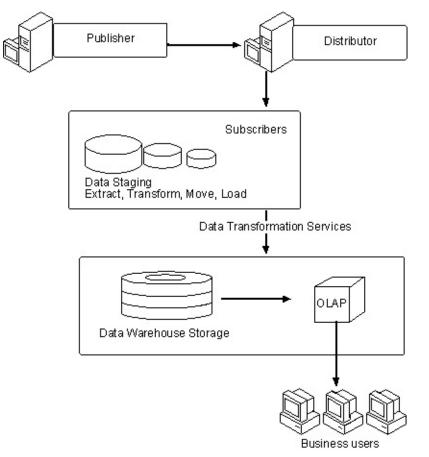
Replication becomes an integral part of the data warehousing and decision support environment when it is used during data staging and as a data warehousing management and deployment tool. You can use replication to update data marts and data warehouses, distribute data to read-only databases used for queries and analyses, distribute data to an online analytical processing (OLAP) database, and consolidate data so it can be transformed and moved into the data warehousing environment.

Replication can also be used to partition data that has been consolidated in a data warehousing environment and distribute the data to data marts or databases inside or outside of the data warehousing environment.

Although Microsoft® SQL Server<sup>™</sup> does not replicate SQL Server 2000 Analysis Services objects (for example, dimensions or cubes), it can help you distribute data from OLTP databases to data staging databases or databases that will be used for reporting, decision support or analysis purposes, and if needed, you can use the capabilities of Data Transformation Services (DTS) during replication.

Providing consistent data to data warehousing and decision support systems is critical to the success of those operations. Within a reporting, decision support query, or OLAP environment, different user groups have different requirements for the data, and replication provides several options for distributing, updating, and synchronizing data.

Because data used in decision support is predominantly read-only (used for queries and analysis), snapshot replication or transactional replication are often the types of replication used. With snapshot replication, data and database objects are copied and distributed exactly as they appear at a specific moment in time. If data transformations are needed for data that is replicated into a data mart or data warehouse, you can use Data Transformation Services (DTS) as part of the replication process when using snapshot replication or transactional replication.



#### Example

#### See Also

**Snapshot Replication** 

Transactional Replication

**Transforming Published Data** 

# **Online/Offline Applications**

Microsoft® SQL Server<sup>™</sup> 2000 replication offers a solution to the problem of data accessibility while traveling and at disconnected locations. Business users often need to use laptops or handheld computers when traveling and need a way to access data, often on demand, when using a modem to dial into corporate networks or connect to an intranet or the Internet.

Working online, using replication, mobile users can receive data from the central server (which would typically be the Publisher) when they connect to the corporate wide area network (WAN) or local area network (LAN), or over the Internet. They can then make changes to data immediately, or they can modify data offline and propagate those changes to the originating database and to other locations when they reconnect to the network.

Because data modifications made at Subscribers are performed asynchronously at the original server and then sent to other servers, transactional replication using the queued updating option, or merge replication are often the types of replication used for mobile or disconnected users.

**Note** When planning for an online/offline application that uses replication, plan for occasional maintenance in the deployment of the application and a way to transfer new datasets to the disconnected Subscribers.

#### Example

Because the sales representatives make frequent data modifications and are often disconnected, Northwind administrators decide to use merge replication. When the sales representatives reconnect to the network, they can synchronize their data changes with changes made at other locations.

### See Also

Merge Replication
Merge Replication or Updatable Subscriptions

Planning for Merge Replication

# **Web-Based Applications**

Replicating data over the Internet allows remote, disconnected, and anonymous users to access data when they need it using a connection to the Internet. For example, if a Web site allows users to browse items for sale, users will need to browse high volumes of data. Using replication, administrators can make that data available for read purposes on multiple Web servers. Browsing can take place at any server and the site can then handle more traffic.

Another use of replication and Web-based applications is allowing individual Subscribers to download or upload data changes using an application that uses an Internet browser, or by using a connection to the corporate network or share where the data resides. Ways for users to connect to replicated data over the Internet using Microsoft® SQL Server<sup>TM</sup> 2000 include:

- Using Virtual Private Networks (VPNs), such as those based on the Microsoft Windows NT® Server version 4.0 operating system, the Microsoft Windows® 2000 Server operating system, or a third party provider.
- Integrating replication with Microsoft Proxy Server.
- Using TCP/IP and File Transfer Protocol (FTP) to transfer the <u>initial snapshot</u> of data over the Internet.

VPNs allow users who are not connected directly to a corporate network to access the corporate network remotely through the Internet. A VPN connects the components of one network over another network. This is achieved by allowing the user to tunnel through the Internet or another public network (using a protocol such as Microsoft Point-to-Point Tunneling Protocol (PPTP)). This process provides the same security and features previously available only in a private network.

Using VPNs is the most secure method for replicating data over the Internet. You can use Windows Authentication as though you were on a local area network (LAN).

Integrating Microsoft SQL Server 2000 replication with Microsoft Proxy Server allows for replication over the Internet with security configured on Windows NT version 4.0, Windows 2000 Server, Proxy Server, and SQL Server 2000.

SQL Server 2000 can use the TCP/IP Sockets or the Multiprotocol Net-Libraries over TCP/IP to establish an ODBC or OLE DB connection between the Publisher or Distributor and the Subscriber. You can then configure the publication and pull subscriptions or <u>anonymous subscriptions</u> to access the FTP site to apply the initial snapshot files (incremental changes are propagated using ODBC or OLE DB on TCP/IP).

For more information, see <u>Implementing Replication Over the Internet</u>.

# **Keeping Data Close to Users**

Data distribution is the process of ensuring that data is available to people when they need it. Here are examples of using replication to give data to the users who need it. Publish data:

- From a central site, partitioning it, and distributing it to various regional offices.
- To a read-only database so that users can execute queries and analyses without interrupting transaction processing on a production database.
- From multiple databases into a central database, which could be a data mart or data warehouse.
- To a backup database as part of a standby solution.
- To support mobile, disconnected users.
- Over the Internet, so it can be available on-demand with <u>anonymous subscription</u>s.

#### Example

Using replication, Northwind Traders can replicate partitions of the central orders online transaction processing (OLTP) database to each region, and filter the data based on the city, region, or user who is accessing the data.

If the regional office only needs to read the data and not make changes, the central office can filter the data to create the appropriate partitions based on region or other criteria and then publish that data to Subscribers. Depending on how often and the how much data is modified at the publishing site, this type of application could use snapshot replication or transactional replication.

If a regional office will make changes to the data and needs autonomy, the data

can be filtered, replicated to the region, and the regional office can make changes to its data as needed. When the changes need to be propagated to the corporate office or to other regions, the regional office can synchronize with the corporate office and those changes will be propagated automatically to the other regions when they synchronize with, and are accepted by, the corporate office. If the regional office needs to distribute the corporate data to its sales force within the region, it can republish the data to the necessary sites.

There are several options for scheduling distribution of the data and modifying the data at the different regional offices. If the regional offices are continuously and reliably connected, multiple offices can update the data and propagate changes to the corporate office immediately. The data is then propagated to other regions within seconds (immediate updating), or if a site is disconnected for a limited amount of time, data modifications can be stored in a queue until the connection with the corporate office is reestablished (queued updating).

# **How Replication Works**

There are several ways to implement and monitor replication, and the process of replication is different depending on the type of replication and the options you choose. In general, replication is composed of the following stages: configuring replication, generating and applying the <u>initial snapshot</u>, modifying replicated data, and synchronizing and propagating data.

## **Configuring Replication**

Replication deployment begins when you configure a Publisher and Distributor. The Distributor can be a separate server from the Publisher, or it can be the same server. In general, replication is composed of the following stages: configuring replication, generating and applying the <u>initial snapshot</u>, modifying replicated data, and synchronizing and propagating data. The Distributor is a primary component during snapshot replication and transactional replication; however, the role of the Distributor is limited during merge replication. The Distributor is used only for agent history reporting and monitoring purposes. During merge replication, the Publisher and Distributor are usually the same server. This is called using a local Distributor.

After the Publisher and Distributor are configured, you can create publications based on data, subsets of data, and/or database objects. When you create the publication, you determine what type of replication you want to use, the type of databases that will be Subscribers to the publication, the data and database objects that will be published, where the snapshot files will be stored, when the initial snapshot synchronization will occur, and options that will be used with the publication.

After you create a publication, you can create push and/or pull subscriptions at either the Publisher or the Subscriber and configure your replication schedule and options.

### Generating and Applying the Initial Snapshot

Whether you choose snapshot replication, transactional replication, or merge replication, SQL Server 2000 creates an initial snapshot of schema and data and

saves it to the snapshot folder and location you chose when creating the publication. The two exceptions to this process are with dynamic filters in merge replication and subscriptions for which the snapshot will be applied manually. After the subscription is created, when the initial snapshot is applied is based on the schedule you indicated when creating the publication, or you can apply the snapshot manually.

The Snapshot Agent prepares snapshot files containing schema, data, and database objects, stores the files in the snapshot folder, and records synchronization jobs in the distribution database on the Distributor for snapshot replication or transactional replication, and in the <u>publication database</u> for merge replication. The Snapshot Agent does not prepare these files when the merge publication uses dynamic filters and does not use <u>dynamic snapshots</u> when the subscription specifies that the snapshot will be applied manually.

With snapshot replication and transactional replication, the Distribution Agent moves the snapshot from the distribution database to the destination tables at the Subscribers and applies the scripts, schema, and data necessary for replication. With merge replication, the Merge Agent moves the snapshot to Subscribers when it is run for the first time or when the subscription is set for reinitialization. It then applies the scripts, schema, and data necessary for replication.

## **Modifying Replicated Data**

Depending on the type of replication and the options you chose when configuring the publication, the Subscriber may be able to modify data after the initial snapshot has been replicated and propagate changes to the Publisher, which can then propagate the changes to other Subscribers.

The following replication types and options allow Subscribers to modify replicated data:

- Merge replication
- Snapshot replication or transactional replication with immediate updating
- Snapshot replication or transactional replication with queued updating

• Any type of replication in which data is filtered so partitions of data can be modified at individual sites autonomously and without conflicts occurring between sites

#### Synchronizing and Propagating Data Changes

How data is synchronized and data modifications propagated to Publishers and other Subscribers depends on the type of replication and options you choose. Synchronizing data refers to the process of data being propagated between Publisher and Subscribers after the initial snapshot has been applied at the Subscriber.

For snapshot replication, synchronize means to reapply the snapshot at the Subscriber so that schema and data at the <u>subscription database</u> is consistent with the publication database. For transactional replication, synchronizing data means that data INSERTS, UPDATES, and DELETES, and other data modifications, are distributed between Publisher and Subscribers. For merge replication, synchronization means that data modifications made at multiple sites are merged, conflicts (if any) are detected and resolved, and data eventually converges to the same data values at all sites.

#### See Also

Applying the Initial Snapshot Generating the Initial Snapshot Implementing Replication Replication Options Synchronizing Data Types of Replication

# **Methods of Implementation**

Methods for implementing replication, developing replication applications, and maintaining replication are: replication wizards and properties, replication programming interfaces, scripting of system stored procedures, and Windows Synchronization Manager.

### **Replication Wizards and Properties**

SQL Server Enterprise Manager includes several wizards and properties dialog boxes you can use to simplify the installation and maintenance of replication. SQL Server Enterprise Manager allows you to view and modify the properties of replication, and provides graphical navigational tools. It also provides the replication folder and Replication Monitor, which help you monitor and troubleshoot replication activity.

The following replication wizards and properties dialog boxes provide a guided approach to implementing replication:

- The Configure Publishing and Distribution Wizard helps you specify a server to use as a Distributor and, optionally, specify other replication components. After the Publisher and Distributor are configured initially, changes can be made in the **Publisher and Distributor Properties** dialog box.
- The Create Publication Wizard guides you through the process of choosing the type of replication and replication options, specifying the data or database objects that you want to replicate, the types of Subscribers that will access the publication, as well as other properties of the publication. After the publication is created using the Create Publication Wizard, changes can be made in the **Publication Properties** dialog box.
- The Push Subscription Wizard helps you create a subscription to a publication that will be distributed to a specified Subscriber. You can view the options selected for a push subscription in the **Subscription**

#### **Properties** dialog box.

- The Pull Subscription Wizard helps you create a subscription to a publication requested by a Subscriber. After the subscription is created, you can view the options in the **Pull Subscription Properties** dialog box.
- The Disable Publishing and Distribution Wizard helps you disable publishing, distribution, or both, on a server.

After replication is configured using wizards, you can script different configuration processes of replication. For example, after creating a standard subscription to a publication for one Subscriber, you can script the set up of the subscription, run it at various Subscribers, and substitute the correct Subscriber name in the script as necessary. For more information, see <u>Scripting Replication</u>.

### **Replication Programming Interfaces**

Another method of replication implementation and administration is by using one of, or a combination of, the replication programming interfaces:

- SQL-DMO
- Microsoft® ActiveX® controls for replication
- Replication Distributor Interface

SQL-DMO has more options available than the replication wizards (which are based on SQL-DMO), and you can create custom applications using Microsoft Visual Basic® or Microsoft Visual C++® that allow you to configure or maintain a replication topology. SQL-DMO can be used to program replication administration such as configuring distribution, creating subscriptions, and so on.

ActiveX controls for replication enable you to control Snapshot Agent, Merge Agent, and Distribution Agent activity programmatically. This allows users to

program replication into their applications. The controls also offer some lightweight administration options to create, delete, and reinitialize subscriptions, and to control, monitor, and troubleshoot replication agents. These controls can be used to program activity needed to operate replication. For example, for an application that provides online and offline capabilities, you may want to expose a **Synchronize** button. That button can be associated with the merge ActiveX control, and whenever the users click the button, the Merge Agent connects to the Publisher, and data is synchronized for the specified publication.

The Replication Distributor Interface provides the capability to replicate data from heterogeneous data sources such as Microsoft Access or Oracle. The Replication Distributor Interface is used primarily by independent service vendors, or others who need to develop a custom replication application based on proprietary data sources.

Essentially, this interface allows a custom solution while employing the replication distribution system, but developers assume the data modification detection capabilities that would typically be conducted by the Log Reader Agent.

## **Replication System Stored Procedures**

Replication system stored procedures are documented and available as a method for implementing replication in special circumstances or for use in batch files and scripts. In most cases, however, you are better served by using the programming interfaces SQL-DMO and replication ActiveX controls for programming replication. SQL-DMO provides an easier method and higher-level solution than direct use of stored procedures.

The stored procedures are typically used if you use the scripting features from SQL Server Enterprise Manager. When you script replication, SQL Server generates Transact-SQL batches that re-create the replication environment (configuring publishing and distribution, creating publications and subscriptions, and so on). After the scripts are generated, you can edit them as needed using SQL Query Analyzer.

### Windows Synchronization Manager

Windows Synchronization Manager is a utility available with the Microsoft Windows® 2000 operating system and anywhere Microsoft Internet Explorer version 5.0 or later is installed. It allows you to synchronize data between instances of Microsoft SQL Server<sup>™</sup>. You can use SQL Server Enterprise Manager to enable pull subscriptions for use in Windows Synchronization Manager, or you can enable subscriptions programmatically for use in Windows Synchronization.

Using Windows Synchronization Manager, you can schedule synchronizations or instruct Windows to synchronize selected items automatically when you log on to the computer or when the computer is idle for a specified length of time. Windows Synchronization Manager is located under the Accessories folder on the Windows **Start** menu.

#### See Also

**Developing SQL-DMO Applications** 

**Getting Started with Replication Programming** 

**Replication Tools** 

# **Agents and Monitors**

Agents used with Microsoft® SQL Server<sup>™</sup> 2000 replication carry out the tasks associated with copying and distributing data. SQL Server 2000 replication uses SQL Server Agent as well as agents that are specific to replication.

## **SQL Server Agent**

SQL Server Agent hosts and schedules the agents used in replication, and provides an easy way to run replication agents. SQL Server Agent also controls and monitors several other operations outside of replication, including monitoring the SQL Server Agent service, maintaining error logs, running jobs, and starting other processes.

## **Snapshot Agent**

The Snapshot Agent is used with all types of replication. It prepares schema and initial data files of published tables and stored procedures, stores the snapshot files, and records information about synchronization in the distribution database. The Snapshot Agent typically runs under SQL Server Agent at the Distributor and can be administered using SQL Server Enterprise Manager.

## Log Reader Agent

The Log Reader Agent is used with transactional replication. It moves transactions marked for replication from the transaction log on the Publisher to the distribution database. Each database published using transactional replication has its own Log Reader Agent that runs on the Distributor and connects to the Publisher.

### **Distribution Agent**

The Distribution Agent is used with snapshot replication and transactional replication. It moves the snapshot jobs and transactions held in the distribution database to Subscribers. The Distribution Agent typically runs at either the Distributor for push subscriptions or at the Subscriber for pull subscriptions.

### Merge Agent

The Merge Agent is used with merge replication. It applies the initial snapshot to the Subscriber, and moves and reconciles incremental data changes that occur. Each merge subscription has its own Merge Agent that connects to both the Publisher and the Subscriber and updates both. The Merge Agent typically runs at either the Distributor for push subscriptions or the Subscriber for pull subscriptions. The Merge Agent typically uploads changes from the Subscriber to the Publisher and then downloads changes from the Publisher to the Subscriber during a typical bidirectional merge. Changes can also be moved in one direction by configuring the exchange type of the agent.

## **Queue Reader Agent**

The Queue Reader Agent is used with snapshot replication or transactional replication with the queued updating option, or if the immediate updating with queued updating as a failover option is enabled.

The Queue Reader Agent is a multithreaded agent that runs on the Distributor. It is responsible for taking messages from a queue and applying them to the appropriate publication.

Unlike the Distribution Agent and the Merge Agent, only one instance of the Queue Reader Agent exists to service all Publishers and publications for a given Distributor.

## **Miscellaneous Agents**

Clean up agents listed under the **Miscellaneous Agents** folder in Replication Monitor complete scheduled and on-demand maintenance of replication.

Clean up agent	Description	Default schedule
Agent History Clean	Removes replication agent	Runs every 10
Up: Distribution	history from the distribution	minutes
	database.	
Distribution Clean Up:	Removes replicated	Runs every 10
Distribution	transactions from the	minutes
	distribution database.	
Expired Subscription	Detects and removes	Runs every day at

-	expired subscriptions from publication databases.	1:00 A.M.
Reinitialize	Reinitializes all	No default schedule
Subscriptions Having	subscriptions that have data	(not enabled by
Data Validation Failures	validation failures.	default).
Replication Agents	Detects replication agents	Runs every 10
Checkup	that are not actively logging	minutes
	history.	

## **Replication Monitor**

Through Replication Monitor in SQL Server Enterprise Manager, you can view and manage replication agents responsible for various replication tasks. For example, you can set up transactional replication so that the log on the Publisher is read continuously, transactions are distributed to Subscribers every ten minutes (although this is often also continuously), and initial snapshots are generated every night at midnight. You can also execute replication agents on demand.

Replication Monitor provides a way to set alerts on replication events. When the event occurs, Replication Monitor responds automatically, either by executing a task that you have defined or by sending an e-mail or a pager message to a specified individual.

#### See Also

Administering and Monitoring Replication

**Replication Agents** 

**Types of Replication** 

# **Planning for Replication**

Careful planning before replication deployment can maximize data consistency, minimize demands on network resources, and prevent troubleshooting later.

Consider these areas when planning for replication:

- Whether replicated data needs to be updated, and by whom.
- Your data distribution needs regarding consistency, autonomy, and latency.
- The replication environment, including business users, technical infrastructure, network and security, and data characteristics.
- Types of replication and replication options.
- Replication topologies and how they align with the types of replication.

# **Distributed Update Factors**

If distributed data does not need to be updated at more than one site, data can easily maintain the ACID properties of transactions. However, when you need to update data at multiple sites, you should consider how the ACID properties of transactions and site autonomy are going to be affected.

### **ACID Properties**

To qualify as a transaction, a single unit of work must adhere to the ACID properties of atomicity, consistency, isolation, and durability.

- Atomic. For a transaction to be atomic, all of its data modifications are performed or none of them are performed.
- Consistent. To be consistent, a completed transaction must leave all data in a consistent, logically correct state.
- Isolation. To meet the isolation property, a transaction reads data in the state it was in before another concurrent transaction modified it (without yet committing the transaction). Concurrent modifications that are in progress do not affect the transaction.
- Durable. To meet the durability property, the modifications of a transaction will persist (for example, remain in the database, even if there is a system failure). After a commit is acknowledged, the system must guarantee that the transaction persists.

Your needs for strict adherence to ACID properties are significant when planning for replication because when data modifications are made at multiple Subscribers independently, conflicts can occur. If conflicts are allowed, strict ACID characteristics cannot be guaranteed even with conflict detection and resolution. If you are considering merge replication or transactional replication with the queued updating option, you need to prepare for how to handle transactions that do not meet these properties. Two-phase commit protocol (2PC) is required to guarantee ACID properties in a distributed, multiple-update environment. However, this means that the sites are dependent on one another for completion of an update, and they will give up site autonomy.

For more information about ACID properties, see <u>Transactions</u>.

Questions relating to ACID properties include:

- Do multiple Subscribers need to make updates? If replicated data is going to be read-only, ACID properties will not be affected.
- If updates need to be made at multiple sites, can you allow conflicts? Is the data filtered into different partitions for different sites? If you need to preserve transaction isolation and durability, you must avoid conflicts.
- Is it acceptable for a committed transaction to be undone to resolve a conflict?
- Is it acceptable that subsequent transactions are changed based on the value of a transaction that was undone due to a conflict?

If ACID properties must be preserved, you can use 2PC so that the Publisher accepts any changes before a conflict could exist, execute all updates at one site, or filter data so sites can update unique subsets of data and avoid conflicts with other sites.

## **ACID Properties and Replication**

When designing replication, determine whether ACID properties need to be maintained and how much autonomy is required by your application.

When thinking about ACID properties in regards to replication, consider whether data at any participating site must be the same data that would have resulted had all transactions been performed at only one site. If you made all data modifications at one site, your transactions would typically be consistent, isolated, and durable. Consider if you also have those needs in your distributed environment.

Latency refers to the period of time between when data is updated at one site (the Publisher) and when those changes appear at another site (the Subscriber). The latency can vary from a few seconds to hours, days, or longer.

Questions relating to ACID properties in your replication application include:

- Does data need to be updated at Subscribers?
- How much latency is acceptable?

To maintain strict ACID properties, you will often have to give up site autonomy because servers must be continuously and reliably connected. That is the only way to guarantee you avoid conflicts. If you allow conflicts, some transactions must be altered or undone to resolve the conflict. Therefore, at least some transactions were not durable, and perhaps other transactions that read the values of the non-durable transaction were not isolated.

### Autonomy

Autonomy is the degree of dependence one site has on another. Complete autonomy occurs when one site does not depend on any other site to complete its work, and it is independent of the operations at any other site.

2PC is an example of a nonautonomous process because every data change is dependent on every other participating site being able to accept the transaction successfully and immediately. But in replication, 2PC is optimized to be dependent on only two servers in the replication topology: the Publisher and the Subscriber making the update, with the Publisher as the arbiter.

Merge replication or transactional replication with queued updating is often used when sites need to modify data autonomously and then later merge changes with changes made at the Publisher and at other Subscribers.

With merge replication, data converges and all sites end up with the same values; however, because conflicts can occur and are resolved, the values are not necessarily the ones that would have resulted had all the work been done at only one site. All sites may work offline and when all sites have synchronized data, all sites will eventually have the same data. However, because the same data is being changed at multiple locations, conflicts can occur and some transactions from one site will be committed while others will be rejected and resolved. Those transactions by definition are not durable.

#### Example

A sales representative in the Northwest office changes the customer information for the company named White Clover Markets by changing the value of the customer phone number to (206) 554-2341. A sales representative in the Southwest office uses replicated data at that site and changes the fax number for White Clover Markets to (206) 555-8314. Another sales representative at the publishing site in the corporate headquarters changes the phone number of White Clover Markets to (206) 554-2241. When the Subscribers merge with the Publisher, the conflicts will be detected and depending on the conflict resolution policy, it will be resolved. It is possible that White Clover Markets ends up with the new phone number that was entered at the corporate office, (206) 554-2241, and that the new fax number entered at the Southwest office is rolled back and the original fax number is maintained in the database.

If sites are autonomous, ACID properties cannot be assured. For example, merge replication allows sites to be autonomous and to update replicated data whether online or offline. It does not, however, guarantee durability. If conflicts are to be resolved, then a committed transaction must be altered in order to resolve the conflict. Instead, it focuses on data convergence, the merging of changes made at various sites into a new result set.

Questions relating to autonomy include:

- How independent do the various sites need to be?
- Are sites continuously and reliably connected, or are they disconnected for periods of time?
- Is preserving ACID properties more important than autonomy?

# **Evaluating the Replication Environment**

The replication environment is composed of the business units, people, technical structure, and applications that will either host or use replication. At this stage of replication planning, you should talk to the people who will be affected by replication as well as gather information about the technical infrastructure including how and what data is stored, where, how, and when the data needs to be replicated, and how replication will be administered and maintained.

# **Business Objectives and Requirements**

Asking questions in the areas of data distribution and data modification helps you determine how to distribute data, what type of replication to use, what replication options to use, what the business needs are for replication, and who will be affected by replication.

#### **Data Distribution**

Answering questions about the objective of distributing data ensures that replication is the correct solution for the problem you need to solve or goal you hope to attain. Answering questions about the needs of the organization regarding data distribution helps you plan where replication is needed and how often, and determine the type of replication to use. These questions include:

- What is the core problem or objective that replication might help solve? For example, do you want to distribute data for reporting servers, do you have applications that need to be updated online and offline, or do you want a standby solution?
- How will data distribution affect existing technology, administrative resources, people who currently access the data, and costs of data administration?
- What data is needed and where is it needed?
- How often is the data needed?
- Are entire refreshes of the data required or just incremental updates?
- Are entire tables needed, or can you filter the data according to site or data usage? Do you want to replicate database objects such as stored procedure definitions or execution, views, triggers, or user-defined

functions?

- Where will data be published and what Subscribers need to receive the data?
- How many Subscribers need the data?
- Are data transformations necessary during replication?

### **Data Modification**

Answering questions about modifying data helps you determine what types of replication to use, what replication options to use, and when to schedule updates.

- Do Subscribers need to update the data?
- If multiple Subscribers update the same sets of data, are conflicts allowed?
- Do transactions have dependencies? Will dependent transactions be affected if a transaction has dependencies and that transaction is undone due to a conflict?
- If data modifications are made at Subscribers, what is the rate of data modification?
- Will Subscribers have continuous, reliable connections to the publication database or will they be disconnected for periods of time?
- Can data be partitioned logically so that various sites can modify their own subsets of data without the possibility of updates causing conflicts with updates made at other sites?

- If multiple sites are updating the same data independently, how will conflicts be handled?
- How quickly must changes be replicated to other sites?
- How quickly must the initial snapshot and data be applied at the Subscriber?
- How often will Subscribers synchronize data or propagate changes?
- How many updates are you sending?

## **Network Considerations**

The following replication issues affect the performance of your networks:

- The volume and typical size of data flowing over the network.
- The number of Subscribers to a particular Publisher.
- The speed and reliability of the line.
- The processing power of the Publisher, Distributor, and Subscribers.

If you are replicating over a slow link, the profiles for the agents involved in replication can be customized. For example, you can configure behavior such as the batch size, the polling interval, the timeout period, and the number of buffers available. The configuration options vary with the particular agent whose profile is being configured.

Network speed is often the most important issue when applying the initial snapshot. The volume of incremental data changes may be low, but the volume of data initially distributed may be high. Transferring the snapshot using a CD-ROM or tape device is one solution to this situation. Compressing the snapshot files can also help preserve network speed.

Knowing the processing power of the servers in your replication topology helps you decide whether to use remote agent activation. If you are using push subscriptions and there is greater processing power at the Subscriber, you may want to use remote agent activation so that the Distribution Agent or Merge Agent runs at the Subscriber rather than at the Distributor. If you are using pull subscriptions and there is greater processing power at the Distributor, you may want to use remote agent activation so that the Distributor Agent or Merge Agent runs at the Distributor rather than at the Distribution Agent or Merge Agent runs at the Distributor rather than at the Subscriber.

You may also want to perform transformations on published data specific to individual Subscribers that discard some data at the Distributor. The transformation could discard the data before placing it on the network, and this could be a significant benefit for replication performance, especially if the network bandwidth is low.

### See Also

Agent Profiles Generating the Initial Snapshot Remote Agent Activation Transferring Snapshots Transforming Published Data

# **Security Considerations**

When considering security in Microsoft® SQL Server<sup>™</sup> 2000, replication is similar to other applications in SQL Server 2000. Your determining factors will be a balance between how secure the data needs to be, and how accessible the data needs to be for your environment.

Additional security issues need to be considered in the following areas:

- SQL Server Agent.
- Location of snapshot files.
- Testing agent connectivity.
- Security mode of the Publisher.

#### **SQL Server Agent**

The SQL Server Agent service (SQLServerAgent) at the client should not use the **LocalSystem** account. It needs to use a standard domain account. The **SQLAgent** account is the security context under which the Snapshot Agent, Merge Agent, and Distribution Agent are running by default.

The account used by the SQL Server Agent is defined at the time SQL Server 2000 is installed and can be changed at any time.

On the Microsoft Windows<sup>®</sup> 98 operating system, SQL Server Agent and the replication agents run under the security account of the user logging on to the Windows operating system. On Microsoft Windows NT<sup>®</sup> version 4.0 and Microsoft Windows 2000 operating systems, the replication agents run under the login or security context of the SQLServerAgent service. Neither the SQLServerAgent service nor the SQL Server service needs to run under a Windows 2000 Administrator account.

Each agent connects to one or more servers (Publisher, Distributor, or

Subscribers depending on the agent) and must have a valid login to that instance of SQL Server to complete the connection. For more information, see <u>Agent</u> <u>Login Security</u>.

### **Location of Snapshot Files**

The folder in which the snapshots are stored must be available to all Subscribers on the network. To ensure secure access to the initial snapshot files of your replicated data, it is recommended you use an explicit share instead of an administration share (for example, C\$) for which you cannot grant specific permissions. The administrative share is used as a default only because it will always exist on Windows NT 4.0 and Windows 2000 (but it cannot be accessed except by an administrator account).

When configuring distribution, you can define the default location for all snapshot files. After creating a publication, you can define the location of the snapshot files using the publication properties dialog box.

## **Testing Agent Connectivity**

When implementing replication, make sure that the replication agents can communicate with all servers involved in the replication topology. One way to test agent connectivity is to log in to the required server and database using SQL Query Analyzer or **osql** using the same login that the replication agent will be using (or typically the login that SQL Server Agent is using).

You must be a SQL Server 2000 system administrator to enable the server for replication. After replication is enabled, you do not need to be a SQL Server 2000 system administrator to set up publications and subscriptions, or to invoke or schedule the replication agents. You must be in the **db\_owner** role to create publications. Anyone who is added to the publication access list (PAL) can create pull subscriptions to that publication (but only to that publication).

## Security Mode of the Publisher

Connections to a server (Publisher, Distributor, or Subscribers) can use Windows Authentication or SQL Server security. Windows Authentication is generally preferred for greater security and ease of use; however, connections to Windows 98 servers must use SQL Server security because Windows Authentication is a feature only on Windows NT 4.0 and Windows 2000.

It is recommended that the Subscriber connection have **dbo** permissions in the subscription database to make sure the proper permissions are granted, and for overall simplification; however, **dbo** permissions are not required.

#### See Also

Generating the Initial Snapshot

Managing Security

**Replication Security** 

Transferring Snapshots

# **Data Needs and Characteristics**

While examining the data that you are replicating, consider the following:

- Collation (defines code page or character set and data sorting)
- Data types

### **Character Sets**

If replication is implemented between servers using different character sets, Microsoft® SQL Server<sup>TM</sup> 2000 does not convert any of the replicated data and may mistranslate the data as it is replicated because it is impossible to map all characters between character sets.

If you can guarantee that all characters you use will have identical codes on all code pages, replication would be successful, but it would not be guaranteed. Similarly, the comparison style specified by the collation you select can affect the accuracy of replicated transactions. To guarantee successful data replication, servers are best when configured using the same code pages and comparison styles.

Generally, if you have an environment where you have different character sets, you should consider using Unicode data types for which no conversion is necessary.

## Data Types

When determining data to replicate, consider the data type. You should understand the following:

• **timestamp** columns. For merge replication or transactional replication with the queued updating option, when articles contain a **timestamp** column, the **timestamp** column is replicated, but the literal **timestamp** values are not. The **timestamp** values are regenerated when applying the initial snapshot rows at the Subscriber. This allows **timestamp** to continue using optimistic concurrency control (a frequent usage). For

snapshot and transactional publications, and publications that allow immediate updating, the literal values for a **timestamp** column are replicated, but the data type for the replicated values is changed to binary (8) on the Subscriber. For more information, see <u>Replication Data Considerations</u>.

- **uniqueidentifier** columns. If you are using merge replication, or if you are using snapshot replication or transactional replication with queued updating and the table that is being replicated does not have a **uniqueidentifier** column, SQL Server 2000 will add one when you create a publication. In merge replication, this occurs when the initial snapshot is generated. In snapshot replication or transactional replication using the queued updating option, this occurs when the publication is created. In the case of queued updating, a predefined **uniqueidentifier** column will be added for row versioning irrespective of the presence of a globally uniqueidentifier (GUID) column in the table. The use of the uniqueidentifier in queued updating is conceptually like a global timestamp. To ensure that merge replication will reuse an existing **uniqueidentifier** column to uniquely identify replicated rows, make sure that your **uniqueidentifier** column is created with the column property **ROWGUIDCOL**. The use of the GUID in merge replication is conceptually like the use of a global primary key.
- Columns with **text** or **image** data types. These columns can take longer to replicate because they can be very large. When using snapshot replication or transactional replication with the immediate updating or queued updating options, updates made at the Subscriber to replicated data with **text** or **image** data types are not supported. However, replication and updating of these columns is fully supported when not using updatable subscriptions. Publishing **text** and **image** data types is also supported in merge replication.
- Case sensitivity. Generally, you should choose the same collation scheme (as the most common setting is case sensitivity) at the Publisher and at the Subscriber. For more information, see <u>Specifying Collations</u>.

For example, suppose you are publishing data about customers and you do not choose the same collation scheme at the Publisher that is at the Subscriber. Data is then filtered based on state="Ca" for a particular Subscriber. The data that is published to the Subscriber may not be the data that you intended because of differences in collation. Choosing the same collation scheme is not required, and depending on your application requirements, you may want to choose a different collation scheme (for example, a Publisher might have data that is case-sensitive, but a Subscriber that is a reporting server may have data that is case-insensitive).

- Triggers. Consider triggers that reside on the publishing table. By default, the triggers will be published with data from that table. If you do not want triggers on the publishing table to be published with data, you can change an option in the properties for a specific publication. For more information, see <u>Publishing Data and Database Objects</u> and <u>Using NOT FOR REPLICATION</u>.
- Row size. Is the row size greater than the maximum of 6,000 characters for merge replication and 8,000 characters for transactional replication? (Size limits exclude columns with **text** and **image** data types.)
- Data type mapping. Do you need to support Subscribers running on an instance of SQL Server 7.0 or earlier, or Subscribers that are not running on a version of SQL Server? SQL Server 2000 has new data types that servers running earlier versions of SQL Server cannot replicate. If so, you should know how the data types map between the different databases. For more information, see <u>Data Type Mapping</u>.
- Column-level or database collations. Depending on which collation you use, retrieving the data may be different at different Subscribers.

# **Planning for Application Development**

When planning replication applications, consider the following:

- Design your application to minimize conflicts. If the Subscribers need to read data and do not need to update data, conflicts will be avoided. Partitioning data logically according to geographic locations or business uses can also prevent users from updating the same data values, thus avoiding conflicts.
- For online/offline applications where you expect conflicts can and will occur, merge replication is usually the best choice for your application. Merge replication allows for a variety of conflict detection and resolution policies, evaluates updates row by row, and results in data convergence.
- Snapshot replication or transactional replication with the immediate updating or queued updating option is recommended for applications that are mostly read with occasional updates. Immediate updating uses two-phase commit (2PC). Queued updating provides policies for conflict resolution and evaluates updates and conflicts on a transaction basis.
- When using merge replication, or when using snapshot replication or transactional replication with the queued updating option, determine the conflict resolution policy before implementing replication.
- Research how disconnecting from the database will affect mobile or disconnected users. What happens if users do not immediately see the updates they make at the Subscriber?
- How fast is data synchronization? How long does it take to apply the initial snapshot and how long does it take for periodic updates? Test the

initial snapshot by applying it over the actual network that will be used. Consider applying the initial snapshot manually using a CD-ROM or removable media device if transferring it over the network takes too long.

- Manage **identity** values by using **identity** ranges when using merge replication or when using snapshot replication or transactional replication and allowing queued updating subscriptions. If you create data partitions and assign different identity ranges to the partitions, conflicts will be avoided because different sites will be working with different subsets of data.
- Ensure that your applications use column names in INSERT statements before enabling merge replication or transactional replication with immediate updating or queued updating options, because these types of replication may add columns to your publishing table. If you do not list the column names in INSERT statements for these types of replication, an error will occur.
- If you are using transactional replication with the immediate updating or queued updating option, Subscribers will not be able to update values with the **text** or **image** data types. The publication can contain **text** or **image** columns, but those columns may be updated only at the Publisher.
- Be aware of maximum column and row sizes. A table used in snapshot replication or transactional replication can have a maximum of 255 columns and a maximum row size of 8,000 bytes. A table used in a merge publication can have a maximum of 246 columns and a maximum row size of 6,000 bytes. The reason the restriction for merge replication is stricter than the restriction for transactional replication is because conflict tables have the same structure with additional columns that store information about the origin of the conflict and the specific reason for the conflict. Because additional space is needed to record this conflict information, the maximum row size is less than the maximum

row size for transactional replication.

• If you will have a high volume of transactions, always design your application to use stored procedures to modify data at the Publisher and publish the execution of stored procedures.

#### See Also

Filtering Published DataMerge Replication Conflict Detection and ResolutionMerge Replication or Updatable SubscriptionsQueued Updating Conflict Detection and Resolution

# **Planning for Each Type of Replication**

Each type of replication (snapshot replication, transactional replication, and merge replication) has specific requirements and issues that you should consider before implementation.

Because an initial snapshot must be applied for all types of replication, you should be familiar with the planning considerations for snapshot replication even if you choose to implement transactional replication or merge replication.

When considering transactional replication, allocate adequate disk space in the distribution database to handle the number of transactions that will be stored there.

When considering merge replication, Microsoft® SQL Server<sup>™</sup> 2000 uses a globally unique identifier (GUID) column to identify each row during the merge replication process. If the table that is replicated does not have a **uniqueidentifier** column with the ROWGUIDCOL property and a unique index, SQL Server 2000 will add one to the table, and you will need to account for the additional data that is stored there. If the table already has a **uniqueidentifier** column, you can add the ROWGUIDCOL property to signal that it can be used during merge replication. You must also add a unique index on this column or make it the primary key for the table. Distributed applications can benefit greatly from using the **uniqueidentifier** column because it guarantees that no two sites will generate the same key value.

#### See Also

Planning for Snapshot Replication

Types of Replication

# **Planning for Snapshot Replication**

Snapshot replication requires planning in the following areas:

- Transferring and storing snapshot files.
- Scheduling snapshots.

### **Transferring and Storing Snapshot Files**

You have the option of storing snapshot files in a location other than or in addition to the default location, which is often located on the Distributor. Alternate locations can be on another server, on a network drive, or on removable media (such as CD-ROM or removable disks). You can also save the snapshot files to a File Transfer Protocol (FTP) site for retrieval by the Subscriber at a later time.

Additionally, you can compress the snapshot files to improve network performance by writing data in the Microsoft® CAB file format. For more information, see <u>Compressed Snapshot Files</u>.

When planning to transfer and store snapshot files, estimate the disk space required at the snapshot file location and at the Subscriber that will receive the snapshot files.

The amount of space required for one snapshot can be affected by several factors including the size and number of articles published. You can create snapshot files in the default snapshot folder on the Distributor and in an alternate location. Compressing the snapshot files in the alternate location can reduce the overall space required.

When snapshot files are created in both the default folder and in an alternate location on the same drive, each file is created initially in the default folder and then copied to the alternate location. If you are using compressed snapshot files, the files are copied and compressed before they are placed in the alternate snapshot location. The total space required for all snapshot files in this situation is the size of the original snapshot files in the default location plus the size of the compressed snapshot files in the alternate location.

If the alternate storage location is on a different drive than the default location, the space required at the default location is the size of the snapshot files. The space required at the alternate location is the total size of the compressed snapshot files.

For more information, see <u>Transferring Snapshots</u>.

# **Scheduling Snapshots**

Concurrent snapshot processing is provided for transactional replication, and an optimized merge snapshot generation is provided for merge replication. Concurrent snapshot processing is conceptually similar to how a database backup can be performed while updates on the database continue.

With concurrent snapshot processing and transactional replication, at the time the Snapshot Agent runs, it places temporary shared locks on the publication tables that are released quickly so that data modifications at the database can continue. The data modifications made at this time are included as part of the initial snapshot. The snapshot is applied at the Subscriber, and the Distribution Agent reconciles each captured transaction to see if it has already been delivered to the Subscriber. During this reconciliation, the tables on the Subscriber are also temporarily locked.

To minimize the user from being temporarily unable to add to or update the table:

- Choose the concurrent snapshot processing with transactional replication when possible. Shared locks on the Publisher are only held for seconds.
- Identify times when the least amount of updates to data are needed and schedule the agent accordingly. Like a backup, the generation of the snapshot can be quite resource-intensive and that overhead will reduce the rest of the system performance during that time.

To plan the optimum schedule for running the Snapshot Agent, estimate the length of time it takes the Snapshot Agent to complete the snapshot. Because the

snapshot is created using **bcp**, perform a test bulk copy of your data set and time how long it takes to complete. If your data set is very large, perform the bulk copy on a sample of the data set and extrapolate the lapse time to the entire data set.

Not applying a snapshot is another option if you are concerned about interrupting activity on your database. You can set up a Subscriber manually such as from a database dump. This is known as manually applying the initial snapshot.

#### See Also

Copying Data Between Different Collations

# **Planning for Transactional Replication**

Transactional replication requires planning in the following areas:

- Transaction log space.
- Disk space for the distribution database.
- Primary keys for each table to be published.
- Immediate updating and queued updating.
- Transforming replicated data.
- **text** and **image** data types in transactional replication.
- Identity ranges.
- Constraints and NOT FOR REPLICATION.

#### **Transaction Log Space**

For each database that will be published in transactional replication, ensure that the transaction log has enough space allocated. The transaction log of a published database may require more space than the log of an identical, unpublished database. This is because the log records may not be purged until they have been moved to the distribution database.

If the distribution database is unavailable, or if the Log Reader Agent is not running, the transaction log of a publication database continues to grow. The log cannot be truncated past the oldest published transaction that has not been passed into the distribution database (unless replication is turned off completely for that database). It is recommended that you set the transaction log file to autogrow so that the log can accommodate these circumstances.

## **Disk Space for the Distribution Database**

If you plan to create transactional publications and make the snapshot files available to Subscribers immediately, allow enough disk space for the distribution database to store all of the transactions after the last snapshot. Although making the snapshot available to Subscribers immediately improves the speed with which new Subscribers have access to the publication, the option does require a larger disk storage area for the distribution database. It also means that a new snapshot will be generated each time the Snapshot Agent runs. If the option is not used, and if anonymous subscriptions are not allowed, a new snapshot needs to be generated only if there is a new subscription.

The distribution database begins collecting transactions immediately and continues to store them until the second time the Snapshot Agent is run (either scheduled or run manually). After the second time the Snapshot Agent is run, the cleanup task begins to clean up and reduce the size of the distribution database by deleting the rows from the first snapshot. Thus, if you use the default schedule of once a day for running the Snapshot Agent, you must have enough disk space to store all the transactions that occur in one day.

Similarly, if you plan to create transactional publications and allow anonymous subscriptions to a publication, you must allow enough disk space for the distribution database to store all of the transactions since the last snapshot. Allowing anonymous subscriptions also means that a new snapshot will be generated every time the Snapshot Agent runs.

An alternative to allocating more disk space in both of these situations is to run the Snapshot Agent more frequently than once a day (the default) so fewer commands must be retained in the distribution database. However, generating a snapshot can be resource-intensive and can affect performance temporarily. Reducing the distribution retention period (in Publisher and Distributor Properties) can also help maintain fewer commands because the Distribution Clean Up Agent is controlled by the distribution retention period and will remove replicated transactions from the distribution database.

### **Primary Keys**

All published tables in transactional replication must contain a declared primary key. Existing tables can be prepared for publishing by adding a declared primary key using the Transact-SQL statement ALTER TABLE.

#### text and image Data Types in Transactional Replication

The process of replicating **text** and **image** data types in a transactional publication is subject to the following considerations:

- INSERT, UPDATE, and DELETE statements at the Publisher on **text** and **image** columns are supported with no special considerations. However, these columns cannot be updated by Subscribers that use snapshot replication or transactional replication and immediate updating or queued updating subscriptions.
- Logged text operations can be replicated by using WRITETEXT and UPDATETEXT with the WITH LOG option on tables that are published for replication. A **text** or **image** column that is published for replication using WRITETEXT and UPDATETEXT operations with the WITH NO\_LOG option is not supported because replication reads the transaction log.
- UPDATETEXT operations can be performed only if all Subscribers are running Microsoft® SQL Server<sup>™</sup> version 6.0 or later Subscribers. WRITETEXT operations are replicated as UPDATE statements, enabling replication of WRITETEXT to ODBC Subscribers as well as to SQL Server. (UPDATETEXT operations are replicated as only UPDATETEXT.)
- Custom procedures are not used if multiple **text** columns are being modified because the other **text** column values are not logged. Instead, a standard UPDATE statement is generated.
- A configurable parameter, **max text repl size**, controls the maximum size (in bytes) of **text** and **image** data that can be replicated. This

permits support of ODBC drivers and instances of SQL Server that cannot handle large **text** and **image** values, and Distributors that have system resource (virtual memory) constraints. When a **text** or **image** column is published and an INSERT, UPDATE, WRITETEXT, or UPDATETEXT operation is run that exceeds the configured limit, the operation fails.

- Using the **sp\_configure** system stored procedure sets the **max text repl size** parameter.
- When publishing **text** and **image** columns, the text pointer should be retrieved within the same transaction as the UPDATETEXT or WRITETEXT operation (and with read repeatability). For example, do not retrieve the text pointer in one transaction and then use it in another. It may have moved and become invalid.
- In addition, when the text pointer has been obtained, you should not perform any operations that can alter the location of the text pointed to by the text pointer (such as updating the primary key), before executing the UPDATETEXT or WRITETEXT statement.

This is the recommended way of using UPDATETEXT and WRITETEXT operations with data to be replicated:

- 1. Begin the transaction.
- 2. Obtain the text pointer with read repeatable isolation.
- 3. Use the text pointer in the UPDATETEXT or WRITETEXT operation.
- 4. Commit the transaction.

**Note** If you do not obtain the text pointer in the same transaction, modifications are allowed at the Publisher, but changes are not

published to Subscribers.

An important consideration when sizing Subscriber databases is that the text pointer for replicated **text** and **image** columns must be initialized on Subscriber tables, even when they are not initialized on the Publisher. Consequently, each **text** and **image** column added to the Subscriber table by the distribution task will consume at least 43 bytes of database storage even if the contents are empty.

# **Planning for Merge Replication**

Merge replication requires planning in the following areas:

- timestamp columns.
- Identity ranges.
- Data integrity.
- Primary keys.
- Synchronizing with alternate synchronization partners.
- Row-level tracking and column-level tracking.
- Triggers and business rules.
- **text** and **image** data types in merge replication.
- Conflict resolution.
- Occassional maintenance for online/offline applications

#### timestamp Columns

Merge replication supports **timestamp** columns. The **timestamp** column is replicated, but the literal **timestamp** values are not. The **timestamp** values are regenerated when the initial snapshot rows are applied at the Subscriber. This allows **timestamp** values to be used by client applications at the Subscriber for functions such as optimistic concurrency control. In those cases, the ODBC

driver, OLE DB provider, DB-Library cursor, or server cursor used to implement optimistic concurrency control compares the **timestamp** value of the row being updated with the current local value of the original row. If the **timestamp** values are different, indicating a row has changed, the application can take appropriate action (such as rolling back the transaction or rereading the data). Because the **timestamp** values are regenerated at the Subscriber, **timestamp** columns are filtered out when performing article validation.

### **Data Integrity**

Because merge replication propagates changes made at the Subscriber, you must ensure that the application integrity is preserved at each Subscriber. All controls used to validate data changes at the Publisher should also be present at the Subscriber.

There are options to ensure that the login used by the Merge Agent to connect to the Publisher can also be used to control that only authenticated users can propagate data changes made at the Subscriber to the Publisher.

# Foreign Keys

When creating a merge publication, specify the tables that are included as articles in that publication. If you include tables that contain foreign keys, the referenced table should also be included in the publication. If an attempt is made to add new rows to an article referencing a primary key in a missing table, the insert fails because SQL Server 2000 cannot find the required primary key. If an attempt is made to update data in an existing row(s) of the article, the update succeeds because SQL Server 2000 does not have to add a new row(s) and key(s).

After they are created, merge publications can be modified to include additional articles. You can add any missing, referenced tables to a publication if you discover that an article must be updated with additional rows and not just with modifications to existing rows. Use the publication properties dialog box to add the missing table.

# Synchronizing with Alternate Synchronization Partners

Subscribers to merge publications can synchronize with servers other than the

Publisher where the subscription originated. Synchronizing with alternate synchronization partners provides the ability for a Subscriber to synchronize data even if the primary Publisher is unavailable, or if you can connect to another synchronization partner because of physical location (for example, if you are visiting a remote office and can connect to an alternate synchronization partner there).

Determine whether it will be necessary for merge replication Subscribers to have alternate synchronization partners, and then prepare those alternate servers for the synchronization.

For more information, see <u>Alternate Synchronization Partners</u>.

### **Conflict Detection and Resolution**

When determining merge replication conflict detection and resolution, you can specify whether you want the conflicts recognized at the row level or at the column level.

Whether to use row-level or column-level tracking should be decided based upon whether you want to consider any change within a row as a conflict (rowlevel tracking) or if different users will be allowed to update the same row simultaneously, but not the same column between synchronizations (columnlevel tracking).

The choice to use row-level versus column-level tracking should be based on your application and whether you want to consider any change to the same row in a table as a conflict or whether it is okay for different users to simultaneously update the same row, but not the same column, between synchronizations. For example, it might be considered acceptable in some applications that changes to different columns can be merged by using column-tracking. This means that if the Publisher changes column 1 and the Subscriber changes column 2, the merge process accepts the change to column 1 from the Publisher and change to column 2 from the Subscriber. Or some applications might require that changes to the same row at multiple sites (even if the values are in different columns) should be considered conflicts, detected and resolved at the row level.

For more information, see <u>Merge Replication Conflict Detection and Resolution</u>.

### **Triggers and Business Rules**

You should be aware of all triggers and constraints on a table that is replicated. Without planning, the triggers and constraints can be replicated along with the table and can cause recurring conflicts during merge replication. For more information, see <u>Publishing Data and Database Objects</u> and <u>Using NOT FOR REPLICATION</u>.

### text and image Data Types in Merge Replication

Merge replication supports the replication of **text**, **ntext**, and **image** columns only if they have been updated explicitly by an UPDATE statement because it causes a trigger to fire that updates meta data ensuring that the transaction gets propagated to other Subscribers.

Using only the WRITETEXT and UPDATETEXT operations will not propagate the change to other sites. If your application uses WRITETEXT and UPDATETEXT to update the **text** or **ntext** columns, explicitly add a dummy UPDATE statement after the WRITETEXT or UPDATETEXT operations, within the same transaction, to fire the trigger and thereby guarantee that the change will be propagated to other sites.

#### Example

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ BEGIN TRAN DECLARE @mytextptr varbinary(16) SELECT @mytextptr = textptr(Notes) FROM Employees WHERE EmployeeID = '7' IF @mytextptr IS NOT NULL BEGIN UPDATETEXT Employees.Notes @mytextptr 0 NULL 'Terrific job th -- Dummy update to fire trigger that will update meta data and ensure t UPDATE Employees -- Set value equal to itself. SET Notes = Notes WHERE EmployeeID = '7'

#### END COMMIT TRAN SET TRANSACTION ISOLATION LEVEL READ COMMITTED

#### **Occasional Maintenance for Online/Offline Applications**

When planning for an online/offline application that uses replication, plan for occasional maintenance in the deployment of the application and for a way to transfer new datasets to the disconnected Subscribers.

Although SQL Server 2000 replication allows for rich data access for occasionally connected Subscribers, or for Subscribers using a slow link, there will still be a need to plan for occasional maintenance of the application and possibly for reapplying a snapshot at the Subscriber.

# **Planning for Replication Options**

The replication options of immediate updating, queued updating, immediate updating with queued updating as a failover, and transforming replicated data require additional considerations during replication planning. If users do not need to update data at the Subscriber, consider using snapshot replication or transactional replication without immediate updating or queued updating options, and then replication will be easier to configure and administer.

#### **Considerations for Immediate Updating or Queued Updating Subscriptions**

Here are planning considerations for immediate updating or queued updating subscriptions:

- INSERT statements used to add rows of data to a table must include a column list.
- Subscribers using immediate updating or queued updating options cannot republish replicated data at the Subscriber.
- The Subscriber cannot update or insert **text** or **image** values. For more information, see <u>Updatable Subscriptions</u>.
- After a publication is enabled for either immediate updating subscriptions or queued updating subscriptions, the option cannot be disabled for the publication (although subscriptions do not need to use it); to delete the option, the publication must be deleted and a new one created.
- Snapshot replication does not require the use of primary keys in a table. However, transactional replication by itself or snapshot replication with any updatable subscriptions does require the use of primary keys.

• If you enable immediate updating and/or queued updating on a publication, you cannot also use transformable subscriptions. The Transform Published Data page will not show in the Create Publication Wizard if you have already chosen to use immediate updating and/or queued updating.

### Additional Considerations for Immediate Updating Subscriptions

Immediate updating allows snapshot replication and transactional replication Subscribers to update the replicated data at the Subscriber and propagate those changes to the Publisher, which then propagates to all other Subscribers.

Consider the following when planning to use snapshot replication or transactional replication with immediate updating:

- A uniqueidentifier column is used to track updates. The uniqueidentifier column is added automatically to any tables used in the publication. The addition of this column requires INSERT statements to have column lists. If you used immediate updating in Microsoft® SQL Server<sup>™</sup> version 7.0 and are upgrading to SQL Server 2000, you will need to subscribe to the publication again. For more information, see <u>Replication and Upgrading</u>.
- Using this option, the update is distributed and performed at both the Publisher and Subscriber using two-phase commit protocol (2PC): one locally at the Subscriber and one at the Publisher. This requires that the Publisher and the Subscriber making the change be available and connected.
- The immediate updating subscription connection to the Publisher (controlled by **sp\_link\_publication**) can use security mode 0 for SQL Server Authentication or 2 for linked server definition to create login mappings. The publication access list (PAL) must include at least one SQL Server Authentication account unless you use security mode 2 and configure delegation (it is possible to set up Windows Authentication in mode 2 by configuring delegation). You can make connections to the Publisher under Windows user accounts invoking the INSERT,

UPDATE, and DELETE triggers at the Subscriber using delegation. To set up delegation, see <u>sp\_addlinkedsrvlogin</u>.

### **Additional Considerations for Queued Updating Subscriptions**

Queued updating allows snapshot replication and transactional replication Subscribers to modify published data without requiring a continuous connection to the Publisher.

When you create a publication with the queued updating option enabled and a Subscriber that is enabled for queued updating performs inserts, updates, or deletes on published data, the changes are stored in a queue. The queued transactions are applied asynchronously at the Publisher when network connectivity is restored.

Consider the following when planning to use snapshot replication or transactional replication with queued updating:

- Because the updates are propagated asynchronously to the Publisher, the same data may have been updated by the Publisher or by another Subscriber and conflicts can occur when applying the updates. You will need to choose an appropriate conflict resolution policy when creating the publication.
- For snapshot replication, tables should have, at least, a unique index and preferably a primary key. For transactional replication, tables must have a primary key.
- If the Subscriber database is partitioned horizontally and there are rows in the partition that exist at the Subscriber, but not at the Publisher, the Subscriber cannot update the preexisting rows. Attempting to update these rows returns an error. The rows should be deleted from the table and then added again.
- Manage **identity** values with **identity** ranges to ensure that different Subscribers have different **identity** values. For more information, see <u>Replication Data Considerations</u>.

#### **Considerations for Transforming Published Data**

You can transform data during the replication process by leveraging the capabilities of Data Transformation Services (DTS). Examples of transforming published data are creating custom horizontal and vertical data partitions and creating data transformations such as data type mappings, column manipulations, and string manipulations.

Consider the following when planning to transform replicated data:

- Snapshot data for a transformable subscription is limited to character mode only; native format (which is usually faster to apply) cannot be used with DTS.
- After a publication is enabled for transformable subscriptions, the option cannot be disabled; the existing publication must be deleted and a new one created, but if the option is enabled, subscriptions do not need to use it.
- You cannot use immediate updating or queued updating options with transformable subscriptions (transformations are mapped in one direction, from Publisher to Subscriber).
- Although using the Transform Published Data Wizard creates a DTS package, this type of DTS package is not available for execution outside of replication (from DTS Designer or at the command prompt). However, you can use a package created with DTS tools during replication of snapshot and transactional publications that allow transforming of published data.
- Introducing DTS transformations into replication adds overhead and reduces the distribution performance. The amount depends on the complexity of the transformation. It does not affect Log Reader Agent performance.

### See Also

Filtering Published Data Immediate Updating Queued Updating Transforming Published Data

# **Merge Replication or Updatable Subscriptions**

When replicated data needs to be updated at the Subscribers, you can use snapshot replication or transactional replication with updatable subscription options or you can use merge replication. The method you choose depends on your replication topology and the needs of your application and its users.

Use merge replication when	Use snapshot replication or transactional replication with immediate updating or queued updating when
• Replicated data is read and updated at the Subscriber.	<ul> <li>Replicated data is mostly read- only at the Subscriber.</li> </ul>
• Subscriber and Publisher are only occasionally connected.	• Subscriber, Distributor, and Publisher are connected most of the time, but this is not necessary for queued updating subscriptions.
• Conflicts caused by multiple updates to the same data are handled and resolved.	• Conflicts caused by multiple updates to the same data are infrequent.
• You need updates to be propagated on a row- by-row basis, and conflicts to be evaluated and resolved at the row level.	• You need updates to be propagated on a transaction basis, and conflicts to be evaluated and resolved on a transaction basis (the entire transaction is either committed or rolled back).

### See Also

Merge ReplicationPlanning for Merge ReplicationPlanning for Replication OptionsUpdatable Subscriptions

# **Designing a Replication Topology**

A replication topology defines the relationship between servers and the copies of data, along with the logic that determines how synchronization occurs between copies. Designing a replication topology helps you determine how long it takes for changes to get from a Publisher to a Subscriber, whether the failure of one update prevents other Subscribers from being updated, and the order in which updated information arrives at a Subscriber, which can affect analysis and reporting.

To determine your replication topology:

- Select the physical replication model (central Publisher, central Publisher with remote Distributor, publishing Subscriber, or central Subscriber).
- Determine where snapshot files will be located and how Publishers and Subscribers will synchronize initially.
- Determine whether the Distributor will be local or remote, and determine whether the distribution database will be shared.
- Determine if multiple Publishers will share a Distributor, each use its own distribution database on the Publisher, or share a distribution database.
- Determine the type of replication and options to use.
- Determine whether replication is initiated at the Publisher (using push subscriptions) or at the Subscriber (using pull subscriptions).

The replication topology is not limited to the physical connections between servers because it also includes data paths between copies of the data. A Subscriber can receive multiple copies of data from different Publishers, and all of those data copies can exist on one server, incorporating a complicated topology.

### See Also

Synchronizing Data Transferring Snapshots Types of Replication

# **Physical Replication Models**

The physical replication model is the map for how data will be distributed across your enterprise and for how you will configure your servers during replication implementation. Based on all the factors and considerations outlined in Distributed Data Factors, Evaluating the Replication Environment, and Planning for Each Type of Replication, you should be able to determine the best solution for your replication model.

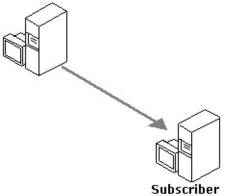
The following are examples of replication models:

- Central Publisher.
- Central Publisher with remote Distributor.
- Republisher.
- Central Subscriber.

# **Central Publisher**

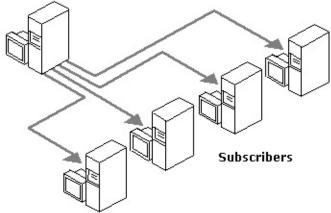
The simplest Microsoft® SQL Server<sup>™</sup> 2000 replication topology model places one Publisher and one Distributor on the same server and one Subscriber on a separate server.

Publisher and Distributor



The scenario becomes more complex as you add Subscribers to the Publisher and Distributor. The Publisher owns the data being published and becomes a central Publisher for all the Subscribers. For example, this scenario might be used to distribute master data, lists, or reports from a central Publisher to any number of Subscribers.

#### Publisher and Distributor



The roles of Publisher and Subscriber are not exclusive; servers can perform both simultaneously. For example, suppose Server A publishes Publication 1, and Server B publishes Publication 2. In this case, Server A could act both as a Publisher of Publication 1 and a Subscriber to Publication 2. This is an example of filtering data and publishing partitions.

# **Central Publisher with Remote Distributor**

As the level of replication activity increases or as server or network resources become constrained, there may be performance reasons to place the Publisher and Distributor on separate servers. This may be appropriate when a busy online transaction processing (OLTP) server is configured as a Publisher. Using a separate Distributor reduces local processing and disk usage on the Publisher, although it increases overall network traffic.

This scenario is similar to the central Publisher scenario, except that separate computers perform the publishing and distribution tasks. This is useful when the Publisher (for example, a heavily used OLTP server) should be freed from the distribution tasks because of performance and storage space considerations. The Publisher should be connected to the Distributor by a reliable, high-speed communications link.

#### See Also

Central Publisher

### Republisher

The republisher model uses two servers to publish the same data. The Publisher sends data to a Subscriber, which then republishes the data to any number of Subscribers. This is useful when a Publisher must send data to Subscribers over a slow or expensive communications link. If there are a number of Subscribers on the far side of that link, using a republisher shifts the bulk of the distribution load to that side of the link.

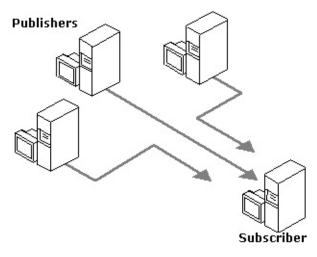
In this diagram, both the Publisher and the republisher (publishing Subscriber) are acting as their own local Distributors. If each were set up to use a remote Distributor, each Distributor would need to be on the same side of the slow or expensive communications link as its Publisher. Publishers must be connected to remote Distributors by reliable, high-speed communications links.

Any server can act as both Publisher and Subscriber. For example, consider the publication of a table that exists in New York and needs to be distributed to four different cities in Europe: London, Oslo, Paris, and Lisbon. The server in London is chosen to subscribe to the published table originating in New York, because the London site meets these conditions:

- The network link back to New York is relatively reliable.
- The New York-to-London communication costs are acceptable.
- There are good network communications lines from London to all other European Subscriber sites.

### **Central Subscriber**

In a central Subscriber model, a number of Publishers replicate information to a common destination table at a Subscriber. The destination table is partitioned horizontally and contains a location-specific column as part of the primary key. Each Publisher replicates rows containing location-specific data.



For example, this replication configuration may be useful for rolling up inventory data from a number of servers at local warehouses into a central Subscriber at corporate headquarters. It could also be used to roll up information from autonomous business divisions within a company, or to consolidate order processing from dispersed locations.

### **Types of Replication**

Microsoft® SQL Server<sup>™</sup> 2000 provides the following types of replication that you can use in your distributed applications:

- Snapshot replication
- Transactional replication
- Merge replication

Each type provides different capabilities depending on your application, and different levels of ACID properties (atomicity, consistency, isolation, durability) of transactions and site autonomy. For example, merge replication allows users to work and update data autonomously, although ACID properties are not assured. Instead, when servers are reconnected, all sites in the replication topology converge to the same data values. Transactional replication maintains transactional consistency, but Subscriber sites are not as autonomous as they are in merge replication because Publishers and Subscribers generally should be connected continuously for updates to be propagated to Subscribers.

It is possible for the same application to use multiple replication types and options. Some of the data in the application may not require any updates at Subscribers, some sets of data may require updates infrequently, with updates made at only one or a few servers, while other sets of data may need to be updated daily at multiple servers.

Which type of replication you choose for your application depends on your requirements based on distributed data factors, whether or not data will need to be updated at the Subscriber, your replication environment, and the needs and requirements of the data that will be replicated. For more information, see <u>Planning for Replication</u>.

Each type of replication begins with generating and applying the snapshot at the Subscriber, so it is important to understand snapshot replication in addition to any other type of replication and options you choose.

# **Snapshot Replication**

Snapshot replication distributes data exactly as it appears at a specific moment in time and does not monitor for updates to the data. Snapshot replication is best used as a method for replicating data that changes infrequently or where the most up-to-date values (low latency) are not a requirement. When synchronization occurs, the entire snapshot is generated and sent to Subscribers.

Snapshot replication would be preferable over transactional replication when data changes are substantial but infrequent. For example, if a sales organization maintains a product price list and the prices are all updated at the same time once or twice each year, replicating the entire snapshot of data after it has changed is recommended. Creating new snapshots nightly is also an option if you are publishing relatively small tables that are updated only at the Publisher.

Snapshot replication is often used when needing to browse data such as price lists, online catalogs, or data for decision support, where the most current data is not essential and the data is used as read-only. These Subscribers can be disconnected if they are not updating the data.

Snapshot replication is helpful when:

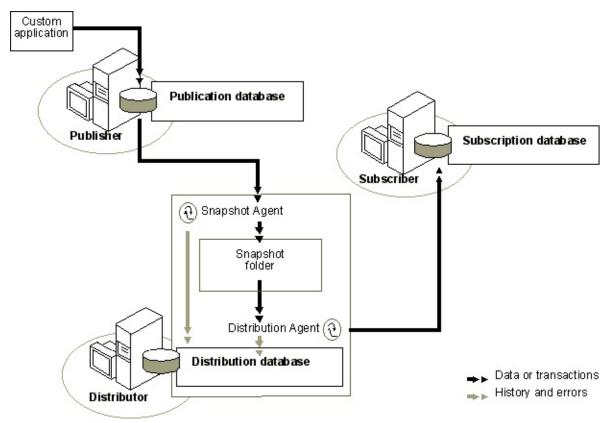
- Data is mostly static and does not change often. When it does change, it makes more sense to publish an entirely new copy to Subscribers.
- It is acceptable to have copies of data that are out of date for a period of time.
- Replicating small volumes of data in which an entire refresh of the data is reasonable.

Snapshot replication is mostly appropriate when you need to distribute a readonly copy of data, but it also provides the option to update data at the Subscriber. When Subscribers only read data, transactional consistency is maintained between the Publisher and Subscribers. When Subscribers to a snapshot publication must update data, transactional consistency can be maintained between the Publisher and Subscriber because the data is propagated using twophase commit protocol (2PC),a feature of the immediate updating option. Snapshot replication requires less constant processor overhead than transactional replication because it does not require continuous monitoring of data changes on source servers. If the data set being replicated is very large, it can require substantial network resources to transmit. In deciding if snapshot replication is appropriate, you must consider the size of the entire data set and the frequency of changes to the data.

# **How Snapshot Replication Works**

Snapshot replication is implemented by the Snapshot Agent and the Distribution Agent. The Snapshot Agent prepares snapshot files containing schema and data of published tables and database objects, stores the files in the snapshot folder, and records synchronization jobs in the distribution database on the Distributor. By default, the snapshot folder is located on the Distributor, but you can specify an alternate location instead of or in addition to the default. For more information, see <u>Alternate Snapshot Locations</u>.

The Distribution Agent moves the snapshot held in the distribution database tables to the destination tables at the Subscribers. The distribution database is used only by replication and does not contain any user tables.



#### **Snapshot Agent**

Each time the Snapshot Agent runs, it checks to see if any new subscriptions have been added. If there are no new subscriptions, no new scripts or data files

are created. If the publication is created with the option to create the first snapshot immediately enabled, new schema and data files are created each time the Snapshot Agent runs. All schema and data files are stored in the snapshot folder and then either the Distribution Agent or Merge Agent transfers them to Subscriber or you can transfer them manually. The Snapshot Agent performs the following steps:

- 1. Establishes a connection from the Distributor to the Publisher and sets a share-lock on all tables included in the publication. The share-lock ensures a consistent snapshot of data. Because the locks prevent all other users from updating the tables, the Snapshot Agent should be scheduled to execute during off-peak database activity.
- 2. Establishes a connection from the Publisher to the Distributor and writes a copy of the table schema for each article to an .sch file. If you request that indexes and declarative referential integrity be included, the agent scripts out the selected indexes to an .idx file. Other database objects, such as stored procedures, views, user-defined functions, and others, can also be published as part of replication.
- 3. Copies the data in the published table on the Publisher and writes the data to the snapshot folder. If all Subscribers are instances of Microsoft® SQL Server<sup>™</sup> 2000, the snapshot is stored as a native bulk copy program file. If one or more Subscribers is a heterogeneous data source, the snapshot is stored as a character mode file. The files are the synchronization set that represents the table at one point in time. There is a synchronization set for each article within a publication.
- 4. Appends rows to the MSrepl\_commands and MSrepl\_transactions tables in the distribution database. The entries in the MSrepl\_commands tables are commands indicating the location of the synchronization set (.sch and .bcp files) and references to any specified pre-creation scripts. The entries in the MSrepl\_transactions table are commands referencing the Subscriber synchronization task.

5. Releases the share-locks on each published table and finishes writing the log history tables.

After the snapshot files are generated, you can view them in the Snapshot Folder using the Snapshot Explorer. In SQL Server Enterprise Manager, expand the Replication and Publications folders, right-click a publication, and then click **Explore the Latest Snapshot Folder**. For more information, see <u>Exploring Snapshots</u>.

#### **Distribution Agent**

Each time the Distribution Agent runs for a snapshot publication, it moves the schema and data to Subscribers. The Distribution Agent performs the following steps:

- 1. Establishes a connection from the server where the agent is located to the Distributor. For push subscriptions, the Distribution Agent is usually run on the Distributor, and for pull subscriptions, the Distribution Agent is usually run on the Subscriber.
- 2. Examines the **MSrepl\_commands** and **MSrepl\_transactions** tables in the distribution database on the Distributor. The agent reads the location of the synchronization set from the first table and the Subscriber synchronization commands from both tables.
- 3. Applies the schema and commands to the subscription database. If the Subscriber is not an instance of Microsoft SQL Server 2000, the agent converts the data types as necessary. All articles of a publication are synchronized, preserving transactional and referential integrity between the underlying tables (presuming the subscription database, if not SQL Server, has the transactional capabilities to do so).

When handling a large number of Subscribers, running the Distribution Agent at the Subscriber, either by using pull subscriptions or by using remote agent activation, can save processing resources on the Distributor. With remote agent activation, you can choose to run the Distribution Agent at the Subscriber for push subscriptions or at the Distributor for pull subscriptions. For more information, see <u>Remote Agent Activation</u>.

Snapshots can be applied either when the subscription is created or according to a schedule set at the time the publication is created.

**Note** For agents running at the Distributor, scheduled synchronization is based on the date and time at the Distributor (not the date and time at the Subscribers). Otherwise, the schedule is based on the date and time at the Subscriber.

Because automatic synchronization of databases or individual tables requires increased system overhead, a benefit of scheduling automatic synchronization for less frequent intervals is that it allows the initial snapshot to be scheduled for a period of low activity on the Publisher.

The Snapshot Agent is usually run by SQL Server Agent and can be administered directly by using SQL Server Enterprise Manager. The Snapshot Agent and Distribution Agent can also be embedded into applications by using Microsoft ActiveX® controls. The Snapshot Agent executes on the Distributor. The Distribution Agent usually executes on the Distributor for push subscriptions, or on Subscribers for pull subscriptions, but remote agent activation can be used to offload Distribution Agent processing to another server.

### **Cleaning Up Snapshot Replication**

When the distribution database is created, SQL Server 2000 adds the following tasks at the Distributor:

- Agent checkup
- Transaction cleanup
- History cleanup

These tasks help replication to function effectively in a long-running environment. After the snapshot is applied at all Subscribers, replication cleanup deletes the associated .bcp file for the initial snapshots automatically.

If the publication is enabled for anonymous subscriptions or with the option to create the first snapshot immediately, at least one copy of the snapshot files are

kept in the snapshot location. This ensures that if a Subscriber with an anonymous subscription to a snapshot publication synchronizes with the Publisher, the most recent snapshot will be available.

### See Also

Planning for Snapshot Replication

**Replication Options** 

## **Transactional Replication**

With transactional replication, an initial snapshot of data is applied at Subscribers, and then when data modifications are made at the Publisher, the individual transactions are captured and propagated to Subscribers.

Transactional replication is helpful when:

- You want incremental changes to be propagated to Subscribers as they occur.
- You need transactions to adhere to ACID properties.
- Subscribers are reliably and/or frequently connected to the Publisher.

Transactional replication uses the transaction log to capture incremental changes that were made to data in a published table. Microsoft® SQL Server<sup>™</sup> 2000 monitors INSERT, UPDATE, and DELETE statements, or other modifications made to the data, and stores those changes in the distribution database, which acts as a reliable queue. Changes are then propagated to Subscribers and applied in the same order as they occurred.

With transactional replication, incremental changes made at the Publisher flow according to the Distribution Agent schedule. This schedule can be set to continuously for minimal latency, or set at scheduled intervals to Subscribers. Because changes to the data must be made at the Publisher (when transactional replication is used without immediate updating or queued updating options), update conflicts are avoided. This guarantees ACID properties of transactions will be maintained. Ultimately, all Subscribers will achieve the same values as the Publisher. If immediate updating or queued updating options are used with transactional replication, updates can be made at the Subscriber, and with queued updating, conflicts might occur.

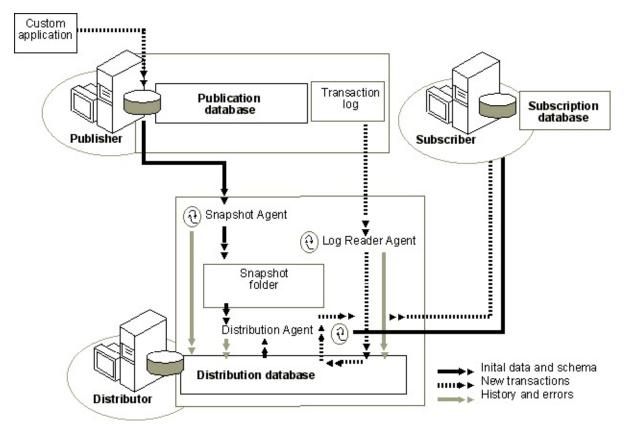
If Subscribers need to receive data changes in near real-time, they need a network connection to the Publisher. Transactional replication can provide very low latency to Subscribers. Subscribers receiving data using a push subscription usually receive changes from the Publisher within one minute or sooner, provided that the network link and adequate processing resources are available (latency of a few seconds can often be achieved).

However, Subscribers can also pull changes down as needed. A traveling sales representative can be a Subscriber and request incremental changes to a price list, which is only modified at the corporate office, once each evening. The use of transactional replication for disconnected users can be very effective for readonly data.

## **How Transactional Replication Works**

Transactional replication is implemented by the Snapshot Agent, Log Reader Agent, and Distribution Agent. The Snapshot Agent prepares snapshot files containing schema and data of published tables and database objects, stores the files in the snapshot folder, and records synchronization jobs in the distribution database on the Distributor.

The Log Reader Agent monitors the transaction log of each database configured for transactional replication and copies the transactions marked for replication from the transaction log into the distribution database. The Distribution Agent moves the initial snapshot jobs and the transactions held in the distribution database tables to Subscribers.



#### **Initial Snapshot**

Before a new transactional replication Subscriber can receive incremental changes from a Publisher, the Subscriber must contain tables with the same

schema and data as the tables at the Publisher. Copying the complete current publication from the Publisher to the Subscriber is called applying the initial snapshot. Microsoft® SQL Server<sup>™</sup> 2000 will create and apply the snapshot for you, or you can choose to apply the snapshot manually. For more information, see <u>Applying the Initial Snapshot</u>.

When snapshots are distributed and applied to Subscribers, only those Subscribers waiting for initial snapshots are affected. Other Subscribers to that publication (those that are already receiving inserts, updates, deletes, or other modifications to the published data) are unaffected.

#### **Concurrent Snapshot Processing**

Typically with snapshot generation, SQL Server will place shared locks on all tables published as part of replication for the duration of snapshot generation. This can prevent updates from being made on the publishing tables. Concurrent snapshot processing, available only with transactional replication, does not hold the share locks in place during the entire snapshot generation, therefore, it allows users to continue working uninterrupted while SQL Server 2000 creates initial snapshot files.

When you create a new publication using transactional replication and indicate that all Subscribers will be instances of SQL Server 7.0 or SQL Server 2000, concurrent snapshot processing is available.

After replication begins, the Snapshot Agent places shared locks on the publication tables. The locks prevent changes until a record indicating the start of the snapshot is entered in the log file. After the transaction is received, the shared locks are released and data modifications at the database can continue. The duration for holding the locks is very brief (a few seconds) even if a large amount of data is being copied.

At this point, the Snapshot Agent starts to build the snapshot files. When the snapshot is complete, a second record indicating the end of the snapshot process is written to the log. Any transactions that affect the tables while the snapshot is being generated are captured between these beginning and ending tokens and forwarded to the distribution database by the Log Reader Agent.

When the snapshot is applied at the Subscriber, the Distribution Agent first applies the snapshot files (schema and .bcp files). It then reconciles each

captured transaction to see if it has already been delivered to the Subscriber. During this reconciliation process, the tables on the Subscriber are locked. Depending on the number of transactions captured at the Publisher while the snapshot was created, you should expect an increase in the amount of time required to apply the snapshot at the Subscriber. Conceptually, this is similar to the process of recovery that SQL Server uses when it is restarted.

UPDATETEXT statements cannot be performed on data marked for replication while it is being extracted during concurrent snapshot processing. If you initiate an UPDATETEXT statement, you will get an error indicating that the operation is not allowed because of concurrent snapshot processing. After the snapshot is complete, UPDATETEXT statements can be performed again.

As mentioned earlier, use caution when concurrent snapshot processing occurs on systems where business logic is indicated through triggers or constraints on the subscription database. Concurrent snapshot processing uses bulk inserts of tables followed by a series of special INSERT and DELETE statements that bring the table to a consistent state. These operations are performed as one transaction so that database users do not see the data in an inconsistent state; however, constraints at the Subscriber will be executed within the transaction and may evaluate changes that are not based on a consistent set of data. To prevent this, it is generally recommended that you specify the NOT FOR REPLICATION option on all constraints and columns with the IDENTITY property on the Subscriber database. Business logic implemented using custom stored procedures will not be affected because custom stored procedures are not used during concurrent snapshot processing until the Subscriber tables are in a consistent state.

Foreign key constraints, check constraints, and triggers at the Subscriber do not require the NOT FOR REPLICATION option because they will be disabled during the concurrent snapshot generation and will be enabled after the snapshot is generated.

**IMPORTANT** The Log Reader Agent must run after the snapshot is generated with concurrent processing. If the Log Reader Agent does not run, the Distribution Agent will continue to return an error stating that the snapshot is not available and will not apply it to Subscribers. The Log Reader Agent needs to propagate all changes that occurred during snapshot generation to the distribution database before the Distribution Agent can apply the snapshot to Subscribers. Usually the

Log Reader Agent runs in continuous mode, so it will run automatically soon after the snapshot is generated, but this is not a concern. If you choose not to run the Log Reader Agent in continuous mode, you must run it manually.

Although concurrent snapshot processing allows updates to continue on publishing tables, the performance will be lowered due to the overhead of the snapshot itself. It is recommended that you generate the snapshot during periods of lowest general activity whenever possible (similar to when you would choose to do a database backup).

**IMPORTANT** If the publishing table has a primary key or unique constraint not contained within the clustered index, replication could fail if data modifications occur on the clustering key during concurrent snapshot processing. It is recommended that you enable concurrent snapshot processing only when unique and primary key constraints are contained within the clustered index or you ensure that data modifications are not made to the columns of the clustering index while the snapshot is generated.

Concurrent snapshot processing is available only with transactional replication and for Subscribers running instances of SQL Server 7.0 or later on the Microsoft Windows® 98, Microsoft Windows NT® 4.0 and Microsoft Windows 2000 operating systems.

If you are publishing to Subscribers running SQL Server 7.0, the Distributor must be running SQL Server 2000, and you must use push subscriptions to use concurrent snapshot processing. The Distribution Agent runs at the Distributor, and is able to execute the concurrent snapshot processing. If you used a pull subscription, the Distribution Agent would run at the Subscriber on SQL Server 7.0 where concurrent snapshot processing is not available. If you use pull subscriptions with Subscribers running SQL Server 7.0, concurrent snapshot processing must be disabled.

Because of these restrictions, the Create Publication Wizard does not make concurrent snapshot processing the default when you create a transactional publication; however, if your application meets these criteria, it is recommended that you enable this option. To enable concurrent snapshot processing, change the snapshot generation mode. Open **Publication Properties**, click the **Snapshot** tab, and then select the **Concurrent access during snapshot generation** checkbox.

#### **Snapshot Agent**

The procedures by which the Snapshot Agent implements the initial snapshot in transactional replication are the same procedures used in snapshot replication (except as outlined earlier with regard to concurrent snapshot processing). After the snapshot files have been generated, you can view them in the Snapshot Folder using the Snapshot Explorer. In SQL Server Enterprise Manager, expand the Replication and Publications folders, right click a publication, and then click **Explore the Latest Snapshot Folder**. For more information, see <u>Exploring Snapshots</u>.

#### Modifying Data and the Log Reader Agent

The Log Reader Agent runs either continuously or according to a schedule you establish at the time the publication is created. When executing, the Log Reader Agent first reads the publication transaction log (the same database log used for transaction tracking and recovery during regular SQL Server 2000 operations) and identifies any INSERT, UPDATE, and DELETE statements, or other modifications made to the data transactions that have been marked for replication. Next, the agent batch copies those transactions to the distribution database at the Distributor. The Log Reader Agent uses the internal stored procedure **sp\_replcmds** to get the next set of commands marked for replication from the log. The distribution database then becomes the store-and-forward queue from which changes are sent to Subscribers. Only committed transactions are sent to the distribution database.

There is a one-to-one correspondence between transactions on the Publisher and replication transactions in the distribution database. One transaction stored in **MSrepl\_transactions** can consist of one or more commands and each command can be broken up along a 500-Unicode-character boundary in the **MSrepl\_commands** table. After the entire batch of transactions has been written successfully to the distribution database, it is committed. Following the commit of each batch of commands to the Distributor, the Log Reader Agent calls **sp\_repldone** to mark where replication was last completed. Finally, the agent marks the rows in the transaction log that are ready to be truncated. Rows still waiting to be replicated are not truncated. The transaction log on the Publisher can be dumped without interfering with replication, because only transactions not marked for replication are purged.

Data modifications made at the Subscriber will always be propagated as a series of single row statements, provided they do not modify a uniquely constrained column. If an UPDATE does modify a uniquely constrained column, the UPDATE will be propagated as a series of DELETE statements followed by a series of INSERT statements. A uniquely constrained column is any column participating in a unique index or clustered index, even if the clustered index is not declared as unique. UPDATES made to indexed views or base tables that indexed views are based on will be propagated as DELETE/INSERT pairs.

The Log Reader Agent usually runs under SQL Server Agent at the Distributor and can be administered directly by accessing it in SQL Server Enterprise Manager under Replication Monitor and the Agents folder.

### **Distribution Agent**

Transaction commands are stored in the distribution database until the Distribution Agent propagates them to all Subscribers or a Distribution Agent at the Subscriber pulls the changes. The distribution database is used only by replication and does not contain any user tables. You should never create other objects in the distribution database. Subscribers will receive transactions in the same order in which they were applied at the Publisher.

The Distribution Agent is a component of SQL Server Agent and can be administered directly by using SQL Server Enterprise Manager. The Snapshot Agent and Distribution Agent can also be embedded into applications by using Microsoft ActiveX® controls. The Snapshot Agent executes on the Distributor. The Distribution Agent usually executes on the Distributor for push subscriptions, or on Subscribers for pull subscriptions, but remote agent activation can be used to offload agent processing to another server. For more information, see <u>Remote Agent Activation</u>.

SQL Server can validate the data being updated at the Subscriber as the replication process is occurring so that you can ensure that data is the same at the Publisher and at the Subscribers. For more information, see <u>Validating</u> <u>Replicated Data</u>.

### **Skipping Errors in Transactional Replication**

The **-skiperrors** agent command line parameter for transactional replication

allows you to specify errors that can be skipped during the distribution process. Typically, when the Log Reader Agent and Distribution Agent are running in continuous mode and one of them encounters an error, the agent, and the distribution process, stops. By specifying expected errors or errors that you do not want to interfere with replication, with the **-skiperrors** parameter, the Distribution Agent will log the error information and then continue running. For more information, see <u>Handling Agent Errors</u>.

### **Cleaning Up Transactional Replication**

When the distribution database is created, SQL Server adds the following tasks to SQL Server Agent at the Distributor to purge the data no longer required:

- Agent checkup
- Agent history cleanup
- Transaction cleanup
- Distribution cleanup
- History cleanup
- Expired subscription cleanup

After all Subscribers have received transactions, the Distribution Cleanup Agent removes delivered transactions in the distribution database. Delivered transactions are kept in the distribution database for a defined period known as the retention period. Setting a retention period while scheduling backups can ensure that information required to recover a destination database automatically is available within the distribution database.

For example, if a Subscriber has scheduled a transaction log dump of a destination database every 24 hours, you could set the retention period to 48 hours. Even if the Subscriber experiences a failure immediately before a scheduled backup, all transactions necessary to restore the replicated tables

automatically will still be available to the distribution process of the Distributor.

#### See Also

Planning for Transactional Replication Replication Options

# **Merge Replication**

Merge replication is the process of distributing data from Publisher to Subscribers, allowing the Publisher and Subscribers to make updates while connected or disconnected, and then merging the updates between sites when they are connected.

Merge replication allows various sites to work autonomously and at a later time merge updates into a single, uniform result. The initial snapshot is applied to Subscribers, and then Microsoft® SQL Server<sup>™</sup> 2000 tracks changes to published data at the Publisher and at the Subscribers. The data is synchronized between servers continuously, at a scheduled time, or on demand. Because updates are made at more than one server, the same data may have been updated by the Publisher or by more than one Subscriber. Therefore, conflicts can occur when updates are merged.

Merge replication includes default and custom choices for conflict resolution that you can define as you configure a merge publication. When a conflict occurs, a resolver is invoked by the Merge Agent and determines which data will be accepted and propagated to other sites.

Merge Replication is helpful when:

- Multiple Subscribers need to update data at various times and propagate those changes to the Publisher and to other Subscribers.
- Subscribers need to receive data, make changes offline, and later synchronize changes with the Publisher and other Subscribers.
- You do not expect many conflicts when data is updated at multiple sites (because the data is filtered into partitions and then published to different Subscribers or because of the uses of your application). However, if conflicts do occur, violations of ACID properties are acceptable.

Insert Diagram (servers and data flow – laptops to indicate occasionally connected)

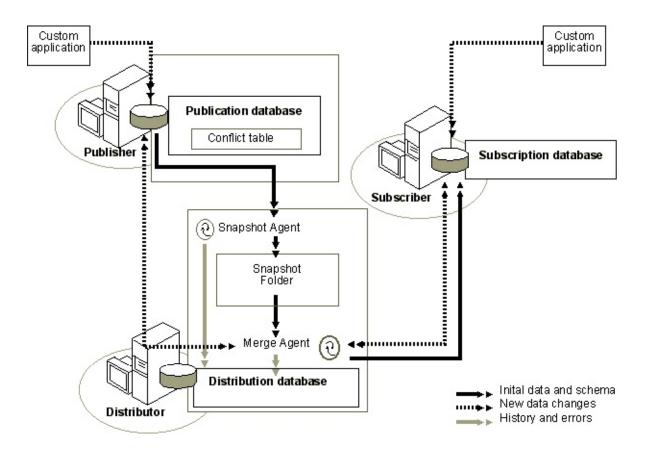
Both queued updating and merge replication allow updates at the Publisher and at Subscribers while offline; however, there are significant differences between the two methods. For more information, see <u>Merge Replication or Updatable</u> <u>Subscriptions</u>.

### **How Merge Replication Works**

Merge replication is implemented by the Snapshot Agent and Merge Agent. The Snapshot Agent prepares snapshot files containing schema and data of published tables, stores the files in the snapshot folder, and inserts synchronization jobs in the publication database. The Snapshot Agent also creates replication-specific stored procedures, triggers, and system tables.

The Merge Agent applies the initial snapshot jobs held in the publication database tables to the Subscriber. It also merges incremental data changes that occurred at the Publisher or Subscribers after the initial snapshot was created, and reconciles conflicts according to rules you configure or a custom resolver you create.

The role of the Distributor is very limited in merge replication, so implementing the Distributor locally (on the same server as the Publisher) is very common. The Distribution Agent is not used at all during merge replication, and the distribution database on the Distributor stores history and miscellaneous information about merge replication.



#### **UNIQUEIDENTIFIER Column**

Microsoft® SQL Server<sup>™</sup> 2000 identifies a unique column for each row in the table being replicated. This allows the row to be identified uniquely across multiple copies of the table. If the table already contains a column with the ROWGUIDCOL property that has a unique index or primary key constraint, SQL Server will use that column automatically as the row identifier for the publishing table.

Otherwise, SQL Server adds a **uniqueidentifier** column, titled **rowguid**, which has the ROWGUIDCOL property and an index, to the publishing table. Adding the **rowguid** column increases the size the publishing table. The **rowguid** column and the index are added to the publishing table the first time the Snapshot Agent executes for the publication.

#### Triggers

SQL Server then installs triggers that track changes to the data in each row or each column. The triggers capture changes made to the publishing table and

record the changes in merge system tables. Tracking triggers on the publishing tables are created while the Snapshot Agent for the publication runs for the first time. Triggers are created at the Subscriber when the snapshot is applied at the Subscriber.

Different triggers are generated for articles that track changes at the row level or the column level. Because SQL Server supports multiple triggers of the same type on the publishing table, merge replication triggers do not interfere with application-defined triggers.

#### **Stored Procedures**

The Snapshot Agent also creates custom stored procedures that update the subscription database. There is one custom stored procedure for INSERT statements, one for UPDATE statements, and one for DELETE statements. When data is updated and the new records need to be entered in the subscription database, the custom stored procedures are used rather than individual INSERT, UPDATE, and DELETE statements. For more information, see <u>Using Custom Stored Procedures in Articles</u>.

#### **System Tables**

SQL Server then adds several system tables to the database to support data tracking, efficient synchronization, and conflict detection, resolution and reporting. For every changed or created row, the table **MSmerge\_contents** contains the generation in which the most recent modification occurred. It also contains the version of the row as a whole and every attribute of the row. **MSmerge\_tombstone** stores DELETEs to the data within a publication. These tables use the **rowguid** column to join to the publishing table.

The **generation** column in these tables acts as a logical clock indicating when a row was last updated at a given site. Actual **datetime** values are not used for marking when changes occur, or deciding conflicts, and there is no dependence on synchronized clocks between sites. This makes the conflict detection and resolution algorithms more resilient to time zone differences and differences between physical clocks on multiple servers. At a given site, the generation numbers correspond to the order in which changes were performed by the Merge Agent or by a user at that site.

**MSmerge\_genhistory** and **MSmerge\_replinfo** allow SQL Server to determine the generations that need to be sent with each merge.

There are several tracking columns added to a merge publication table. If your publishing table has column names reserved for merge processing, you will not be able to generate an initial snapshot because of duplicate column names. Reserved column names are:

- reason\_code
- source\_object
- reason\_text
- Pubid
- conflict\_type
- origin\_datasource
- tablenick
- create\_time

#### **Initial Snapshot and the Snapshot Agent**

Before a new Subscriber can receive incremental changes from a Publisher, the Subscriber must contain tables with the same schema and data as the tables at the Publisher. Copying the complete current publication from the Publisher to the Subscriber is called applying the initial snapshot. SQL Server will create and apply the snapshot for you, or you can choose to apply the snapshot manually. For more information, see <u>Applying the Initial Snapshot</u>.

Even when creating a subscription for which the snapshot is not applied

automatically (sometimes referred to as a nosync subscription), portions of the snapshot are still applied. The necessary tracking triggers and tables are created at the Subscriber, which means that you still need to create and apply a snapshot even when subscriptions specify that the snapshot will not be applied automatically.

Replication of changed data occurs only after merge replication ensures that the Subscriber has the most recent snapshot of the table schema and data that has been generated. When snapshots are distributed and applied to Subscribers, only those Subscribers needing initial snapshots are affected. Subscribers that are already receiving INSERTs, UPDATEs, DELETEs, or other modifications to the published data are unaffected unless the subscription is marked for reinitialization or the publication is marked for a reintialization, in which case all subscriptions corresponding to a given publication are reintialized during the next merge process.

A subscription table can subscribe only to one merge publication at a time. For example, suppose you publish the **Customers** table in two publications, and then you subscribe to both publications from one Subscriber, indicating the same subscription database will receive data from both publications. One of the Merge Agents will fail during the initial synchronization.

The initial snapshot can be an attached subscription database in snapshot replication, transactional replication, and merge replication. If you use attachable subscription database, a subscription database and its subscriptions will be copied and you can apply them at another Subscriber. For more information, see <u>Attachable Subscription Databases</u>.

The Snapshot Agent implements the initial snapshot in merge replication using similar steps to the Snapshot Agent in snapshot replication. For more information, see <u>Snapshot Replication</u>.

After the snapshot files have been generated, you can view them in the Snapshot Folder using the Snapshot Explorer. In SQL Server Enterprise Manager, expand the Replication and Publications folders, right-click a publication, and then click **Explore the Latest Snapshot Folder**. For more information, see <u>Exploring Snapshots</u>.

#### **Dynamic Snapshots**

Dynamic snapshots provide a performance advantage when applying the snapshot of a merge publication with dynamic filters. By using SQL Server 2000 bulk copy programming files to apply data to a specific Subscriber instead of a series of INSERT statements, you will improve the performance of applying the initial snapshot for dynamically filtered merge publications.

For more information, see **Dynamic Snapshots**.

### Merge Agent

After the initial snapshot has been applied to a Subscriber, SQL Server triggers will begin tracking INSERT, UPDATE and DELETE statements made at the Publisher and at Subscribers.

Every table that participates in merge replication is assigned a generation slot in the **MSmerge\_articles** table. When a row is updated in a merge publication at the Publisher or at Subscribers, even if they are not connected, a trigger updates the **generation** column in the **MSmerge\_contents** system table for that row to the appropriate generations slot for the given base table. When the Publisher and Subscriber are reconnected and the Merge Agent runs, the Merge Agent collects all the undelivered row changes (with new generation values) into one or more groups and assigns generation values that are higher than all previous generations. This allows the Merge Agent to batch changes to different tables in separate generations and process these batches to achieve efficiency over slow networks.

The Merge Agent at each site keeps track of the highest generation it has sent to each of the other sites, and the highest generation that each of the other sites has sent to it. These provide starting points, so that each table can be examined without looking at data already shared with the other site. The generations stored in a given row can differ between sites because the numbers at a site reflect the order in which changes were processed at that site.

You can limit the number of merge processes running simultaneously by setting the *@max\_concurrent\_merge* parameter of *sp\_addmergepublication* or *sp\_changemergepublication*. If the maximum number of merge processes is already running, any new merge processes will wait in a queue. You can set – *StartQueueTimeout* on the Merge Agent command line to specify how long the agent should wait for the other merge processes to complete. If the –

**StartQueueTimeout** period is exceeded, and the new merge process is still waiting, it will stop and exit.

### Synchronization

Synchronization occurs when Publishers and Subscribers in a merge replication topology reconnect and changes are propagated between sites, and if necessary, conflicts detected and resolved. At the time of synchronization, the Merge Agent sends all changed data to the Subscriber. Data flows from the originator of the change to the site that needs to be updated or synchronized.

The direction of the exchange controls whether the Merge Agent uploads changes from the Subscriber (-ExchangeType='Upload'), downloads changes to the Publisher (-ExchangeType='Download') or executes an upload followed by a download (-ExchangeType='Bidirectional'). If the number of changes applied must be controlled, the Merge Agent command line parameters – MaxUploadChanges and –MaxDownloadChanges can be configured. In this case, the data at the Publisher and Subscribers converges only when all changes are propagated.

At the destination database, updates propagated from other sites are merged with existing values according to conflict detection and resolution rules. A Merge Agent evaluates the arriving and current data values, and any conflicts between new and old values are resolved automatically based on the default resolver, a resolver you specified when creating the publication or a custom resolver. Merge replication in SQL Server 2000 offers many out-of-the-box custom resolvers that will help you implement the business logic.

Changed data values are replicated to other sites and converged with changes made at those sites only when synchronization occurs. Synchronizations can occur minutes, days, or even weeks apart and are defined in the Merge Agent schedule. Data is converged and all sites ultimately end up with the same data values, but for this to happen, you would have to stop all updates and merge between sites a couple of times.

The retention period for subscriptions specified for each publication controls how often the Publisher and Subscribers should synchronize. If subscriptions do not synchronize with the Publisher within the retention period, they are marked as 'expired' and will need to be reinitialized. This is to prevent old Subscriber data from synchronizing and uploading these changes to the Publisher. The default retention period for a publication is 14 days. Because the Merge Agent cleans up the publication and subscription databases based on this value, care must be taken to configure this value appropriate to the application.

**Note** The merge process requires an entry for the Publisher in the **sysservers** table on the Subscriber. If the entry does not exist, SQL Server will attempt to add this entry. If the login used by the Merge Agent does not have access to add the entry (such as db\_owner of the subscription database), an error will be returned.

### **Reinitializing Subscriptions**

Merge replication Subscribers update data based on the original snapshot provided to them unless you mark the subscription for reinitialization. When you mark the subscription for reinitialization, the next time the Merge Agent runs, it will apply a new snapshot to the Subscriber. Optionally, changes made at the Subscriber can be uploaded to the Publisher before the snapshot is reapplied. This ensures that any data changes at the Subscriber are not lost when the subscription is reinitialized.

If you created a subscription and indicated no initial snapshot was to be applied at the Subscriber (the **@sync\_type** parameter set to **nosync** in **sp\_addmergesubscription** system stored procedure), and you reinitialize the subscription, the snapshot will be reapplied to the Subscriber. This functionality ensures that Subscribers have data and schema identical to data and schema at the Publisher.

If you reinitialize all subscriptions to a merge publication, the subscriptions specified with no initial snapshot synchronization will be reinitialized the same way the subscriptions with synchronization type of 'automatic' are reinitialized. To prevent the reapplication of the snapshot to the Subscriber, drop the subscription specified with no initial snapshot synchronization, and then recreate it after reinitialization.

For more information about synchronization, see <u>Synchronizing Data</u>.

The Merge Agent is a component of SQL Server Agent and can be administered directly by using SQL Server Enterprise Manager. The Snapshot Agent and Merge Agent can also be embedded into applications by using Microsoft

ActiveX® controls. The Snapshot Agent executes on the Distributor. The Merge Agent usually executes on the Distributor for push subscriptions and on Subscribers for pull subscriptions. Remote agent activation can be used to offload agent processing to another server. For more information, see <u>Remote Agent Activation</u>.

SQL Server can validate the data at the Subscriber as the replication process is occurring so that you can ensure that data updates applied at the Publisher are applied at Subscribers. For more information, see <u>Validating Replicated Data</u>.

### Validating Permissions for a Subscriber

SQL Server 2000 provides the option to validate permissions for a Subscriber to upload data changes to a Publisher. This verifies that the Merge Agent login has the permissions to perform INSERT, UPDATE, and DELETE commands on the publication database. Validating permissions requires that the Merge Agent login be a valid user with the appropriate permissions in the publication database.

This permissions validation is in addition to the verification that the logins used at the Subscriber are in the publication access list (PAL).

Validating permissions for a Subscriber can be set using the **@check\_permissions** property in **sp\_addmergearticle** or by using the CheckPermissions Property in SQL-DMO. For more information, see <u>CheckPermissions Property</u>. You can specify one or more of the following values for the **@check\_permissions** parameter in **sp\_addmergearticle**.

Value	Description
0 (Default)	Permissions will not be checked.
1	Check permissions at the Publisher before INSERTs made at a Subscriber can be uploaded.
2	Check permissions at the Publisher before UPDATEs made at a Subscriber can be uploaded.
4	Check permissions at the Publisher before DELETEs made at a Subscriber can be uploaded.

**Note** If you set the *@check\_permissions* parameter after the initial snapshot

has been generated, a new snapshot must be generated and reapplied at the Subscriber in order for permissions to be validated when data changes are merged.

### **Cleaning Up Merge Replication**

When the distribution database is created, SQL Server adds the following tasks automatically to SQL Server Agent to purge the data no longer needed:

- Subscription cleanup at the Publisher
- History cleanup at the Distributor

These tasks help replication to function effectively in a long-running environment; therefore, administrators should plan for this periodic maintenance. The cleanup tasks delete the initial snapshot for each publication and remove history information in the **Msmerge\_history** table.

#### Merge Meta Data Cleanup

The **sp\_mergecleanupmetadata** system stored procedure allows administrators to clean up meta data in the **MSmerge\_contents** and **MSmerge\_tombstone** system tables. Although these tables can expand infinitely, in some cases it improves merge performance to clean up the meta data. This procedure can be used to save space by reducing the size of these tables at the Publisher and Subscribers.

Before executing this stored procedure, merge all data from Subscribers with the Publisher to load all the Subscriber data changes that must be saved. Snapshot files for all merge publications involved at all levels must be regenerated after executing this stored procedure. If you try to merge without running the snapshot first, you will receive a prompt to run the snapshot.

**CAUTION** After **sp\_mergecleanupmetadata** is executed, by default, all subscriptions at the Subscribers of publications that have meta data stored in the two tables are marked for reinitialization, changes at the Subscriber are lost, and the current snapshot is marked obsolete.

The reinitialization propagates the merge topology automatically. The

administrator does not have to reinitialize all subscriptions at every republisher manually. When using SQL Server 7.0 with Service Pack 2, the reinitialization does not propagate through the merge topology automatically.

By default, the **@reinitialize\_subscriber** parameter of **sp\_mergecleanupmetadata** is set to TRUE, and all subscriptions are marked for reinitialization. If you set the **@reinitialize\_subscriber** parameter to FALSE, the subscriptions are not marked for reinitialization. Setting the parameter to FALSE should be used with caution because if you choose not to have the subscriptions reinitialized, you must make sure that data at the Publisher and Subscribers is synchronized.

If **sp\_mergecleanupmetadata** is executed with the **@reinitialize\_subscriber** parameter set to TRUE, the snapshot will be reapplied at the Subscriber even if the subscription was created without an initial snapshot applied (for example, if the snapshot data and schema were manually applied or already existed at the Subscriber). If you do not want the subscription to be reinitialized and the snapshot reapplied, the subscription must be dropped and re-created as a subscription with no initial synchronization after ensuring that the data is in synchronization between Publisher and Subscriber.

If you want to run **sp\_mergecleanupmetadata** without the subscriptions being marked for reinitialization:

- 1. Synchronize all Subscribers.
- 2. Stop all updates to the publication and subscription databases.
- 3. It is recommended that you execute a merge that validates the Subscriber data with the Publisher by running the Merge Agent with the **-Validate** command line option at each Subscriber.
- 4. Execute the **sp\_mergecleanupmetadata** system stored procedure. After the stored procedure has executed, you can allow users to update the publication and subscription databases again.

Execute **sp\_mergecleanupmetadata** after all merges, including continuous

mode merges, have been completed. One method for controlling this is to deactivate the publication and activate it after the merge cleanup has been completed.

For example, execute code similar to the following at the Publisher:

EXEC central..sp\_changemergepublication 'publicationname', 'status', '

This ensures that all continuous mode merges that are polling for the publication status will fail if the publication has been inactivated. Execute the following after all continuous mode merges have terminated:

```
EXEC central..sp_mergecleanupmetadata 'publicationname',
@reinitialize_subscriber='false'
```

EXEC central..sp\_changemergepublication 'publicationname', 'status', '

If the merge cleanup is propagated to a republisher that is not yet inactive, an error message is returned stating that cleanup of merge meta data could not be performed.

To use this stored procedure, the Publisher and all Subscribers must be running Microsoft SQL Server 7.0 with Service Pack 2 or later. Only members of **sysadmin** and **db\_owner** role can use this stored procedure. To clean up merge meta data, execute the **sp\_mergecleanupmetadata** system stored procedure. If you specify a **@tablename** parameter, only the merge meta data for that table will be cleaned. If no table name is specified, all merge meta data in **MSmerge\_contents** and **MSmerge\_tombstone** will be cleaned.

**IMPORTANT** If there are multiple publications on a database, and any one of those publications uses an infinite publication retention period (@retention=0), running **sp\_mergecleanupmetadata** will not clean up the merge replication change tracking meta data for the database. For this reason, use infinite publication retention with caution.

#### See Also

Planning for Merge Replication

**Replication Options** 

# **Merge Replication Conflict Detection and Resolution**

When Publisher and Subscribers are reconnected and synchronization occurs, the Merge Agent detects conflicts and then determines which data will be accepted and propagated to other sites based on a resolver specified when the merge publication was implemented.

In merge replication, a conflict exists when:

- Changes are made to the same column(s) in the same row (using INSERT, UPDATE or DELETE statements) in more than one copy, with column-level conflict tracking in effect.
- Changes are made to a row in both replicas, and row-level tracking is in effect (the columns affected in the corresponding rows need not be the same).

**Note** Although a Subscriber is merging with the Publisher, a conflict typically occurs between updates made at different Subscribers and not necessarily updates made at a Subscriber and at the Publisher.

#### **Conflict Detection**

The Merge Agent detects conflicts through lineage values in the **MSmerge\_contents** tables for the database of the article. Each entry in **MSmerge\_contents** contains information about a row that has been updated. The **lineage** column in **MSmerge\_contents** represents the history of changes in an updated row; its value is updated automatically by the Merge Agent whenever the row is synchronized.

When the Merge Agent is merging changes, it examines the **lineage** values of the version of the row at each site. The agent compares the **lineage** value for the updated row between **MSmerge\_contents** tables (**MSmerge\_contents** Publisher table, **MSmerge\_contents** Subscriber table) to determine whether the row has been updated in multiple locations. If the row has not been updated in multiple locations, there is no conflict and the updated value is merged. If the row has been updated in multiple locations, a conflict has occurred, and the conflict

resolution process is invoked.

If column-level tracking is enabled, the Merge Agent also needs to compare the **COLV** values in the **MSmerge\_contents** table with the updated rows.

### **Resolving Conflicts**

After a conflict is detected, the Merge Agent launches the selected conflict resolver. The winner of the conflict is chosen according to a user-specified priority scheme, a first wins solution (with the first to synchronize winning the conflict), or a custom resolver consisting of a COM object or stored procedure. Unless the interactive conflict resolver is used, conflicts are resolved immediately after the resolver executes. The losing row is written to a conflict table named **conflict\_<PublicationName>\_<ArticleName>\_usertablename** (the winning row is applied at the Publisher and Subscriber).

### **Conflict Resolvers**

Microsoft<sup>®</sup> SQL Server<sup>™</sup> 2000 allows you to choose how to resolve merge conflicts. Options available include:

- The default priority-based conflict resolver supplied with SQL Server 2000. When using this resolver, you can assign priority values to individual Subscribers (global subscriptions), or use the default priority assignments (local subscriptions), where the Publisher takes ownership of the changes upon data synchronization. These changes then have priority over changes made at other local Subscribers on a first-merge basis.
- A custom resolver, which implements specific data or business-decision rules to resolve the conflict. Custom resolvers can be built either as stored procedures or as COM objects written in languages such as Microsoft Visual C++® or Microsoft Visual Basic®. A set of out-of-the-box custom conflict resolvers and examples of custom conflict resolvers are supplied with SQL Server 2000.
- Other Microsoft Resolvers including additive, averaging, DATETIME,

maximum, merge text, minimum, and Subscriber Always Wins resolvers.

In addition, SQL Server 2000 supplies an Interactive Resolver that you can use in conjunction with either the priority-based resolver or a custom resolver. When performing an on-demand synchronization, the Interactive Resolver displays conflict data at run-time, and lets you choose which data to use to resolve the conflict. You can also use the Conflict Viewer, which has a similar user interface to the Interactive Resolver, to view the results of conflicts that have been resolved. This means that a user must be available to respond to the Interactive Resolver when a merge occurs. This would therefore not be appropriate for an application independent of human interaction.

In merge replication, conflict resolution takes place at the article level (property of an article) for a single row of data at a time. For publications composed of several articles, you can have different conflict resolvers serving different articles, or the same conflict resolver serving one article, several articles, or all the articles comprising a publication.

If you plan to use the default priority-based conflict resolver, you do not have to set the resolver property of an article. If you want to use a custom resolver instead of the default resolver, you must set the resolver property (by selecting an available custom resolver on the Publisher) for the article that will use it. Any specific information that needs to be passed to the custom resolver can also be specified in the resolver information property.

### **Viewing Conflicts**

Replication creates several tables that can be used to review information on conflicts and their resolution. In addition, the Conflict Viewer displays conflicting rows and can be used as a conflict reviewing tool.

SQL Server 2000 creates a conflict table for each table in a merge article. For example, if there is a table named **Customers** that is published as an article named "Customer-Article" in the "Northwind-Customers" publication, the conflict table named **conflict\_Northwind-Customers\_Customers-Article** will be generated.

Conflict tables have the same structure as the tables on which they are based. A

row in one of these tables consists of a losing version of a conflict row (the winning version of the row residing in the actual user table). The **sysmergearticles** table identifies which user tables have conflict tables, and provides information about the conflict tables. SQL Server also provides stored procedures that allow the conflict tables to be queried.

Another conflict table generated during merge replication setup is **MSmerge\_delete\_conflicts**. The table is a log for deleted conflicts. It contains information for deleted rows that conflicted with an update and lost the conflict, or because a delete was undone to achieve data convergence.

Concepts necessary for understanding merge conflict resolution include:

- Row-level tracking versus column-level tracking, which specifies whether the Merge Agent identifies changes to any values in corresponding rows, or changes to the same columns in corresponding rows as a conflict.
- Subscriber type, which describes whether a user assigns a priority value to a Subscriber (global), or whether the Subscriber uses the priority value of the Publisher when the changes are synchronized (local).

# **Row-Level Tracking and Column-Level Tracking**

Several options are available for specifying how the Merge Agent recognizes a conflict. One option is specifying whether conflicts are recognized at the row level or at the column level.

When conflicts are recognized at the row level, changes made to corresponding rows are judged a conflict, whether or not the changes are made to the same column. For example, suppose one change is made to the address column of a Publisher row, and a second change is made to the phone number column (in the same table) of the corresponding Subscriber row. With row-level tracking, a conflict is detected, because changes were made to both rows. With columnlevel tracking, no conflict is detected, because changes were made to different columns in the rows.

Resolution of the conflict is the same, regardless of which tracking option is used; the entire row of data is overwritten by data from the conflict winner. In the earlier example, suppose the phone number is changed at both the Publisher and Subscriber, and the address is changed only in the Subscriber row. If the Publisher wins the conflict, the entire Publisher row overwrites the Subscriber row for both row-level tracking and column-level tracking; thus, the original value for phone number in the Publisher overwrites the changed value in the Subscriber. If you are using column-level tracking, and one user changes the address for a particular row, and another user changes the phone number for the same row, there is no conflict and both changes will be accepted.

The application semantics usually determine which tracking option to use. For example, if you are updating customer data that is generally entered at the same time, such as an address and phone number, row-level tracking should be chosen. If column-level tracking were chosen in this situation, changes to the customer address in one location and to the customer phone number in another location would not be detected as a conflict: the data would be merged on synchronization and the error would be missed. In other situations, updating individual columns from different sites may be the most logical choice. For example, two sites may have access to different types of statistical information on a customer, such as income level and total dollar amount of credit card purchases. Selecting column-level tracking ensures that both sites can enter the statistical data for different columns without generating unnecessary conflicts.

Row-level tracking involves less tracking overhead. Column-level tracking may result in fewer conflicts being detected by the Merge Agent, but can be more resource intensive in terms of the storage needed to track changes. Column-level tracking may generate less network traffic during synchronization because only the changed columns are transferred to the partner database (the publication database or the subscription database).

To set row- or column-level tracking for an article

# **Subscriber Types and Conflicts**

When you create a subscription, you can either assign it a priority value or use the priority value of the Publisher.

A subscription with an assigned priority value is called a global subscription; a subscription using the priority value of the Publisher is called a local subscription. This table summarizes the main differences and uses of each type.

Туре	Priority Value	Used
Global	Assigned by user	When you want different Subscribers to have different priorities.
anonymous)	0.00, but change assumes priority value of Publisher after synchronization	When you want all Subscribers to have the same priority, and the first Subscriber to merge with the Publisher to win the conflict. Anonymous subscriptions are helpful when you expect to have a large number of Subscribers and you do not want to keep track of them at the Publisher/Distributor.

When you change a row in a global subscription, the subscription priority is stored in the meta data for the change. This priority value travels with the changed row as it merges with changes at other Subscribers. This assures that a change made by a higher priority subscription does not lose to a change made by a subscription with a lower priority.

If a row is changed in a local subscription, no priority is assigned to the change until the row merges with the other changes at a Publisher. During the merge process at the Publisher, the changes from the Subscriber are assigned the priority of the Publisher and travel with that priority as it merges with changes at other Publishers and Subscribers. In a sense, the Publisher assumes authorship of the change. Global subscriptions provide a greater number of options and allow for greater sophistication to a conflict resolution scheme than local subscriptions. Using global subscriptions ensures that priority values are preserved throughout the enterprise.

Local subscriptions are also appropriate (and usually required) in a topology with several levels, where Subscribers are leaf nodes. In these topologies, any nodes that republish data must be global Subscribers; local Subscribers can be used only at the leaf nodes.

# **Example of Merge Conflict Resolution Based on Subscriber Type and Assigned Priorities**

To understand how conflicts resolve according to assigned priority values and whether a subscription is global or local, consider the following example, which describes a series of updates to a row over several merge synchronizations.

Here are the initial priority values for four sites in a basic merge replication topology (one Publisher, two global Subscribers, and one local Subscriber).

Site	Туре	Priority Value
А	Publisher	100.00
В	Global Subscriber	75.00 (assigned)
С	Global Subscriber	50.00 (assigned)
D	Local Subscriber	0.00 (default)

#### **Phase 1: Initial Values**

Initially, Site A (the Publisher) creates version one of the row containing value='Nebraska', which is replicated to Sites B, C, and D during the next merge synchronization. After synchronization, here are the values for the row.

Site	Priority Value	Row Value
A (Publisher)	100.00	Nebraska
B (Global Subscriber)	75.00	Nebraska
C (Global Subscriber)	50.00	Nebraska

D (Local Subscriber)	0.00	Nebraska
----------------------	------	----------

#### Phase 2: Publisher and Global Subscriber Both Update Row

Site A updates the row value to Texas and site B updates the row value to New Jersey. When the next merge synchronization occurs, there is a conflict between sites A and B. Site A wins the conflict (the Publisher always wins an update conflict, even if the priority values are the same by default, but there is also the option that Subscriber wins the conflict). The conflict winner value from site A is propagated to sites B, C, and D.

Site	Priority Value	Row Value
A (Publisher)	100.00	Texas
B (Global Subscriber)	75.00	Texas
C (Global Subscriber)	50.00	Texas
D (Local Subscriber)	0.00	Texas

#### Phase 3: Multiple Changes Made to the Same Row

Suppose site C updates the row (changes it to North Carolina) and synchronizes with the Publisher. This is not a conflict because C already successfully merged the last update from A (with the row value='Texas' successfully merged). Then suppose Site B updates the row (changes it to Idaho).

Site	Priority Value	Row Value
A (Publisher)	100.00	North Carolina
B (Global Subscriber)	75.00	Idaho
C (Global Subscriber)	50.00	North Carolina
D (Local Subscriber)	0.00	Texas

When site B synchronizes with the Publisher, there is an update conflict. Because both B and C are global subscriptions and the priority of B is greater than that of C, site B wins the conflict. After the other two sites are also merged, the value of B is propagated to the other Subscribers.

Site	Priority Value	Row Value
A (Publisher)	100.00	Idaho
B (Global Subscriber)	75.00	Idaho
C (Global Subscriber)	50.00	Idaho
D (Local Subscriber)	0.00	Idaho

#### Phase 4: Local and Global Subscribers Both Update Row

Suppose site D updates the row (changes it to New Mexico) and synchronizes with the Publisher. Then suppose Site B updates the row (changes it to California).

Site	Priority Value	Row Value
A (Publisher)	100.00	New Mexico
B (Global Subscriber)	75.00	California
C (Global Subscriber)	50.00	Idaho
D (Local Subscriber)	0.00	New Mexico

When site B synchronizes with the Publisher, there is an update conflict. Unlike the previous example, because D is a local Subscriber, it assumes the priority value of the Publisher (site A) upon synchronization. Because the priority of A is greater than B, B loses the conflict; the value initially entered into D wins. (Had the global Subscriber B synchronized with A before the local Subscriber D did, site B would have won the conflict.) Site D winning the conflict relies on the Publisher not having made a change or received another change since the version of the row updated at Site D was last synchronized. If any global Subscriber or any other local Subscriber synchronizes first, the rule of highest priority or first in to the Publisher wins is followed.)

The final values after all the sites are synchronized are shown here.

Site	Priority Value	Row Value

A (Publisher)	100.00	New Mexico
B (Global Subscriber)	75.00	New Mexico
C (Global Subscriber)	50.00	New Mexico
D (Local Subscriber)	0.00	New Mexico

Synchronization order and priority value determine the outcome of conflicts when mixing global and local Subscribers at the same level in your topology. This last set of updates illustrates why caution must be exercised. Although the local Subscriber had the lowest priority value of the three Subscribers, it won the conflict because it synchronized with the Publisher (thus assuming the Publisher priority value of 100.00) first. Had site C (global Subscriber with a priority value of 50.00) entered New Mexico instead of site D, site B (global Subscriber with a priority value of 75.00) would have won the conflict, and the result would have been California.

## **Default Resolver and Custom Resolvers**

When you create a merge publication, the conflict resolver is set to the default resolver for all articles in the publication (if you do not plan on using a custom resolver with an article, you do not need to choose a resolver). For each article, you can use the default merge resolver or select an available custom resolver. After an article in a publication is assigned a resolver, that association must be maintained across all publications (for when the same table is in multiple publications). You cannot assign different resolvers to the same article across different publications.

### **Default Resolver**

When you create a push or pull subscription, you specify the behavior of the default resolver by choosing to make the subscription global or local.

By default, SQL Server defines a subscription as local, with a priority value of 0.00. On the Set Subscription Priority page in the Push Subscription Wizard or Pull Subscription Wizard, this selection corresponds to the option for using the priority value of the Publisher when a conflict occurs. If this is retained for all Subscribers, the result is that the Publisher updates win the conflict and between Subscribers that have conflicts, the first Subscriber to synchronize, wins the conflict. For the default merge resolver, the Publisher always wins a conflict; however, a custom resolver can override this rule.

You can also assign a specific priority value to a subscription. On the Set Subscription Priority page in either the Push Subscription Wizard or the Pull Subscription Wizard, this selection corresponds to the option for assigning a specific priority value from 0.00 through 99.99 to the Subscriber. When you make this selection and specify a priority greater than 0.0, you define a global subscription.

If both global and local Subscribers are connected to the Publisher, and changes from a global Subscriber with a priority value greater than 0.00 are synchronized first, subsequent conflicting changes from local Subscribers are rejected. The priority value for the global Subscriber is greater than the priority value of any local Subscriber (which would be 0 prior to merge synchronization). If a local Subscriber synchronizes with the Publisher first, subsequent conflicting changes from global Subscribers or other local Subscribers will be rejected. The priority value for the local Subscriber that was first synchronized with the Publisher assumes the priority value of the Publisher, which always wins a conflict; however, a custom merge resolver can override these rules.

### **Custom Resolvers**

Merge replication allows you to use a variety of custom resolvers to deal with conflict situations. Custom resolvers are always executed where the Merge Agent runs. You can select from a number of out-of-the-box custom resolvers supplied with SQL Server, write a custom stored procedure resolver, or write a COM object resolver in a language such as Microsoft Visual C++® or Microsoft Visual Basic®. If you plan to use a COM object resolver, make sure the DLL is registered at the computer where the Merge Agent runs. For a push subscription, this is the Distributor, and for a pull subscription, it is the Subscriber. For applications that use the Merge ActiveX® Control, the resolver should be registered at the computer where the application executes.

Custom resolvers can be loaded in one of the following ways:

- By selecting the custom resolver you want in the Create Publication Wizard. This option is found on the **Resolver** tab, in the **Properties** dialog box.
- If you are using stored procedures to set up and configure merge replication, the *@resolver\_info* parameter of the **sp\_addmergearticle** system stored procedure contains the name of the custom conflict resolver to use with the article.

#### To choose a resolver

## **COM Custom Resolvers**

A COM custom conflict resolver is a dynamic-link library (DLL) that implements the **ICustomResolver** interface, its methods and properties, and other supporting interfaces and type definitions designed specially for conflict resolution.

These interfaces and type definitions are defined in Microsoft® Visual C++® header files (Sqlres.h and Sqlresid.h), supplied with the merge conflict resolver samples in \Microsoft SQL Server\Tools\DevTools\Samples\Sqlrepl (available through a custom installation of Microsoft SQL Server<sup>TM</sup> 2000). If you are a Visual C++ developer, you can view these samples to get an idea of how to build a custom COM resolver in Visual C++.

Because the conflict resolver interfaces are COM-based and therefore languageneutral, it is possible to create custom COM resolvers in other languages than C++. To build a custom COM resolver in Visual Basic®, you can use the type library that is provided in the replrec.dll.

Before writing a custom COM resolver, you need to decide:

- The types of row changes you want to resolve, such as updates, inserts, and deletes, and for the upload of merge changes, the download, or both. You can specify one type of change, all changes, or any combination. The default merge conflict resolver handles any conflicts not covered by a custom resolver.
- Whether to use column tracking when resolving the conflict. No column tracking means that changes are tracked at the row level. Changes to any columns in both rows are flagged as a conflict. To resolve the conflict, the priority winner overwrites the entire row of data.

When column-level tracking is on, only data in those columns where a conflict exists are flagged as a conflict, otherwise the data is merged. However, conflicts are resolved in the same way as row-level tracking: the priority winner overwrites the entire row of data (but the data can be a mix of values from the Publisher, Subscribers, or some altered values that were from neither Publisher nor Subscribers).

When using a custom COM resolver with merge replication on a cluster, you must register the custom resolver on both nodes of the cluster regardless of whether the configuration is active-active or active-passive. This is required to ensure that the custom resolver will be able to properly load the reconcile following a failover.

# **Specifying a Custom Resolver**

A custom resolver can be specified from the Create Publication Wizard or with replication stored procedures.

### **Using the Create Publication Wizard**

When a merge replication is created with the Create Publication Wizard, custom resolvers can be specified on the Specify Articles page. When a table is selected for publication, a properties (...) button is presented, which when clicked, displays the **Properties** dialog box for the article (table).

On the **General** tab, select whether changes to the same row or to the same column are regarded as conflicts. When changes to the same column are conflicts, changes to different columns in the same row are merged.

On the **Resolver** tab, select whether to use the default resolver, or a custom resolver, and then select one in the list. If the resolver references a specific column, enter its name in the **Information for the custom resolver** dialog box.

### **Using Replication Stored Procedures**

When a merge replication is created with replication stored procedures, custom resolvers are specified from **sp\_addmergearticle** or **sp\_changemergearticle**.

In **sp\_addmergearticle**, if a custom resolver is to be used, the resolver name from the table in the Microsoft Resolver Descriptions topic is entered with the *@article\_resolver* parameter. The name must be typed exactly as it appears in the table. If a column name is required, it is entered with the *@resolver\_info* parameter.

This example specifies that the Microsoft SQL Server Averaging Conflict Resolver be used with article **ProductsArticle** in publication **ProductsPublication** for source table **Products** to calculate the average of the **UnitPrice** column when conflicts occur.

exec @ret = sp\_addmergearticle @publication='ProductsCatalog',

@article='ProductsArticle', @source\_object='Products', @article\_resolver='Averaging Conflict Resolver', @resolver\_info='UnitPrice'

**sp\_changemergearticle** is used to change one property of an existing merge article. The *@***property** parameter specifies the property to be changed, the *@***value** parameter specifies the new value for the property.

This example changes the article **ProductsArticle** in publication **ProductsPublication** to use the Microsoft SQL Server Additive Conflict Resolver to calculate the sum of the **UnitsOnOrder** column when conflicts occur.

```
exec @ret = sp_changemergearticle @publication='ProductsCatalog',
    @article='ProductsArticle', @property='article_resolver',
    @value='Additive Conflict Resolver'
exec @ret = sp_changemergearticle
```

@publication='ProductsCatalog', @article='ProductsArticle', @property='resolver\_info', @value='UnitsOnOrder'

### See Also

Microsoft Resolver Descriptions

# **Interactive Resolver**

Microsoft SQL Server replication provides an interactive resolver, which allows you to resolve conflicts manually during on-demand synchronization. Activated at run-time, the Interactive Resolver displays data for each conflicting row, and provides options for viewing and editing the conflict data, and resolving each conflict individually.

#### **Interactive Resolver and the Conflict Viewer**

The Interactive Resolver resembles the Conflict Viewer. However, the Conflict Viewer displays the results of conflicts that are already resolved after merge synchronization, and the Interactive Resolver displays each conflict prior to resolution, allowing you to determine the outcome of each conflict during merge synchronization. Someone must be available to monitor the Interactive Resolver when a conflict occurs.

### Article Resolvers and the Interactive Resolver

Conflict resolvers (either the Microsoft® SQL Server<sup>™</sup> 2000 default resolver or a custom resolver) are assigned to specific articles when a publication is created, and use a set of predetermined rules to determine which set of data should be used when conflicting row data is entered.

The Interactive Resolver is not a separate conflict resolver with rules for determining conflict winners and losers, but a tool used in conjunction with the default and custom merge resolvers. The article resolver still determines the winning and losing row, but the Interactive Resolver allows user intervention to accept, reject, or modify the results. Use the Interactive Resolver to review individual conflicts occurring during synchronization, edit the conflict data, or make individual determinations of conflict winners and losers. In other words, an Interactive Resolver can be used in conjunction with the default or custom resolvers.

The option to allow the Interactive Resolver to be used on a subscription is enabled as a publication property. However, invoking the Interactive Resolver when a conflict is detected requires setting this option as a subscription property (in the Create Pull Subscription Wizard, using replication stored procedures, or using ActiveX® controls). After these properties are set for both the publication and subscription, the Interactive Resolver is used when a conflict is detected during merge synchronization.

**Note** Because user intervention is required, the Interactive Resolver should be used only during an on-demand synchronization, never during a scheduled synchronization.

To enable activation of the Interactive Resolver

### **Custom Stored Procedure Conflict Resolver**

You can create custom resolvers consisting of special queries and code to examine conflicts and override the default way in which conflicts are resolved by Microsoft® SQL Server<sup>™</sup> 2000. You can override the default conflict resolver by substituting your own program with the same name. For example, suppose multiple sites participate in monitoring a chemical process and each records the low and high temperatures achieved in a test. Rather than use a priority or first wins solution, such an application might want to accept the lowest low and the highest high value.

You can use Transact-SQL to build your custom conflict resolver as a stored procedure at each Publisher. Custom conflict resolvers are always executed at the Publisher. The stored procedure should accept the following required parameters.

Parameter	Data Type	Description
@tableowner	Sysname	Name of the owner of the table for
		which a conflict is being resolved -
		this is the owner for the table in the
		publication database.
@tablename	sysname	Name of the table for which a
		conflict is being resolved.
@rowguid	uniqueidentifier	Unique identifier for the row having
		the conflict.
@subscriber	sysname	Name of the server from where a
		conflicting change is being
		propagated.
@subscriber_db	sysname	Name of the database from where
		conflicting change is being
		propagated.
@log_conflict	int	Whether the merge process should
OUTPUT		log a conflict for later resolution:
		0 = Do not log the conflict

	<ul><li>1 = Subscriber is the conflict loser</li><li>2 = Publisher is the conflict loser</li></ul>
@conflict_message nvarchar(512) OUTPUT	Message to be given about the resolution if the conflict is logged.

The stored procedure uses these parameters to examine the values contained in the row at both the Publisher and Subscriber. The stored procedure can also examine any additional information you specify and manipulate the values to determine what column values the resolved row should have. The stored procedure then returns a single row result set that is identical in structure to the base table and contains the data values for the winning version of the row. The stored procedure can potentially use distributed queries or other mechanisms to query the value from the remote database.

The stored procedure must be located either in the published database at the Publisher or in the **master** database and marked as a system object. Execute permission should be granted to **public** or to a list of all Subscribers.

**Note** SQL Server stored procedure resolvers will be invoked only to handle update conflicts. They cannot be used to handle other types of conflicts such as insert failures due to PRIMARY KEY violations or unique index constraint violations.

After the stored procedure is created, you must configure an article to use that stored procedure as its custom resolver. You can specify a custom resolver for an article by executing **sp\_addmergearticle** to associate the stored procedure with the article. You must set the **@article\_resolver** parameter to **Microsoft SQLServer Stored Procedure** and set the **@resolver\_info** parameter to the name of stored procedure.

For more information, see <u>Developing Replication Merge Conflict Resolvers</u> <u>Through a Custom Resolver</u>.

# **Other Microsoft Resolvers**

When conflicts occur during the merge process, a conflict resolver must determine how the conflict is resolved. Microsoft® SQL Server<sup>TM</sup> 2000 includes several custom COM-component resolvers that can be used for this purpose, in addition to the default priority-based resolver and the stored procedure resolver:

- Microsoft SQL Server Additive Conflict Resolver
- Microsoft SQL Server Averaging Conflict Resolver
- Microsoft SQL Server DATETIME (Earlier Wins) Conflict Resolver
- Microsoft SQL Server DATETIME (Later Wins) Conflict Resolver
- Microsoft SQL Server Maximum Conflict Resolver
- Microsoft SQL Server Merge Text Conflict Resolver
- Microsoft SQL Server Minimum Conflict Resolver
- Microsoft SQL Server Subscriber Always Wins Conflict Resolver

The resolvers are installed during the installation process for SQL Server 2000. The **sp\_enumcustomresolvers** stored procedure can be used to view all the conflict resolvers registered on that computer. In SQL Query Analyzer, run:

exec sp\_enumcustomresolvers

This displays the description and globally unique identifier (GUID) for each resolver in a separate result set.

The resolver must be registered on the computer from which the Merge Agent is invoked. For push subscriptions, the resolver is registered at the Distributor. For pull subscriptions, the resolver should be registered at the Subscriber.

# **Microsoft Resolver Descriptions**

All of the resolvers in Microsoft® SQL Server<sup>™</sup> 2000 handle update conflicts, and where indicated, they also handle insert and delete conflicts. They all handle column tracking; most also handle row tracking. These and all other custom conflict resolvers declare the types of conflict they can handle, and the merge replication agent uses the default resolver for all other conflict types.

The following table describes the attributes of the specific resolvers. For information about how to specify the required input, see <u>Specifying a Custom</u> <u>Resolver</u>.

Name	<b>Required Input</b>	Description	Comments
Microsoft SQL	Name of the	Conflict winner	Supports
Server	column to be	determined from priority	update
Additive	summed. It must	value. Specified column	conflicts,
Conflict	have an arithmetic	values set to sum of source	column
Resolver		and destination column	tracking
	int, smallint,	values. If one is set to	only.
	numeric, and do	NULL, they are set to the	
	on.).	value of the other column.	
Microsoft SQL	Name of the	Conflict winner	Supports
Server	column to be	determined from priority	update
Averaging	averaged. It must	value. Resulting column	conflicts,
Conflict	have an arithmetic	values set to average of	column
Resolver	data type (such as	source and destination	tracking
	int, smallint,	column values. If one is	only.
	numeric, and so	set to NULL, they are set	
	on.).	to the value of the other	
		column.	
Microsoft SQL	Name of the	Column with the earlier	Supports
Server	column to be used	datetime value determines	update
DATETIME	to determine the	the conflict winner. If one	conflicts,
(Earlier Wins)	conflict winner. It	is set to NULL, the row	row, and
Conflict	must have a	containing the other is the	column

Resolver	DATETIME data type.		tracking. The column values are compared directly and an adjustment is not made for different time zones.
Microsoft SQL	Name of the	Column with the later	Supports
Server		datetime value determines	
DATETIME	to determine the	the conflict winner. If one	conflicts,
(Later Wins)	conflict winner. It	is set to NULL, the row	row, and
Conflict	must have	containing the other is the	column
Resolver	DATETIME data	winner.	tracking.
	type.		
Microsoft SQL	Name of the	Column with the larger	Supports
Server	column to be used	numeric value determines	row and
Maximum	to determine the	the conflict winner. If one	column
Conflict	conflict winner. It	is set to NULL, the row	tracking.
Resolver	must have an	containing the other is the	
	arithmetic data	winner.	
	type (such as <b>int</b> ,		
	smallint, numeric,		
	and so on.).		
-	No inputs. Text	Conflict winner	Supports
Server Merge	columns in	determined from priority	update
Text Conflict	conflict are	value. Text columns in	conflicts,
Resolver	merged.	conflict are set to merged	column
		value consisting of	tracking
		common prefix followed	only.
		by unique part of source,	
		newline character	
		(linefeed), and then unique	
		part of destination.	

Microsoft SQL Server Minimum Conflict Resolver	column to be used to determine the	Column with the smaller numeric value determines the conflict winner. If one is set to NULL, the row containing the other is the winner.	Supports update conflicts, row, and column tracking.
Microsoft SQL Server Subscriber Always Wins Conflict Resolver	No inputs. No data type restrictions.	Subscriber, regardless of whether it is the source or destination, is the winner.	Supports all conflict types.

# **Choosing a Resolver**

When choosing a resolver, you need to consider the importance of conflict resolution in your replication application and whether you will need to build a custom resolver.

If your data is partitioned without multiple users writing to the same partitions, and your replication topology is relatively basic (one Publisher and a few Subscribers), conflicts should be rare or nonexistent. In these environments, you may not need a complex conflict resolution strategy. A strategy using the default settings for conflict resolution, using local Subscribers and a first change in wins policy, is recommended.

Another factor is determining whether to build a custom resolver or use the default merge conflict resolver. Using a custom resolver is the recommended option if your business needs require a more finely tuned solution than is available with the default resolver, and the table associated with the custom resolver is relatively stable, or updating the custom resolver is not an issue.

Choosing whether to use the default resolver, or a custom resolver and the logic used in a custom resolver, should be based on the data. For example, suppose the employees entering customer-ranking data into a set of nonpartitioned replicas span various job categories (branch managers, line managers, sales staff), and job category determines whose data should be given priority. In this case, a custom resolver can be built that uses job category data from the article to determine the priority winner when a conflict occurs.

Custom resolvers are usually specific to a particular table; if the table used in the article is modified (for example, renaming the column name that is used in conflict resolution), the custom resolver may have to be modified and recompiled.

If conflicts are likely to occur with some frequency, here are the most important decisions you will need to consider when implementing a conflict resolution strategy.

Conflict Resolution Issue	Recommendation

Different categories of users require different priority values.	<ul> <li>Use the default merge resolver and create global Subscribers with different priority values. Or</li> <li>Use a custom resolver that recognizes an authority value column in the article to help resolve a conflict.</li> </ul>
First change in wins conflict solution wanted.	Use the default merge resolver and create local Subscribers.
Multiple users changing the same data row acceptable, as long as no conflicting changes made to the same column.	Use either the default merge resolver or a custom resolver with column-level tracking enabled.
Flag multiple changes to any value in a row as a conflict.	Use either the default merge resolver or a custom resolver with row-level tracking.
Conflict outcome data needs to be different from original conflict data.	Use a custom resolver that calculates new values. Alternatively, optionally use the stored procedure resolver and write a custom procedure that returns a result set that contains the new data.

# **Replication Tools**

Microsoft® SQL Server<sup>™</sup> 2000 provides several methods for implementing and administering replication, including SQL Server Enterprise Manager, programming interfaces, and other Microsoft Windows® components.

SQL Server Enterprise Manager includes a graphical organization of replication objects, several wizards, and dialog boxes you can use to simplify the configuration and administration of replication. SQL Server Enterprise Manager allows you to view and modify the properties of replication configuration, and monitor and troubleshoot replication activity.

You can also implement, monitor, and maintain replication using programming interfaces such as Microsoft ActiveX® controls for replication, SQL-DMO, and scripting of Transact-SQL system stored procedures.

Components such as Windows Synchronization Manager and Active Directory<sup>™</sup> Services enable you to synchronize data, subscribe to publications, and organize and access replication objects from within Windows applications.

# **Replication and SQL Server Enterprise Manager**

You can use SQL Server Enterprise Manager to implement, administer, and monitor a complete replication environment across your enterprise.

SQL Server Enterprise Manager provides the Replication folder as a central location to organize and administer your publications and subscriptions. If you have heterogeneous publishing services from Microsoft or other companies installed, the Heterogeneous Replication folder will appear under the Replication folder as a location to manage publications and subscriptions based on heterogeneous data sources.

## **Replication Monitor**

Through Replication Monitor, you can view and manage replication agents responsible for various replication tasks. Replication Monitor appears as a node below the Replication folder in SQL Server Enterprise Manager on the Distributor after you have configured publishing and distribution.

For example, using Replication Monitor, you can set up replication so that the Publisher log is read continuously, transactions are distributed to Subscribers every ten minutes, and initial snapshots are generated every night at midnight. You can also execute replication agents on demand.

Replication Monitor provides a way to set alerts on replication events. When the event occurs, Replication Monitor responds automatically, either by executing a task that you have defined or by sending an e-mail or a pager message to a specified individual.

Events in the task history can also be written to the Microsoft Windows NT® 4.0 or Windows 2000 application log if the task is set to use Windows NT logging, and can be viewed by using Event Viewer. For information about using Event Viewer, see Windows NT 4.0 or Windows 2000 Help.

SQL Server Agent is an internal SQL Server 2000 tool that hosts and schedules the agents used in replication, and provides an easy way to run replication agents. SQL Server Agent also controls and monitors several other operations outside of replication including monitoring the SQLServerAgent service, maintaining error logs, running jobs, and starting other processes.

Another tool accessible through SQL Server Enterprise Manager is the replication Conflict Viewer. The Conflict Viewer helps you view and resolve conflicts that occurred during the merge replication or queued updating process.

#### See Also

Administering and Monitoring Replication Merge Replication Conflict Detection and Resolution Queued Updating Conflict Detection and Resolution

# **Replication Wizards**

Microsoft® SQL Server<sup>™</sup> includes replication wizards to simplify configuring and implementing replication. The replication wizards can be accessed in SQL Server Enterprise Manager. On the **Tools** menu, point to **Replication**, and then click the appropriate wizard.

### **Configure Publishing and Distribution Wizard**

Through the Configure Publishing and Distribution Wizard, you can:

- Specify the server that you want to configure as the Distributor.
- Configure SQLServerAgent service to start manually or automatically when the computer is started.
- Customize the distribution database properties, enable Publishers, enable Subscribers, and set publishing settings.

### **Create Publication Wizard**

Using the Create Publication Wizard, you can specify:

- The existing publication to be used as a template for the new publication.
- The type of publication to create (snapshot, transactional, or merge).
- The data and database objects (articles) to include in the publication.
- A name and description for the publication.
- Horizontal and vertical data filters, and for merge publications, dynamic

and join filters.

- Whether to allow anonymous Subscribers.
- The Snapshot Agent schedule and whether you want the Snapshot Agent to run immediately.

If you select the **Show advanced options in this wizard** check box on the Welcome page of the wizard, and you create a snapshot or transactional publication, you can specify the following:

- Enabling updatable subscriptions including immediate updating and/or queued updating.
- Enabling transforming published data so data can be transformed before it is distributed to Subscribers.

## **Create Pull Subscription Wizard**

The Create Pull Subscription Wizard allows you to initiate a subscription at a Subscriber and request data to be replicated from a Publisher. Through the Create Pull Subscription Wizard, you can:

- Select the Publisher and publication to which you want to subscribe.
- Select the Subscriber (destination) database that will receive the published data.
- Specify initialization of the subscription so that a snapshot of schema and data is applied at the Subscriber.
- Specify the location of the snapshot files and how to access them at the time the subscription is initialized.

- Set agent schedules for how frequently updates are propagated to the Subscriber.
- Specify whether to transform the data before it is distributed (for snapshot or transactional publications that allow transforming published data).
- Specify if you want required services to start automatically after the subscription is created or if you want to start required services manually.

### **Create Push Subscription Wizard**

The Create Push Subscription Wizard allows you to specify at the Publisher what data you want replicated to specified Subscribers. Through the Create Push Subscription Wizard, you can:

- Select one or more Subscribers or groups of Subscribers to receive published data.
- Specify the database on the Subscriber where data will be published.
- Specify where you want the Distribution Agent to run (for snapshot replication or transactional replication).
- Set agent schedules for how frequently updates are propagated to the Subscriber.
- Specify initialization of the subscription so that a snapshot of schema and data is applied at the Subscriber.
- Specify whether to transform the data before it is distributed (for snapshot or transactional publications that allow transforming published

data).

- Set the priority value of the subscription to determine the winner if conflicts are detected (for a merge publication).
- Specify whether you want required services to start automatically after the subscription is created or if you want to start required services manually.

# **Define Transformation of Published Data**

The Define Transformation of Published Data Wizard is available after you have configured a publication to allow transformation of published data. This wizard allows you to create a Data Transformation Services (DTS) package that defines data transformations. You can specify:

- The Subscriber that will use the package and what authentication that Subscriber uses.
- Column mappings and data transformations that occur as the data is published including Microsoft ActiveX® or Java scripts.
- The location of the DTS package at the server where the Distribution Agent runs.
- The name, description, and security for the package.

**Note** DTS packages created in the Define Transformation of Published Data Wizard cannot be used outside of replication. However, DTS packages created independently of replication using DTS tools can be used to transform published data during replication.

## **Create Dynamic Snapshot Job Wizard**

The Create Dynamic Snapshot Job Wizard guides you through creating a

dynamic snapshot for dynamically filtered merge publications. In this wizard you can:

- Specify the filter criteria page, including any system or user-defined functions used in the dynamic filters of the publication and the value of the login for the Publisher.
- Specify the snapshot file location where you want snapshot files saved.
- Set the dynamic Snapshot Agent schedule.
- Specify the agent name for this dynamic Snapshot Agent.

**Note** You must generate a regular snapshot to the dynamically filtered merge publication before creating a dynamic snapshot.

# **Disable Publishing and Distribution Wizard**

The Disable Publishing and Distribution Wizard allows you to disable publishing, distribution, or both on a server. You can also:

- Specify whether to disable publishing on the server where the wizard is run.
- Confirm the publications that will be dropped.

### See Also

Configuring Replication

**Disabling Publishing and Distribution** 

Dynamic Snapshots

Publishing Data and Database Objects

Subscribing to Publications

**Transforming Published Data** 

# **Replication Properties**

After you configure replication, you can view and modify options by using the properties dialog boxes for replication. Properties are available for the Publisher, its Subscribers, and the Distributor, publications, push subscriptions, pull subscriptions, and replication agents.

## **Publisher and Distributor Properties**

After you have configured a Publisher and Distributor using the Configure Publishing and Distribution Wizard, you can view and modify those options using the Publisher and Distributor properties. The Publisher and Distributor properties include the following tabs and information.

Tab	Information
Distributor	The distributor name, distribution databases,
	properties for the distribution database, buttons to
	create or delete a distribution database, a button to
	see the agent profiles for all replication agents, and
	the administrative link password for Publishers to
	connect to the Distributor.
Publishers	A list of Publishers that have been enabled to use
	this Distributor during replication, and buttons to
	enable, disable, or specify new Publishers.
Publication	A list of databases that are enabled for transactional
Databases	replication (includes snapshot replication) and/or
	merge replication, and buttons to enable or disable
	the databases for transactional replication and/or
	merge replication.
Subscribers	A list of Subscribers configured to receive data
	from this Distributor, and buttons to enable, disable,
	or specify new Subscribers.

#### To open Publisher and Distributor properties

# **Replication Icons**

SQL Server Enterprise Manager uses several icons to represent replication objects, operations, and results.

Description
Publisher
Publisher error
Publisher retrying synchronization
Snapshot publication
Snapshot publication error
Snapshot publication retry
Transactional publication
Transactional publication error
Transactional publication retry
Merge publication
Merge publication error
Merge publication retry
Subscription
Subscription error
Subscription retrying synchronization
Subscription to a merge publication
Subscription (agent not running)
Subscription (agent running)
Database is enabled for publishing
Replication Monitor

	Replication Monitor error
5	Replication Monitor retry
<b>Ø</b> _	Snapshot Agent running
۰.	Snapshot Agent not running
٠	Snapshot Agent retrying
<b>i</b>	Snapshot Agent error
<b>P</b>	Log Reader Agent running
	Log Reader Agent not running
20	Log Reader Agent retrying
₽_	Log Reader Agent error
Þ	Queue Reader Agent running
<b>i</b> •	Queue Reader Agent not running
10	Queue Reader Agent retrying
	Queue Reader Agent error
0	Miscellaneous agents running
Image: Second secon	Miscellaneous agents error
<b>.</b>	Miscellaneous agents retrying
8	Column is a primary key
K	No primary key in the table
•	This table includes a timestamp column and cannot be published by Publishers running SQL Server 7.0 or to Subscribers running SQL Server 7.0.

# **Replication Programming Interfaces**

As an alternative to using SQL Server Enterprise Manager, you can use the following programming interfaces to implement, administer, and monitor replication:

- Microsoft® ActiveX® controls used within custom applications using Microsoft Visual Basic® or Microsoft Visual C++®, provide programmable controls to administer and control the Snapshot Agent, the Distribution Agent, and the Merge Agent. These controls can be used to program activity needed to operate replication. For example, for an application that provides online and offline capabilities, you may want to display a Synchronize button. That button can be associated with the merge ActiveX control, and whenever users click the button, they connect to the Publisher and the Merge Agent for the specified publication merges and synchronizes data.
- SQL-DMO allows you to create custom applications, using Visual Basic or C++, which allow you to configure, implement, or maintain your replication topology. SQL-DMO can be used to program replication administration such as configuring distribution, creating subscriptions, and so on.
- The Replication Distributor Interface provides the capability to replicate data from heterogeneous data sources such as Microsoft Access or Oracle. The Replication Distributor Interface is primarily used by independent service vendors, or others who need to develop a custom replication application based on proprietary data sources.
- Scripting replication using Transact-SQL system stored procedures enables you to automate some replication tasks, configure replication, and implement subscriptions on multiple servers. Stored procedures are frequently used in scripts that can be run when configuring replication on multiple servers (for example, creating subscriptions to a publication

on multiple Subscribers).

# **Programming Replication with ActiveX Controls**

Microsoft® ActiveX® controls allow custom applications to invoke replication agent functionality. The controls support all types of subscriptions and can be monitored using SQL Server Enterprise Manager at the Distributor.

Programmers can use ActiveX controls for replication, similar to any standard built-in control. The controls provided are the SQL Snapshot control, the SQL Distribution control, and the SQL Merge control.

The following list describes the benefits of using ActiveX controls for replication:

- Replication can be part of your application intrinsically. For example, you can place a **Synchronize Now** command on a menu that controls when a specified agent associated with the type of replication you are using runs.
- An application can use a progress bar to provide feedback on the progress of the replication control.
- An application can determine how to obtain login information (for example, hard-coded or interactive).
- Replication controls can be embedded in applications, providing a way to distribute mobile applications without the complexity of Subscriber setup.
- Controls can be programmed to add or drop subscriptions and create or attach databases at the Subscriber.
- An application can be programmed to register the synchronization of a subscription in Windows Synchronization Manager.

- The client has no dependency on SQL Server Agent, which is responsible for executing jobs in addition to replication.
- If you start a replication agent using SQL Server Agent, other jobs can also run. If you are replicating to heterogeneous Subscribers using pull or anonymous subscriptions, SQL Server Agent is not available at the Subscriber.
- ActiveX replication controls can be invoked from many programming environments, including Microsoft Visual Basic®, Visual Basic Scripting Edition, Java, and Microsoft Visual C++®.

If a subscription is registered in Windows Synchronization Manager, there is often no need to embed the controls in the application. All synchronization can then be controlled by this central application, if that meets the needs of your application.

For more information, see <u>Developing Replication Applications Using ActiveX</u> <u>Controls</u>.

# **Programming Replication with SQL-DMO**

SQL Distributed Management Objects (SQL-DMO) allows you to control replication components for implementation, administration, and monitoring. SQL-DMO encapsulates Microsoft® SQL Server<sup>™</sup> 2000 components as objects. Using programming languages, such as Microsoft Visual C++® or Microsoft Visual Basic®, you can write SQL-DMO applications based on these objects and the properties and methods associated with the objects.

For example, a replication component can be a Distributor, and using SQL-DMO, you can program the SQL-DMO Distributor Object to install a local distributor or configure remote distribution for a Publisher. You can then use the **DistributionDatabase** Object to create a new distribution database or change the properties of a distribution database.

After distribution is configured, you can use the **DistributorAvailable** property to find out the state of a Distributor or the **Distribution Database** property to identify the distribution database used at the Distributor.

For more information, see <u>Developing SQL-DMO Applications</u>.

# **Programming Replication with the Replication Distributor Interface**

The Replication Distributor Interface is an OLE DB service provider that allows heterogeneous data sources to publish data to Microsoft® SQL Server<sup>™</sup> Subscribers using snapshot replication or transactional replication. Often used as component in third-party tools, the Replication Distributor Interface allows heterogeneous Publishers to inherit the features of SQL Server replication such as heterogeneous Subscribers, anonymous subscriptions, monitoring and troubleshooting tools in SQL Server Enterprise Manager, alerts and notifications, and others.

You can program C++ applications to use the Replication Distributor Interface and store transactions published from databases other than SQL Server. You can program the Distribution Agent and forward those transactions to Subscribers.

### See Also

**Replication and Heterogeneous Data Sources** 

Programming Replication from Heterogeneous Data Sources

# **Transact-SQL System Stored Procedures**

Replication system stored procedures and replication agent executable files are documented and available as a method for implementing replication in special circumstances or for use in batch files and scripts. In most cases, however, you are better served by using the programming interfaces SQL-DMO and replication Microsoft® ActiveX® controls for programming replication rather than writing direct calls to the system stored procedures.

An advantage to using scripts based on system stored procedures is that you can implement replication, create publications and subscriptions on a server, generate the script automatically through SQL Server Enterprise Manager, and then use that script at other servers to implement replication components. Executing a script can be faster and more efficient than manually performing the same steps repeatedly using SQL Server Enterprise Manager.

For more information, see <u>Scripting Replication</u>.

# Windows Synchronization Manager

Windows Synchronization Manager is a utility available with Microsoft® Windows® 2000 and anywhere Microsoft Internet Explorer version 5.0 is installed. It allows you to synchronize or distribute data between instances of Microsoft SQL Server<sup>TM</sup> 2000 when using snapshot replication, transactional replication, or merge replication.

Windows Synchronization Manager is also a central location for synchronizing other applications including e-mail and offline Web pages. You can use Windows Synchronization Manager to schedule synchronizations or instruct Windows to synchronize selected items automatically when you log on or log off a computer, or when you undock a portable computer.

Windows Synchronization Manager allows you to:

- Choose an alternate synchronization partner.
- Add a new subscription.
- Remove a subscription from Windows Synchronization Manager only.
- Delete a subscription.
- Reinitialize a subscription.
- Reinitialize a subscription preceded by an upload of changes.
- Change the update mode of an updatable subscription.
- Attach a subscription database.

You can also use SQL Server Enterprise Manager to enable pull subscriptions for

use in Windows Synchronization Manager, or you can programmatically enable subscriptions for use in Windows Synchronization Manager by using replication Microsoft ActiveX® Controls, SQL-DMO, or Transact-SQL system stored procedures.

For more information about Windows Synchronization Manager, see the Windows 2000 documentation.

#### Example

#### **To open Windows Synchronization Manager**

# **Active Directory Services**

Replication publications can be accessed using Active Directory<sup>™</sup> Services on the Microsoft<sup>®</sup> Windows<sup>®</sup> 2000 operating system. Through Active Directory, you can view replication objects, such as a publication, and, if allowed, subscribe to that publication.

Typically, if a user wants to subscribe to a publication, they must know the name of the instance of Microsoft SQL Server<sup>™</sup> and the database where the publication is published. Having publication information available in Active Directory allows users to browse based on publication properties and, if allowed, to subscribe to publications using pull subscriptions. Users do not need to know the server name and database where the publication is located.

Active Directory is a central component of the Windows 2000 operating system and provides a place to store information about network-based entities, such as applications, files, printers, and people.

The properties listed in the Active Directory may not always be exactly the same as they are in SQL Server. If there is discrepancy between the publication and attributes in the Active Directory and publication properties in SQL Server, the publication properties in SQL Server are the correct settings to use. This also applies to database and server listings in the Active Directory.

## Adding or Removing a Server Object to the Active Directory

Adding an instance of SQL Server object to the Active Directory requires a login with local administrator privileges on the server. If the login used to register the instance of SQL Server does not have sufficient permissions, and the server is a local server, the **Connect to SQL Server** dialog box is displayed requesting a login with the required permissions. If the login does not have sufficient permissions, and the server is a remote server, an error message is displayed.

#### To add a SQL Server object to the Active Directory

1. On the **SQL Server Properties** dialog box, **Active Directory** tab, click **Add**.

After the SQL Server object is added, click **Refresh** to update the attributes of the instance of SQL Server object in the Active Directory. (Clicking **Refresh** does not add new objects to the Active Directory; it only refreshes the attributes of the server object).

#### To remove a SQL Server object from the Active Directory

• On the **SQL Server Properties** dialog box, **Active Directory** tab, click **Remove**. Removing this SQL Server object from the Active Directory also removes its databases and publications from the Active Directory.

# To add or remove a SQL Server object from the Active Directory using Transact-SQL

• Execute **sp\_ActiveDirectory\_SCP** and set **@action**='CREATE' to add to Active Direcotry or set **@action**='DELETE' to remove from Active Directory.

### **Adding Publications as Active Directory Objects**

After the SQL Server object is enabled for the Active Directory, you can add publications as Active Directory objects. To add publications the SQL Server service account must have at least Power User privileges.

#### To add a publication to the Active Directory

- 1. Right-click the publication, and then click **Publication Properties**.
- 2. On the **General** tab, select **List this publication in the Active Directory**. Or when you are creating a publication using the **Create Publication Wizard**, on the Select Publication Name and Description page, select **List this publication in the Active Directory**. If a Subscriber has access to the publication, you can subscribe to the publication using Active Directory.

#### To add a publication to Active Directory using Transact-SQL

• For new publications, execute **sp\_addpublication** or **sp\_addmergepublicatio**n and set

#### @add\_to\_active\_directory='TRUE'.

 For existing publications, execute sp\_changepublication or sp\_changemergepublication and set @property=publish\_to\_ActiveDirectory, @value='TRUE'.

#### **Browsing or Subscribing to Publications in Active Directory**

You can browse publications in Active Directory and, if allowed, subscribe to publications using the Pull Subscription Wizard or Windows Synchronization Manager.

#### **Browsing or Subscribing to Publications in Active Directory Using the Pull Subscription Wizard**

- 1. At the Subscriber, start the Pull Subscription Wizard, and then on the Look for Publication page, select **Look at publications in the Active Directory or specify publication information.**
- 2. On the Specify Publication page, click the browse button (...) to browse for publications in Active Directory.

#### **Browsing or Subscribing to Publications in Active Directory Using Windows Synchronization Manager**

- 3. In Windows Synchronization Manager, double-click **To create a subscription**, and then select **By browsing Active Directory for publications**.
- 4. On the **Create Anonymous Subscription (Browse the Active Directory)** dialog box, view the publication listed in Active Directory or click the browse (...) button to browse for publications in Active Directory.

For more information, see the Windows 2000 Server documentation.

# **Implementing Replication**

The following stages will help you implement replication, whether you are using snapshot replication, transactional replication, or merge replication.

Stage	Tasks	
Configuring Replication	<ul> <li>Identify the Publisher, Distributor, and</li> <li>Subscribers in your topology. Use SQL Server</li> <li>Enterprise Manager, SQL-DMO, or Transact-</li> <li>SQL system stored procedures and scripts to</li> <li>configure the Publisher, create a distribution</li> <li>database, and enable Subscribers.</li> </ul>	
Publishing Data and Database Objects	Create the publication and define the data and database object articles in the publication, and apply any necessary filters to data that will be published.	
Subscribing to Publications	Create push, pull, or anonymous subscriptions to indicate what publications need to be propagated to individual Subscribers and when.	
Generating the Initial Snapshot	<ul><li>Indicate where to save snapshot files, whether they are compressed, and scripts to run before or after applying the initial snapshot.</li><li>Specify to have the Snapshot Agent generate the snapshot one time, or on a recurring schedule.</li></ul>	
Applying the Initial Snapshot	Apply the snapshot automatically by synchronizing the subscription using the Distribution Agent or the Merge Agent. The snapshot can be applied from the default snapshot folder or from removable media that can be transported manually to the Subscriber before application of the snapshot.	
Synchronizing Data	Synchronizing data occurs when the Snapshot Agent, Distribution Agent, or Merge Agent runs	

and updates are propagated between Publisher and Subscribers.
For snapshot replication, the snapshot will be reapplied at the Subscriber.
For transactional replication, the Log Reader Agent will store updates in the distribution database and updates will be propagated to Subscribers by the Distribution Agent.
If using updatable subscriptions with either snapshot replication or transactional replication, data will be propagated from the Subscriber to the Publisher and to other Subscribers.
For merge replication, data is synchronized during the merge process when data changes at all servers are converged and conflicts, if any, are detected and resolved.

# **Configuring Replication**

Configuring replication is the process of identifying Publishers, Distributors, and Subscribers across your enterprise, configuring them for replication using Microsoft® SQL Server<sup>™</sup> 2000 tools, and then later modifying or disabling replication if necessary.

The steps for configuring replication are:

- 1. Identifying a Distributor.
- 2. Creating a distribution database on the Distributor.
- 3. Enabling Publishers that will use the Distributor.
- 4. Enabling publication databases.
- 5. Enabling Subscribers that will receive published data.

For ease of implementation, you can use the Configure Distribution and Publishing Wizard, script configuration of distribution and publishing using Transact-SQL system stored procedures, or SQL-DMO. After replication is configured, you can use the Publisher and Distributor properties dialog box, Transact-SQL system stored procedures or SQL-DMO to modify the settings.

# **Publishers, Distributors, and Subscribers**

Before you configure publishing and distribution, consider the roles and requirements of the servers in your replication topology.

### Publisher

The Publisher is a server that makes data available for replication to other servers. In addition to being the server where you specify which data is to be replicated, the Publisher also detects which data has changed and maintains information about all publications at that site. Usually, any data element that is replicated has a single Publisher, even if it may be updated by several Subscribers or republished by a Subscriber.

The publication database is the database on the Publisher that is the source of data and database objects to be replicated. Each database used in replication must be enabled as a publication database either through the Configure Publishing and Distribution Wizard, the Publisher and Distributor properties, by using the **sp\_replicationdboption** system stored procedure, or by creating a publication on that database using the Create Publication Wizard.

### Distributor

The Distributor is a server that contains the distribution database and stores meta data, history data, and/or transactions. The Distributor can be a separate server from the Publisher (remote Distributor), or it can be the same server as the Publisher (local Distributor). The role of the Distributor varies depending on which type of replication you implement, and in general, its role is much greater for snapshot replication and transactional replication than it is for merge replication.

Type of Replication	Distributor role
Snapshot Replication or Transactional Replication	• Stores replicated transactions temporarily for transactional replication.

	• Hosts most of the replication agents unless remote agent activation or pull subscriptions are used.
	• Stores meta data and history data.
Merge Replication	<ul> <li>Stores meta data and synchronization history.</li> </ul>
	• Hosts the snapshot agent and merge agent for push subscriptions.

A Distributor may require additional resources to:

- Store the snapshot files for a publication. The default snapshot folder location is on the Distributor; however, you can change the default location or choose an alternate snapshot location. For more information, see <u>Alternate Snapshot Locations</u>.
- Host one or more distribution databases.
- Host processing for most replication agents (for pull subscriptions, the Merge Agent or Distribution Agent runs at the Subscriber). You can however, choose to offload agent processing. For more information, see <u>Remote Agent Activation</u>.

#### **Remote Distributors**

A remote Distributor is a computer that is physically separate from the Publisher and is configured as a Distributor of replication. A local Distributor is a computer that is configured to be both a Publisher and a Distributor of replication.

When you create a publication, the default snapshot folder location is on the

Distributor. If you use this default location, and you use a remote Distributor, make sure the Snapshot Agent at the Publisher can access the snapshot folder. Without access, the Snapshot Agent cannot write the snapshot files to the Distributor.

Similarly, if pull subscriptions access data on a remote Distributor, make sure the Distribution Agent or Merge Agent that runs on the Subscriber has read permission on the snapshot folder if it is located on the Distributor.

Typically, you would choose to use a remote Distributor when you want to offload processing to another computer, when you want minimal impact from replication on the Publisher (for example, if the Publisher is an OLTP server), or if you want a centralized Distributor for multiple Publishers.

The Distributor could be configured as a separate instance of SQL Server, and therefore, could run on the same computer as the Publisher. This would technically be correct and be a remote Distributor, but this is not advised.

### Subscribers

Subscribers are servers that receive replicated data. Subscribers subscribe to publications, not to individual articles within a publication, and they subscribe only to the publications that they need, not necessarily all of the publications available on a Publisher.

If you have applications using transactional replication built with Microsoft® SQL Server<sup>TM</sup> version 6.5 or later, and those applications subscribe directly to articles instead of to publications, the applications will continue to work in SQL Server 2000. However, you should begin to migrate your subscriptions to the publication level where each publication is composed of one or more articles.

#### To configure publishing and distribution

# **Disabling Publishing and Distribution**

Disabling publishing and distribution includes disabling the Distributor and Publishers, deleting the distribution database, and deleting publications and subscriptions.

By using the Disable Publishing and Distribution Wizard, SQL-DMO, or scripts with Transact-SQL system stored procedures on the Distributor, you can:

- Delete all distribution databases on the Distributor.
- Disable all Publishers that use the Distributor and delete all publications on those Publishers.
- Delete all subscriptions to the publications. Subscription information at the Subscriber will not be deleted, and you should delete it manually. Data in the publication and subscription databases will not be deleted; however, it loses its synchronization relationship to any publication databases. If you want the data at the Subscriber to be deleted, you need to delete it manually.

#### To disable publishing and distribution

# **Publishing Data and Database Objects**

When creating a publication, you can choose the tables, filtered partitions of data, and database objects that you want to publish.

A table used in a snapshot or transactional publication can have a maximum of 255 columns and a maximum row size of 8,000 bytes. A table used in a merge publication can have a maximum of 246 columns and a maximum row size of 6,000 bytes.

Horizontal, vertical, dynamic, and join filters enable you to create partitions of data to be published. By filtering published data, you can:

- Minimize the amount of data sent over the network.
- Reduce the amount of storage space required at the Subscriber.
- Achieve better security; the Subscriber sees only data that they need to see.
- Customize publications and applications based on individual Subscriber requirements.
- Avoid conflicts because the different data partitions can be sent to different Subscribers (limiting the number of Subscribers likely to be updating the same data values).
- Restrict visibility of sensitive data to Subscribers. For example, the **Employees** table might be vertically filtered to exclude the employee salary or review information because that is sensitive and might be information that is not necessary at the Subscriber.

Horizontal (row) filters and vertical (column) filters are available for snapshot replication, transactional replication, and merge replication. Dynamic and join

filters are available for merge replication. However, by using transformable subscriptions, you can create custom partitions for snapshot replication and transactional replication that are similar to dynamic partitions. For information about creating filtered partitions of data, see <u>Filtering Published Data</u>. For information about dynamic partitions in snapshot or transactional replication, see <u>Using Transformable Subscriptions to Create Custom Data Partitions</u>.

For information about the specific data types, see <u>Data Needs and Characteristics</u> and <u>Planning for Each Type of Replication</u>.

**Note** When you create a publication using an existing publication as the template, the Publication Access List (PAL) of the original publication will not be copied to the second publication. You must re-create any PAL settings manually using publication properties and the **Publication Access List** tab after the publication is created.

### **Publishing Database Objects**

Database Object	Snapshot Replication or Transactional Replication	Merge Replication
Tables	X	Х
Stored Procedures – Definition	X	X
Stored Procedures – Execution	X	
Views	X	X
Indexed Views	X	X
Indexed Views as Tables	X	
User-Defined Functions	X	Х

The following database objects can be published with Microsoft® SQL Server<sup>™</sup> 2000 replication.

When you publish these objects, their definitions are copied to Subscribers. When you add or drop columns to a publication database, those changes to the definitions of the objects will be propagated to Subscribers. Changes to the definition of other types of objects may not be copied to Subscribers automatically.

When you change data in a published table, or run a stored procedure published for execution, the data changes that are made will be propagated to Subscribers.

If you are publishing a database object that references other database objects, you must publish all objects referenced by the object. For example, the Products Above Average Price view on the **Northwind** database retrieves data from the PRODUCTS tables. If you publish this view, you must also publish the PRODUCTS tables as part of the publication.

A publication containing a stored procedure definition might be replicated even if you do not publish the database objects that the stored procedure references; however, when trying to execute that stored procedure at the Subscriber, you will get an error. This occurs because of deferred name resolution, where object dependencies are checked when the stored procedure is executed rather than when the stored procedure is created.

### **Publishing Views, User-Defined Functions, Stored Procedure Definitions, and Triggers**

After you create views, user-defined functions, and stored procedure definitions in a database, they will appear as objects in the Create Publication Wizard in the **Specify Articles** dialog box.

When you replicate these objects, the definitions are replicated as part of the initial snapshot applied at the Subscriber. Subsequent changes to the definition of these objects are not copied automatically to Subscribers. However, replicating the definition of these objects can provide a convenient mechanism for deploying these components of your application to Subscribers.

When publishing indexed views that are not schema-only articles for snapshot replication or transactional replication, you do not have to replicate the view as a table. When the view is published to the Subscriber, a table is created on the Subscriber that contains the data the view is based upon. Indexing a view as a table at a Subscriber can be a convenient way of replicating the contents of a view without requiring that each of the tables that comprise the view definition are replicated as well. An indexed view published as a table article cannot be

partitioned vertically using column filters.

Triggers are defined as part of a table and are published as a schema option when that table is replicated as part of a publication. To publish triggers for a table article that is being published:

- 1. Right click a publication, and then click **Properties**.
- 2. On the **Articles** tab, click the properties button (...) for a specific table article.
- 3. In the **Table Articles Properties** dialog box, on the **Snapshot** tab, select the **User triggers** check box under **Copy objects to destination**.

**Note** If you are publishing to a Subscriber running an earlier version of SQL Server, you are limited to the functionality of that version. For example, you will not be able to publish views, user-defined functions, triggers and schema objects to Subscribers running SQL Server 7.0.

### **Schema Objects**

In addition to the database objects listed in the table earlier, you can also specify if you want schema objects to be copied, such as declared referential integrity (primary key constraints, reference constraints, unique constraints), clustered indexes, nonclustered indexes, user triggers, extended properties, and collation.

### **Encrypted Database Objects**

Stored procedures, views, triggers, and user-defined functions that are marked with ENCRYPTION or WITH ENCRYPTION cannot be published as part of SQL Server replication.

#### To create publications and define articles

## **Publishing Stored Procedure Execution**

If you include one or more stored procedures as articles in a snapshot or transactional publication, SQL Server 2000 can replicate the execution of the stored procedures rather than the data changes caused by the execution of those stored procedures. This is useful in replicating the results of maintenanceoriented stored procedures that may affect large amounts of data.

If replicated as a series of data manipulation language (DML) SQL statements, these procedures can require significant amounts of network resources, distribution database space, and server processing time. Replicating the changes as one stored procedure statement can greatly increase the efficiency of your application, but this feature should be used with care.

There are two different ways in which the execution of a stored procedure can be published:

- Procedure execution article. Replicates the procedure execution to all Subscribers of the article. This occurs regardless of whether individual statements in the stored procedure were successful. Furthermore, because changes made to data by the stored procedure can occur within multiple transactions, data at the Subscribers cannot be guaranteed to be consistent with data at the Publisher.
- Serializable procedure execution article. Replicates the procedure execution only if the procedure is executed within the context of a serializable transaction. If the stored procedure is executed from outside a serializable transaction, changes to data in published tables are replicated as a series of DML statements. This behavior contributes to making data at the Subscriber consistent with data at the Publisher. This is especially useful for batch operations, such as large cleanup operations.

#### **Procedure Execution Articles**

If a stored procedure execution is replicated, no new data changes or procedure

executions from the current connection are replicated until that stored procedure finishes executing. For example, if a stored procedure that modifies data in a published table is executed, and the procedure execution is replicated, the individual DML changes to the published table are not replicated.

Similarly, if a stored procedure that executes another published stored procedure is executed, and the execution is replicated, the EXEC statement of the stored procedure called by the first procedure is not replicated. However, if a published stored procedure modifies data within another database and the underlying table is replicated, those data changes are replicated as DML statements.

By default, the stored procedure definition at the Publisher is propagated to each Subscriber. However, you can also define the stored procedure logic to be different at a Subscriber. This is useful if you want different logic to be executed at the Publisher and Subscriber. For example, consider **sp\_big\_delete**, a stored procedure at the Publisher that has two functions: it deletes 1,000,000 rows from the replicated table **big\_table1** and updates the nonreplicated table **big\_table2**. To reduce the demand on network resources, you should propagate the 1 million row delete as a stored procedure by publishing **sp\_big\_delete** and creating subscriptions at the Subscribers. At the Subscriber, you can define **sp\_big\_delete** to **big\_table2**.

Each time a published stored procedure is executed at the Publisher, the execution and the parameters passed to it for execution are forwarded to each Subscriber to the publication.

For example, if you execute a stored procedure that contains actions on several different tables, only the execution of that procedure (along with its parameters) is forwarded to each Subscriber. If you publish the underlying tables instead of the stored procedure, each data modification (insert, update, or delete) generated by the procedure is marked for replication and forwarded to each Subscriber. During the execution of a published stored procedure, SQL Server 2000 temporarily suspends marking transactions or commands for replication within that procedure to avoid duplication of effort.

Stored procedure replication both reduces the volume of commands requiring forwarding to Subscribers and increases the performance of your application by executing fewer dynamic SQL statements at each Subscriber.

For example, assume you created a stored procedure:

CREATE PROC give\_raise AS UPDATE EMPLOYEES SET salary = salary \* 1.10

This procedure gives each of the 10,000 employees in your company a 10 percent pay increase. When you execute this stored procedure at the Publisher, it updates the salary for each employee. Without stored procedure replication, the update is sent to Subscribers as a large, multistep transaction:

BEGIN TRAN UPDATE EMPLOYEES SET salary = salary \* 1.10 WHERE PK = 'en UPDATE EMPLOYEES SET salary = salary \* 1.10 WHERE PK = 'en

And so on for 10,000 updates.

With stored procedure replication, SQL Server 2000 sends only the execution of the stored procedure:

### EXEC give\_raise

**IMPORTANT** Stored procedure replication is not appropriate to all applications. If an article is filtered horizontally, so that there are different sets of rows at the Publisher than at the Subscriber, executing the same stored procedure at both returns different results. Similarly, if an update is based on a subquery of another, nonreplicated table that has different values at both the Publisher and Subscriber, executing the same stored procedure at both returns different results.

To ensure that the same results are achieved at both the Publisher and Subscriber, the default behavior of SQL Server 2000 is to send the resultant data changes as a series of singleton statements in a transaction.

### **Serializable Procedure Execution Articles**

The following example illustrates why it is recommended that you set up replication of procedures as serializable procedure articles.

```
BEGIN TRANSACTION T1
SELECT @var = max(col1) FROM tableA
```

UPDATE tableA SET col2 = <value> WHERE col1 = @var

```
BEGIN TRANSACTION T2
WHERE col1 = @var
INSERT tableA VALUES
COMMIT TRANSACTION T2
```

In the previous example, it is assumed that the SELECT in transaction T1 happens before the INSERT in transaction T2.

If the procedure is not executed within a serializable transaction (for example, with isolation level set to SERIALIZABLE), transaction T2 will be allowed to insert a new row within the range of the SELECT statement in T1 and it will commit before T1. This also means that it will be applied at the Subscriber before T1. When T1 is applied at the Subscriber, the SELECT can potentially return a different value than at the Publisher and can result in a different outcome from the UPDATE.

If the procedure is executed within a serializable transaction, transaction T2 will not be allowed to insert within the range covered by the SELECT statement in T2. It will be blocked until T1 commits ensuring the same results at the Subscriber.

Locks will be held longer when you execute the procedure within a serializable transaction and may result in reduced concurrency.

To replicate a stored procedure when it is executed inside a serializable transaction, in the article properties for the stored procedure to be published, click the **Other** tab, and then select **Only when it is executed inside a serializable transaction**.

Using Transact-SQL system stored procedures, you can indicate that the stored procedure is to be replicated when it is executed inside a serializable transaction by setting the **@type** parameter of **sp\_addarticle** to a value of **serializable proc exec.** 

## **Using Custom Stored Procedures in Articles**

When the Log Reader Agent encounters an INSERT, UPDATE, or DELETE statement marked for replication in the transaction log of a publication database, it usually reconstructs one row Transact-SQL statement from the recorded data changes. The Distribution Agent then sends that reconstructed Transact-SQL statement to each Subscriber and applies the statement to the destination table in each destination database. This is the default data replication mechanism used by Microsoft® SQL Server<sup>TM</sup> 2000 when there are one or more heterogeneous Subscribers.

If all Subscribers are instances of SQL Server 2000, SQL Server 2000 can override the INSERT, UPDATE, and DELETE statements from the transaction log with custom stored procedures at each Subscriber. For each published table, there are three ways you can handle each type of statement (INSERT, UPDATE, or DELETE) detected by the Log Reader Agent. You can:

- Leave the default replication mechanism in place.
- Specify that no action will be taken at any Subscriber. Transactions of that type are not replicated. For example, if you select **Replace DELETE statements with this stored procedure** and enter NONE, DELETE statements are not replicated for that article.
- Specify that a custom procedure be called at all Subscribers. When the Log Reader Agent encounters a statement of the specified type (INSERT, UPDATE, or DELETE) in a transaction marked for replication, it constructs a stored procedure call based on this syntax and passes column values to the referenced stored procedure. This is the default behavior for SQL Server 2000 Subscribers.

### **About Custom Stored Procedures**

Depending on the requirements of the application, the parameters of the stored procedures can be specified using:

- CALL syntax
- XCALL syntax
- MCALL syntax

Each method differs in the amount of data that is propagated to the Subscriber. For example, MCALL will pass in values only for the columns that are actually affected by the update, and a bitmask representing the changed columns and XCALL will pass in all columns (whether affected by an update or not) and all the old data values for each column. This allows flexibility to application developers with diverse requirements. When using XCALL, the before image values for **text** and **image** columns are expected to be NULL.

To implement custom stored procedure—based replication for a published table, stored procedures must be created either by replication or by the user. These custom stored procedures expect to receive and process these parameters:

#### call Syntax

**INSERT** stored procedures

Stored procedures handling INSERT statements will be passed the inserted values for all columns:

c1, c2, c3,... cn

UPDATE stored procedures

Stored procedures handling UPDATE statements will be passed the updated values for all columns defined in the article, followed by the original values for the primary key columns:

c1, c2, c3,... cn, pkc1, pkc2,... pkcn

**Note** No attempt is made to determine which columns were changed.

DELETE stored procedures

Stored procedures handling DELETE statements will be passed values for

the primary key columns:

pkc1, pkc2,... pkcn

#### mcall Syntax

UPDATE stored procedures

Stored procedures handling UPDATE statements will be passed the updated values for all columns defined in the article, followed by the original values for the primary key columns, followed by a bitmask (**binary(n)**) parameter that indicates the changed columns:

c1, c2, c3,... cn, pkc1, pkc2,... pkcn, bitmask

#### xcall Syntax

UPDATE stored procedures

Stored procedures handling UPDATE statements will be passed the original (the before image) values for all columns defined in the article, followed by the update (the after image) values for all columns defined in the article.

old-c1, old-c2, old-c3,... old-cn, c1, c2, c3,... cn,

DELETE stored procedures

Stored procedures handling UPDATE statements will be passed the original (the before image) values for all columns defined in the article.

old-c1, old-c2, old-c3,... old-cn

If you want your INSERT, UPDATE, or DELETE stored procedure to return an error when a failure status is encountered, you must add a RAISERROR statement so that the Distributor will capture the failure status coming back. If the severity is greater than 12, the Distributor stops the distribution process to that Subscriber. If this procedure definition is distributed as part of the article schema definition file, it will be sent using ODBC. In this case, only single quotation marks (') can be used to define the RAISERROR message string. The use of double quotation marks ('') generates an error.

You can also program a custom stored procedure to skip specified errors. For

more information, see <u>Handling Agent Errors</u>.

Indicate whether you want to use single quotation marks or double quotation marks when you specify article properties in the Create Publication Wizard. You can also make this choice in the **Properties** dialog box for the article.

# **Subscribing to Publications**

A subscription is the request for data or database objects to be published to a specific Subscriber. A Subscriber can have several subscriptions to different publications.

A subscription defines what publication will be replicated, where and when. A subscription can be created either at the Publisher (a push subscription) or at the Subscriber (a pull subscription). Push subscriptions are then created and synchronized at the Publisher/Distributor and the synchronizing agent (Distribution Agent or Merge Agent) is typically run at the Distributor. Pull subscriptions and anonymous subscriptions are created and synchronized at the Subscriber and the synchronizing agent is typically run at the Subscriber.

When planning for subscriptions, consider where you want administration of the subscription to take place and where you want agent processing to occur. The type of subscription you choose controls where the agent runs, but in some circumstances, using remote agent activation, you can offload the synchronization agent processing to another server.

Additionally, be aware of publication and distribution database properties for subscription deactivation and expiration. For more information, see <u>Subscription</u> <u>Deactivation and Expiration</u>.

Subscription	Characteristics	Use When
Push Subscription	With a push subscription, the Publisher propagates changes to a Subscriber without a request from the Subscriber. Changes can be pushed to Subscribers on demand, continuously, or on a scheduled basis. By default, the Distribution Agent or Merge Agent runs at the Distributor.	<ul> <li>Data will typically be synchronized on demand or on a frequently recurring schedule.</li> <li>Publications require near real-time movement of data without polling.</li> </ul>

	Because a Subscriber must explicitly be enabled at the Publisher to receive a push subscription, push subscriptions are known as named subscriptions.	<ul> <li>The higher processor overhead at a Publisher using a local Distributor does not affect performance.</li> <li>You need easier administration from a centralized location (the Distributor).</li> <li>The centralized Distributor will establish the schedule on which connections will be made with remote, occasionally connected Subscribers.</li> </ul>
Pull Subscription	With a pull subscription, the Subscriber requests changes made at the Publisher. Pull subscriptions allow the user to determine when the data changes are synchronized. By default, the Distribution Agent or the Merge Agent runs at the Subscriber. Because a Subscriber must explicitly be enabled at the Publisher to receive a push subscription, pull subscriptions are known	<ul> <li>Administration of the subscription will take place at the Subscriber.</li> <li>The publication has a large number of Subscribers (for example, Subscribers (for example, Subscribers using the Internet), and when it would be too resource-intensive to run all the agents at one site or all at the Distributor.</li> </ul>

	as named subscriptions.	<ul> <li>Subscribers are autonomous, disconnected, and/or mobile. Subscribers will determine when they will connect to the Publisher/Distributor and synchronize changes.</li> <li>Data will typically be synchronized on demand or on a schedule rather than continuously.</li> </ul>
Anonymous Subscription	An anonymous subscription is a type of pull subscription. Detailed information about the subscription and the Subscriber is not stored at the Publisher when using an anonymous subscription.	<ul> <li>All of the rules for pull subscriptions apply to anonymous subscriptions.</li> <li>Applications have a large number of Subscribers.</li> </ul>
	Instead, the Subscriber keeps information about the subscription and what the data was when the subscription was last synchronized. This information is then passed on to the	• You do not want the overhead of maintaining extra information at the Publisher or Distributor.
	Distributor when the next synchronization occurs. The Subscriber does not	• If Subscribers use the Internet to access

need to be explicitly named at the Publisher when using anonymous subscriptions.	publications.
---	---------------

# **Push Subscriptions**

Push subscriptions can simplify and centralize subscription administration because you do not need to administer each Subscriber individually. The Distribution Agent or Merge Agent runs at the Distributor when synchronizing a push subscription. Push subscriptions are created at the Publisher, and the replication agents propagate data and updates to a Subscriber without a request from the Subscriber. Changes can also be pushed to Subscribers on a scheduled basis.

Use push subscriptions when:

- Data will typically be synchronized on demand or on a frequently recurring schedule.
- Publications require near real-time movement of data without polling.
- The higher processor overhead at a Publisher using a local Distributor does not affect performance.
- You need easier administration from a centralized location (the Distributor).

The centralized Distributor will establish the schedule on which connections will be made with remote, occasionally connected Subscribers. With push subscriptions, the Distribution Agent (for snapshot and transactional publications) or the Merge Agent (for merge publications) runs at the Distributor. However, if you need to offload agent processing from the Distributor but retain some of the benefits of easier administration, you can run the agent at the Subscriber. For more information, see <u>Remote Agent Activation</u>.

Because remote agent activation is available, the determining factors to consider when setting up subscriptions is what type you will need (push, pull, or anonymous) and where the replication agent will run.

Users who are members of the **sysadmin** or **db\_owner** roles at that Subscriber

can set up a push subscription. However, for a member of the **db\_owner** role to set up a push subscription, a member of the **sysadmin** role must register the Subscribers.

For a subscription to be created, you must have a publication at the Publisher and a subscription database at the Subscriber. You can create the subscription database before creating the subscription, or specify a new subscription database in the Create Push Subscription Wizard. You can create a push subscription for any Subscribers that are enabled in the Publisher and Distributor properties.

Push subscriptions and pull subscriptions are known as named subscriptions because information about the subscription and the Subscriber is stored at the Publisher, and performance information about the Subscriber is stored at the Distributor. This is in contrast to anonymous subscriptions (which are a type of pull subscription) for which little or no information about the subscription and the Subscriber is stored.

When you create a push subscription, you specify:

- The name of the Subscriber.
- The name of the subscription database.
- Whether the Distribution Agent or Merge Agent runs at the Distributor (default) or at the Subscriber using remote agent activation.
- Whether the Distribution Agent or Merge Agent runs continuously, on a scheduled basis or on demand only.
- If the Snapshot Agent should create an updated initial snapshot for the subscription and if the Distribution Agent or Merge Agent should apply that snapshot at the Subscriber.
- For snapshot or transactional publications that allow immediate updating or queued updating, the options that this subscription will use (available if you enable advanced options in the Push Subscription

Wizard).

- For merge replication, the priority value for the changes made in the subscription database to be used during conflict detection and resolution.
- For snapshot replication and transactional replication, specify that the subscription will use immediate updating, queued updating, or transform published data options (these must first be enabled when creating the publication).
- Services that will be started to create the subscription.

#### To create a push subscription

# **Pull Subscriptions**

Pull subscriptions are created at the Subscriber, and the Subscriber requests data and updates made at the Publisher. Pull subscriptions allow the user at the Subscriber to determine when the data changes are synchronized, which can be on demand or scheduled.

Use pull subscriptions when:

- Administration of the subscription will take place at the Subscriber.
- The publication has a large number of Subscribers (for example, Subscribers using the Internet), and when it would be too resource-intensive to run all the agents at one site or all at the Distributor.
- Subscribers are autonomous, disconnected, and/or mobile. Subscribers will determine when they will connect to the Publisher/Distributor and synchronize changes.

Data will typically be synchronized on demand or on a schedule rather than continuously. One feature of pull subscriptions is that the Distribution Agent for snapshot and transactional publications and the Merge Agent for merge publications all run at the Subscriber. This can result in a reduction of the amount of processing overhead on the Distributor. However, if you need the Distribution Agent or Merge Agent to run at the Distributor, you can offload agent processing from the Subscriber.

For example, you might use this option if the Subscriber will determine when it is connected to the network and ready to synchronize, but you want to run the agent at the Distributor to make use of better processing power at the Distributor. For more information, see <u>Remote Agent Activation</u>.

Another feature of pull subscriptions is that members of the **sysadmin** or **db\_owner** roles at the Subscriber decide which publications are received and when. Each Subscriber can have subscriptions to multiple publications at different Publishers.

For a subscription to be created, you must have a publication at the Publisher and a subscription database at the Subscriber. You can create the subscription database before creating the subscription, or specify a new subscription database in the Create Pull Subscription Wizard. You can create a pull subscription to any publication that has been enabled for pull subscriptions on a registered Publisher.

When you create a pull subscription, you specify:

- The name of the subscription database.
- Whether the Snapshot Agent should create an initial snapshot and the Distribution Agent or Merge Agent should apply that snapshot at the Subscriber.
- The location of the snapshot files to apply when initializing the subscription.
- The priority of the subscription for merge.
- For snapshot replication and transactional replication, specify that the subscription will use immediate updating, queued updating, or transform published data options (these must first be enabled when creating the publication).
- Whether the Distribution Agent or Merge Agent runs continuously, on demand, or on a scheduled basis.
- Services that will be started to create the subscription.

Push subscriptions and pull subscriptions are known as named subscriptions because information about the subscription and the Subscriber is stored at the Publisher and performance information about the Subscriber is stored at the Distributor. This is in contrast to anonymous subscriptions (which are a type of pull subscription) for which information about the subscription and the Subscriber is not stored. When you create a pull subscription and a push subscription for the publication already exists for the Subscriber, an error message informs you that the push subscription already exists and that you should drop any push subscriptions before proceeding. When you create a pull subscription, and another pull subscription to the same publication already exists, you will be required to drop the existing subscription before adding the new one unless the first subscription has expired.

#### To create a pull or anonymous subscription

## **Anonymous Subscriptions**

An anonymous subscription is a type of pull subscription for which detailed information about the subscription and the Subscriber is not stored. Initiated at the Subscriber, the Subscriber is responsible for keeping an anonymous subscription synchronized.

Use anonymous subscriptions when:

- Applications have a very large number of Subscribers.
- You do not want the overhead of maintaining extra information at the Publisher or Distributor.
- If Subscribers use the Internet to access publications.
- All the rules for pull subscriptions apply to anonymous subscriptions.

A defining factor for deciding to use anonymous subscriptions with snapshot replication and transactional replication is the clean up of the distribution database. The distribution database is cleaned up by the Distribution Clean Up Agent, which by default is scheduled to run every 10 minutes. The Distribution Clean Up Agent removes replicated transactions from the distribution database; however, if you are using anonymous subscriptions, the transactions are kept for the retention period of the subscription to given anonymous subscriptions time to synchronize.

With merge, the significant factor affecting scale relates to whether or not Subscribers know about other Subscribers. Meta data is stored for all global subscriptions in the **Sysmergesubscriptions** system table. Information can be viewed about all Subscribers, the Publisher, and any global subscriptions to publications to which they are subscribed.

If you enable anonymous subscriptions for the publication, the user creating the pull subscription can specify that the subscription should be anonymous in the Create Pull Subscription Wizard, by using Windows Synchronization Manager

or in the stored procedure.

To create a pull or anonymous subscription

# **Applying the Initial Snapshot**

After a publication and subscription have been created, you need to create and transfer an initial snapshot to the Subscriber. The snapshot transfers schema and data to the Subscriber, as well as constraints, extended properties, indexes, triggers and system tables necessary for replication.

The snapshot consists of different files depending on the type of replication and the articles in your publication. The files can be viewed using the Snapshot Explorer. For more information, see <u>Exploring Snapshots</u>.

Type of Replication	Common Snapshot Files
Snapshot Replication	schema (.sch); data (.bcp); constraints and indexes
or Transactional	(.dri); constraints (.idx).
Replication	
	schema (.sch); data (.bcp); constraints and indexes (.dri); triggers (.trg); system table data (.sys); conflict tables (.cft)

Applying the initial snapshot can take additional time if you are transferring a large amount of data over the network, or if you have a slow link. In that case, you may want to consider saving the snapshot to removable media and transferring it to Subscribers manually.

Additionally, SQL Server 2000 has improved performance of applying the initial snapshots with: the ability to compress snapshots; concurrent snapshot processing for transactional replication; and dynamic snapshot for merge publications that use dynamic filters. For more information, see Improving Performance While Generating and Applying Snapshots.

## **Generating the Initial Snapshot**

Snapshots can be created:

- Manually by running the Snapshot Agent after creating the publication.
- Automatically when the publication is created by selecting **Create the first snapshot immediately** on the Set Snapshot Agent Schedule page in the Create Publication Wizard.
- At a scheduled time, as specified by the Snapshot Agent Schedule page in the Create Publication Wizard.

By default, snapshots are saved in the default snapshot folder located on the Distributor. On a Distributor running Microsoft® Windows NT® version 4.0 or Windows 2000, the snapshot folder defaults to using the <drive>\$ share and a path of \\<computer>\<drive>\$\Program Files\Microsoft SQL Server\Mssql\Repldata.

On a Distributor running the Microsoft Windows 98 operating system, the snapshot folder defaults to using the <drive> without a share and a path of \\ <computer>\<drive>\Program Files\MicrosoftSQL Server\Mssql\Repldata. If your application requires the ability to create pull subscriptions on a server running the Windows 98 operating system, you must change the snapshot folder to a network path accessible by replication agents running at the Publisher and Subscribers. You can change the local path to a network path by sharing the folder.

**IMPORTANT** The <drive>\$ share is a special administration-only share, and you will not be able to grant rights to it; only administrators on the computer can access it. It is recommended that you change the default snapshot location to a network location or shared folder that the Subscriber can access. This also applies if you are going to allow pull or anonymous subscriptions because remote Subscribers or Subscribers over the Internet will rarely be administrators. You can test the Subscriber connection to the snapshot folder by mapping a network drive in Windows Explorer at the Subscriber.

You can also save snapshot files on removable media such as removable disks, CD-ROMs, or in locations other than in the default snapshot folder on the Distributor, such as File Transfer Protocol (FTP) servers. Additionally, you can view and transfer the snapshot files using the Snapshot Explorer, compress the files so that they are easier to store and transfer, and execute scripts before or after snapshot synchronization.

To view or modify the default snapshot folder location

### **Alternate Snapshot Locations**

Alternate snapshot locations enable you to store snapshot files in a location other than, or in addition to, the default location, which is often located on the Distributor. Alternate locations can be on another server, on a network drive, or on removable media such as CD-ROMs or removable disks.

Saving snapshot files in an alternate location can alleviate disk overhead on the Distributor, offers an administrative advantage, and allows you to transfer files using removable media.

Alternate snapshot locations are stored as a property of the publication. You can view this information in the publication properties on the **Snapshot Location** tab. Because the alternate snapshot location is a publication property, the Distribution Agent and the Merge Agent are able to locate the proper snapshot as part of the synchronization process. However, if you change the alternate location after creating the initial snapshot, the Distribution Agent and the Merge Agent may not be able to find the alternate location and you may have to reinitialize the snapshot.

Subscribers running earlier versions of Microsoft® SQL Server<sup>™</sup> cannot use the alternate snapshot location. Therefore, continue to use the default snapshot location to store snapshot files for those Subscribers.

If you want to specify an alternate snapshot folder location or if you want to compress snapshot files, create the publication without creating the initial snapshot immediately, set the publication properties for the snapshot location, and then run the Snapshot Agent for that publication.

**Note** Do not specify an alternate location in publication properties that is the same as the default snapshot folder location. You will receive an error message, and should specify a different alternate location or use the default location.

You can also increase the availability of replication when using failover clustering by saving the snapshot files to a share on a server running Microsoft Cluster Server. For more information, see <u>Failover Clustering</u>.

#### To specify alternate snapshot locations

## **Compressed Snapshot Files**

When snapshot files are too large to fit on removable media or require transmission over slow networks, compressing the snapshot files is an option. Compressing snapshot files can reduce network traffic but it increases the time to generate and apply the snapshot.

Compression writes data in the Microsoft® CAB file format. You can compress snapshot files when you are saving them to an alternate location or when Subscribers are accessing them using FTP. Snapshot files written to the default snapshot folder on the Distributor cannot be compressed.

### **Disk Space Requirements**

The amount of space required for a single snapshot can be affected by several factors including the size and number of articles published. You can create snapshot files in the default snapshot folder on the Distributor and in an alternate location. Compressing the snapshot files in the alternate location can reduce the overall space required.

When snapshot files are created in both the default directory and in an alternate location on the same drive, each file is created initially in the default directory and then copied to the alternate location. If you are using compressed snapshot files, the files are copied and compressed before they are placed in the alternate snapshot location. The total space required for all snapshot files in this situation is the size of the original snapshot files in the default location, plus the size of the compressed snapshot files in the alternate location.

If the alternate storage location is on a different drive than the default location, the space required at the default location is the size of the snapshot files. The space required at the alternate location is the total size of the compressed snapshot files.

When using only the alternate snapshot location, the Snapshot Agent writes files directly to that location. After the Snapshot Agent generates the files, the files are compressed by the CAB utility and become part of the compressed snapshot file with the extension .cab. After each file is compressed successfully and included in the compressed snapshot (.cab) file, the original, noncompressed file

is deleted. The space required in the alternate location is the size of the last file in the default snapshot location (usually a .bcp file) plus the size of the compressed snapshot (.cab) file.

When the Subscriber receives a compressed snapshot file, the file is written initially to a temporary location. The default client working directory can be used, or an alternate location can be specified in the subscription properties. After the compressed snapshot file is copied to the Subscriber, the file is decompressed, in order, one file at a time by the CAB utility.

The uncompressed files are read by either the Merge Agent or the Distribution Agent and then executed or applied to the Subscriber. As each file is applied successfully, it is deleted and the next file in the snapshot directory is decompressed. Space required at the Subscriber is the size of the compressed snapshot file plus the largest uncompressed file.

#### To compress and deliver snapshot files

## **Exploring Snapshots**

Exploring snapshots allows you to use Windows Explorer to review or customize current snapshot files or copy them to another location. After the Snapshot Agent has created the snapshot files containing the schema and data of published tables, the files are stored in the snapshot folder on the Distributor or an alternate location. You can then use the Windows Explorer to view and transfer these snapshot files.

**Note** You may not see a snapshot for named Subscribers that have received synchronization objects. In this case, the Distribution Cleanup Agent may have removed all the contents of the directory. When viewing snapshots for Subscribers with named subscriptions, you may not see a current snapshot if snapshot processing has not completed or the Snapshot Agent has not generated a new snapshot file for that Subscriber.

#### To browse and copy snapshot files

## **Transferring Snapshots**

Before a new Subscriber can receive incremental changes from a Publisher, it must contain tables with the same schema and data as the tables at the Publisher. After the snapshot is created at the Publisher and stored, you need to transfer the snapshot to the Subscriber, either using Microsoft® SQL Server<sup>™</sup> 2000 replication agents or manually.

### **SQL Server Applies the Initial Snapshot**

When SQL Server 2000 applies the snapshot to Subscribers, either the Distribution Agent (for snapshot replication and transactional replication) or the Merge Agent (for merge replication) applies the schema and data files to the subscription database on the Subscriber.

Unless you are using transactional replication with concurrent snapshot processing, share locks are held while the snapshot is generated so a full, logical, and consistent set of data is produced. This means that while the data can be queried, it cannot be updated during the time it takes to generate the snapshot. To minimize any inconvenience to your operations, always plan to generate a snapshot when updates are minimal. If you are using transactional replication, concurrent snapshot processing allows you to continue data modifications while the snapshot is generated. For more information, see <u>Improving Performance</u> <u>While Generating and Applying Snapshots</u>.

For merge replication, the process is similar to concurrent snapshot processing for transactional replication because locks are in place only for the duration of the copy of the merge contents table. The tables are not locked when the snapshot is being bulk copied and updates at the publication database are not prevented for the duration of the entire snapshot.

When snapshots are distributed and applied to Subscribers, only those Subscribers waiting for initial or new snapshots are affected. Other Subscribers to that publication (those that are already receiving inserts, updates, deletes, or other modifications to the published data) are unaffected.

You can specify that SQL Server 2000 should initialize the schema and data on the Initialize Subscription page in the Create Push Subscription or Create Pull

Subscription Wizard.

When the first synchronization occurs (which you specify to occur immediately in the subscription wizards), the Distribution Agent or Merge Agent applies the initial snapshot and then proceeds to propagate updates and other data modifications.

### **Applying the Snapshot Manually**

If the publication is large, it may be more efficient to load the snapshot from a compact disc, or other storage device.

For example, if you have a 20 GB database, it may be easier and faster to dump the database to removable media, express courier it to the Subscriber location, and reload the database instead of sending the file over a slow network. If you decide to load the snapshot this way, SQL Server 2000 will not synchronize the published articles with the destination tables.

For this example to work effectively in merge replication, you must have precreated and populated the **ROWGUIDCOL** column or have already run the Snapshot Agent at the Publisher. Applying the snapshot is still required so that system tracking data and objects necessary for merge replication are at the Subscriber.

It is recommended that you use attachable subscription databases when you need to apply a large snapshot rather than using a combination of standard and dynamic snapshots and alternate snapshot locations with compression. For more information, see <u>Attachable Subscription Databases</u>.

With SQL Server 2000, you can store snapshots in a location other than or in addition to the default location, and you can browse snapshot folders, so it is easier to view, copy and move snapshot files.

To apply the snapshot manually, you can:

- Save the snapshot files to removable media such as a compact disc, tape device, or removable disk and then send the media to the Subscriber location.
- Base the initial snapshot off a database dump.

You can specify that the Subscriber already have the schema and data on the Initialize Subscription page in the Create Push Subscription or Create Pull Subscription Wizard.

The Distribution Agent or Merge Agent then assumes that the Publisher and Subscriber are already synchronized, and starts sending inserts, updates, deletes, or other modifications to the published data immediately.

If a current snapshot is not already waiting, SQL Server will wait until the next time the Snapshot Agent runs according to its schedule (by default, that is once a day at 1 A.M.) before applying the snapshot to the new Subscriber.

If you create a publication and enable it for anonymous subscriptions or if you specify that the snapshot should be retained in the snapshot location (both of these are options in the Create Publication Wizard), the snapshot will run at its scheduled time and it will be retained in the snapshot location. If you do not choose one of these options, the snapshot will not be retained, therefore, when a new Subscriber attempts to synchronize for the first time, it will have to wait until the next time a snapshot is generated to have the snapshot applied.

### **Attachable Subscription Databases**

The attachable subscription databases feature allows you to transfer a database with published data and subscriptions from one Subscriber to another. After the database is attached to the new Subscriber, the database at the new Subscriber will automatically receive its own pull subscriptions to the publications at those Publishers.

Attachable subscription databases requires the following steps:

- 1. Configuring a publication to allow copying.
- 2. Copying the subscription database.
- 3. Transferring and attaching the subscription database to a new Subscriber.

Subscription databases copied and attached to other Subscribers can contain multiple pull subscriptions for multiple publications using snapshot replication, transactional replication, or merge replication. Attachable Subscription databases are not compatible with heterogeneous databases or instances of Microsoft® SQL Server<sup>TM</sup> version 6.5. This feature is not available with push subscriptions.

## **Configuring a Publication to Allow Copying**

To use attachable subscription databases, you must first configure subscription options for each publication that propagates data to a Subscription database that will be copied. These options allow new subscriptions to be created after the subscription database is attached to a different Subscriber.

**IMPORTANT** After you create a subscription to the publication, you cannot change the subscription options. If you want to use attachable subscription databases, configure the subscription options for the publication before creating subscriptions.

To configure a publication to allow copying of subscription databases

## **Copying a Subscription Database**

After you have configured the subscription options in the publication, created, and synchronized a pull subscription to a subscription database, you can copy the subscription database.

The copy of the subscription database includes data, views, stored procedures, user-defined functions, schema, and all objects that are not replicated that comprise the database. Only subscription databases that are contained in one, primary file group can be copied. You must synchronize at least one subscription to the subscription database before copying.

During the copy process, a compressed Microsoft Subscription File (.msf) is created. The .msf file contains subscription information up to the last synchronization of the subscription database.

When creating the .msf file, save the file to a location. The file can then be picked up and transferred over the network, transferred using removable media, or attached to an e-mail message.

#### To copy a subscription database

## **Attaching a Subscription Database**

After copying the subscription database and saving it as an .msf file, you can transfer and attach it to any Subscriber. When you attach it to the new Subscriber, the file is decompressed and then attached. You must have database owner permissions to attach a subscription database to a new Subscriber.

Merge replication .msf files are valid only for the retention period set for those publications, and transactional replication .msf files are valid only for the maximum retention period of the distribution database. If a subscription expires, new snapshot files must be generated and the subscription reinitialized.

If the subscription database has subscriptions to publications that allow queued updating with auto identity range articles, you will need to run the distribution agents to obtain new identity ranges on the Subscriber after attaching the subscription database. If the subscription database has subscriptions to merge publications, you will need to run the merge agents on the Subscriber after attaching the subscription database to prevent conflicts.

**Note** Detaching a database created with SQL Server 7.0 with subscriptions to transactional replication and attaching it on a server running SQL Server 2000 is not recommended. If this is required, run

**sp\_vupgrade\_subscription\_databases**, a system stored procedure in the **master** database, to upgrade the replication schema after attaching the database.

The procedures for attaching a subscription database are different depending on whether you are using anonymous or named subscriptions.

## **Attaching Databases with Named Subscriptions**

For subscriptions that are not anonymous, you must enable the Subscriber so it can receive data from the Publisher, attach the subscription database, and add the subscription at the Publisher using SQL-DMO or stored procedures. You can add the subscription at the Publisher before or after attaching the database.

#### To enable a Subscriber to receive published data

## **Attaching Databases with Anonymous Subscriptions**

After the subscription database is attached to the new Subscriber, the subscriptions to the original publications will be generated automatically. This allows the synchronization process to begin immediately for anonymous subscriptions. If you are using anonymous subscriptions, you do not need to enable the Subscriber.

To attach a subscription database with anonymous subscriptions

# Improving Performance While Generating and Applying Snapshots

Depending on the amount of data in your publication and your network connection and resources, applying the initial snapshot to Subscribers can be time- and resource-consuming. Concurrent snapshot processing for transactional replication, dynamic snapshots (merge replication) and the **–UseInprocLoader** property have been added to Microsoft® SQL Server<sup>™</sup> 2000 to improve performance while generating the initial snapshot and applying it at Subscribers.

### **Concurrent Snapshot processing for Transactional Replication**

Typically, with snapshot generation, SQL Server places shared locks on all tables published as part of replication for the duration of snapshot generation. This can prevent updates from being made on the publishing tables. Concurrent snapshot processing, available only with transactional replication, does not hold the share locks in place during the entire snapshot generation, thus allowing users to continue working uninterrupted while SQL Server 2000 creates initial snapshot files.

When you create a new publication using transactional replication and indicate that all Subscribers will be instances of SQL Server 7.0 or SQL Server 2000, concurrent snapshot processing is enabled automatically.

For more information, see <u>How Transactional Replication Works</u>.

### **Snapshot Processing for Merge Replication**

For merge replication, the process is similar to concurrent snapshot processing for transactional replication because locks are in place only for the duration of the copy of the merge contents table. The tables are not locked when the snapshot is being bulk copied and updates at the publication database are not prevented for the duration of the entire snapshot.

### **Dynamic Snapshots**

Dynamic snapshots provide a performance advantage when applying the

snapshot of a merge publication with dynamic filters. By using SQL Server 2000 bulk copy programming files to apply data to a specific Subscriber instead of a series of INSERT statements, you will improve the performance of applying the initial snapshot for dynamically filtered merge publications.

For more information, see <u>Dynamic Snapshots</u>.

### Add a ROWGUIDCOL to Merge Publications

By planning ahead and creating a column that can be used to help track changes during merge replication, you will avoid the sometimes significant time (and disk and log) decrease in performance that could occur from waiting for the Snapshot Agent to alter the tables for you.

Merge replication requires that each published table have a ROWGUID column. If a ROWGUID column does not exist in the table before the Snapshot Agent creates the initial snapshot files, the agent must first add and populate the ROWGUID column. To gain a performance advantage when generating and applying snapshots during merge replication, create the ROWGUID column on each table published during merge replication. When creating the column, specify:

- The column title as ROWGUID.
- The data type as UNIQUEIDENTIFIER.
- The default as NEWID().
- The ROWGUIDCOL property.
- An index on the column.

The ROWGUID column is used frequently for relating to merge tracking data during tracking and synchronization of changes made at the Publisher and at Subscribers.

### -UseInProcLoader

The **–UseInprocLoader** agent property improves performance of the initial snapshot for snapshot replication, transactional replication, and merge replication.

When you apply this property to either the Distribution Agent (for snapshot replication or transactional replication) or the Merge Agent (for merge replication), the agent will use the in-process BULK INSERT command when applying snapshot files to the Subscriber.

The **–UseInprocLoader** property cannot be used with character mode **bcp**, and it cannot be used by OLE DB or ODBC Subscribers.

**IMPORTANT** When using the **-UseInprocLoader** property, the SQL Server 2000 account under which the Subscriber is running must have read permissions on the directory where the snapshot .bcp data files are located. When the **-UseInprocLoader** property is not used, the agent (for heterogeneous Subscribers) or the ODBC driver loaded by the agent (for SQL Server 2000 Subscribers) reads from the files, so the security context of the Subscriber SQL Server 2000 account is not used.

#### To set the -UseInprocLoader property

# **Executing Scripts Before and After the Snapshot is Applied**

You can specify scripts to execute necessary procedures at the Subscriber before or after snapshot synchronization. Possible uses of executing scripts before or after synchronization could be to create logins at each Subscriber, to create userdefined data types at the Subscriber so that data with those data types can be replicated, or to update statistics after snapshot synchronization.

When a file location and script name entry is specified, the Snapshot Agent copies the script files to the current snapshot folder each time snapshot processing occurs. The Distribution Agent or Merge Agent will run the presnapshot script before any of the replicated object scripts when applying an initial synchronization. The Distribution Agent or Merge Agent will run the post-snapshot script after all the other replicated object scripts and data have been applied during an initial synchronization. The script is run by launching the **osql** utility. Test your script by running it with **osql** to be sure it executes as expected. It is recommended that you make sure that the contents of scripts that are executed before and after the snapshot is applied are repeatable and can be executed more than once. If you need to reinitialize a subscription for which the script has already been applied, the script will be applied again when the new snapshot is applied during reinitialization.

If you are compressing the snapshot file (by putting it in CAB file format), the scripts are also compressed and placed in the CAB file. After the compressed snapshot file is transferred to the Subscriber and decompressed to a working directory on the Subscriber, any scripts indicated as a pre-snapshot script will be executed. Likewise, any post-snapshot script will be decompressed and executed at the Subscriber as the last step in applying the snapshot. After initial synchronization is complete and script files run successfully, the script files are removed from the working directory on the Subscriber.

**IMPORTANT** You can execute scripts when applying the snapshot to Subscribers running SQL Server 7.0 if you use push subscriptions and the Distributor is running SQL Server 2000. You cannot execute scripts when applying the snapshot to Subscribers running SQL Server 7.0 if you use pull subscriptions or anonymous subscriptions. With pull subscriptions, the agent is created and run

on the Subscriber. Agents in SQL Server 7.0 do not have the capability of running scripts while applying the snapshot. However, if you use push subscriptions, the agent is run at the Distributor by default. If the Distributor is running SQL Server 2000, the agent running there will be able to execute the scripts before and after applying the snapshot.

#### To execute scripts before and after the snapshot is applied

# **Reinitializing Subscriptions**

When a subscription is marked for reinitialization, the snapshot schema and data are applied at the Subscriber after the next time the Snapshot Agent prepares a snapshot and Distribution Agent (for snapshot replication or transactional replication) applies it or the Merge Agent (for merge replication) runs.

For example, merge replication Subscribers update data based on the original snapshot provided to them unless you mark the subscription for reinitialization. When you mark the subscription for reinitialization, the next time the Merge Agent runs, it will apply the most recent snapshot to the Subscriber.

By default, a new snapshot is applied at the Subscriber as the first step on the next synchronization after it is marked for reinitialization. This means that any changes made at the Subscriber, but not yet synchronized with the Publisher, will be overwritten by the application of the new snapshot. Merge replication provides an option that can preserve the changes made at a Subscriber for which subscriptions are being reinitialized.

If you have a subscription to a merge publication, you can choose to have all the data changes uploaded from the Subscriber before the snapshot is reapplied. Any updates that have been made at the Subscriber since the last synchronization will be propagated to the Publisher before the snapshot is reapplied.

If you created a subscription and indicated no initial snapshot was to be applied to the Subscriber (specifying in the Create Push Subscription or Create Pull Subscription Wizard that the Subscriber already has the schema and data), and you reinitialize the subscription, the most recent snapshot will be applied to the Subscriber.

This functionality ensures that Subscribers have data and schema identical to data and schema at the Publisher. To prevent the reapplication of the snapshot to the Subscriber, drop the subscription specified with no initial snapshot synchronization and then re-create it after the reinitialization of any other Subscribers.

Reinitialization of push subscriptions is administered at the Publisher, while reinitialization of pull subscriptions is administered at the Subscriber.

**Note** At this stage, it is easy to confuse reinitialize with synchronize. Reinitialize marks the subscription. The next time the subscription is synchronized (the Distribution Agent or Merge Agent runs), the snapshot will be reapplied at the Subscriber.

#### To reinitialize a subscription

# Synchronizing Data

Synchronizing data refers to the process of data being propagated between Publisher and Subscribers after the initial snapshot has been applied at the Subscriber. When a subscription is synchronized, different processes occur depending on the type of replication you are using and whether the subscription has been marked for reinitialization.

For snapshot replication, synchronize means to reapply the snapshot at the Subscriber so that schema and data at the subscription database is consistent with the publication database. For transactional replication, synchronizing data means that data updates, inserts, deletes, and other modifications are distributed between Publisher and Subscribers. For merge replication, synchronization means that data updates made at multiple sites are merged, conflicts (if any) are detected and resolved, and data eventually converges to the same values.

The Distribution Agent and the Merge Agent move changes to data that occur at the Publisher or at Subscribers. For consistency, Microsoft® SQL Server<sup>TM</sup> 2000 replication uses the term synchronize to refer to when one of these replication agent runs.

## **Snapshot Replication Synchronization**

When a subscription to a snapshot publication is synchronized, the Distribution Agent (using distrib.exe or the Distribution ActiveX® Control) runs and the most recent snapshot will be applied at the Subscriber. If modifications to data have been made, a new snapshot will need to be generated before the new data can be applied to the Subscriber.

### **Transactional Replication Synchronization**

When a subscription to a transactional publication is synchronized, the Distribution Agent (using distrib.exe or the Distribution ActiveX Control) runs and UPDATE, INSERT and DELETE statements that have been logged at the Distributor are propagated to the Subscriber.

If the subscription has been marked for reinitialization, the Snapshot Agent and Distribution Agent must run so that a new snapshot is generated and propagated

to Subscribers.

### **Merge Replication Synchronization**

Synchronization occurs when Publishers and Subscribers in a merge replication topology reconnect using the Merge Agent (replmerg.exe or the Merge ActiveX Control) and updates are propagated between sites, and if necessary, conflicts detected and resolved. At the time of synchronization, the Merge Agent sends all changed data to the other sites. Data flows from the originator of the change to the sites that need to be updated or synchronized.

At the destination database, updates propagated from other sites are merged with existing values according to extensible and flexible conflict detection and resolution. A Merge Agent evaluates the arriving and current data values, and any conflicts between new and old values are resolved automatically based on the default resolver (a resolver you specified when creating the publication or a custom resolver).

Changed data values are replicated to other sites and converged with changes made at those sites only when synchronization occurs. Synchronizations can occur minutes, days, or even weeks apart. Data is converged and all sites eventually end up with the same data values. However, if conflicts were detected and resolved, it means that work that was committed by some users was altered or undone to resolve the conflict according to your defined policies.

## Synchronizing Schema Changes

Microsoft® SQL Server<sup>™</sup> 2000 supports limited schema changes to an existing publication database. You can add columns to and drop columns from a published table without dropping and re-creating the publications and subscriptions referencing that table.

Replication of schema changes is supported for snapshot replication, transactional replication, and merge replication. Column additions and deletions are implemented at the table level and propagated to all Subscribers that receive data from that table.

For more information, see <u>Schema Changes on Publication Databases</u>.

# **On Demand Script Execution**

On demand script execution allows you to post a SQL script, and then during the distribution or merge process, the script can be executed at all Subscribers to a specific publication.

On demand script execution is available for snapshot replication, transactional replication, and merge replication.

To specify a script to run for all Subscribers to a merge publication, execute **sp\_addrepImerge\_script**. The next time the Merge Agent runs, the script will execute at each Subscriber.

To specify a script to run for all Subscribers to a snapshot or transactional publication, execute **sp\_addscriptexec**. The next time the Distribution Agent runs, the script will execute at each Subscriber.

The following parameters need to be specified when executing either **sp\_addscriptexec** or **sp\_addrepImerge\_script**.

Parameter	Data Type	Description	
@publication	sysname	Specifies a valid publication. Required.	
		No default.	
@scriptfile	nvarchar(8000)	Specifies the UNC path where the SQL	
		script is located. Required. No default.	

On demand script execution copies the script to the replication working directory and then uses osql.exe to apply the script at the Subscriber. If there is a failure when applying the script for snapshot or transactional publications, the Distribution Agent will stop. The **sp\_addscriptexec** system stored procedure has an additional parameter, **@SkipError**, to specify whether the Distribution Agent should stop if an error is encountered (@SkipError = 0) or if the error should be logged and the Distribution Agent should continue (@SkipError = 1).

#### To synchronize a push or pull subscription

# **Scripting Replication**

You can script commonly performed replication functions such as configuring publishing and distribution, and creating or deleting publications and subscriptions. After you configure or create a replication component, you can automate the creation of a script by using SQL Server Enterprise Manager.

The script contains the Transact-SQL system stored procedures necessary to implement the replication component. Composed primarily of a series of stored procedures, you can view, execute, and/or modify and run the script using SQL Query Analyzer or **osql**.

You can choose to script creation or deletion of one or a combination of the following:

- Distributor properties
- Publications and push subscriptions
- Pull subscriptions

If you need to delete multiple push subscriptions or a mix of push and pull subscriptions, you can automate the process by creating a script to delete the publication. All subscriptions to the publication will be deleted with the publication. If you are deleting pull subscriptions, you can generate a script that deletes one or more pull subscriptions without deleting the publication.

#### Example

Because Northwind Traders has more than 50 sales representatives in different territories, it would be time-consuming to create the different subscriptions needed at each Subscriber. Instead, the replication administrator can set up the necessary merge publications (with static or dynamic partitions based on the sales representative or their territory), and then create a pull subscription, generate a script based on that pull subscription, and then run that script at multiple Subscribers to generate the necessary pull subscriptions.

To script replication

# **Schema Changes on Publication Databases**

Microsoft® SQL Server<sup>™</sup> 2000 supports common schema changes to an existing publication database. You can add columns to, and drop columns from, a published table without dropping and recreating the publications and subscriptions referencing that table.

Schema changes can be replicated during snapshot replication, transactional replication, and merge replication. Column additions and deletions are implemented at the table level and propagated to all Subscribers that receive data from that table. For snapshot replication, the schema change is propagated when a new snapshot is reapplied at the Subscriber. For transactional replication and merge replication, the schema change is propagated incrementally when the Distribution Agent or Merge Agent runs.

**IMPORTANT** Schema changes to a published table must be made only through the replication publication properties dialog box in SQL Server Enterprise Manager or through replication stored procedures. Do not make schema changes to published tables using the SQL ALTER TABLE statements in a tool such as SQL Query Analyzer or by using SQL Server Enterprise Manager visual database tools. Changes made to the schema of a published table using these tools will not be propagated to Subscribers.

It is recommended that you back up the publication database after making schema changes or using **sp\_mergecleanupmetadata**. This will ensure that you can recover the publication database in its correct state if there is a failure of the Publisher.

# **Adding Columns**

You can add a column:

• To an article in one or more publications.

Here, you add a column and apply the schema change immediately to one or more existing publications; the change is propagated to the Subscribers of those publications.

• To the underlying table, without including it in the published article.

You may want to make a schema change to the underlying table but not to the published article. For example, if you want to add a column that includes sensitive or proprietary data, this choice allows you to make a schema change without propagating the information to Subscribers. This option also lets you defer inclusion of a new column in a published article until a later date.

• To a published article, using a column that exists in an underlying table.

Whenever you add a column to a transactional publication, the appropriate ALTER TABLE statement (or **sp\_repladdcolumn** or **sp\_repldropcolumn** if the table is republished at the Subscriber) will be propagated and run at the Subscribers to complete the schema changes at the subscription databases.

Reinitialization of the subscription is necessary only when you add an existing column to a published article. When creating a new column and immediately adding it to a published article, a reinitialization is not required. This is because the Merge Agent re-executes the **sp\_repladdcolumn** stored procedure (or **sp\_repldropcolumn** for the dropping of a column), including all of its original syntax, at each affected Subscriber at the time of the next synchronization. The Distribution Agent re-executes the ALTER TABLE statement if the destination table is not republished at the Subscriber, otherwise, it re-executes the **sp\_repladdcolumn** or **sp\_repldropcolumn**, including all the original syntax, at each affected Subscriber at the time of the next synchronization.

When you add a column to the publishing table, but do not include the column in a publication, no further action is required. However, if you add the column to a publication later, subscriptions to the publication will need to be reinitialized for all types of publications. To avoid reinitializing subscriptions, add the column to the published article immediately, instead of waiting to add it to an existing article.

# **Additional Considerations**

When defining the new column through the replication user interface or through replication stored procedures, you must do one of the following:

• Allow NULL values for the new column.

• Specify a default value for the column.

# **Adding Articles to a Merge Publication**

When you add articles to a merge publication, a reinitialization of existing subscriptions is not required for the new article schema and data to be propagated to Subscribers. When adding an article to a merge publication for which there are active subscriptions, you must run the Snapshot Agent after adding the article before any Subscribers can synchronize. If the publication already has subscriptions, Subscribers will receive the schema and data for the new article based on this snapshot the next time they synchronize. The Merge Agent will then synchronize any data changes for the subscription.

When adding an article to a publication that has active subscriptions, you can filter the article using a subset filter clause without requiring that subscriptions be reinitialized. However, you cannot add any join filter clauses to a publication that has active subscriptions without also reinitializing all subscriptions to the publication.

When adding the article using **Publication Properties** in SQL Server Enterprise Manager, you will receive a message indicating that subscriptions will be prevented from synchronizing until a new snapshot has been generated for the publication. When you apply the changes, you will be advised to run the Snapshot Agent immediately.

If you are using stored procedures to add articles, you must authorize the addition of the article to a publication by setting @force\_invalidate\_snapshot=1 in **sp\_addmergearticle**. You should then run the Snapshot Agent for the publication immediately.

Whether you use **Publication Properties** in SQL Server Enterprise Manager or stored procedures, you can defer running the Snapshot Agent, but you must run it before any existing subscriptions to the changed publication can synchronize and receive the new schema and data.

# **Dropping Columns**

When dropping a column from a published article, take into consideration any constraints or properties of the column that could affect the database.

- You cannot drop columns with primary key or unique constraints, and you cannot drop **UNIQUEIDENTIFIER** (or **ROWGUIDCOL**) columns, which are used by the replication agents.
- The column to be dropped cannot be used in the filter clauses of any article of any publication in the database.
- Other types of constraints, such as foreign key and check constraints, will not prevent you from dropping a column. However, for most constraints, you are prompted with a warning message identifying the constraints on a column and requesting validation before you can drop the column. After you confirm the action, SQL Server 2000 drops all constraints on the column, and then drops the column.

**Note** Replication does not warn you of every possible dependency related to a column that is being dropped. If a column you are considering dropping is referenced by a constraint on another column, SQL Server 2000 does not inform you of the dependency and you are allowed to drop the column. Therefore, you should have a thorough understanding of the underlying database schema and use caution before dropping a published column.

### How Schema Changes are Applied

After adding or dropping a column on the publishing table in merge replication, the schema change will be propagated to Subscribers the next time the subscription is synchronized. In transactional replication, the schema change will be propagated to Subscribers the next time the Log Reader Agent and the Distribution Agent run. When adding a new article or reinitializing an existing article to a transactional publication using concurrent snapshot processing, when the Snapshot Agent starts, the Distribution Agent stops to wait for the synchronization process including the time it takes for the Snapshot Agent and Log Reader Agent to run. When the synchronization is complete, the Distribution Agent will resume.

By default, in transactional replication, the custom stored procedures will be recreated at the Subscriber automatically. The current snapshot with old schema information is invalidated by default for all types of replication. If you do not want the custom stored procedures to be re-created at the Subscriber after a schema change to a transactional publication, you should specify that when creating the publication.

**Note** When columns are added to or dropped from a publication that allows transformations on published data, the DTS packages will need to be regenerated.

# To disable automatic creation of custom stored procedures during initial synchronization (transactional replication):

- 1. In the Create Publication Wizard, on the Specify Articles page, select the articles you want to publish, and for a specific table article, click the properties (...) button associated with that table article.
- 2. On the **Commands** tab, clear the **Create the stored procedures during initial synchronization of subscriptions** check box.

# To change default properties for forcing reinitialization and invalidation of the current snapshot (transactional replication):

- Execute **sp\_repladdcolumn** or **sp\_repldropcolumn** with a value of **1** for the @force\_reinit\_subscription parameter. When set equal to 1, schema changes commands will not be propagated to Subscribers. All subscriptions affected by the schema change will be reinitialized except for nosync subscriptions, for which no action is taken.
- Execute **sp\_repladdcolumn** or **sp\_repldropcolumn** with a value of **0** for the @force\_invalidate\_snapshot parameter. When set equal to 0, current snapshot with previous schema information is still available in case it is needed. This parameter affects only publications created with the **immediate\_sync** option.

### **Applying Schema Changes to Specific Publications**

• Usually, schema changes flow to all Subscribers and republishers when included in an article. You can optionally select the publications on which to add a column, and the schema change will be propagated only

to Subscribers of those publications.

• When dropping a column, all publications and Subscribers are affected; you cannot selectively implement the change on a specific publication.

To apply schema changes on publication databases

# **Implementing Replication Over the Internet**

Replicating data over the Internet allows remote, disconnected, and anonymous users to access data when they need it using a connection to the Internet. Ways to replicate data over the Internet using Microsoft® SQL Server<sup>TM</sup> 2000 include:

- Using a Virtual Private Network (VPN) included with the Microsoft Windows NT® Server version 4.0 operating system or the Microsoft Windows® 2000 Server operating system, as well as offered by several third parties.
- Integrating replication with Microsoft Proxy Server.
- Using TCP/IP and File Transfer Protocol (FTP) to access data live on the Internet (if there is no firewall or proxy server used).

# **Publishing Data Over the Internet Using VPN**

Virtual Private Networking (VPN) technology allows users working at home, branch offices, remote clients, and other companies to connect to a corporate network over the Internet, while maintaining secure communications. Using VPNs is the most secure method for publishing data over the Internet. Users can use Windows Authentication as though they were on a Local Area Network (LAN).

VPNs include client software so that computers connect over the Internet (or in special cases, even an intranet) to software in a dedicated computer or a server. Optionally, encryption at both ends as well as user authentication methods keep data safe. The VPN connection over the Internet logically operates as a Wide Area Network (WAN) link between the sites.

A VPN connects the components of one network over another network. This is achieved by allowing the user to tunnel through the Internet or another public network (using a protocol such as Microsoft Point-to-Point Tunneling Protocol (PPTP) available with the Microsoft® Windows NT® version 4.0 or Microsoft Windows® 2000 operating system, or Layer Two Tunneling Protocol (L2TP) available with Windows 2000). This process provides the same security and features previously available only in a private network.

For the user, the intermediate routing infrastructure of the Internet is not visible, and it appears as though the data is being sent over a dedicated private link. As far as users are concerned, the VPN is a point-to-point connection between the user computer and a corporate server.

After you have your remote client configured to connect using a VPN, and the client has Internet access and is logged in to the corporate LAN, you can configure replication as though the remote client is connected directly on the LAN. For security reasons, it is possible to have different network resources available to users connected over VPN and to those connected directly on the LAN.

For more information about setting up VPN, see Virtual Private Networks in the Windows 2000 documentation.

# **Publishing Data Over the Internet Using Microsoft Proxy Server**

Integrating Microsoft® SQL Server<sup>™</sup> 2000 replication with Microsoft Proxy Server allows for replication over the Internet with security configured on the Microsoft Windows NT® version 4.0 or Microsoft Windows® 2000 Server operating systems, Proxy Server, and SQL Server 2000.

Using this approach, Proxy Server provides a connection between the Internet and the server where data is stored in SQL Server 2000. The Subscriber connects to Proxy Server over the Internet and uses a pull subscription to receive the data. Proxy Server is configured so that unauthorized Internet users cannot gain access to internal network resources, and the Subscriber must connect to a port on the Proxy Server that limits Subscriber access only to the services where permission is been granted.

For information about how to configure Microsoft Proxy Server for replication, search for the white paper titled Configuring Proxy Server for SQL Server Replication at <u>Microsoft Web site</u>.

# Publishing Data Over the Internet Using TCP/IP and FTP

Microsoft® SQL Server<sup>™</sup> 2000 can use the TCP/IP Sockets or the Multiprotocol Net-Libraries over TCP/IP to establish an ODBC connection between the Publisher or Distributor and the Subscriber. You can then configure the publication and pull subscriptions or anonymous subscriptions to access the FTP site where the data will be replicated.

Configuring your application for Internet publishing requires:

- Configuring a Publisher or Distributor to listen on TCP/IP.
- Configuring a publication to allow Subscribers to retrieve snapshots using FTP.
- Creating a subscription to use FTP for retrieving snapshots.
- Configuring a subscription agent to use TCP/IP.

# Configuring a Publisher or Distributor to Listen on TCP/IP

Before you can publish articles over the Internet, the servers where the Publisher and Distributor are located must be enabled to listen on either TCP/IP or Multiprotocol network protocol. Microsoft® SQL Server<sup>™</sup> 2000 uses the TCP/IP Sockets or the Multiprotocol Net-Libraries over TCP/IP to establish an ODBC connection between the Publisher or Distributor on one side of the Internet and the Subscriber on the other. In pull or anonymous subscriptions to transactional publications, the Distributor to synchronize. In pull or anonymous subscriptions to merge publications, the Merge Agent executes at the Subscriber and connects through the Internet to the Publisher and Distributor to synchronize.

The TCP/IP Sockets Net-Library is enabled by default during the typical SQL Server 2000 Setup, but may not have been enabled if you performed a custom installation. You can specify the FTP paths and ports as the snapshot folder location under Publication Properties so that a server already configured as an FTP site is used as the snapshot folder location. Or you can set the snapshot folder to be the FTP home directory (by default, \Microsoft SQL Server\Mssql\Repldata\Ftp) and configure the FTP home directory as an FTP site.

#### To specify FTP information

# **Configuring a Publication to Allow Subscribers to Retrieve Snapshots Using FTP**

After you have completed configuring your servers to listen on the TCP/IP or Multiprotocol connection, you are ready to configure your publications for publishing over the Internet. Any publication you create can be enabled for Internet publishing by setting the **@enabled\_for\_internet** property on the publication. Setting **@enabled\_for\_internet** to TRUE tells the Snapshot Agent to place the files associated with the initial snapshot into the FTP location specified in Publication Properties.

The Distribution Agent or Merge Agent uses FTP to download the snapshot of the schema and data to the Subscriber. The image of the entire publication flows to the destination database where it is re-created as an exact duplicate. After the snapshot files arrive at the Subscriber, the agent applies the files to the appropriate tables at the Subscriber. The agent moves through each table taking out exclusive locks on a set of rows, copying in the new rows, releasing the locks on the rows, and then repeating the process on the next blocks of rows. Because the agent locks only a small number of rows at one time, other users should be able to continue using the tables with minimal disruption.

You can configure a publication through SQL Server Enterprise Manager by selecting **Allow snapshots to be downloaded using FTP** on the **Subscriptions Option** tab of the *publication* **Properties** dialog box. You can also set the **@enabled\_for\_internet** property programmatically through the replication stored procedures that support replication over the Internet:

- sp\_addpublication
- sp\_addmergepublication
- sp\_changemergepublication
- sp\_helpmergepublication

To publish data over the Internet

# Configuring a Subscription to Use FTP to Retrieve a Snapshot

After a publication has been enabled for publishing on the Internet, you must create a pull or anonymous subscription to the publication. Subscriptions using the Internet are created the same way as other subscriptions. The only difference in subscribing to a Publication over the Internet is that you must also configure the FTP addressing properties (**FtpAddress**, **FtpPassword**, **FtpPort**, and **FtpUserName**) for the Distribution Agent or Merge Agent to use.

You can configure the FTP addressing through SQL Server Enterprise Manager on the **Snapshot Location** tab in Publication Properties.

# **Replication Between Different Versions of SQL Server**

Because you can upgrade servers running instances of Microsoft® SQL Server<sup>TM</sup> 2000 one at a time, you may have circumstances where servers in your replication topology are running different versions of SQL Server. You can replicate between different versions of SQL Server, but you are often limited to the functionality of the earliest version used.

For example, if you upgrade a Distributor to an instance of SQL Server 2000, but you have a Publisher running an instance of SQL Server version 7.0, and a Subscriber running an instance of SQL Server version 6.5, you are limited to the replication functionality of SQL Server 6.5 and unable to use features introduced in SQL Server 7.0 or SQL Server 2000. To use the new functionality, upgrade all servers used for replication to SQL Server 2000.

Features available in SQL Server 2000 are not supported with Subscribers running earlier versions of SQL Server. For example, if a merge publication contains features valid only in SQL Server 2000, and you use a push subscription to a Subscriber running SQL Server 7.0, backward compatibility is checked, and the Merge Agent will fail and display an error message indicating that the Subscriber does not meet the compatibility level. If a transactional publication contains features valid only in SQL Server 7.0, backward compatibility is not checked, and the Distribution Agent may fail with an error message not related to backward compatibility, or the Distribution Agent may succeed, but transactional processing will fail at another point.

If a publication has active subscriptions to Subscribers running earlier versions of SQL Server, and you add a feature to the publication that is valid only for SQL Server 2000, the Merge Agents or Distribution Agents for the SQL Server 7.0 subscriptions will fail. Even if the SQL Server 2000 feature is installed, the agents will not run successfully. You must delete the subscription and re-create the publication and subscription.

Following are the different combinations of SQL Server versions you can have in a replication topology. When using SQL Server 6.5, you must have SQL Server Service Pack 4 or later installed, and when using SQL Server 7.0, you must have SQL Server Service Pack 1 or later installed. SQL Server version 6.0 can be used as an ODBC Subscriber in snapshot replication or transactional replication, but it cannot be a Publisher.

This table lists the combinations for snapshot replication and transactional replication.

	Combination 1	Combination 2	Combination 3
Publisher	SQL Server 6.5	SQL Server 7.0	SQL Server 2000
Distributor	SQL Server 6.5 or	SQL Server 7.0 or	SQL Server 2000
	SQL Server 7.0	SQL Server 2000	
Subscriber	SQL Server 6.0,	SQL Server 6.0,	SQL Server 6.0,
	SQL Server 6.5,	SQL Server 6.5,	SQL Server 6.5,
	SQL Server 7.0, or	SQL Server 7.0, or	SQL Server 7.0,
	SQL Server 2000	SQL Server 2000	or SQL Server
			2000

This table lists the combinations for merge replication.

	Combination 1	Combination 2	Combination 3
Publisher	SQL Server 7.0	SQL Server 2000	SQL Server 2000
Distributor	SQL Server 2000	SQL Server 2000	SQL Server 2000
Subscriber	SQL Server 7.0	SQL Server 7.0	SQL Server 2000

#### See Also

**Replication and Upgrading** 

# SQL Server 7.0 Publisher/Distributor to SQL Server 6.5 Subscriber

You can implement replication from a Microsoft® SQL Server<sup>™</sup> version 7.0 Publisher/Distributor to a SQL Server 6.5 Subscriber using either SQL Server Enterprise Manager or stored procedures. Both creating the publication and creating the subscription are subject to certain restrictions.

## **Creating a Publication**

When you create a SQL Server 7.0 publication that has SQL Server 6.5 subscriptions, the following restrictions apply:

- Replicated tables cannot contain any Unicode or **uniqueidentifier** data types.
- Replicated tables cannot have names longer than 30 characters.
- The name of the custom stored procedure cannot be longer than 21 characters. When creating a transactional publication that has only SQL Server 6.5 Subscribers, the Create Publication Wizard defaults to using custom stored procedures to apply transactions at the Subscriber. This configuration is applied to each article in a publication. The name of the insert/update/delete stored procedures to be created and called at the Subscriber defaults to the table name, with a prefix of **sp\_Msins\_**, **sp\_Msupd\_**, or **sp\_Msdel\_**. If a published table name is longer than 21 characters, this prefix causes the custom stored procedure name to be too long to be created on a SQL Server 6.5 Subscriber. The work around is to change the default custom stored procedure names so that they are 30 characters or less. This is done by going to the **Commands** tab on the property page for each article in the publication and changing the name of the custom stored procedure. Alternatively, you can choose not to use custom stored procedures at the Subscriber, or set up subscriptions using stored procedures (**sp\_addarticle**), where it is more efficient to override the defaults.

# **Creating a Subscription**

Before creating a subscription from a SQL Server 7.0 Publisher to a SQL Server 6.5 Subscriber, you must run Replp70.sql at the Subscriber, and then execute **sp\_addpublisher70** at the Subscriber. **sp\_addpublisher70** registers the SQL Server 7.0 Publisher at the SQL Server 6.5 Subscriber (a necessary step for SQL 6.*x* replication). Replp70.sql is located in the \Microsoft SQL Server\Mssql\Install directory. **sp\_addpublisher70** takes two parameters: **@publisher** and **@dist\_account**. **@publisher** is the name of the SQL Server 7.0 Publisher. **@dist\_account** is the domain account name that SQL Server Agent runs under at the SQL Server 7.0 Distributor. For example, the syntax may look as follows:

### EXEC sp\_addpublisher70 'PUBSERV', 'REDMOND\repladmin'

It is also necessary to enable a SQL Server 6.5 subscribing database for replication. This can be done through the SQL Server Enterprise Manager in SQL Server 6.5, or by executing:

### EXEC sp\_dboption <dbname>, 'subscribed', true

SQL Server 7.0 replication supports push subscriptions to SQL Server 6.5 servers, but does not support pull subscriptions from SQL Server Enterprise Manager 6.5. To configure a push subscription to a SQL Server 6.5 Subscriber, you must first register the Subscriber at the Publisher. You can do this using the SQL Server Enterprise Manager in SQL Server 7.0, or executing **sp\_addsubscriber**.

**Note** Subscribers running SQL Server 6.5 do not support nullable **bit** columns, so NULL values in **bit** columns published by a Publisher running SQL Server 7.0 or SQL Server 2000 cannot be represented at the Subscriber. If you have Subscribers running SQL Server 6.5 and you need to use nullable **bit** columns, use custom stored procedures to change incoming NULL values to 0.

# SQL Server 7.0 Publisher/Distributor to SQL Server 6.0 Subscriber

Replication from a Microsoft® SQL Server<sup>™</sup> version 7.0 Publisher/Distributor to a SQL Server 6.0 Subscriber is implemented in much the same way as to a SQL Server 6.5 Subscriber. The only difference is that the SQL Server 6.0 Subscriber must be configured as an ODBC data source rather than as a native SQL Server Subscriber.

# SQL 6.5 Publisher/Distributor to SQL Server 7.0 Subscriber

Microsoft® SQL Server<sup>™</sup> version 7.0 can act as a Subscriber to a SQL Server 6.5 Publisher. You must add the SQL Server 7.0 Server as a Subscriber at the SQL Server 6.5 publishing server.

This action will also add an entry in the console tree of SQL Server Enterprise Manager. This is for replication purposes only, and you cannot administer this server using SQL Server Enterprise Manager in SQL Server 6.5. If you click this server in the console tree, the following warning message will be returned (and can be ignored for replication purposes):

A connection cannot be established to <SERVER> - (SQL Server) You

You cannot use the replication topology or pull subscription features of SQL Server Enterprise Manager in SQL Server 6.5 to manage the SQL Server 7.0 Subscriber.

If you did not upgrade the ODBC driver at the Distributor (thus using the SQL Server 6.5 ODBC driver), you may encounter a login failure when the Distribution Agent connects to the SQL Server 7.0 Subscriber. You should upgrade the ODBC driver to successfully start the SQL Server 7.0 Subscriber.

To publish from a Publisher running SQL Server 6.5 to a Subscriber running SQL Server 7.0, add the SQL Server Authentication login, **repl\_publisher**, with a blank password at the SQL Server 7.0 Subscriber.

# SQL Server 6.5 Publisher to SQL Server 7.0 Distributor

It is possible to configure a Microsoft® SQL Server<sup>™</sup> version 6.5 Publisher to use a SQL Server 7.0 installation as a remote Distributor. This topology provides a way to stagger the upgrade of SQL Server installations participating in a replication application. In addition, when using a SQL Server 7.0 Distributor to service a SQL Server 6.5 Publisher, you can use the monitoring capability of SQL Server 7.0 replication.

You can configure a SQL Server 6.5 Publisher to use a SQL Server 7.0 Distributor by registering the SQL Server 6.5 Server in SQL Server Enterprise Manager 7.0 and configuring it as a Publisher.

# **Replication with SQL Server 2000 Windows CE Edition**

Using Microsoft® SQL Server<sup>™</sup> 2000 and merge replication, you can publish data to mobile devices running SQL Server 2000 Windows CE Edition (SQL Server CE). Merge replication is suited for replication with mobile, disconnected Subscribers because it allows updates to be made at the Subscriber while the Subscriber is disconnected from the network and the Publisher. Later, when the device is reconnected, the changes made at the Subscriber can be merged with other changes made at the Publisher and at other Subscribers.

Replication with SQL Server CE is possible with merge publications using anonymous subscriptions. Administration of the subscription is conducted at the Subscriber, and information about the Subscriber running SQL Server CE and the subscription is not stored at the Publisher.

## How Replication to SQL Server CE Works

Publishing to Subscribers running SQL Server CE is similar to publishing to other types of Subscribers using anonymous subscriptions.

Create a merge publication, and on the Specify Subscriber Types page of the Create Publication Wizard, select **Servers running SQL Server CE** as a type of Subscriber that can subscribe to this publication. If you select this option, anonymous subscriptions will be enabled for the publication automatically.

A SQL Server CE application can subscribe to the publication using the SQL Server CE Replication Object. When the subscription is created, the initial snapshot is applied to create the subscription database on the device running SQL Server CE.

Users can modify data in the subscription database online or offline. When reconnected, the data modifications made at the Subscriber are sent to the Publisher and merged with changes made at the Publisher and at other Subscribers. Changes made at the Publisher or propagated to the Publisher since the last synchronization are sent to the Subscriber.

The SQL Server CE Replication Object, within SQL Server CE, controls the

execution of the SQL Server Merge Agent to complete synchronization. If conflicts occur because of changes to the same data, it will resolve the conflicts using the conflict resolvers you chose when creating the publication.

For more information, see the SQL Server CE documentation.

## See Also

Anonymous Subscriptions

Merge Replication

# **Replication Options**

Replication options allow you to configure replication in a manner best suited to your application and environment.

Option	Type of Replication	Benefits
<b>Option</b> Filtering Published Data	Replication Snapshot Replication Transactional Replication Merge Replication	<ul> <li>Benefits</li> <li>Filters allow you to create vertical and/or horizontal partitions of data that can be published as part of replication. By distributing partitions of data to different Subscribers, you can: <ul> <li>Minimize the amount of data sent over the network.</li> </ul> </li> <li>Reduce the amount of storage space required at the Subscriber.</li> <li>Customize publications and applications based on individual Subscriber requirements.</li> <li>Reduce conflicts because the</li> </ul>
		different data partitions can be sent to different Subscribers.
Updatable Subscriptions (Immediate Updating, Queued Updating)	Snapshot Replication Transactional Replication	Immediate updating and queued updating options allow users to update data at the Subscriber and either propagate those updates to the Publisher immediately or store the updates in a queue. Updatable subscriptions are best for replication topologies where replicated

		data is mostly read, and occasionally updated at the Subscriber when Publisher, Distributor, and Subscriber are connected most of the time and when conflicts caused by multiple users updating the same data are infrequent.
Updatable Subscriptions (Merge Replication)	Merge Replication	Merge replication allows users to update data at the Subscriber or Publisher and synchronize changes continuously, on- demand, or at scheduled intervals. Merge replication is well suited for topologies where replicated data is frequently updated at the Subscriber even when the Subscriber is disconnected from the Publisher. Conflicts caused by multiple users updating the same data should be infrequent, but merge replication provides a rich set of options for handling conflicts that do occur. For more information, see <u>Merge Replication</u> .
Transforming Published Data	Snapshot Replication Transactional Replication	<ul> <li>You can leverage the data movement, transformation mapping and filtering capabilities of Data Transformation</li> <li>Services (DTS) during replication. With transformable subscriptions, you can: <ul> <li>Create custom partitions for snapshot and transactional publications.</li> </ul> </li> <li>Transform the data as it is being published with data type mappings (for example, integer to real data type), column manipulations (for example,</li> </ul>

		concatenating first name and last name columns into one), string manipulations, and functions.
Alternate Synchronization Partners	Merge Replication	Alternate synchronization partners allow merge Subscribers to synchronize data with servers other than the Publisher at which the subscription originated. This allows the Subscriber to synchronize data when the original Publisher is unavailable, and is also useful for mobile Subscribers that may have access to a faster or more reliable network connection with an alternate server.
Optimizing Synchronization	Merge Replication	By optimizing synchronization during merge replication, you can store more information at the Publisher instead of transferring that information over the network to the Subscriber. This improves synchronization performance over a slow network connection, but requires additional storage at the Publisher.

## **Filtering Published Data**

Horizontal, vertical, dynamic, and join filters enable you to create partitions of data to be published. By filtering published data, you can:

- Minimize the amount of data sent over the network.
- Reduce the amount of storage space required at the Subscriber.
- Customize publications and applications based on individual Subscriber requirements.
- Avoid or reduce conflicts because the different data partitions can be sent to different Subscribers (no two Subscribers will be updating the same data values).

Row and column filters can be used with snapshot, transactional, and merge publications. Row filters use the WHERE clause of an SQL statement and restrict the rows included in a publication based on specific criteria. Column filters restrict the columns that are included in a publication.

Dynamic and join filters extend the capabilities of merge replication. Dynamic filters are row filters that use a function to retrieve a value from the Subscriber and filter data based on that value. The filter is defined once for a publication, but the qualifying result set can be different for each Subscriber and allows the user at a Subscriber to receive only the subset of data customized for their needs.

Join filters extend a row filter from one published table to another. A join filter defines a relationship between two tables that will be enforced during the merge process; it is similar to specifying a join between two tables.

## **Row Filters**

Using row filters, you can specify a subset of rows from a table to be published. Row filters can be used when only specific rows need to be propagated to Subscribers, to eliminate rows that users do not need to see (such as rows that contain sensitive or confidential information), or to create different partitions of data that are sent to different Subscribers. For those applications that can, publishing different partitions of data to different Subscribers can also help avoid conflicts that would otherwise be caused by multiple Subscribers updating the same data values.

Row filtering is convenient because it can be applied to existing applications where a site-specific attribute is present to filter on either in the table to be published or in one of its related tables.

In this diagram, the published table is filtered so that only rows 2, 3, and 6 are included in the publication sent to the Subscriber.

Row filters are available with snapshot replication, transactional replication, and merge replication. Row filters in transactional publications may add significant overhead because the article filter clause is evaluated for each log row written for a published table to determine whether it should be marked for replication. Row filters in transactional publications should be avoided where each site can support the full data load, the overall data set is reasonably small, and the number of insert, update, and delete transactions per day is low.

Row filters in snapshot replication and transactional replication are static and the WHERE clause criteria you set in the Create Publication Wizard or the publication properties dialog box stays the same until you modify it. If you had two Subscribers that require different rows of data from the publishing table, you would need two different publications each with a different row filter to retrieve the correct rows for each Subscriber.

Although you can put a subquery into a row filter, it is not a join filter. If you update a row in a table referenced by a subquery, the query will not be re-evaluated and the row will not be propagated as part of replication. Replication join filters exist only for merge replication. For more information, see Join

#### Filters.

An alternative to creating multiple publications is to use a dynamic filter for merge replication or create a transformable subscription with a custom filter for snapshot replication or transactional replication that dynamically creates data partitions based on information from individual Subscribers. For more information, see <u>Dynamic Filters</u> and <u>Transforming Published Data</u>.

## Example

Because the sales representatives need to update the data frequently and make updates while connected and while disconnected from the Publisher, the replication administrator at Northwind decides to use merge replication and create different publications with row filters based on region. The publication would include data from the customers, orders, and order details table. For example, one of the publications would be restricted for the Northwest region of the United States. In the **Specify** Filter dialog box in the Create Publication Wizard, the WHERE clause would read:

SELECT <published\_columns> FROM [dbo].[Customers] WHERE Region = 'WA'

Because data partitions based on region will be sent to Subscribers that have exclusive, logical ownership of each region, conflicts that could occur when multiple Subscribers update the same data will be avoided. However, conflicts may still occur if the Publisher and Subscriber update the same data. For more information, see <u>Merge Replication</u> and <u>Merge Replication Conflict Detection</u> and <u>Resolution</u>.

An alternate, often preferable approach to this type of situation is to use a dynamic filter for a merge publication or a transformable subscription for a snapshot or transactional publication. For more information, see <u>Dynamic Filters</u> and <u>Transforming Published Data</u>.

#### To filter publications horizontally

# **Column Filters**

Column filters restrict the columns to be included as part of a snapshot, transactional, or merge publication. Column filters can reduce the time it takes to propagate data updates to Subscribers, reduce the storage space needed at the Subscriber, and limit the data in a publication to data that is needed by individual Subscribers.

This illustration shows a publication that has a column filter to restrict all columns except columns A, B, and D.

You can also use row and column filtering together, as illustrated here.

When you add a column to a vertical partition, the table structure changes and any INSERT statements on the publishing table will require column lists.

Columns that cannot be vertically filtered from a publication are:

- Columns with primary key constraints.
- Non-null columns without a default.
- Columns included in a unique index.
- The **ROWGUID** column for merge publications and the **ROWGUID** column for snapshot or transactional publications that allow immediate updating subscriptions.

For snapshot replication and transactional replication, you can use transformable subscriptions to create custom filters that produce different vertical partitions for different Subscribers using one publication. For more information, see <u>Using</u> <u>Transformable Subscriptions to Create Custom Data Partitions</u>.

**Note** If the snapshot or transactional publication allows updatable subscriptions

and the publication has a column filter, you cannot filter non-nullable columns without defaults from the publication.

## Example

Currently, the sales information is distributed to all sales representatives, but Northwind managers do not want the sales representatives to see the commission amounts paid. The replication administrator can use a column filter to exclude the COMMISSION column from the publication.

#### To filter publications vertically

# **Dynamic Filters**

Dynamic filters allow you to create a merge publication and then filter data from the publishing table providing different partitions of data to different Subscribers. Benefits of using dynamic filters in merge publications are:

- Fewer publications stored at the Publisher. This reduces the overhead of administering multiple publications.
- Employing user-defined functions in the dynamic filter enables you to filter criteria.
- The Subscriber receives only the information needed because data is filtered based on the connection properties of the Merge Agent for the subscription.

In the dynamic filter, you specify a Microsoft® SQL Server<sup>™</sup> 2000 function or a user-defined function that is evaluated differently for each Subscriber based on the connection properties of the Merge Agent when the merge process is replicating data between the Subscriber and Publisher. The most common system functions used for this purpose are **SUSER\_SNAME()** and **HOST\_NAME()**. You can use a user-defined function in a dynamic filter, but unless the user-defined function definition includes **SUSER\_SNAME()**, **HOST\_NAME()**, or the user-defined function evaluates one of these system functions in the filter criteria (such as **MyUDF(SUSER\_SNAME())**, the user-defined function will be static.

Dynamic filters are row filters (restricting rows of data) and are created on a single table basis (they do not cross or join tables). You can, however, use both dynamic filters and join filters in the same publication and on the same published tables.

Dynamic filters are available only with merge replication, so when using them, you should consider employing a dynamic snapshot as well. By default, dynamic filtered publications rely on INSERTs from the Publisher to apply data to the Subscriber as part of the initial snapshot. Dynamic snapshots provide the

performance advantage of using SQL bulk copy program (**bcp**) files to apply data to a specific Subscriber when applying the initial snapshot while using dynamic filters. For more information, see <u>Dynamic Snapshots</u>.

If you are using snapshot replication or transactional replication, you can create custom filters using transformable subscriptions, which will filter data based on individual Subscriber requirements. For more information, see <u>Transforming Published Data</u>.

## Example

Instead of creating a separate publication for each sales representative, Northwind will use the **SUSER\_SNAME()** function in the dynamic filter on the **CUSTOMERS** table article to return the user ID of the sales representative assigned to each customer and filter published data based on it. The **SALES\_REP** column could be added to the **CUSTOMERS** table to identify the sales representative responsible for servicing each customer.

The **CUSTOMERS** table at the Publisher.

CustomerID	CompanyName	SALES_REP
GREAL	Great Lakes Food Market	WestRegion\Robert King
RATTC	Rattlesnake Canyon Grocery	Janet Leverling

The row filter for the CUSTOMERS article in the Northwind merge publication is:

## WHERE SALES\_REP = SUSER\_SNAME()

If the merge process is initiated using the WestRegion\Robert King integrated security account, the **SUSER\_SNAME()** function evaluates to this account in the dynamic filter only when the Merge Agent is run by the user WestRegion\Robert King. As a result, Robert King receives only data regarding the customers for which he is assigned as a sales representative.

The **CUSTOMERS** table at the Subscriber after using the dynamic filter when publishing data.

CUST_ID	CUSTNAME	SALES_REP
GREAL	Great Lakes Food Market	West Region\Robert
		King

The behavior of dynamic filters is different depending on whether you use Windows Authentication or SQL Server Authentication. With SQL Server Authentication, the **–PublisherLogin** parameter specified in the Merge Agent command line (or PublisherLogin property in the SQL Merge ActiveX® Control) is the key property returned when using **SUSER\_SNAME()** in a dynamic filter.

With Windows Authentication, SQL Server Agent initiates the merge process and the **SUSER\_SNAME()** function in SQL Server 2000 returns the account under which the SQLServerAgent service is running. This may be different from the Microsoft Windows NT® security account of the user. If the merge process is initiated using the Microsoft ActiveX control or by calling Replmerge.exe independently of SQL Server Agent, the **SUSER\_SNAME()** function in SQL Server 2000 returns the login account of the user.

When using dynamic filters, the filtering logic expression is evaluated within the context of the merge connection to the Publisher, not the connection to the Subscriber. If the merge process uses the SQL Server 2000 login Janet Leverling to connect to the Publisher, and the **sa** login to connect to the Subscriber, the **SUSER\_SNAME()** function will evaluate to Janet Leverling in the filtering logic.

The **CUSTOMERS** table at the Subscriber (using the dynamic filter).

CUST_ID	CUSTNAME	SALES_REP
RATTC	Rattlesnake Canyon Grocery	Janet Leverling

# **Dynamic Snapshots**

Dynamic snapshots provide a performance advantage when applying the snapshot of a merge publication with dynamic filters. Performance is improved by using Microsoft® SQL Server<sup>™</sup> 2000 bulk copy files to apply data to a specific Subscriber instead of a series of INSERT statements.

Generating a dynamic snapshot for a subscription also allows the flexibility of saving and transferring the snapshot on removable media (such as a CD-ROM) and applying the snapshot at the Subscriber from the media rather than applying the initial snapshot over a slow network connection.

## **How Dynamic Snapshots Work**

When dynamic filters are used in merge publications, data is filtered from the publishing table based on the connection properties of the Merge Agent for the publication during the merge process. By default, dynamically filtered publications rely on INSERTs from the Publisher to apply data to the Subscriber as part of the initial snapshot. This can be a lengthy and resource-intensive process because the Merge Agent will have to determine row-by-row which data to include in the snapshot based upon the dynamic filter criteria.

Dynamic snapshots provide the performance advantage of using SQL bulk copy program (**bcp**) files to apply data to a specific Subscriber when applying the initial snapshot while using dynamic filters. When you create a dynamic snapshot, you pre-generate a snapshot that will be customized to a specified Subscriber. Because the data values are already copied and extracted, applying the snapshot will be just as fast as applying snapshots without dynamic filters. There is, however, additional time and space required when generating and storing the dynamic snapshot.

Although it takes longer to prepare a dynamic snapshot (you will need to generate two snapshots), the process of applying the snapshot at Subscribers is faster than applying a standard snapshot for a dynamically filtered merge publication. You will need to generate a standard snapshot first, before the dynamic snapshot is created by filtering the standard snapshot.

Dynamic snapshots can be implemented using SQL Server Enterprise Manager and the Create Publication and Create Dynamic Snapshot Job wizards, Transact-SQL system stored procedures and scripts, Microsoft ActiveX® controls or SQL-DMO.

## **Dynamic Snapshot Considerations**

When planning for dynamically filtered merge publications and dynamic snapshots, consider:

- Dynamic snapshots can be used with all types of subscriptions. You can generate the dynamic snapshot using the Create Dynamic Snapshot Job Wizard and/or running the Snapshot Agent with the appropriate parameters. Applying a dynamic snapshot is done using the Merge Agent or Merge ActiveX Control and setting the **DynamicSnapshotLocation** properties.
- You can use the **-DynamicSnapshotLocation** command line parameter for the Merge Agent or the **DynamicSnapshotLocation** property in the Merge ActiveX Control to apply a pre-generated dynamic snapshot.
- Dynamic filters and dynamic snapshot are available only with merge replication.
- To generate a dynamic snapshot, the publication must be enabled for dynamic filters and a standard snapshot must be generated.
- Dynamic snapshot files will also be compressed if the standard snapshot is compressed. To compress a standard snapshot, and therefore the dynamic snapshot, open publication properties, and on the **Snapshot Location tab**, select **Generate snapshots in the following location**, specify a snapshot location in the text box, and then select **Compress snapshot files in this location**.
- The login specified as the value of the Publisher login must be in the

Publication Access List (PAL) or be a member of the publication database sysadmin role or **db\_owner** group. This login can be specified in the Create Dynamic Snapshot Job Wizard or by using the **- DynamicFilterLogin** parameter of the Snapshot Agent.

- Because SQL Server adds and drops temporary logins in the Snapshot Agent, the Publisher login of the Snapshot Agent must be a member of the **securityadmin** server role and be a member of the **db\_owner** group on the publication database to be able to generate dynamic snapshots.
- Dynamic filter logins specified for dynamic snapshot generation must be members of the corresponding publication access list (PAL).
- SQL Server on the Publisher must be running under mixed security mode.
- Changing publication properties without regenerating a standard snapshot for a dynamically filtered publication will invalidate all subsequent dynamic snapshots that are generated.

For example, if you have a sales representative who receives customer management information based on a SalesPersonLogin, which is really the integrated login used at the Subscriber to connect to the Publisher. In this example, there are two users, DOMAIN\JohnSmith and DOMAIN\BobJohnson. The administrator of the Publication can specify the **-DynamicFilterLogin** property of the Snapshot Agent to be DOMAIN\JohnSmith and generate a dynamic snapshot for the user named John Smith. Similarly, they can specify the **-DynamicFilterLogin** property to be DOMAIN\BobJohnson and generate the snapshot for the user named Bob Johnson. However, the dynamic filter must be expressed using the **SUSER\_SNAME()** function for this to occur.

If the dynamic filter used previously was SalesPersonLogin = SUSER\_SNAME(), the dynamic filter must now be SalesPersonLogin = **SUSER\_SNAME()** to use the dynamic snapshot functionality.

• Do not use parameters in the **SUSER\_SNAME()** system function used

with dynamic snapshots, such as 'SUSER\_SNAME(SID)'.

- Functions that implicitly rely on SUSER\_SNAME() or the current user, such as USER\_NAME(), CURRENT\_USER(), SYSTEM\_USER(), USER\_ID(), or SUSER\_SID() will not work as expected and should not be used with dynamic snapshots (use SUSER\_SNAME() or HOST\_NAME() instead).
- You can use user-defined functions in a dynamic filter; however, if the user-defined filter evaluates to the same value for all Subscribers, it is a type of static filter, and there is no need to use dynamic snapshots because all Subscribers would receive the same snapshot of data.
- You can use the SUSER\_SNAME() system function nested in a userdefined function in the filter criteria for a dynamic filter, and you can use a dynamic snapshot (for example, MyUDF(SUSER\_SNAME()) where the MyUDF user-defined function evaluates the SUSER\_SNAME() system function). The system function must be visible in the dynamic filter criteria. If the system function exists in the definition of the user-defined function, and you enter only the userdefined function in the dynamic filter, you will not be able to use a dynamic snapshot.

#### To create a dynamic snapshot

## Validate Subscriber Information

With merge replication dynamic filters, you use a function that references Subscriber information. Microsoft® SQL Server<sup>™</sup> 2000 validates Subscriber information based on that function before each merge. This ensures that information is partitioned consistently with each merge.

For example, when a publication is dynamically filtered using the function **SUSER\_SNAME()**, the Merge Agent applies the initial snapshot to each Subscriber based on data that is valid for the **SUSER\_SNAME()** expression.

When the Subscriber reconnects to the Publisher for the next synchronization, the Merge Agent validates the information at the Subscriber and ensures that the same partitions are synchronized as was originally sent as part of the initial snapshot. If the Merge Agent detects that the filtering expression returns a different value, the merge fails. Because the value of the function used in the dynamic filter has changed, the subscription at the Subscriber may need to be reinitialized or the original login or host\_name value must be used before synchronization will be permitted. This will prevent problems that may arise if the merge settings of a Subscriber are changed.

You can choose to create the dynamic filter and then validate Subscriber information while creating a publication using the Create Publication Wizard or after the publication is created and enabled for dynamic filters by using the publication properties.

## Example

If a laptop used by Northwind sales representative Bob Jones is the Subscriber to the merge publication with the dynamic filter, each time Bob Jones logs in and synchronizes data with the Publisher, he will receive data based only on the user ID he enters when logging on to his laptop. Because he is a sales representative receiving customer and orders information, he receives data only for the customers he services.

## To validate Subscriber information using the Create Publication Wizard

## **Join Filters**

Join filters allow cross table relationships to be used in merge replication filters when the filter of one table is based on another table in the publication. A join filter defines a relationship between two tables that will be enforced during the merge process; it is similar to specifying a join between two tables. The join filter names two articles, and specifies the join condition to represent the relationship between the two tables in the articles. The join condition is usually in the form:

## ARTICLE1\_TABLE.COLUMN = ARTICLE2\_TABLE.COLUMN

Join filters are typically used in conjunction with row filters and allow the merge process to maintain the referential integrity between the two tables. If a table published with a row filter is referenced by a foreign key in another published table, the foreign key table's article must have a join filter to represent the referential dependency on the primary key table article.

SQL Server Enterprise Manager uses this rule when creating a publication to suggest the join filter logic automatically for the foreign key table based in the foreign key reference. For this reason and also for ease of use, it is recommended that you declare the proper primary key to foreign key relationships and then let the join filters be generated automatically when you create a publication using the Create Publication Wizard.

**Note** The syntax for creating FOREIGN KEY constraints with CREATE TABLE or ALTER TABLE allows the NOT FOR REPLICATION option. When this option is set, Microsoft® SQL Server<sup>™</sup> 2000 assumes that the reference was validated when the user made the data change; therefore, SQL Server 2000 does not perform the extra processing steps to verify the reference when the merge process synchronizes the data. If this option is used, a merge filter must be defined to avoid invalid foreign key rows at the subscriber.

Join filters are not limited strictly to primary key/foreign key relationships. The join filter can be based on any comparison logic that associates the data in the two article tables, but the logic should use indexed columns if possible for best performance.

The merge process has special performance optimizations depending on whether the join condition is based on a unique column, as is the case when the join filter represents a foreign key relationship. If the join condition is based on a unique column, the **join\_unique\_key** property should be set for the article for best performance.

Although you can put a subquery into a row filter, it is not a join filter. If you update a row in a table referenced by a subquery, the query will not be re-evaluated and the row will not be propagated as part of replication. Replication join filters exist only for merge replication.

**WARNING** Join filters with several tables (such as dozens or hundreds of tables) will seriously impact performance during merge processing. It is recommended that if you are generating join filters of five or more tables that you consider other solutions. Another strategy might be to not filter tables which are primarily lookup tables, smaller tables, and tables that are not subject to change. Make those tables part of the publication in their entirety. It is recommended that you use join filters only between tables for which it is important they carefully partition among Subscribers.

#### Example

CustomerID	CustomerName	Status
ALFKI	Alfreds Futterkiste	Active
ANATR	Ana Trujillo Emparedados	Inactive
ANTON	Antonio Moreno Taqueria	Active

#### The CUSTOMERS table

#### The ORDERS table

OrderID	CustomerID	OrderDate
10643	ALFKI	1997-08-25
11077	RATTC	1998-05-06
10926	ANATR	1998-03-04
11000	RATTC	1998-04-06
11010	REGGC	1998-04-09

10569	RATTC	1997-06-16

The join filter for these tables would be defined for the ORDERS article. The join article would be the CUSTOMERS article, and the join filter clause would be:

## CUSTOMERS.CUSTOMERID=ORDERS.CUSTOMERID

If the CUSTOMERS table article in the publication has a row filter clause of **Status** = 'Active', the merge process publishes only the Alfreds Futterkiste and Antonio Moreno Taqueria customer data to the Subscriber.

If no join filter is present to restrict the ORDERS table data to the filtered customers, the merge process fails with a primary key violation for the **CustomerID** column in the ORDERS table. This is because the process attempts to insert the inactive customers' transaction rows that have no valid **CustomerID** in the CUSTOMERS table at the Subscriber.

OrderID	CustomerID	OrderDate
10643	ALFKI	1997-08-25
11077*	RATTC	1998-05-06
10926	ANATR	1998-03-04
11000*	RATTC	1998-04-06
11010*	REGGC	1998-04-09
10569*	RATTC	1997-06-16

The ORDERS table data with no join filter applied to the Subscriber.

\*These rows violate the foreign key on the CustomerID column at the Subscriber.

To avoid this problem, add a join filter to the ORDERS table that represents the referential dependence on the CUSTOMERS table. The merge process replicates only the ORDERS data for the active customers.

The CUSTOMERS table at the Publisher.

CustomerID	CustomerName	Status
ALFKI	Alfreds Futterkiste	Active
ANATR	Ana Trujillo Emparedados	Inactive

	•	
ANTON	Antonio Moreno Taqueria	Active

The ORDERS table at the Publisher.

OrderID	CustomerID	OrderDate
10643	ALFKI	1997-08-25
11077	RATTC	1998-05-06
10926	ANATR	1998-03-04
11000	RATTC	1998-04-06
11010	REGGC	1998-04-09
10569	RATTC	1997-06-16

The CUSTOMERS table at the Subscriber with a row filter clause for Active customers.

CustomerID	CustomerName	Status
ALFKI	Alfreds Futterkiste	Active
ANTON	Antonio Moreno Taqueria	Active

The ORDERS table at the Subscriber with a join filter to Active customers.

OrderID	CustomerID	OrderDate
10643	ALFKI	1997-08-25
10926	ANTON	1998-03-04

## **User-Defined Functions and Static Filters**

User-defined functions are subroutines composed of encapsulated sets of Transact-SQL logic. You can use them in row static or dynamic filters.

By accessing user-defined functions, you increase your filtering capability because you can create filters based on frequently performed logic, table-driven business rules, or any set of complex instructions that returns a value.

You can specify user-defined functions that return a scalar value (such as **int**, **char**, or **decimal**) when filtering horizontally (row filtering replicates a subset of the rows in a table) in snapshot replication, transactional replication, or merge replication.

To create a user-defined function for use as a publication filter, use the **CREATE FUNCTION** command on the database containing the data you want to publish, and build a function with Transact-SQL. You can then use the function in a filter when you create a new publication using the Create Publication Wizard or when configuring an existing publication using the publication properties dialog box. If the publication has subscribers, you must drop all subscriptions to the publication before you can create or modify row filters. You do not have to replicate the function to use it as part of a filter in a publication.

## Example

```
CREATE FUNCTION fn_wknum(@Parm datetime)
RETURNS int
AS
BEGIN
DECLARE @ReturnVar int
SELECT @ReturnVar = CAST((DATEPART(dy,@Parm) + DATEPA
RETURN @ReturnVar
END
```

To implement the **fn\_wknum** example in a publication based on the **Northwind** database, create the function on that database. Start the Create Publication Wizard, select **Define Data Filters**, and then in the **Filter Table Rows** dialog

box, click the properties button (...) for the **Orders** article.

In the **Specify Filter** dialog box, you can complete the WHERE clause to filter for the first 12 weeks of any year based on the **orderdate** column:

SELECT \* FROM [dbo].[Orders] WHERE dbo.fn\_wknum(orderdate)

To filter with a user-defined function using the Create Publication Wizard

## **User-Defined Functions and Dynamic Filters**

You can gain greater flexibility when filtering merge publications and improve dynamic filtering performance by invoking user-defined functions to determine the different partitions of data. Dynamic filters allow you to define different partitions of one publication replicated to different Subscribers.

Dynamic filters can use an intrinsic function (such as **SUSER\_SNAME()**) that is evaluated based on each Subscriber to a publication. Different partitions of data are replicated to different Subscribers based on the value returned by the function.

User-defined functions expand on this capability by allowing you to define the function used in the dynamic filter. This enhancement allows you to define business rules, scalar, or table values to use when partitioning published data based in a dynamic filter.

For example, in a sales environment, each customer is assigned a region code representing the region where they are located. Sales representatives in the Northwest need to see orders only for the customers in their region. To publish only the orders placed in the Northwest to the Subscribers in that region, you could write a user-defined function that retrieved the region code from the Subscriber and then use that code to partition the data dynamically depending on which Subscriber is receiving the data.

For more information, see **Dynamic Filters**.

#### See Also

**CREATE FUNCTION** 

**User-Defined Functions and Static Filters** 

# **Updatable Subscriptions**

With snapshot replication or transactional replication, replicated data is by default read only; however, you have the ability to modify replicated data at the Subscriber by using updatable subscriptions. If you need to modify data at the Subscriber using snapshot or transactional replication, you can choose one of the following options depending on your requirements.

Updatable Subscription	Requirements
Immediate Updating	Publisher and Subscriber must be
	connected to update data at the Subscriber.
Queued Updating	Publisher and Subscriber do not have to be
	connected to update data at the Subscriber.
	Updates can be made while offline.
Immediate Updating with	Publisher and Subscriber are connected
Queued Updating as a Failover	most of the time, but you may
	occasionally need to make updates offline.

# **Immediate Updating**

Immediate updating allows snapshot replication and transactional replication Subscribers to update the replicated data at the Subscriber and send those changes back to the Publisher and to other Subscribers. Immediate updating benefits applications in which snapshot or transactional publications are preferred but occasional updates need to be made at the Subscriber. If using immediate updating, the Publisher and Subscribers must be available and connected.

The immediate-updating option:

- Ensures that there are no conflicts. A Subscriber can perform inserts, updates, and deletes on replicated data only if it can perform a two-phase commit protocol (2PC) transaction with the Publisher. The Publisher must accept every update before it is made at the Subscriber. Conflicts do not occur because they are detected before a transaction is committed.
- Initiates two-phase commit (2PC) automatically.
- Replicates the committed update down to all other Subscribers through the standard snapshot replication or transactional replication mechanism.
- Lets the Subscriber continue working without waiting for the successful update to propagate to other Subscribers.
- Forestalls the requirement for the updating Subscriber to have a distribution database or log reader and get involved in the administrative issues of replication publishing.
- Has fewer failure points with every site than with full 2PC involving every Subscriber, and it is also more scalable.

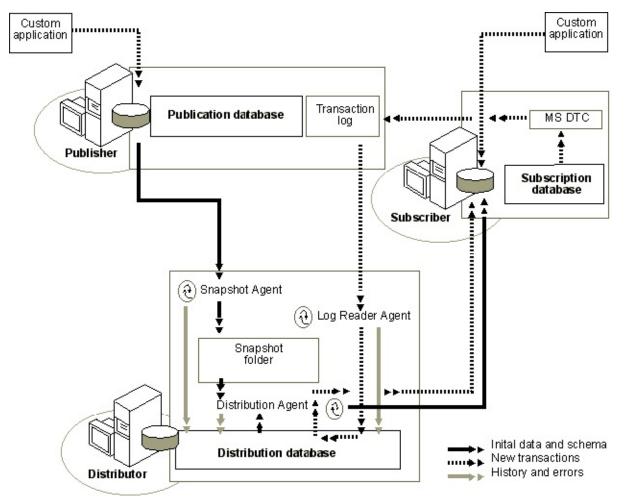
- Because there are no conflicts, there is no loss of ACID properties.
- Registers a **uniqueidentifier** column in the publishing table named MSrepl\_tran\_version. This column is used for tracking changes to replicated data and to perform conflict detection at the Publisher. Adding this **uniqueidentifier** column will cause INSERT statements without column lists to fail and increase the size of the publishing table.

If you were using the immediate updating option with Microsoft® SQL Server<sup>™</sup> version 7.0 and are upgrading to SQL Server 2000, there are additional upgrade requirements. For more information, see <u>Replication and Upgrading</u>.

# **How Immediate Updating Works**

When a publication is enabled to support immediate updating, a Subscriber can modify replicated data if the transaction can be performed by using the twophase commit protocol (2PC) with the Publisher. The 2PC transaction back to the Publisher is completed automatically, so an application can be written as though it is updating just one site.

This approach does not have the large availability limitations of using 2PC with all participating sites because only the Publisher needs to be available. After the change is made at the Publisher under 2PC, it will eventually be published to all other Subscribers to the publication.



2PC is managed by Microsoft Distributed Transaction Coordinator (MS DTC). If

the update can be performed using 2PC, the Publisher propagates those changes to all other Subscribers according to the Distribution Agent schedule (or at the time of the next snapshot refresh, if it is a snapshot publication). Because the Subscriber making the update already has the data changes reflected locally, the user can continue working with the updated data secure in the guarantee that data at the Publisher also reflects the change. There is no loss of ACID properties.

An application using immediate updating should be able to deal with a failure in the transaction, just as it would in a non-replication environment for issues such as a uniqueness violation. The most common failure is that data has been changed at the Publisher, and Subscribers need to refresh their copies. In many cases, the preferred choice might be to retry the update after a few seconds. If the transaction is successful, the Subscriber can work with the changed values immediately, and know that the update has been accepted at the Publisher without conflict and will eventually be propagated to every Subscriber of the publication. A Subscriber performing updates does not have full autonomy; however, because the Publisher must be available at the time of the update. Autonomy is higher than the full 2PC case where every site must be available for any site to perform changes.

Instead of using a **timestamp** column to track updates (as in SQL Server 7.0), a **uniqueidentifier** column, added automatically to any tables used in the publication, is used to track updates. The addition of this column requires INSERT statements to have column lists.

The **uniqueidentifier** column MSrepl\_tran\_version is used in place of **timestamps** to provide a reliable method of detecting conflicts even when an update is made offline (such as in the case of queued updating). Any server in the enterprise can assign a **uniqueidentifier** and it will not be duplicated. If an update occurs and the **uniqueidentifier** columns do not match, a conflict is detected. If the **uniqueidentifier** columns match, the update is completed.

## **Immediate Updating Components**

Immediate updating is supported using:

- Triggers
- Stored procedures
- Microsoft Distributed Transaction Coordinator (MS DTC)
- Conflict detection
- Loopback detection

#### Triggers

An update trigger at the Publisher updates the MSrepl\_tran\_version column for the updated rows when needed.

Triggers at the Subscriber capture transactions and submit them to the Publisher using a remote stored procedure call within a 2PC that is controlled by MS DTC. The triggers are created using the NOT FOR REPLICATION parameter of the CREATE TRIGGER statement so that changes applied by the Distribution Agent do not themselves cause the trigger to fire. The logic of the INSERT, UPDATE, and DELETE triggers is:

- Extract values from inserted or deleted tables at the Subscriber.
- Call the BEGIN DISTRIBUTED TRANSACTION statement.
- Execute a remote procedure to call the relevant stored procedure at the Publisher, passing values from inserted or deleted tables.

- Manage **identity** and **timestamp** values at the Subscriber. In the case of immediate updating subscriptions, the new values generated at the Publisher for these types of columns are propagated to the Subscriber as part of the 2PC transaction.
- If the remote stored procedure call succeeds, commit the transaction, reflecting exactly the same changes at both the Subscriber and the Publisher.

The Publisher then ensures that the changes are propagated to all other Subscribers. Otherwise, roll back the transaction and return an error to the user.

If you subscribe to a transactional publication and use the immediate updating option, but choose not to initialize the subscription, the immediate updating triggers are not automatically applied to the Subscriber. Instead, you must create the triggers manually at the Subscriber using **sp\_addsynctrigger**. You can use **sp\_script\_synctran\_commands** to script out the immediate-updating trigger commands at the Publisher and then use those commands when running **sp\_addsynctrigger** at the Subscriber.

When creating synchronization triggers for immediate updating or queued updating subscriptions, additional calls to the **sp\_settriggerorder** system stored procedure are made to specify the firing order for the INSERT, UPDATE, and DELETE triggers so that these triggers fire first during synchronization. If there is already a trigger set to fire first, an error will be returned and the subscription will be marked inactive. If you receive this error, you should either remove the existing trigger or set the firing order to none. Restart the Distribution Agent so that the initial snapshot and triggers are applied at the Subscriber.

#### **Stored Procedures**

Stored procedures at the Publisher apply transactions only if they do not conflict with changes made at the Publisher after the Subscriber last received its copy of the changes. If a conflict is detected, the transaction is rejected and rolled back at both sites. INSERT, UPDATE, and DELETE procedures are created for each article. The logic of the immediate updating subscription stored procedure at the Publisher is:

• Insert procedure

Attempt to insert rows. Check **@@ROWCOUNT** and **@@ERROR**, and return success or failure to calling trigger. May also return an **identity** value to the Subscriber if required.

• Delete procedure

Attempt to delete rows, with a WHERE clause that qualifies the current row with values from deleted table. Check @@ROWCOUNT and @@ERROR, and return success or failure to the calling trigger.

Update procedure. Attempt to update row, with a WHERE clause that qualifies the unique index and uniqueidentifier column in current row, with unique index and uniqueidentifier value from deleted table. Check @@ROWCOUNT and @@ERROR, and return success or failure to the calling trigger. May also return an identity value to the Subscriber if required.

**Note** A transaction that affects multiple rows must have all rows reflected at both sites to succeed.

#### **Microsoft Distributed Transaction Coordinator**

Microsoft Distributed Transaction Coordinator (MS DTC) manages the twophase commit operation between a Subscriber and Publisher inside a Microsoft® SQL Server<sup>™</sup> 2000 remote stored procedure call using the BEGIN DISTRIBUTED TRANSACTION statement in Transact-SQL.

## **Conflict Detection**

The Publisher stored procedure uses the **uniqueidentifier** column to detect whether a row has changed after it was replicated to the Subscriber. When the Subscriber requests an immediate-update transaction, it passes the **uniqueidentifier** value (generated at the Subscriber) to the Publisher, along with all other columns in the row. Within the Publisher's stored procedure, this value is compared to the current **uniqueidentifier** value for the row in question. If the values are the same, the row has not been modified after it was replicated to the Subscriber, and so the transaction is accepted. If a conflict is detected, the transaction is rejected, and the application should treat it like any transaction rollback. This usually means that the Subscriber needs to synchronize with the latest data changes at the Publisher before attempting to update the same data locally.

## **Loopback Detection**

If a transaction is applied successfully to a Subscriber and Publisher, it is unnecessary to propagate the change back to the originating Subscriber using the standard asynchronous transaction replication mechanisms. SQL Server 2000 replication has a loopback detection mechanism to handle this situation.

The information used to perform loopback detection is stored on a transactionby-transaction basis. Consequently, tables that reside in different databases at the Subscriber with immediate updating subscriptions or tables that reside in different databases across Subscribers with immediate updating subscriptions should not be updated in the same transaction.

**WARNING** Using the same transaction to update tables that reside in different databases at the Subscriber or to update tables that reside in different databases across Subscribers that have immediate updating subscriptions will delete the information necessary to control loopback detection and may cause replication to fail. Loopback detection is tracked at the transaction level. If the transaction involves more than one subscription database, SQL Server will attempt to mark the transaction with the Subscriber server name and database name multiple times. The last entry will overwrite all previous entries.

## **Immediate Updating Considerations**

Immediate updating can be enabled using SQL Server Enterprise Manager, or programmatically by using Transact-SQL system stored procedures or SQL-DMO.

#### **Immediate Updating Restrictions**

The following restrictions exist with immediate updating:

- Published tables must have a **uniqueidentifier** column. The **uniqueidentifier** column MSrepl\_tran\_version is added to the publishing table automatically. If the MSrepl\_tran\_version column already exists on the publishing table, it will be used.
- INSERT statements used to add rows of data to a table must include a column list.
- If you create two or more articles on the same table in a publication database and then create subscriptions to those articles in the same subscription database, the following additional restrictions apply:
  - If multiple articles based on the same table are in one publication enabled for immediate updating, you cannot create an immediate updating subscription to this publication. Warning message 21293 will be issued.
  - If multiple articles based on the same table are in different publications and you want to create subscriptions to all publications in the same subscription database, only one of the subscriptions can be immediate-updating.
- The immediate updating subscription connection to the Publisher (controlled by **sp\_link\_publication**) can use security mode 0 for SQL Server Authentication or 2 for linked server definition to create login

mappings. The publication access list (PAL) must include at least one SQL Server Authentication account unless you use security mode 2 and configure delegation (it is possible to set up Windows Authentication in mode 2 by configuring delegation). You can make connections to the Publisher under Windows user accounts invoking the INSERT, UPDATE, and DELETE triggers at the Subscriber using delegation. To set up delegation, see <u>sp\_addlinkedsrvlogin</u>.

- If the snapshot or transactional publication allows immediate updating subscriptions and the publication has a column filter, you cannot filter non-nullable columns without defaults from the publication.
- Subscribers using immediate updating subscriptions cannot republish data to other Subscribers.

#### **Data Modifications at Subscribers**

When modifying data at Subscriber sites using the immediate-updating Subscribers option, consider the following issues:

- The Subscriber should not update **timestamp** or **identity** values directly. Those values are generated by the Publisher as part of the 2PC transaction between the Publisher and Subscriber. Default constraints are applied to these columns at the Subscriber.
- The Subscriber cannot update or insert **text** or **image** values because they cannot be read from the inserted or deleted tables inside the trigger. Similarly, the Subscriber cannot update or insert **text** or **image** values using WRITETEXT or UPDATETEXT because the data is overwritten by the Publisher. Instead, you could partition the **text** and **image** columns into a separate table and modify the two tables within a transaction. You could use merge replication to synchronize these values if updates to **text** or **image** columns are needed at the Subscriber. You can be assured there are no conflicts if all updates follow this guideline because the update of the **text** or **image** table cannot occur unless the main table was updated, which is protected by 2PC.

- When loopback detection is in effect, modified rows are not sent back to the originating Subscriber (thereby reducing overhead).
- It is recommended that Subscriber tables have at least a unique index and preferably a primary key for snapshot replication. This is required for transactional replication.
- Although snapshot replication without immediate updating does not require the use of primary keys in a table, snapshot replication with immediate updating or transactional replication with immediate updating requires you to use primary keys on publishing tables. (Transactional replication always requires the use of primary keys on publishing tables).
- If the subscription database is horizontally filtered and there are rows in the partition that existed at the Subscriber separate from the data propagated to the Subscriber by the Publisher, and that partition is not at the Publisher, the Subscriber cannot update the pre-existing rows. Attempting to update these rows returns an error. The rows should be deleted from the table and added again.

#### **Configuration Modes**

The immediate updating option supports either dynamic RPC mode or static RPC mode for the 2PC connection from the synchronization triggers back to the Publisher. In dynamic RPC mode, synchronization triggers connect dynamically to the Publisher, using a supplied server name, login, and password. This mode offers increased security for users who do not want a statically defined linked server/remote server connection from a Subscriber to Publisher. It is also easier to use when setting up push subscriptions because the Publisher does not have to be predefined at the Subscriber. In static RPC mode, synchronization triggers connect to the Publisher over a statically defined server name defined as a linked server or remote server in the **sysservers** table. This entry is added by an administrator at the Subscriber server.

The configuration mode is set automatically when creating push or pull subscriptions:

- When setting up a push subscription using the Push Subscription Wizard in SQL Server Enterprise Manager or the **sp\_addsubscription** stored procedure, the default configuration uses dynamic RPC at the Subscriber. The dynamic RPC defaults to using the **sa** login with no password. This is done to avoid sending logins or passwords over the network, and should be changed at the Subscriber using **sp\_link\_publication**.
- When setting up a pull subscription using the Pull Subscription Wizard in SQL Server Enterprise Manager, the user chooses the desired configuration mode. If you choose static RPC, the Publisher must be configured as a linked server or remote server at the Subscriber. If you choose dynamic RPC, you must supply a login and password that the synchronization triggers will use to connect to the Publisher.
- When setting up a pull subscription using stored procedures, you need to explicitly call **sp\_link\_publication** after calling **sp\_addpullsubscription** at the Subscriber.

#### **User-Defined Triggers**

If you are adding user-defined, cascading triggers to tables that are published and allow immediate updating, you can place the triggers at either the Publisher or Subscriber. Adding the triggers at the Publisher requires no special programming considerations. For example, you may have two tables, **customer** and **orders**, where **customerid** is a primary key in the **customers** table and a foreign key in the **orders** table. You can use a user-defined trigger on the **customers** table to cascade changes to the **customerid** in the **orders** table. Updating the **customerid** in the **customers** table at the Subscriber causes the immediate updating trigger to propagate the update to the Publisher. When the update is applied to the Publisher, the user-defined trigger fires at the Publisher, and cascades the update to the **orders** table at the Publisher. When the Distribution Agent runs, the update to the **orders** table is propagated down to the Subscriber. The cascaded changes are reflected accurately at the Subscriber, but with some latency because the **orders** table is not immediately up to date.

If your application requires that the cascaded table at the Subscriber immediately reflect the change in the cascading table (that is, avoid the latency of the round-trip to the Publisher), you also can add the cascading triggers at the Subscriber. However, when you add user-defined triggers at both the Publisher and the Subscriber, both sets of triggers must be created using the NOT FOR REPLICATION option. With the NOT FOR REPLICATION option active, an update to one of the tables at the Subscriber is cascaded to the other table by the user-defined triggers on each table. Because the user-defined cascading triggers at the Publisher are marked NOT FOR REPLICATION, these triggers do not fire.

**Note** SQL Server 2000 replication supports the automatic transferring of triggers from the table at the Publisher to the table at the Subscriber; however, they will not be marked automatically as NOT FOR REPLICATION on the Subscriber, which has to be done manually. The triggers will be marked as NOT FOR REPLICATION if that is how they are defined on the Publisher.

You can also add user-defined triggers to update columns in the row currently being modified. Programming insert and update triggers is challenging because the immediate updating triggers may also need to update the same row. For example, an immediate updating trigger must insert the new **timestamp** or **identity** value received from the Publisher as part of a two-phase-commit transaction.

If both the user-defined trigger and the immediate updating trigger apply an update to the same row and you have not included a subroutine for special case handling, the transaction could terminate. Without special handling, the update process continues in a loop with each trigger update firing the other trigger until the maximum nesting level (32) is reached and the transaction terminates.

To avoid this situation, you must allow immediate updating insert and update triggers to fire before any user-defined triggers. The user-defined trigger should determine if it is being fired in the context of an immediate updating trigger and, if so, terminate without firing. Add the following lines of code to the beginning of the trigger:

DECLARE @retcode int, @trigger\_op char(10)

EXEC @retcode = sp\_check\_for\_sync\_trigger @table\_id, @tablename IF @retcode = 1 RETURN

# **Queued Updating**

Queued updating allows snapshot replication and transactional replication Subscribers to modify published data without requiring an active network connection to the Publisher.

When you create a publication with the queued updating option enabled and a Subscriber performs INSERT, UPDATE, or DELETE statements on published data, the changes are stored in a queue. The queued transactions are applied asynchronously at the Publisher when network connectivity is restored.

Because the updates are propagated asynchronously to the Publisher, the same data may have been updated by the Publisher or by another Subscriber and conflicts can occur when applying the updates.

Conflicts are detected and resolved according to a conflict resolution policy that is set when creating the publication. The transaction is then propagated to other Subscribers using typical replication mechanisms (loopback detection avoids sending the update to the Subscriber that originated the transaction).

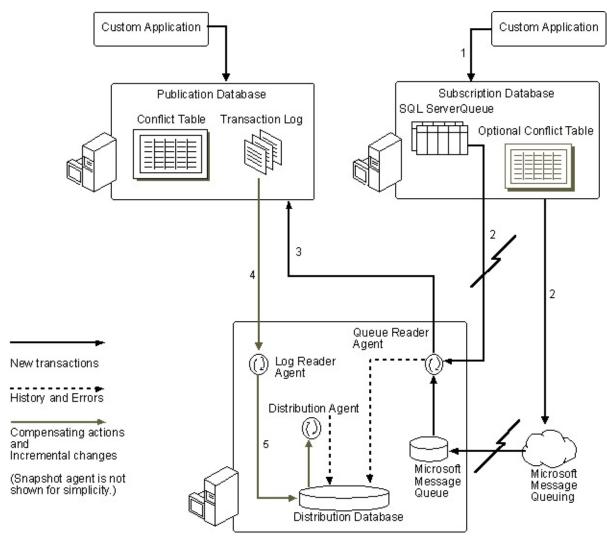
Queued updating is most appropriate for applications where users mostly read data and only occasionally update data. Subscribers should be connected most of the time, but if they are offline, updates can continue without interruption.

Both queued updating and merge replication allow updates while offline; however, there are significant differences between the two features. For more information, see <u>Merge Replication or Updatable Subscriptions</u>.

## **How Queued Updating Works**

When you create a publication and enable it for queued updating, data modifications can be made at the Subscriber and then held in a queue until they can be applied to the Publisher and then propagated to other Subscribers. The queue is implemented as a Microsoft® SQL Server<sup>TM</sup> 2000 table but on Microsoft Windows® 2000, it can optionally be implemented using Microsoft Message Queuing. For more information, see <u>Queued Updating Components</u>.

The following illustration shows how triggers, queues, and the Queue Reader Agent work together to complete this process.



- 1. Updates made at the Subscriber are captured by triggers on the subscribing tables. The triggers store these updates in a queue, which by default is a SQL Server queue. The triggers are created automatically when the subscription is created.
- 2. If you are using SQL Server queues, updates will be stored in a table designated as the queue (called **MSreplication\_queue**), which is created automatically when the subscription is configured. If you are using Message Queuing version 2.0, the updates will be stored in a message queue at the Distributor. If the Subscriber is disconnected from the network, it can continue to generate messages destined for other computers. Message Queuing stores the messages locally, and automatically sends them to the queue at the Distributor when network connection is restored.
- 3. The Queue Reader Agent applies queued transactions to the appropriate publication. When using SQL Server 2000 queues, the queued transactions are read directly from the queue stored on the Subscriber. When using Message Queuing, the queued transactions are read from a queue stored at the Distributor.
- 4. While applying the queued transactions, conflicts (if any) are detected and resolved according to a conflict resolution policy that is set when the publication is created. As a result, compensating commands may be generated to rollback a transaction to a Subscriber using the standard transactional replication distribution process, but they are sent only to the Subscriber that caused the conflict.
- 5. Any changes made at the Publisher are propagated to all other Subscribers according to the Distribution Agent schedule.

# **Queued Updating Components**

Triggers, stored procedures, queues, and the Queue Reader Agent are the components used with queued updating.

## Triggers

When immediate updating, queued updating, or immediate updating with queued updating as a failover is enabled, triggers are attached to the replicated table at the Subscriber. With queued updating, the triggers capture transactions initiated at the Subscriber, and then package the transactions into messages and place them in a queue. This occurs within the same transaction to ensure that the update to the local database and the queuing of the update is atomic.

The triggers are created using the NOT FOR REPLICATION modifier of the CREATE TRIGGER statement so that the changes applied by the Distribution Agent do not cause the trigger to fire.

If you subscribe to a transactional publication and use the queued updating option but do not initialize the subscription; the queued updating triggers are not applied to the Subscriber automatically. Instead, you must create the triggers manually at the Subscriber using **sp\_addsynctrigger**.Manual initial synchronization of a queued updating subscription is discussed later in this topic.

When creating synchronization triggers for immediate updating or queued updating subscriptions, additional calls to the **sp\_settriggerorder** system stored procedure are made to specify the firing order for the INSERT, UPDATE, and DELETE triggers so that these triggers fire first during synchronization. If there is already a trigger set to fire first, an error will be returned and the subscription will be marked inactive. If you receive this error, you should either remove the existing trigger or set the firing order to none. Restart the Distribution Agent so that the initial snapshot and triggers are applied at the Subscriber.

#### **Stored Procedures**

When you create a publication and enable it for queued updating by default, stored procedures to insert, update, and delete data in the published table are

created automatically on the publication database.

The stored procedures are called by the Queue Reader Agent to apply transactions at the Publisher, detect conflicts, and if needed, generate compensating commands, which are posted to the distribution database and then delivered to the Subscriber. INSERT, UPDATE, and DELETE stored procedures are created for each article.

A stored procedure for logging conflict information at the Publisher, and optionally sending conflict information to relevant Subscribers, is also created at the Publisher. This is invoked by the Queue Reader Agent if a conflict is detected.

## **Storing Messages in a Queue**

Subscribers with the queued updating option can use either a Microsoft® SQL Server<sup>™</sup> 2000 queue or Microsoft Message Queuing version 2.0 on Microsoft Windows® 2000 Server as the queuing mechanism. When selecting queued updating, the default is a SQL Server 2000 queue, which is available to all instances of SQL Server. After creating the publication, you can change the queue to Message Queuing using the publication properties dialog box. This must be done before activating any subscriptions to the publication.

To see which rows have changes that are pending in a queue, execute **sp\_getqueuedrows** in the subscription database at the Subscriber.

#### **SQL Server Queue**

When using SQL Server 2000 queue, each Subscriber has its own queue in the form of a SQL Server 2000 table (**MSreplication\_queue**) in the subscription database. The triggers store all messages in the SQL Server 2000 queue until the Subscriber reconnects to the network after updating published data. The Subscriber and the Publisher must be connected and available for the updates to occur.

The Subscriber is dependent on the Queue Reader Agent to read and empty the queue. The Queue Reader Agent reads messages on a Subscriber, finds modifications, and propagates the changes to the Publisher. It then repeats this process at each Subscriber.

Using SQL Server 2000 queues requires that all three servers (Subscriber, Distributor, and Publisher) are connected and available when queued updates need to be applied at the Publisher. Updates made at the Subscriber can be queued without the Subscriber, Distributor and Publisher being connected. SQL Server 2000 queues at the Subscriber can be monitored using the **sp\_replqueuemonitor** stored procedure.

SQL Server queues:

- Work with all SQL Server platforms (Windows 98, Windows NT® 4.0, and Windows 2000).
- Do not have any additional components that need to be installed.
- Are faster for updates made at the Subscriber to queue.

The **sp\_getqueuedrows** stored procedure returns a result set consisting of rows in the user table that have pending updates in the queue not yet picked up by the Queue Reader Agent. This procedure can be used to identify the rows that can be considered tentative.

## **Microsoft Message Queuing**

If you are running Windows 2000 Server on the Distributor and Subscriber, you have the option to use Microsoft Message Queuing as the queuing mechanism at the Subscriber. Message Queuing provides additional routing, centralized monitoring, and administrative capabilities beyond what is available with SQL Server 2000 queues.

When using Message Queuing as the queuing mechanism, the update is packaged as a message and is placed in a queue on the Distributor under a twophase commit protocol (2PC) transaction managed by Microsoft Distributed Transaction Coordinator (MS DTC).

When the Subscriber is disconnected from the network, Message Queuing stores transactions as messages in a cache on the Subscriber until they can be sent to a corresponding queue on the Distributor. You must enable Message Queuing on both the Subscriber and the Distributor. The Queue Reader Agent, which runs at

the Distributor, reads the queued messages asynchronously and applies them as transactions to the appropriate publication.

Using Message Queuing provides some advantages over SQL Server 2000 queues. In addition to routing capabilities, it offers centralized queue administration and monitoring. This is not possible with SQL Server 2000 queues because the queues are distributed at each Subscriber instead of consolidated at the Distributor.

Message Queuing provides better offline capabilities including propagating offline changes to the queue at the Distributor without SQL Server running on the Subscriber. In addition, Message Queuing does not require availability of the Publisher when the Subscriber reconnects to the network after updating published data. Message Queuing propagates messages automatically when the Subscriber comes online without relying on the Queue Reader Agent to read and empty the queue. It will also be a better choice if there are many Subscribers.

You will need to install Message Queuing on each Subscriber and the Distributor. Queued updating works with Message Queuing installed in workgroup mode on Windows 2000. This eliminates the need to install Message Queuing on a Windows 2000 domain controller and should be the preferred installation method unless you have other Message Queuing requirements that preclude using workgroup mode (for example, Message Queuing in workgroup mode does not allow public queues and cannot use Message Queuing authentication or encryption).

For Message Queuing installed in workgroup mode, install Message Queuing on the Distributor and on the Subscribers. For Message Queuing not installed in workgroup mode, install Message Queuing server on the domain controller and Message Queuing independent client on the Distributor and on the Subscribers.

#### To install Message Queuing on the Distributor and Subscribers

## **Queued Updating Considerations**

When using queued updating, consider the following:

- Queued updating is supported only with Subscribers running SQL Server 2000.
- If you create two or more articles on the same table, and then create subscriptions to those articles in the same Subscriber database, the following restrictions apply:
  - If multiple articles based on the same table are in a single publication enabled for queued updating, you cannot create a queued updating subscription to this publication.
  - If multiple articles based on the same table are in different publications and you want to create subscriptions to all publications in the same database, only one of the subscriptions can be queued updating.
- The publication access list (PAL) must include at least one SQL Server Authentication account.
- Subscribers using immediate updating or queued updating cannot republish replicated data at the Subscriber.
- If a transaction at the Subscriber involves multiple databases, compensating commands are generated only for the updates affecting the subscription database in case of a conflict.
- Tables included in a merge publication cannot also be published as part of a snapshot or transactional publication that allows queued updating subscriptions.

#### Modifying Data at the Subscriber

When modifying published data at the Subscriber, consider the following:

- The Subscriber cannot update or insert **text** or **image** values because they cannot be read from the inserted or deleted tables inside the trigger. Similarly, the Subscriber cannot update or insert **text** or **image** values using WRITETEXT or UPDATETEXT because the data is overwritten by the Publisher. Instead, you could partition the **text** and **image** columns into a separate table and modify the two tables within a transaction. Use merge replication to synchronize these values. You cannot be assured there are no conflicts because the update of the **text** or **image** table can occur if the data is not well partitioned.
- INSERT statements used to add rows of data to a table must include a column list.
- It is recommended that Subscriber tables have at least a unique index and preferably a primary key for snapshot replication. This is required for transactional replication.
- Although snapshot replication without immediate updating does not require the use of primary keys in a table, snapshot replication or transactional replication with an immediate updating subscription requires you to use primary keys on publishing tables. Although snapshot replication does not require the use of primary keys in a table, queued updating does require the use of primary keys.
- Updates made to primary key columns are not recommended when using queued updating because the primary key is used as a record locator for all queries. When the conflict resolution policy is set to Subscriber Wins, updates to primary keys should be made with caution. If updates to the primary key are made at both the Publisher and at the Subscriber, the result will be two rows with different primary keys.

For example, if a row has a value of 'Bill' in the primary key column, and that value is updated to be 'William' at the Publisher and to 'Will' at the Subscriber, both the publication database and the subscription database will end up with two rows (one with the primary key 'William', and the other with the primary key of 'Will'). It is recommended to restrict primary key updates to a single site (for example, you could restrict primary key updates by adding an update trigger at the Subscriber that prevents updates to columns participating in the primary key. The trigger could be added to any necessary Subscribers by using script execution before or after applying the initial snapshot).

- Updates to unique keys (including primary keys) that generate duplicates (for example, an update of the form UPDATE <column> SET <column> =<column>+1) are not allowed and will be rejected because of a uniqueness violation. This is because set updates made at the Subscriber are propagated by replication as individual UPDATE statements for each row affected.
- If the Subscriber database is partitioned horizontally and there are rows in the partition that exist at the Subscriber but not at the Publisher, the Subscriber cannot update the pre-existing rows. Attempting to update these rows returns an error. The rows should be deleted from the table and added again.

# Manual Initial Synchronization of a Queued Updating Subscription

If you subscribe to a transactional publication that allows queued updating subscriptions, but you do not have the subscription initialized automatically by SQL Server, all of the objects (custom stored procedures, change tracking triggers, and conflict table) will not be created. You will need to create them manually with the following steps:

- 1. Script the creation of the table at the Publisher, and using that script, create the table in the **subscription** database. If you create the script manually, include the primary key constraint.
- 2. In the **publication** database, execute the following stored procedures:

- **sp\_scriptinsproc** (specify the @article\_id parameter).
- **sp\_scriptxupdproc** (specify the @article\_id parameter).
- **sp\_scriptxdelproc** (specify the @article\_id parameter).

These will generate scripts for custom stored procedures to be applied to the **subscription** database. Execute these scripts in the **subscription** database. The article ID value can be obtained by executing **sp\_helparticle**.

- 3. In the **publication** database, execute the following system stored procedure:
  - **sp\_makeconflicttable** (specify the @publication and @article parameters).

This stored procedure returns 0 if successful and 1 if not successful. This generates a script for the conflict table for the given article. Execute this script in the **subscription** database.

- 4. At the Subscriber, execute the following system stored procedure:
  - **sp\_addsynctriggers**. For more information, see <u>sp\_addsynctriggers</u>.

# **Queued Updating Conflict Detection and Resolution**

Because queued updating allows modifications to the same data at multiple locations, there may be conflicts when data is synchronized at the Publisher. Conflict detection and resolution is handled differently with queued updating than it is with merge replication. With queued updating, conflict detection and resolution is based on maintaining atomicity of the transaction. Because of this requirement, the number of conflict resolution policies that can be defined by the user is limited as compared with merge replication, which provides a more flexible framework for conflict resolution, but merge replication handles conflicts at the row level, not at the transaction level.

Microsoft® SQL Server<sup>™</sup> 2000 detects the conflict when changes are synchronized with the Publisher. It then follows the resolution policy you selected when creating the publication.

Conflict detection and resolution can be a time-consuming and resourceintensive process, and it is best to minimize conflicts in the application by creating data partitions so that different Subscribers are modifying different subsets of data, and to prevent a user's work from being uncommitted if a conflict occurs.

## **Detecting Conflicts**

When creating a publication and enabling queued updating, SQL Server 2000 adds a **uniqueidentifier** column (**MSrepl\_tran\_version**) with the default of **newid()** to the underlying table. When published data is changed at either the Publisher or the Subscriber, the row receives a new globally unique identifier (GUID) to indicate that a new row version exists. The Queue Reader Agent uses this column during synchronization to determine if a conflict exists.

A transaction in a queue maintains the old and new row version values. When the transaction is applied at the Publisher, the GUIDs from the transaction and the GUID in the publication are compared. If the old GUID stored in the transaction matches the GUID in the publication, the publication is updated and the row is assigned the new GUID that was generated by the Subscriber. By updating the publication with the GUID from the transaction, you have matching row versions in the publication and in the transaction.

If the old GUID stored in the transaction does not match the GUID in the publication, a conflict is detected. The new GUID in the publication indicates that two different row versions exist: one in the transaction being submitted by the Subscriber and a newer one that exists on the Publisher. In this case, another Subscriber or the Publisher updated the same row in the publication before this Subscriber transaction was synchronized.

Unlike merge replication, the use of a GUID column is not used to identify the row itself, but is used to check if the row has changed.

## **Resolving Conflicts**

When you create a publication using queued updating, a conflict resolver instructs the Queue Reader Agent how it should handle different versions of the same row encountered during synchronization. By default, the Publisher wins conflict resolver is set. You can change the conflict resolution policy after the publication is created as long as there are no subscriptions to the publication.

The conflict resolver choices are:

- Publisher wins and the subscription is reinitialized
- Publisher wins
- Subscriber wins

These conflict resolvers maintain transactional consistency at the Subscriber to varying degrees. Reinitializing the Subscriber provides the highest degree of transactional consistency, and Subscriber wins provides the lowest degree of transactional consistency.

Conflicts are recorded and can be viewed using the Conflict Viewer. When using queued updating with snapshot replication, the conflict resolution policy is restricted to reinitializing the Subscriber or Publisher wins. The Subscriber wins conflict resolution policy is not available.

#### **Reinitialize Subscriber**

Reinitializing Subscriber to resolve conflicts maintains strict transactional consistency at the Subscriber, but it can be time consuming if the publication contains large amounts of data.

When the Queue Reader Agent detects a conflict, all remaining transactions in the queue (including the transaction in conflict) are rejected, and the Subscriber is marked for reinitialization. The next snapshot generated for the publication will be applied by the Distribution Agent to the Subscriber.

## **Publisher Wins**

When the conflict resolution is set to Publisher wins, transactional consistency is maintained based on the data at the Publisher. The conflicting transaction is rolled back at the Subscriber that initiated it.

The Queue Reader Agent detects a conflict and compensating commands are generated and propagated to the Subscriber by posting them in the distribution database. The Distribution Agent then applies the compensating commands to the Subscriber that originated the conflicting transaction. The compensating actions update the rows on the Subscriber to match the row on the Publisher.

Until the compensating commands are applied, it is possible to read the results of a transaction that will eventually be rolled back to the Subscriber. This is equivalent to a dirty read (read uncommitted isolation level). There is no compensation for the subsequent dependent transactions that can occur. However, transaction boundaries are honored and all the actions within a transaction are either committed, or in the case of a conflict, rolled back.

#### Subscriber Wins

Conflict detection under the Subscriber wins policy means the last Subscriber transaction to update the Publisher wins. In this case, when a conflict is detected, the transaction sent by the Subscriber is still used and the Publisher is updated. This policy is suitable for applications where such changes do not compromise data integrity.

#### To set the queued updating conflict resolution policy

# **Queued Updating and Identity Ranges**

Normally with snapshot replication and transactional replication (read only or using immediate updating), if the publishing table contains a column with the **identity** data type, the identity property is not propagated to the Subscriber.

The **identity** property is used to provide next number values for data automatically (for example, for columns such as Customer ID or Order ID). When using immediate updating, the Publisher determines this value, and as part of the 2PC transaction initiated by the Subscriber, it is synchronized between Publisher and Subscriber.

With queued updating and immediate updating with queued updating as a failover, identity values must be assigned at the Subscriber because the Subscriber may be offline and updates at the Subscriber may be sent to a queue. In this case, the Publisher will not be able to assign identity values immediately. Therefore, when the initial snapshot is applied at the Subscriber, the identity property is propagated as well.

To avoid different Subscribers assigning the same identity values, you can define identity ranges for each Subscriber. When you define identity ranges, a Subscriber is allowed to assign values only from a specific range.

You can manage identity values using automatic identity ranges (SQL Server 2000 replication handles assigning identity ranges for you) or you can set identity ranges manually using a check constraint and the NOT FOR REPLICATION option on the IDENTITY property of a Transact-SQL CREATE TABLE statement.

For more information about handling identity values in replication, see <u>Replication Data Considerations</u>.

If you are using the attachable subscription database feature and the subscription database has subscriptions to publications that allow queued updating with auto identity range articles, you will need to run the distribution agents to obtain new identity ranges on the Subscriber after attaching the subscription database. For more information, see <u>Attachable Subscription Databases</u>.

## See Also

Identity Ranges with Immediate Updating and Queued Updating

# Immediate Updating with Queued Updating as a Failover

Immediate updating with queued updating as a failover can be used when you expect the Publisher and Subscribers to be connected, but you do not want to lose the ability to make updates at the Subscriber if a system failure results in the loss of network connectivity. Immediate updating with queued updating as a failover allows you to use immediate updating and switch to queued updating when needed.

In this case, 2PC is used to propagate updates made at the Subscriber to the Publisher until you enable the queued updating failover. After the queued updating failover is enabled, transactions from the Subscriber are packaged into messages and sent to a queue. The transactions are recorded asynchronously and are applied to the Publisher when a connection is re-established.

You can invoke queued updating failover at any time, but after you do, you cannot failback to immediate updating until the Subscriber and Publisher (or Distributor and Publisher in the case of Message Queuing) are connected and the Queue Reader Agent has applied all pending messages in the queue to the Publisher. Queued updating is not invoked automatically because it may be easy to fix the problem that is preventing immediate updating (for example, hardware that is disconnected). You may not need or want to allocate resources to switch from queued updating back to immediate updating (which requires emptying the queue).

Pull subscriptions created using on-demand synchronization are added to Windows Synchronization Manager automatically. You can add pull subscriptions that are not using on-demand synchronization to Windows Synchronization Manager by opening the subscription properties, and then on the **Synchronization** tab, selecting **Enable this subscription to be synchronized using the Windows Synchronization Manager**.

To enable immediate updating with queued updating as a failover

# **Transforming Published Data**

Transformable subscriptions (available with snapshot replication or transactional replication) leverages the data movement, transformation mapping, and filtering capabilities of Data Transformation Services (DTS). Using transformable subscriptions in your replication topology allows you to customize and send published data based on the requirements of individual Subscribers.

Examples of how you can use transformable subscriptions include:

- Creating column and horizontal partitions of published data on a per Subscriber basis (custom data partitions).
- Creating data transformations such as data type mappings (for example, integer to real data type), column manipulations (for example, concatenating first name and last name columns), string manipulations, and use of functions.

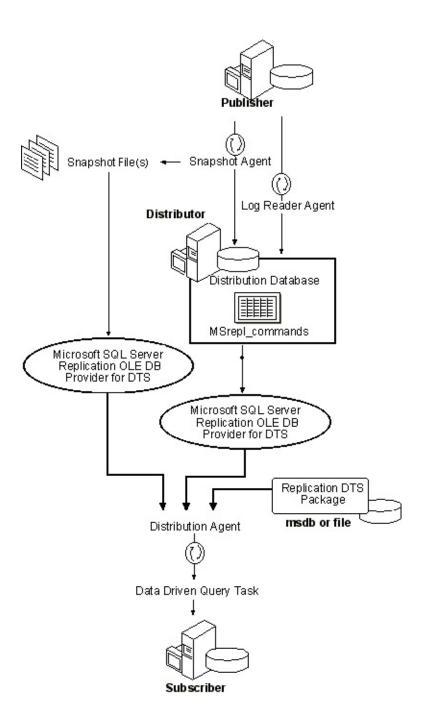
The option to allow transformations is set at the time you create a publication. After the option is set, and a replication DTS package is built, Subscribers to the publication can attach a DTS package and incorporate it as part of the replication data flow. This functionality is supported for Microsoft® SQL Server<sup>™</sup> 2000 and OLE DB Subscribers (ODBC Subscribers are not supported).

You create a DTS package for replication either using the replication wizards or programmatically, such as using Microsoft Visual Basic<sup>®</sup>. You can also customize a DTS package for use with a Subscriber by using DTS Designer.

The transformable subscriptions topics assume you are familiar with DTS. For information about DTS concepts, DTS programming, and using DTS as a part of your data warehousing strategy, see <u>DTS Basics</u>

# **How Transforming Published Data Works**

When a publication is configured to allow DTS transformations, the Subscriber is allowed to specify a DTS package as part of setting up a subscription. The following diagram illustrates how snapshots and subsequent incremental changes are transformed before the data is applied to the Subscriber.



#### **Snapshots**

During the process of applying the snapshot, the Distribution Agent loads the replication DTS package from the **msdb** database (or loads a saved .dts file, in the case of OLE DB pull Subscribers). The SQL Server replication OLE DB Provider for DTS converts snapshot data into an OLE DB rowset that is used to drive a DTS Data Driven Query task, which performs any specified

transformations or filtering operations before applying the data to the Subscriber. This is a special purpose OLE DB provider intended for use only by replication and not a general purpose OLE DB provider.

The following events and processes occur when a DTS package is included in the replication data flow:

- A DTS package is created with the snapshot .bcp (bulk copy) file as the source input to the package.
- The Subscriber table is created from the script in the DTS package Execute SQL task.
- The Data Driven Query task, used in a replication DTS package, moves data using Transact-SQL INSERT statements. When snapshots are applied or reinitialized, the equivalent of an INSERT statement for each row of data is executed by the DTS package.

For publications allowing DTS transformations, the snapshot .bcp data files are generated as character-mode because native format .bcp files cannot be used with DTS.

Heterogeneous Subscribers can subscribe to publications for which the snapshot is created in character-mode, as long as the publication allows transformations of published data.

#### **Incremental Changes**

As incremental changes occur at the Publisher, the Distribution Agent retrieves transactions that need to be replicated from the distribution database, and processes them in the same way described for applying a snapshot. In this case, however, the data source is the **MSrepl\_commands** table rather than a .bcp character-mode data file. For incremental changes, the Data Driven Query task handles UPDATES and DELETES in addition to INSERTS, and applies the incremental changes for individual statements within a transaction according to the type of incremental change and its specified transaction mappings (for example, if the change is mapped to a transaction with INSERT, UPDATE or DELETE statements).

**Note** When columns are added to or dropped from a publication that allows transformations on published data, the DTS packages will need to be regenerated.

# **Creating a Transformable Subscription Using Replication Wizards**

Creating a transformable subscription using the replication wizards requires the following steps:

- 1. Create a new snapshot publication or transactional publication and enable the publication for transformable subscriptions using the Create Publication Wizard.
- 2. Build the replication DTS package (define the columns and rows in the partition, and map the transformations) using the Transform Published Data Wizard (available by right-clicking the publication that allows data transformations and selecting **Define Transformation of Published Data**).
- 3. Create a subscription that incorporates an existing replication DTS package to transform published data, using either the Push Subscription Wizard or the Pull Subscription Wizard (the DTS package must exist at the time a subscription is created). When creating a transformable subscription, you select from the existing DTS packages marked for replication.

After the replication DTS package is created, advanced users can edit the DTS package in DTS Designer to customize it for an individual Subscriber.

**Note** You must enable a subscription to use a DTS package when you create the publication; you cannot modify an existing publication to use DTS packages.

#### To create a transformable subscription

# Using Transformable Subscriptions to Create Custom Data Partitions

In earlier versions of Microsoft® SQL Server<sup>™</sup>, if you needed to create different partitions of data for different Subscribers that subscribe to snapshot or transactional publications, you would have to create a different publication for each Subscriber. With SQL Server 2000, you can use transformable subscriptions to create custom data partitions for a single publication that provide different data based on requirements of individual Subscribers. You can create partitions of column and row data on a per Subscriber basis, with one publication supporting multiple subscriptions.

One method of creating custom data partitions with transformable subscriptions is with the Transform Published Data Wizard. After you create a publication and enable it for transformable subscriptions, you create the data partitions in the Transform Published Data Wizard as part of the process of building a replication DTS package. Most of the work required to build data partitions is done on the Column Mappings and Transformations page of that wizard.

# **Defining a Vertical Partition**

The Define Transformations page of the Transform Published Data Wizard lists the published tables. To filter published data vertically, select a table in that page by clicking its transform (...) button, and then on the Column Mappings and Transformations page, click the **Column Mappings** tab and clear any columns you want to exclude from the partition.

# **Defining a Horizontal Partition**

Using transformable subscriptions, you can exclude certain rows on a per Subscriber basis. To partition data horizontally for a transformable subscription, you must:

- Enable the publication to use horizontal DTS partitions by selecting the **Provide support for horizontal DTS transformation scripts** option in the properties for each article for which you want a horizontal DTS partition.
- Use the Transform Published Data Wizard to build the DTS package. The Define Transformations page of the Transform Published Data Wizard lists the published tables. To partition published data horizontally, select a table in that page by clicking its transform (...) button. On the subsequent Column Mappings and Transformations page, click the **Transformations** tab. The **Transformations** tab contains an edit box you use to write the Microsoft® ActiveX® scripts that define the horizontal partition.
- Include Microsoft ActiveX scripts written to the DTS object model with the DTS package. The ActiveX script needs to specify the filter criterion and be able to check if:
- 1. A newly inserted row needs to be propagated to the Subscriber.
- 2. Rows updated at the Publisher no longer meet the partitioning criterion and need to be deleted at the Subscriber.
- 3. Rows updated at the Publisher meet the filter criteria and need to be inserted at the Subscriber.

Supporting steps 2 and 3 is possible by using XCALL syntax for UPDATE,

which ensures before and after values for the row after an update. This allows the ActiveX script to determine if the row is moving into or out of the partition.

The ActiveX scripts you use to define the horizontal partition need to follow guidelines that are explained in the following example. You would only need to provide the two functions **IsInPartition()** and **Transform()**. You do not need to change the **Main()** function.

You can also use the system stored procedures **sp\_addarticle** and **sp\_changearticle** to specify horizontal partitions for snapshot or transactional publications that allow data transformations.

If a publication allows transformable subscriptions, you can set @status parameter for **sp\_addarticle** to a value of 64 to indicate that the article supports DTS horizontal partitions. If the parameter is not set, it is not possible to insert or delete rows at the Subscriber when rows move into or out of the partition resulting from an update at the Publisher. If the status is set, the before image of an updated row will be sent to the **distribution** database by the Log Reader Agent and then to the DTS package inside the Distribution Agent. This will increase network traffic, and should be used with caution.

If the publication allows data transformations, **sp\_changearticle** accepts the values of 'dts horizontal partitions' and 'no dts horizontal partitions' for the status article property. Changes to this property are not allowed when there are active subscriptions (snapshots have been generated for the subscriptions). If the publication allows immediate updating, the Snapshot Agent needs to be run after this property is changed. Snapshots generated previously will be obsolete.

#### Validation of Custom Data Partitions

Using transformable subscriptions to exclude rows or columns and create horizontal and vertical partitions means that data at the Subscriber will be different than data at the Publisher. ROWCOUNT and CHECKSUM validation will report discrepancies. Typically, you would not want to run validation for those Subscribers.

Dynamic filters available with merge replication do not have this limitation because the validation is compared with the SQL Server view that defines the partition.

#### Example

Function Main()

- ' If the current source data is the old values of the row before an upd
- ' ReplicationChangeType values:
- ' 1 = Insert. Source data is from a row that was inserted at the source
- ' 2 = Update. Source data is from a row that was updated at the sour
- ' 3 = Delete. Source data is from a row that was deleted at the source
- ' 4 = Before Update. Source data is from a row that was updated at t

If DTSGlobalVariables("ReplicationChangeType").Value = 4 Then OldRowInPartition = IsInPartition()

Main = DTSTransformStat\_SkipRow

Else 'Error check to prevent users from forgetting to enable the arti-

If DTSGlobalVariables("RelicationChangeType").Value=2 and IsEm

Err.Raise 1, "Replication DTS ActiveX Script", "The article does n End If

If IsInPartition() Then

'Set default return status, which means using the query type set by the Main = DTSTransformStat\_OK

Transform

' If it is an update, test to see if the row has just moved into the partiti If DTSGlobalVariables("ReplicationChangeType").Value = 2 And \_

Not OldRowInPartition Then

Main = DTSTransformStat\_InsertQuery

End If

Else

' If it is an update, test to see if the row has just moved out of the part ' If so, overwrite the query type to insert from update.

If DTSGlobalVariables("ReplicationChangeType").Value = 2 And \_ OldRowInPartition Then

```
Transform
 Main = DTSTransformStat_DeleteQuery
 Else
 Main = DTSTransformStat_SkipRow
 End If
 "
 End If
 End If
 End Function
Function Transform()
   DTSDestination("CustID") = DTSSource("CustID")
   DTSDestination("LName") = DTSSource("LName")
   DTSDestination("FName") = DTSSource("FName")
   DTSDestination("Item") = DTSSource("Item")
   DTSDestination("SaleAmt") = DTSSource("SaleAmt")
End Function
 Function IsInPartition()
 ' In partition if the first char is uppercase and begins with A, B, C or
  If Left(DTSSource("LName"), 1) <= "D" Then
 IsInPartition = True
 Else
 IsInPartition = False
 End If
 End Function
```

# Using Distributed Agents to Create Efficient Custom Partitions

It is recommended that you use push subscriptions when creating data partitions using transformable subscriptions. If you intend to use transformable pull subscriptions instead, you can decrease the amount of data sent over the network by running the Distribution Agent on the Distributor using DCOM instead of on the Subscriber.

The advantage of this functionality is that the filtering takes place at the Distributor, and only the data included in a partition is sent over the network. If the Distribution Agents were located at each Subscriber, all the published data would travel over the network before being filtered at the Subscriber. Remote agent activation works best on Microsoft® Windows® 2000 and requires additional configuration when using Windows 98 and Windows NT® 4.0. For information about configuring replication for distributed pull agents, see <u>Remote Agent Activation</u>.

# Using Transformable Subscriptions with Data Transformations

You can use transformable subscriptions for basic data transformations and column manipulations between a Publisher and Subscriber, including:

- Changing data types (except for primary key columns).
- Renaming a column.
- Concatenating columns.
- Extracting a substring from, or adding characters to, a column.
- Applying functions to column values.

You map transformations in the Transform Published Data Wizard, on the Column Mappings and Transformations page, on the **Transformations** tab, using Microsoft® ActiveX® scripts written to the DTS object model (the same as when defining partitions). Using scripting code, specify the transformation in the ActiveX script edit box. For example, to concatenate the first name and last name columns of published data to a one name column in the Subscriber, you could use the following Microsoft Visual Basic® Scripting Edition code (the second line of the function shows the column concatenation):

```
Function Main()
DTSDestination("CustID") = DTSSource("CustID")
DTSDestination("Name") = DTSSource("LName") & " " & DTSSou
DTSDestination("Item") = DTSSource("Item")
Main = DTSTransformStat_OK
End Function
```

### **DTS Package Details**

A Data Transformation Services (DTS) package containing a specific configuration of DTS custom tasks, connection settings, and workflow is used to create a transformable subscription.

#### Connections

• To connect to the distribution database and provide published data to the DTS package (source connection), a special OLE DB provider, the Microsoft SQL Server Replication OLE DB Provider for DTS is used. When you create a transformable subscription, this provider is installed automatically on the DTS package, and cannot be changed. This provider can be used only with transformable subscriptions.

The connection from the DTS package to the Subscriber (destination connection) does not use the Microsoft SQL Server Replication OLE DB Provider for DTS; it uses whatever OLE DB provider is required to connect to the Subscriber. For example, you would use the Microsoft OLE DB Provider for SQL Server to send transformed data to a Microsoft® SQL Server<sup>™</sup> 2000 Subscriber.

• Only SQL Server (the Microsoft OLE DB Provider for SQL Server) Subscribers and other OLE DB Subscribers can use transformable subscriptions; ODBC Subscribers will not work with transformable subscriptions.

#### **Package Storage**

A DTS package created through replication is saved by default as a DTS SQL Server package (a DTS save option where the package is saved to local SQL Server tables in the **msdb** database); however, packages can also be saved as a DTS structured storage (.dts) file. Packages used with transformable subscriptions cannot be saved in the other available DTS save formats (repository, Microsoft Visual Basic® Script file, Visual Basic .bas file).

#### **Package Security**

The DTS package can be read or overwritten only by a user operating under the **sysadmin** role, or by the package owner. Thus, Distribution Agents need to be run under the **sysadmin** account or the package owner account at the Distributor or Subscriber site. The optional package password is stored in

MSDistribution\_agents at the Distributor, or in the

**MSSubscription\_properties** table. If the package is stored at the Distributor, all the publication access list (PAL) users will have access to the package passwords defined for subscriptions on the publication. If the package is stored on the Subscriber, members of **db\_owner** for the Subscriber database have access.

#### **Package Components**

A DTS package used for transformable subscriptions consists of several DTS objects:

- Connections for the Subscriber (multiple Subscribers share the same connection) and the Publisher. The Publisher connection is always a Microsoft SQL Server Replication OLE DB Provider for DTS data source. A different Publisher connection is necessary for each article.
- One or more Execute SQL tasks. These include, at minimum, create table scripts, per article, for each published article.
- Data Driven Query task. This task is able to match different types of replication change request (INSERT, DELETE, UPDATE) with the data movement operations required to implement the request on the Subscriber correctly. Each article requires a different Data Driven Query task in the DTS package. For the data movement to work, the destination column values must bind to parameters in the INSERT, DELETE, and UPDATE statements. The Data Driven Query task handles all data movement from the distribution database to the Subscriber through its underlying data pump. For the snapshot, the **InsertQuery** property is used.

- If a replication DTS package is customized later in DTS Designer by adding a Microsoft ActiveX® script transformation, the return status specified in the script must be changed to: Main = DTSTransformStat\_OK
- A global variable, *ReplicationChangeType*, is set by the Distribution Agent to signify the change type of a particular row, and can be accessed programmatically. Here are the available change types.

Change Type	Value	Action
INSERT	1	Source data is from a row that was
		inserted at the source.
UPDATE	2	Source data is from a row that was
		updated at the source. The data are
		values after the update.
DELETE	3	Source data is from a row that was
		deleted at the source.
BEFORE	4	Source data is from a row that was
UPDATE		updated at the source. The data are
		values before the update. This is used
		with horizontal partitions.

Declaring the global variable is optional. When used, its value is set by the Distribution Agent. The global variable can be used with an ActiveX script or other transformation servers to determine the change type associated with the current row. Following is sample Microsoft Visual Basic code you might use to declare and use this global variable:

Dim oConnProperty As DTS.OLEDBProperty Dim gVar As Integer Dim oGlobal As DTS.GlobalVariable Set oGlobal = goPackage.GlobalVariables.New("ReplicationCf oGlobal.Name = "ReplicationChangeType" oglobal.Value = 0 goPackage.GlobalVariables.Add oGlobal

#### Set oGlobal = Nothing

#### **Debugging ActiveX Scripts in DTS Packages**

You can debug ActiveX scripts in transformable subscription DTS packages. To debug your scripts:

- Turn on debugging in the Data Transformation Services Package Properties dialog.
- Run the Distribution Agent from the command line or ActiveX control. Do not run it under the SQL Server Agent NT service.

For more information about how to debug scripts, see <u>Debugging ActiveX</u> <u>Scripts</u>.

To turn on just-in-time debugging

# **Limitations and Considerations**

Limitations to using transformable subscriptions and considerations to take into account include the following:

- Snapshot data for a transformable subscription is limited to character mode only; native format (which is usually faster to apply) cannot be used with Data Transformation Services (DTS).
- After a publication is enabled for transformable subscriptions, the option cannot be disabled; the existing publication must be deleted and a new one created.
- You cannot use the updatable transactional Subscriber (two-way transactional updates) or queued updating Subscriber feature with transformable subscriptions (transformations are mapped in one direction, from Publisher to Subscriber).
- If an existing DTS package is changed, and Distribution Agent is running in continuous mode, the Distribution Agent must be shut down and restarted for the package changes to take effect.
- Although creating a transformable subscription creates a DTS package, this type of DTS package is not available for execution outside of replication (from DTS Designer or at the command prompt).
- You must have the proper access permissions for executing the DTS package to use a transformable subscription.
- Introducing DTS transformations into the replication data flow may affect performance. Performance will usually be somewhat slower than sending data to a Subscriber without a DTS package.

- If you add or drop columns from a published table by using Add Column or Drop Column on the Filter Columns tab of the publications properties dialog box, or by using sp\_repladdcolumn or sp\_repldropcolumn, you will also need to drop and recreate the DTS package to reflect changes to the meta data.
- WRITETEXT and UPDATETEXT statements cannot be used to update columns with **text**, **ntext**, or **image** data types.

# **Alternate Synchronization Partners**

Subscribers to merge publications can synchronize with servers other than the Publisher at which the subscription originated. Synchronizing with alternate partners provides the ability for a Subscriber to synchronize data even if the primary Publisher is unavailable. This feature is also useful when mobile Subscribers have access to a faster or more reliable network connection with an alternate synchronization partner.

The following are requirements when using alternate synchronization partners:

- The feature is available only with merge replication.
- The alternate synchronization partner must have the data and schema required by the subscription.
- It is recommended that the publication created on the alternate server be a clone of the publication created on the original Publisher.
- The publication properties must specify that Subscribers can synchronize with other Publishers.
- For named subscriptions (subscriptions that are not anonymous to the Publisher), the Subscriber must be enabled at the alternate synchronization partner so that the Subscriber can synchronize data with that Publisher. If this is not done, the merge Agent will add the subscription entry at the Publisher automatically.
- For named subscriptions, a subscription with the same attributes as the subscription at the primary Publisher will be added automatically at the alternate synchronization partner.

### **How Alternate Synchronization Partners Works**

To enable alternate synchronization partners, create a publication and then modify its properties to allow Subscribers to synchronize with alternate partners or create a publication with the property enabled using SQL-DMO or Transact-SQL system stored procedures. Next, create a pull subscription with the synchronize on demand option.

When you need to synchronize data using an alternate partner, you can use Windows Synchronization Manager, SQL Server Enterprise Manager, or the SQL Server merge replication ActiveX® control and select from a list of available alternate synchronization partners. When synchronizing published data, the Publisher where the subscription originated is the default Publisher; however, you can choose to specify a different synchronization partner as the default in Windows Synchronization Manager.

If you are using named Subscriptions, you must enable the Subscriber at the alternate synchronization partner and create a subscription identical to the original subscription at the alternate synchronization partner.

When the Subscriber merges its published data with data at an alternate synchronization partner, checks occur verifying that the Subscriber login exists in the publication access list (PAL) and ensuring that the Subscriber is enabled at the alternate synchronization partner (for named subscriptions).

When a Subscriber using an anonymous subscription synchronizes with an alternate synchronization partner for the first time, this subscription is recorded in the subscription database. The subscription will have the same attributes as the subscription at the primary Publisher.

# To enable Subscribers to synchronize with alternate synchronization partners

# **Optimizing Synchronization**

Optimizing synchronization during merge replication allows you to minimize network traffic when determining if recent changes have caused a row to move into or out of a partition for a Subscriber. In merge replication, an option is provided that stores more information at the Publisher instead of transferring that information over the network to the Subscriber. While this option may result in a larger database at the Publisher, it can improve synchronization performance over a slow link. However, more information will be stored at the Publisher and additional storage space will be necessary.

If the optimize synchronization setting is not used, changes in one partition will cause the merge process to verify the partition content of data sent to all Subscribers again, even if the change affects only one or a few Subscribers.

For example, if sales data is filtered based on the state where a customer resides, and a customer moves from Washington to California, that row needs to be removed from the data partition sent to Washington Subscribers and added to the data partition sent to California Subscribers.

If optimize synchronization is not used, the merge process will also check partitions sent to other Subscribers (those in Idaho, Oregon, and so on) for the state value that changed. The merge process cannot know what the California value was before it was changed. Enabling synchronization optimization will allow the merge process to accurately know what partitions were affected and what Subscribers need to be cleaned up.

By storing additional information at the Publisher, Microsoft® SQL Server<sup>™</sup> 2000 can more quickly determine the filtered data that should be sent to a particular Subscriber. When synchronization is optimized, SQL Server 2000 creates before image tables at the Publisher that contain additional information about changes to columns used in horizontal or join filters. These before images from an UPDATE or DELETE to such a column permit the Merge Agent to determine quickly and accurately which Subscriber may need to have rows added to or removed from a specific data partition.

For example, if a sales organization partitions and distributes data based on sales territories, and the publication is enabled to optimize synchronization, the

information about how data is partitioned would be stored in before image tables at the Publisher. If sales territories shift and data needs to be repartitioned to multiple Subscribers, it will be a faster process to update and redistribute the data because the information about how data is currently partitioned is already at the Publisher.

This optimization may be useful if your application allows for the values used in row filters to change frequently for a given row. For example, if you frequently shift or realign sales territory assignments, you may gain a significant performance improvement during synchronization through this optimization.

The amount of information stored at Publisher is based on columns used to define the partition. For example, if the columns in a partition total 20 bytes and there are 10 million rows, approximately an extra 200 MB will be stored at the Publisher. If there are only 10,000 rows, 200 KB will be stored at the Publisher.

**CAUTION** Choosing to maintain this additional information at the Publisher results in an increase in the storage requirements for the merge replication tracking system tables in the **publication** database; however, if UPDATES to columns included in partitions are not atypical, the performance gains are usually worth maintaining the additional information.

#### To minimize the amount of data sent over the network

# **Replication Data Considerations**

Special considerations should be taken when publishing certain data types and properties. This section identifies those data types and properties, and it describes solutions for managing them, including:

- Identity range management. Specifying identity range management can help you control how data modifications are made at different Subscribers during merge replication or during snapshot or transactional replication with updatable subscriptions.
- Data types with specific uses. Different data types such as **uniqueidentifier** and **timestamp** have specific uses during replication processing, including conflict resolution when changes to the same data are made at multiple servers.
- NOT FOR REPLICATION. Using the NOT FOR REPLICATION option allows you to implement ranges of identity values in a partitioned environment. .

# **Using IDENTITY Values with Replication**

When you assign an IDENTITY property to a column, the system automatically generates sequential incrementing numbers for new rows inserted into a table. Because identity values are usually unique, an identity column is frequently defined as a primary key.

In replication topologies, where a publication contains an identity column and new rows can be inserted at Subscribers, additional configuration may be necessary to ensure that no duplicate identity values or constraint violations occur.

To illustrate managing identity values with replicas, suppose three rows of data from Publisher A, containing the identity values 1, 2, and 3,

are replicated to Subscriber A and Subscriber A allows inserts. If two new rows in the same article are inserted, one at Publisher A and one at Subscriber A, and no additional measures are taken by the replication agents, both rows are assigned an identity value of 4. An attempt is made by the replication agents to copy the new rows between the Subscriber and Publisher. If successful, two different rows with an identity value of 4 will exist on each replica. As a result, each published article will contain multiple rows with the same identity values. If the identity column was defined as a primary key, or with a unique constraint, the data will not replicate.

Replication provides several options to ensure the same identity values are not assigned to rows inserted at different replicas, or that a primary key constraint violation does not occur.

# **Managing Identity Values**

You can manage identity values by:

- Allowing Microsoft® SQL Server<sup>™</sup> 2000 replication to automatically manage identity columns by dynamically allocating ranges of identity values to the Publisher and all the Subscribers.
- Using the Transact-SQL NOT FOR REPLICATION option when defining the identity column.
- Using a primary key other than the identity column (for example, a composite key or a **rowguid** column), if an identity column is not necessary. This strategy eliminates the overhead of managing identity columns on the replicated data.

#### **Automatic Identity Range Handling**

The simplest way of handling identity ranges across replicas is to allow SQL Server 2000 to manage identity range handling for you. To use automatic identity range handling, you must first enable the feature at the time the publication is created, assign a set of initial Publisher and Subscriber identity range values, and then assign a threshold value that determines when a new identity range is created.

For example, assigning an identity range from 1000 through 2000 to a Publisher, and a range from 2001 through 3000 to an initial Subscriber, works as follows when combined with a threshold value of 80 percent:

- Newly inserted Publisher rows are assigned identity values from 1000 through 2000. Newly inserted rows on the initial Subscriber will sequence from 2001 through 3000.
- When 80 percent of either the Publisher identity values or the Subscriber identity values are used, a new identity range is created for

forthcoming inserts. In this example, if rows from 1001 through 1800 are used on the Publisher, the threshold has been reached. A new identity range, from 3001 through 4000, is created on the Publisher, and the next inserted row at the Publisher is assigned an identity value of 3001. After the Subscriber reaches the threshold (assuming the Subscriber reached threshold after the Publisher), a new identity range is created on the Subscriber, from 4001 through 5000, and the next inserted row at the Subscriber is assigned an identity value of 4001. The process is repeated as identity ranges are used.

• As each Subscriber is added, an identity range that is the same size as the initial Subscriber range is added, using the next available starting point.

The threshold setting avoids situations where the Subscribers run out of identity values and become unable to insert new rows until the Distribution Agent or Merge Agent synchronizes with the Subscriber. However, setting the threshold value too low can generate large numbers of unused identity values. The threshold value should be set carefully by evaluating the update frequency at the Subscriber and the synchronization schedule.

For transactional articles enabled for identity range management, the identity ranges at both the Publisher and Subscriber need to be checked and adjusted periodically. The Log Reader Agent does this at the Publisher and the Distribution Agent does this at the Subscriber.

If a Log Reader Agent or Distribution Agent is not running in continuous mode, the check and possible adjustment will be done after all the commands have been processed. When one of the agents is in continuous mode, the check and possible adjustment will be done in a time interval of 10 times the polling interval of the agent after all the commands have been processed. After the agent is started, the first check will be done as soon as the commands have been processed.

Run the Log Reader Agent or the Distribution Agent to adjust the Publisher or Subscriber when the server is out of its identity range. If the agent is running in continuous mode, you may need to restart it for the identity range to be adjusted immediately. You can also execute **sp\_adjustpublisheridentityrange** to explicitly adjust the identity range at the Publisher based on threshold value for either transactional or merge publications.

You enable automatic identity range handling:

- In SQL Server Enterprise Manager, in the **Publication Properties** dialog box.
- By setting the following options in the **sp\_addmergearticle** stored procedure.

Parameter	Values	Description
@auto_identity_range	TRUE or FALSE	Enable (TRUE) or
		disable (FALSE)
		automatic identity range
		handling.
<pre>@pub_identity_range</pre>	Integer values of range	Identity range for the
	(for example, from 1001	Publisher.
	through 2000)	
@identity_range	Integer values of range	Identity range for the
	(for example, from 2001	initial Subscriber; length
	through 3000)	of range used for
		additional Subscribers.
@threshold	Integer value for percent	Percent of total identity
	threshold (for example, 90	values used on replica
	is equivalent to 90	that trigger creation of
	percent)	new identity range.

#### **Manual Identity Range Handling**

You can also manage identity values using a check constraint and the NOT FOR REPLICATION option on the IDENTITY property of a Transact-SQL CREATE TABLE statement. Use the NOT FOR REPLICATION option to specify identity ranges programmatically, or if you are upgrading an existing instance of SQL

Server where identity ranges are already being managed through Transact-SQL statements.

Using the NOT FOR REPLICATION statement informs SQL Server 2000 that the replication process gets a waiver when supplying an explicit value and that the local identity value should not be reseeded. Each Publisher using this option receives a reseeding waiver.

The following code example illustrates how to implement identities with different ranges at each Publisher:

- At Publisher A, start at 1 and increment by 1. CREATE TABLE authors ( COL1 INT IDENTITY (1, 1) NOT
- At Publisher B, start at 1001 and increment by 1. CREATE TABLE authors ( COL1 INT IDENTITY (1001, 1) N

After activating the NOT FOR REPLICATION option, connections from replication agents to Publisher A insert rows with values such as 1, 2, 3, 4. These are replicated to Publisher B without being changed (that is, 1, 2, 3, 4). Connections from replication agents at Publisher B receive values 1001, 1002, 1003, and 1004. Those are replicated to A without being changed. When all data is distributed or merged, both Publishers have values 1, 2, 3, 4, 1001, 1002, 1003, and 1004. The next locally inserted value at Publisher A is 5. The next locally inserted value at Publisher B is 1005.

It is recommended that you always use the NOT FOR REPLICATION option along with the CHECK constraint to ensure that the identity values being assigned are within the allowed range. For example:

CREATE TABLE sales (sale\_id INT IDENTITY(100001,1) NOT FOR REPLICATION CHECK NOT FOR REPLICATION (sale\_id <= 200000), sales\_region CHAR(2), CONSTRAINT id\_pk PRIMARY KEY (sale\_id) )

Even if someone used SET IDENTITY INSERT, all values inserted locally must obey the range. However, a replication process is still exempt from the check.

#### Using Other Columns as Primary Keys

If using an identity column is not a requirement, you can eliminate the overhead of managing the uniqueness of identity values in replicated data by using another column as the primary key, or using combinations of columns as the primary key.

For example, you can define a primary key, consisting of an identity column whose values are not unique and a second column, that when combined with the identity column guarantees uniqueness (for example, a site ID column, **pk\_id\_plus\_site**). In this example, the composite key **pk\_id\_plus\_site** is a combination of the identity and site columns. In replication, duplicate identity values can be created only at different sites; therefore, each primary key value in this case will always be unique.

ROWGUIDCOL is a property you can assign to a column with **uniqueidentifier** values, a SQL Server 2000 data type that defines a 128-bit integer guaranteed to be unique. As such, using a **rowguid** column as a primary key is a safe alternative to using an identity column to guarantee uniqueness.

# **Identity Ranges with Immediate Updating and Queued Updating**

For publications that allow immediate updating but not queued updating (in snapshot replication and transactional replication), the Publisher controls identity values. You cannot assign identity ranges with this type of replication because the replication agents do not assign an IDENTITY property to the column on the Subscriber. Create the IDENTITY property at the Publisher only, and have the Subscriber use the base numeric data type (for example, **int**) with a default value of 0. These actions are taken automatically if the Distribution Agent initializes the schema and data (that is, if the synchronization type of the subscription is set to automatic). The next identity value is always generated at the Publisher and assigned to the row inserted at the Subscriber.

With queued updating, identity values must be assigned by the Subscriber because newly inserted rows at the Subscriber may be sent to a queue rather than directly to the Publisher. Because the data is sent asynchronously, there is no mechanism for the Publisher to assign an identity value immediately to a newly inserted row at the Subscriber, as there is for the immediate updating case.

For publications that allow immediate updating with queued updating as a failover option, assign identity ranges to Subscribers either automatically or manually. Inserted rows at a Subscriber will generate identity values from the assigned local identity range. The new Subscriber row will be sent to the queue, where it will be picked up by the Queue Reader Agent and applied to the Publisher with the correct (not reseeded) identity value.

#### **Managing Replicated timestamp Data**

Microsoft® SQL Server<sup>™</sup> 2000 **timestamp** data refers to database-specific incrementing binary numbers that indicate the relative sequence in which data modifications take place in a database; **timestamp** data is unrelated to both chronological time and calendar date.

A **uniqueidentifier** data type column is used to detect conflicts for this replication type; **timestamp** data is no longer used for conflict detection. For information about upgrade issues associated with this change, see <u>Replication</u> and <u>Upgrading</u>.

The literal values for a **timestamp** column are replicated, but the data type for the replicated values is changed to **binary (8)** on the Subscriber.

For merge replication and queued updating Subscriber (including immediate updating with queued updating as a failover) articles containing a **timestamp** column, the **timestamp** column is replicated, but the literal **timestamp** values are not. The **timestamp** values are regenerated at initial synchronization time when the rows are applied at the Subscriber. This allows **timestamps** to be used by client applications at the Subscriber for functions such as optimistic locking. In those cases, the ODBC driver, OLE DB provider, DB-Library cursor, or server cursor used by the application to implement optimistic locking compares the **timestamp** value of the row being updated with the current value of the original row. If the **timestamp** values are different, indicating the row has changed, the application can take appropriate action (rolling back the transaction, rereading the data, and so on).

The processing of **timestamp** data has implications for the detection of conflicts. For a conflict to occur with row-level tracking, the same row must be updated at both replicas. For a conflict to occur with column-level tracking, the same column within the same row must be updated at both replicas. Because **timestamp** values change whenever a row is updated, the distinction between row-level and column-level tracking would disappear with the presence of a **timestamp** column, unless special measures were taken. With column tracking turned on, every time updates were made at both locations, even to different columns within the rows, both **timestamp** values would change, and a columnlevel conflict would be flagged. Effectively, column-level tracking would always work the same as row-level tracking, and no merging of data updated in different columns could take place.

Merge replication tracking resolves this problem by ignoring **timestamp** values The queued updating Subscribers option for transactional replication uses only row-level tracking to detect conflicts, so this is not an issue. For example, suppose a merge replication table contains four columns: a **uniqueidentifier** column, an **integer** column, a **character** column, and a **timestamp** column. The value for the **integer** column of row 1 on the Publisher is updated, and the value for the **character** column of row 1 is changed on the Subscriber. When columnlevel tracking is turned on, the data merges without a conflict. If the Merge Agent did not ignore the **timestamp** values with column level tracking turned on (with row-level tracking, a conflict would have been detected in any case), this non-conflicting update would have been flagged as a conflict, and the data would not have been merged correctly. Therefore, the Merge Agent does not compare the **timestamp** columns and does not take any action if their values changed.

# **Using NOT FOR REPLICATION**

The NOT FOR REPLICATION option is used by Microsoft® SQL Server<sup>TM</sup> 2000 replication to implement ranges of identity values in a partitioned environment. The NOT FOR REPLICATION option is especially useful in transactional or merge replication when a published table is partitioned with rows from various sites.

When a replication agent connects to a table with any login, all of the NOT FOR REPLICATION options on the table are activated. When the option is set, SQL Server 2000 maintains the original identity values on rows added by the replication agent but continues to increment the identity value on rows added by other users. When a user adds a new row to the table, the identity value is incremented in the normal way. When a replication agent replicates that row to a Subscriber, the identity value is not changed when the row is inserted in the Subscriber table.

For example, consider a table that contains rows inserted from two sources: Publisher A and Publisher B. The rows inserted at Publisher A are identified by increasing values from 1 through 1000, and those rows at Publisher B are identified by values from 1001 through 2000. If a process at Publisher A inserts a row locally into the table, SQL Server assigns the first row a value of 1, the next row a value of 2, and so forth, in automatically increasing increments. Similarly, if a process at Publisher B inserts a row locally into the table, the first row is assigned a value of 1001, the next row a value of 1002, and so forth. When rows at Publisher A are replicated to B, the identity values remain 1, 2, and so forth, but local seed values at B are not reset.

Regardless of its role in replication, the IDENTITY property does not enforce uniqueness by itself, but merely inserts the next value. Although you can provide an explicit value using SET IDENTITY INSERT, that function is not appropriate for replication because it also reseeds the value. The NOT FOR REPLICATION option was created specifically for applications using replication. For example, without this option, as soon as the first row from Publisher B (with value 1001) is propagated to Publisher A, Publisher A's next value would be 1002. The NOT FOR REPLICATION option is a way of telling SQL Server 2000 that the replication process gets a waiver when supplying an explicit value and that the local value should not be reseeded. Each Publisher using this option gets the same reseeding waiver.

Custom stored procedures that use INSERT, UPDATE, and DELETE statements with full column lists are required before replication will work with identity properties. If full column lists are not used, an error will be returned.

The following code example illustrates how to implement identities with different ranges at each Publisher:

- At Publisher A, start at 1 and increment by 1. CREATE TABLE authors ( COL1 INT IDENTITY (1, 1) NOT
- At Publisher B, start at 1001 and increment by 1. CREATE TABLE authors ( COL1 INT IDENTITY (1001, 1) N

After activating the NOT FOR REPLICATION option, connections from replication agents to Publisher A insert rows with values such as 1, 2, 3, 4. These are replicated to Publisher B without being changed (that is, 1, 2, 3, 4). Connections from replication agents at Publisher B get values 1001, 1002, 1003, 1004. Those are replicated to A without being changed. When all data is distributed or merged, both Publishers have values 1, 2, 3, 4, 1001, 1002, 1003, 1004. The next locally inserted value at Publisher A is 5. The next locally inserted value at Publisher B is 1005.

It is advisable to always use the NOT FOR REPLICATION option along with the CHECK constraint to ensure that the identity values being assigned are within the allowed range. For example:

```
CREATE TABLE sales
(sale_id INT IDENTITY(100001,1)
NOT FOR REPLICATION
CHECK NOT FOR REPLICATION (sale_id <= 200000),
sales_region CHAR(2),
CONSTRAINT id_pk PRIMARY KEY (sale_id)
)
```

Even if someone used SET IDENTITY INSERT, all values inserted locally must

obey the range. However, a replication process is still exempt from the check.

**Note** If you are using transactional replication with the immediate-updating Subscribers option, do not use the IDENTITY NOT FOR REPLICATION design. Instead, create the IDENTITY property at the Publisher only, and have the Subscriber use just the base data type (for example, **int**). Then, the next identity value is always generated at the Publisher.

## **Administering and Monitoring Replication**

Microsoft® SQL Server<sup>™</sup> 2000 replication provides tools to administer and monitor replication agents, replication alerts, and replication processes, ensuring that replication meets the needs of your organization.

Monitoring replication helps you:

- Set the agent profiles, schedules, properties, and notifications for replication agents.
- View and troubleshoot agent activity, including verifying when agents last ran, monitoring agent activity, and analyzing replication performance.
- Receive notification through a replication alert when an event occurs on a replication agent.
- Validate subscriptions to ensure that data values are the same at the Publisher and at Subscribers.
- Reinitialize one or all subscriptions to a publication as needed.
- Manage replication agents from a central location.

## **Tools for Administering and Monitoring Replication**

To administer and monitor agents, you can use Replication Monitor in SQL Server Enterprise Manager, command prompt utilities, Transact-SQL system stored procedures, or you can use Microsoft® ActiveX® controls for replication or SQL-DMO objects that are programmable in languages such as Microsoft Visual Basic® and Microsoft Visual C++®. Windows NT Performance Monitor or Windows 2000 System Monitor can be used to monitor the rate at which various replication processes occur.

## **Replication Monitor**

Replication Monitor is a component of SQL Server Enterprise Manager designed for viewing the status of replication agents and troubleshooting potential problems at the Distributor. Replication Monitor shows up as a node in SQL Server Enterprise Manager under the server that is enabled as a Distributor when the user is a member of the **sysadmin** fixed server role. Replication Monitor Group can also appear as a top-level node in Enterprise Manager for a central location where you can monitor and administer multiple Distributors.

Additionally, users that are not members of the **sysadmin** fixed server role can monitor replication if they are defined with the **replmonitor** role in the distribution database. A system administrator can add any user to the **replmonitor** role, which allows that user to view replication activity with the Replication Monitor node in SQL Server Enterprise Manager; however, the user will not be able to administer replication.

Users may only be part of the **replmonitor** role in only one distribution database. They will only be able to view agents in the distribution database in which they are part of the **replmonitor** role. However, the status shown on the Replication Monitor node reflects all agents. So there may be times when the user sees an error status (red X on the Replication Monitor node, Publishers or Agents folders), but none of the agents they can view show an error status. This indicates that an agent this user is not able to see is showing an error status.

For example, a user added as part of the **replmonitor** role can view agent history, errors, and analyze error details, but they would not be able to change agent profiles, agent schedules, and so on. Although this user would be able to view a list of publications, the user would only be able to view properties for the publications that include the user in the Publication Access List (PAL).

You can use Replication Monitor to:

- View a list of Publishers, publications, and subscriptions to the publications that are supported by the Distributor.
- View scheduled replication agents, and to monitor real-time status and history for each agent.

- Set up and monitor alerts related to replication events.
- Administer agents and subscriptions including starting and stopping agents and reinitializing subscriptions.

After replication has been configured, you can also use the Microsoft Windows NT® or Windows 2000 Event Viewer to view SQL Server<sup>™</sup> messages. For information about Event Viewer, see Microsoft Windows NT 4.0 Help or Windows® 2000 Help.

#### **To Enable Replication Monitor Group**

- 1. In Enterprise Manager, right-click on a SQL Server, and click **Properties**.
- 2. On the Replication tab, select **Show Replication Monitor Group**. Optionally, you can also select to add this server as a Distributor in the Replication Monitor Group.

## **Replication Agent Utilities**

You can use the replication command prompt utilities to configure and start replication agent activity. Command prompt utilities are installed automatically with Microsoft® SQL Server<sup>™</sup> 2000. The replication agent files are located under \Microsoft SQL Server\80\Com. This table lists the replication utility names and file names.

Command Prompt Utility	File Name
Replication Snapshot Agent Utility	snapshot.exe
Replication Distribution Agent Utility	distrib.exe
Replication Log Reader Agent Utility	logread.exe
Replication Queue Reader Agent Utility	qrdrsvc.exe
Replication Merge Agent Utility	replmerg.exe

**Note** You can modify agent settings by changing the command line available for each agent when administering replication agents through Replication Monitor. To access that command line, right-click a specific agent, click **Agent Properties**, click the **Steps** tab, and then double-click the **Run Agent** step.

For more information, see Getting Started with Command Prompt Utilities.

### **ActiveX Controls for Replication**

Microsoft® ActiveX® controls allow custom applications to configure and invoke replication agent functionality. The controls support all types of subscriptions and can be monitored using SQL Server Enterprise Manager at the Distributor.

Programmers can use ActiveX controls for replication, similar to any standard built-in control. The controls provided are the SQL Snapshot control, the SQL Distribution control, and the SQL Merge control.

Benefits of using ActiveX controls for replication are:

- The client has no dependency on SQL Server Agent, which is responsible for executing jobs in addition to replication.
- If you start a replication agent using SQL Server Agent, other jobs can also run.
- If you are replicating to heterogeneous Subscribers using pull or anonymous subscriptions, SQL Server Agent is not available at the Subscriber.
- ActiveX replication controls can be invoked from many programming environments, including Microsoft Visual Basic®, Visual Basic Scripting Edition, and Microsoft Visual C++®.
- The application can control when replication should take place. For example, you can program a command on a menu or a Web page that uses the replication ActiveX controls.
- An application can use the ActiveX Controls status callback handlers to place a progress bar to provide feedback on the progress of the replication control.

- An application can determine how to obtain login information for running the replication agents automatically (for example, hard-coded or interactive).
- Embedding replication controls in applications provides a way to distribute mobile applications without the complexity of Subscriber setup.
- Controls can be programmed to add, drop, reinitialize, or validate subscriptions, and create or attach databases at the Subscriber.
- An application can be programmed to register the synchronization of a subscription in Microsoft Synchronization Manager.

If a subscription is registered in Microsoft Windows Synchronization Manager, there is no need to embed the controls in the application. All synchronization is then controlled by this central application. Windows Synchronization Manager does not, however, allow you to specify some of the custom properties of a subscription, such as its hostname override and subscription agent settings including FTP. ActiveX Controls are useful for these administrative activities.

For more information, see <u>Developing Replication Applications Using ActiveX</u> <u>Controls</u>.

#### Windows NT Performance Monitor and Windows 2000 System Monitor

Windows 2000 System Monitor and Windows NT Performance Monitor can be used to monitor the rate at which various replication processes are running. Using Performance Monitor or System Monitor, you can use charts and reports to gauge the efficiency of your computer, identify and troubleshoot possible problems (such as unbalanced resource use, insufficient hardware, or poor program design), and plan for additional hardware needs. You can optimize replication performance by using the relevant replication counters.

For more information, see Performance Monitor and System Monitor documentation included with Windows NT and Windows 2000.

#### **Setting Agent Parameters**

Each replication agent supports a set of run-time parameters that you can use to control how the agent runs. For information about parameters available for each agent, see the documentation for replication command prompt utilities.

For example, a parameter that can be helpful when troubleshooting replication agent activity is the **–output** parameter. This parameter writes all actions that occur and error messages for a particular agent to a text file.

The parameters can be set through:

- The command line of the agent job step titled 'Run agent'.
- The properties of a Microsoft® ActiveX® component.
- A centralized agent profile.
- The agent command prompt utility.

#### See Also

Replication Distribution Agent Utility Replication Log Reader Agent Utility Replication Merge Agent Utility Replication Queue Reader Agent Utility Replication Snapshot Agent Utility

#### **Agent Profiles**

When a replication agent is created, it is associated with an agent profile that is maintained at the Distributor. The agent profile contains a set of parameters to be used each time the agent runs. During the startup process, each agent logs in to the Distributor and queries for the parameters in its profile.

The agent profile allows you to change key parameters easily for all agents associated with that profile. For example, if you have 20 Snapshot Agents and need to change the query time-out value, you can update the profile used by the Snapshot Agents and all agents of that type will begin using the new value automatically the next time they are run. You also may have different profiles for different instances of an agent. For example, a Distribution Agent that uses Remote Access Service (RAS) to connect to the Distributor could use a set of parameters that are better suited to the slower communications link.

A set of default and predefined profiles for each agent type is installed when a server is configured as a Distributor. If a specific profile is not associated with an agent, SQL Server Agent uses the default profile for that type of agent.

**Note** The values set in the agent profile are overridden by any values set for the same parameter in the agent command prompt utility.

#### To create a replication agent profile

# **Replication Agents**

The replication agents carry out many of the tasks associated with replication including creating copies of schema and data, detecting updates at the Publisher or Subscriber, and propagating changes between servers. Each replication agent has an agent profile associated with it, agent properties that you can set, an agent schedule, and an agent history.

Replication Monitor provides the capability to administer replication agent activity graphically. You can view a list of all the Snapshot, Log Reader, Distribution, Queue Reader, or Merge Agents supported by a Distributor. You can select a Distributor and click Replication Monitor to display a list of agents. When you click the folder for a specific type of agent, all the agents of that type on the Distributor are displayed. You can then view the detailed activity of a specific agent.

#### **Independent and Shared Agents**

An independent agent is an agent that services one subscription. Latency is reduced when using independent agents because it is ready whenever the subscription needs to synchronize.

A shared agent services multiple subscriptions, and is the default for snapshot replication or transactional replication. When multiple subscriptions using the same shared agent need to synchronize, they wait in a queue, and the shared agent services them one at a time.

All of the agents used during merge replication are independent agents. When using independent agents with snapshot replication or transactional replication, you must take care to prevent transactions that have interdependencies from being delivered out of sequence.

# **SQL Server Agent**

SQL Server Agent hosts and schedules the agents used in replication, and provides an easy way to run replication agents. When choosing to make a trusted connection, the replication agents run under the security context of the SQL Server Agent startup account. SQL Server Agent also controls and monitors several other operations outside of replication including monitoring the SQLServerAgent service, maintaining error logs, running jobs, and starting other processes.

### **Snapshot Agents**

The Snapshot Agent is used with all types of replication. It prepares schema and initial data files of published tables and stored procedures, stores the snapshot files, and inserts information about initial synchronization in the distribution database. The Snapshot Agent typically runs under SQL Server Agent at the Distributor and can be administered using SQL Server Enterprise Manager or the ActiveX® Snapshot Control. There is one Snapshot Agent per publication.

# **Snapshot Agent Profile**

A default profile for the Snapshot Agent is installed when a server is configured as a Distributor. The default profile contains the following parameters and values.

	Default	
Parameter	Value	Description
-BcpBatchSize	100000	When performing a <b>bcp</b> in operation, the batch size is the number of rows to send to the server as one transaction and is also the number of rows that must be sent before the Distribution Agent logs a <b>bcp</b> progress message. When performing a <b>bcp out</b> operation, a fixed batch size of 100,000 is used. A value of 0 indicates no message logging.
- HistoryVerboseLevel	2	The amount of history logged during a snapshot operation can be: 1 = Always update a previous history message of the same status (startup, progress, success, and so forth). If no previous record with the same status exists, insert a new record. 2 = Insert new history records. If the record is for items such as idle messages or long-running job messages, update the previous records. 3 = Always insert new records, unless it is for idle messages. You can minimize the performance effect of history logging by specifying 1.
-LoginTimeOut	15	The number of seconds before the login

		attempted by the agent times out.	
-MaxBcpThreads	1	The number of bulk copy operations that	
		can be performed in parallel. The	
		maximum number of threads and ODBC	
		connections that exist simultaneously is the	
		lesser of <b>MaxBcpThreads</b> or the number	
		of bulk copy requests that appear in the	
		synchronization transaction in the	
		distribution database. MaxBcpThreads	
		must be greater than zero, and has no hard-	
		coded upper limit.	
-QueryTimeOut	300	The number of seconds before the queries	
		issued by the agent time out.	

# **Distribution Agents**

The Distribution Agent is used with snapshot replication and transactional replication. It moves the snapshot files and incremental changes held in the distribution database to Subscribers. The Distribution Agent typically runs under SQL Server Agent at the Distributor for push subscriptions or at the Subscriber for pull subscriptions. It can be administered using SQL Server Enterprise Manager or the ActiveX® Distribution Control. There will either be one Distribution Agent per subscription (an independent agent) or one Distribution Agent per publication database and subscription database pair (a shared agent).

### **Distribution Agent Profile**

A default profile for the Distribution Agent is installed when a server is configured as a Distributor. The default profile contains the following parameters and values.

	Default	
Parameter	Value	Description
-BcpBatchSize	100000	The number of rows to send in a bulk copy operation. When performing a <b>bcp</b> in operation, the batch size is the number of rows to send to the server as one transaction, and is also the number of rows that must be sent before the Distribution Agent logs a <b>bcp</b> progress message. When performing a <b>bcp out</b> operation, a fixed batch size of 1000 is used. A value of 0 indicates no message logging.
-CommitBatchSize	100	The number of transactions to be issued to the Subscriber before a COMMIT statement is issued.
-CommitBatchThreshold	1000	The number of replication commands to be issued to the Subscriber before a COMMIT statement is issued.
-HistoryVerboseLevel	1	The amount of history logged during a distribution operation can be: 1 = Always update a previous history message of the same status (startup, progress, success, and so forth). If no previous record with the same status exists, insert a new record. 2 = Insert new history records unless

		<ul> <li>the record is for such things as idle</li> <li>messages or long-running job</li> <li>messages, in which case update the</li> <li>previous records.</li> <li>3 = Always insert new records, unless</li> <li>it is for idle messages.</li> <li>You can minimize the performance</li> <li>effect of history logging by selecting</li> <li>1.</li> </ul>
-LoginTimeOut	15	The number of seconds before the login attempted by the agent times out.
-MaxBcpThreads	1	The number of bulk copy operations that can be performed in parallel. The maximum number of threads and ODBC connections that exist simultaneously is the lesser of <b>MaxBcpThreads</b> or the number of bulk copy requests that appear in the synchronization transaction in the distribution database. <b>MaxBcpThreads</b> must have a value greater than zero, and has no hard- coded upper limit.
- MaxDeliveredTransactions	0	The maximum number of push or pull transactions applied to Subscribers in one synchronization. A value of 0 indicates that the maximum is an infinite number of transactions. Other values can be used by Subscribers to shorten the duration of a synchronization being pulled from a Publisher.
-PollingInterval	10	Number of seconds the distribution database is queried for replicated

		transactions.
-SkipErrors		The error number(s) that will be
		skipped. The Distribution Agent will
		ignore the error number(s) indicated
		and continue processing according to its schedule.
-SkipFailureLevel	1	The Distribution Agent is enabled to skip errors. A value of 0 indicates that
		the Distribution Agent will not ignore any errors.
-QueryTimeOut	300	The number of seconds before the queries issued by the agent time out.
-TransactionsPerHistory	100	The transaction interval for history logging. If the number of committed transactions after the last instance of history logging is greater than this option, a history message is logged. A value of 0 indicates infinite <b>TransactionsPerHistory</b> .

# **Log Reader Agents**

The Log Reader Agent is used with transactional replication. It moves transactions marked for replication from the transaction log on the Publisher to the distribution database. Each database that is marked for transactional replication will have one Log Reader Agent that runs on the Distributor and connects to the Publisher.

### **Log Reader Agent Profile**

A default profile for the Log Reader Agent is installed when a server is configured as a Distributor. The default profile contains the following parameters and values.

	Default	
Parameter	Value	Description
-HistoryVerboseLevel	1	The amount of history logged during a log reader operation can be:
		<ul> <li>1 = Always update a previous history message of the same status (startup, progress, success, and so forth). If no previous record with the same status exists, insert a new record.</li> <li>2 = Insert new history records unless the record is for such things as idle messages or long-running job messages, in which case update the previous records.</li> </ul>
		You can minimize the performance effect of history logging by selecting 1.
-LoginTimeOut	15	The number of seconds before the login attempted by the agent times out.
-PollingInterval	10	The number of seconds the log is queried for replicated transactions.
-QueryTimeOut	300	The number of seconds before the queries issued by the agent times out.
-ReadBatchSize	500	The maximum number of transactions read out of the source. For the Log Reader Agent, the source is the transaction log of the publishing database.
-	100	The number of replication commands to

ReadBatchThreshold	be read from the transaction log before
	being issued to the Subscriber by the
	Distribution Agent.

### **Queue Reader Agents**

The Queue Reader Agent is used with snapshot replication or transactional replication with the queued updating option, or if the immediate updating with queued updating as a failover option is enabled.

The Queue Reader Agent is a multithreaded agent that runs on the Distributor. It is responsible for taking messages from a queue and applying them to the appropriate publication.

The Queue Reader Agent reads messages from the Microsoft® SQL Server<sup>™</sup> 2000 queue on each Subscriber and applies the transactions to the publication. This agent uses the security context of SQL Server Agent by default. Unlike the Distribution Agent and the Merge Agent, only one instance of the Queue Reader Agent exists to service all Publishers and publications for a given Distributor.

### **Merge Agents**

The Merge Agent is used with merge replication. It applies the initial snapshot at the Subscriber, and moves and reconciles incremental data changes that occurred after the initial snapshot was created. Each merge subscription has its own Merge Agent that connects to and updates both the Publisher and the Subscriber. The Merge Agent typically runs under SQL Server Agent at the Distributor for push subscriptions or at the Subscriber for pull subscriptions. It can be administered using SQL Server Enterprise Manager or the ActiveX® Merge Control.

# **Merge Agent Profile**

A default profile for the Merge Agent is installed when a server is configured as a Distributor. The default profile contains the following parameters and values.

_	Default	
Parameter	Value	Description
-BcpBatchSize	100000	The number of rows to send a bulk copy operation. When performing a <b>bcp</b> in an operation while applying the schema changes, the Merge Agent uses the batch size to determine when to log a progress message. A value of 0 indicates no message logging.
-ChangesPerHistory	100	The threshold beyond which upload and download messages are logged.
-DownloadGenerationsPerBatch	100	The number of generations to be processed in one batch while downloading changes from the Publisher to the Subscriber. A generation is defined as a logical group of changes per article. The default for an unreliable communication link is 10. In all cases, however, the actual number of generations processed per batch will be equal to the greater of the UploadGenerationsPerBatch

		setting or the number of articles published plus 1.
- DownloadReadChangesPerBatch	100	The number of changes to be read in one batch while downloading changes from the Publisher to the Subscriber.
- DownloadWriteChangesPerBatch	100	The number of changes to be applied in one batch while downloading changes from the Publisher to the Subscriber.
-FastRowCount	1	Specifies what type of rowcount calculation method should be used for rowcount validation. A value of 1 (default) indicates the fast method. A value of 0 indicates the full rowcount method.
-HistoryVerboseLevel	1	The amount of history logged during a merge operation can be: 1 = Always update a previous history message of the same status (startup, progress, success, and so forth). If no previous record with the same status exists, insert a new record. This level logs the minimum number of messages. 2 = Insert new history records unless the record is for such things as idle messages or

		long-running job messages, in which case update the previous records. This level logs level 1 messages plus additional in-progress messages. 3 = Always insert new records, unless it is for idle messages. You can minimize the performance effect of history logging by setting the <b>ChangesPerHistory</b> parameter.
-KeepAliveMessageInterval	300	The number of seconds before history thread checks if any of the existing connections is waiting for a response from the server. This value can be increased to avoid getting the agent marked as suspect by the checkup agent when executing a long-running batch.
-LoginTimeOut	15	The number of seconds before the login attempted by the agent times out.
-MaxDownloadChanges	0	The maximum number of changes you want to download during a specific merge session. Because complete generations are processed, the number of rows downloaded may go over the specified maximum.
-MaxUploadChanges	0	The maximum number of

		changes you want to upload during a specific merge session. Because complete generations are processed, the number of rows uploaded may go over the specified maximum.
-NumDeadlockRetries	5	The number of times the merge process attempts to retry an internal operation when it encounters a deadlock error. Can be any value between 1 and 100.
-PollingInterval	60	The number of seconds the Publisher or Subscriber is queried for data changes when in continuous mode.
-QueryTimeOut	300	The number of seconds before the queries issued by the agent times out.
-StartQueueTimeout	0	If the number of merge processes running is at the limit, this indicates the maximum number of seconds that the Merge Agent waits. If the maximum number of seconds is reached and the Merge Agent is still waiting, it will exit. A value of '0' means that the agent waits indefinitely, although it can be cancelled.
-UploadGenerationsPerBatch	100	The number of generations to be processed in one batch while uploading changes from the Subscriber to the

		Publisher. A generation is defined as a logical group of changes per article. The default for an unreliable communication link is 1. In all cases, however, the actual number of generations processed per batch will be equal to the greater of the UploadGenerationsPerBatch setting or the number of articles published plus 1.
-UploadReadChangesPerBatch	100	The number of changes to be read in one batch while uploading changes from the Subscriber to the Publisher.
-UploadWriteChangesPerBatch	100	The number of changes to be applied in one batch while uploading changes from the Subscriber to the Publisher.
-Validate	0	Specifies if validation should be done at the end of the merge session, and, if so, what type of validation. A value of 0 (default) indicates no validation. A value of 1 indicates rowcount-only validation. A value of 2 indicates rowcount and checksum validation. A value of 3 indicates binary checksum validation (available only with SQL Server 2000).
-ValidateInterval	60	The number of minutes the subscription is validated when

Se	et to continuous mode.
----	------------------------

### **Miscellaneous Agents**

The Miscellaneous Agents folder in Replication Monitor lists the agents needed to clean up and monitor different replication processes.

### **Agent History Clean Up Agent**

The Agent History Clean Up Agent removes replication agent history from the distribution database. This agent runs every 10 minutes by default. Running this agent is helpful in managing the size of the distribution database.

### **Distribution Clean Up Agent**

The Distribution Clean Up Agent removed replicated transactions from the distribution database. This agent runs for snapshot and transactional publications every 72 hours by default. The Distribution Clean Up Agent may deactivate a subscription if the subscription has not been synchronized within the maximum distribution retention period. For more information, see <u>Subscription</u> <u>Deactivation and Expiration</u>.

### **Expired Subscription Clean Up Agent**

Detects and removes expired subscriptions from the published databases. If a subscription is deactivated, the subscription will be removed by the Expired Subscription Clean Up Agent, which runs once a day by default. A subscription is marked as expired either during the cleanup process or when the replication agent runs after the publication retention period has been exceeded. For more information, see <u>Subscription Deactivation and Expiration</u>.

### **Reinitialize Subscriptions Having Data Validation Failures Agent**

Reinitializes all subscriptions that have data validation failures. This agent is not set on a schedule by default. Run this agent to automatically detect the subscriptions that failed validation and mark them for reinitialization. After the subscriptions are marked for reinitialization, the next time the Merge Agent or Distribution Agent runs, a new snapshot will be applied at the Subscribers.

### **Replication Agents Checkup Agent**

Detects replication agents that are not actively logging history. This agent runs every 10 minutes by default, and it writes to the Windows event log if the job step fails.

### See Also

Anonymous Subscriptions
Planning for Transactional Replication
Subscription Deactivation and Expiration

# **Viewing Agent History**

The Replication Monitor **Agent History** dialog box displays a summary of the sessions of a selected agent. This is helpful when you need to examine recent agent activity, gauge performance quickly, or detect error trends. The amount of history information stored for a replication agent is determined by the distribution retention period and how often the History Clean Up Agent runs.

Agent history also includes several predefined filters on (or views of) session history, such as:

- All sessions.
- Sessions in the last 24 hours.
- Sessions in the last two days.
- Sessions in the last seven days.
- Sessions with errors.

The following history columns are displayed in the **Agent History** dialog box.

Column	Values
Status	Success icon; Error icon; In Progress icon (only
	one session can be in progress at a given time).
#Actions	Number of actions in each session.
Action Message	If the session ended in an error, the highest level
	error reported.
Start Time	Time this session was started.
End Time	Time this session ended.
Duration	Duration of the agent session.
Delivery Rate	Ratio of delivered commands to the duration of the
	agent. If the agent is still running, this value reflects

	a cumulative count from the beginning of the
	session.
Latency	Latency between when an action occurs at the Publisher and is propagated to the Subscriber. If the agent is still running, this value reflects a cumulative count from the beginning of the session. Not available for Snapshot Agent or Merge Agent.
# Trans	Total number of transactions delivered during the agent session. Not available for Merge Agent.
# Cmds	Total number of commands delivered during the agent session. Not available for Merge Agent.
Publisher_Inserts	Number of inserts that occurred on the Publisher. Available only for Merge Agent.
Publisher_Updates	Number of updates that occurred on the Publisher. Available only for Merge Agent.
Publisher_Deletes	Number of deletes that occurred on the Publisher. Available only for Merge Agent.
Publisher_Conflicts	Number of conflicts that occurred on the Publisher. Available only for Merge Agent.
Subscriber_Inserts	Number of inserts that occurred on Subscribers. Available only for Merge Agent.
Subscriber_Updates	Number of updates that occurred on Subscribers. Available only for Merge Agent.
Subscriber_Deletes	Number of deletes that occurred on Subscribers. Available only for Merge Agent.
Subscriber_Conflicts	Number of conflicts that occurred on Subscribers. Available only for Merge Agent.

# **Handling Agent Errors**

You can monitor details about the current activity and the task history of each replication agent in Replication Monitor. As an agent operates, it writes details of its activity and messages to the history table in the Distributor.

You can display errors if the agent has encountered any during an agent session. The error details are displayed in the **Error Detail** dialog box. You can also display error information in the **Session Details** dialog box, or right-click an agent and then click **Error Details**.

The replication agent error status is represented in SQL Server Enterprise Manager as an icon at each agent and each node under Replication Monitor. To have the icon correctly reflecting the status of the replication agents, you must refresh the node manually or enable automatic refreshing.

You can use the **Refresh Rate and Settings** dialog box to:

- Enable or disable automatic refreshing for the console tree or details pane.
- Specify the refresh period in seconds for the console tree or details pane.
- Specify the inactivity threshold for restarting replication agents.
- Specify the Windows NT Performance Monitor or Windows 2000 System Monitor file setting for replication performance.

You can also customize the columns displayed in the details pane when the selection is on a publication or an agent view. Use the **Select Columns** dialog box to select which columns to display.

You can use the **Select Columns** dialog box to select columns to display when the focus is on any of these nodes:

• A transactional, snapshot, or merge publication

- Snapshot Agent
- Log Reader Agent
- Distribution Agent
- Merge Agent

#### **Skipping Errors in Transactional Replication**

During transactional replication, you can specify errors that can be skipped during the distribution process. Typically, when the Distribution Agent is running in continuous mode and it encounters an error, the agent, and the distribution process, stops. By specifying expected errors or errors that you do not want to interfere with replication, the agent will log the error information and then continue running.

The most typical way to skip errors is using the Distribution Agent profile titled Continue On Data Consistency Errors. To use this profile, right-click on the Distribution Agent, click Profiles, and then select this profile. The Distribution Agent will then skip errors 2601, 2627, and 20598. You can also create your own agent profile and specify the –SkipErrors parameter with the errors you want skipped. For more information on creating profiles, see <u>Agent Profiles</u>.

**CAUTION** Under typical replication processing, you should not experience any errors that need to be skipped. The ability to skip errors during transactional replication is available for the unique circumstances where you expect errors and do not want them to affect replication (for example, when failing over to a secondary Publisher during log shipping). Skipping errors should only be used with caution and with the understanding of what the error is, why it is occurring, and why it needs to be skipped rather than solved.

You can specify the errors that should be skipped using the SQL Distribution ActiveX® Control, in an agent profile (with the **–SkipErrors** parameter on the Distribution Agent profile), or by using the **–SkipErrors** parameter in the

command line for the Distribution Agent.

For example, if the Distribution Agent returns a duplicate key violation error, but you would want the distribution process to continue and log only the error information, you can specify the **–SkipErrors** command line parameter with the number of the error that should be skipped.

Typically, the Distribution Agent is a shared agent servicing multiple publications and multiple articles. If you specify **–SkipErrors** on the agent, all publications that use that Distribution Agent will be affected. If you want to skip an error on one specified publication, set up the publication with an independent agent and then specify the **–SkipErrors** command line parameter for that agent.

#### To specify the –SkipErrors parameter on the agent command line

- At the Distributor, expand Replication Monitor, click the Distribution Agents folder, right-click an agent, and then click Agent Properties.
- 2. On the **Steps** tab, double click the Run agent step.
- 3. In the command text box, type **–SkipErrors** and specify the error numbers that you want skipped if this agent encounters them (errors are delimited; list them with colons between each error number).

#### To change replication monitor refresh rate and settings

## **Remote Agent Activation**

Remote agent activation allows you to reduce the amount of processing on the Distributor or Subscriber by running the Distribution Agent or Merge Agent on another computer and then activating that agent remotely using Distributed Component Object Model (DCOM).

You can implement remote agent activation on either push or pull subscriptions. With each type of subscription, you need to:

- Indicate where the agent will run in the Push Subscription or Pull Subscription Wizard.
- Configure DCOM to activate an agent remotely.
- Configure or create the subscription indicating where the agent should run.

It is recommended that you set up regular push or pull subscriptions before configuring remote agent activation. You are not able to configure remote agent activation on a local computer (for example, when the Subscriber and Distributor reside on the same computer).

Remote agent activation is supported on Microsoft® SQL Server<sup>™</sup> 2000 running on Microsoft Windows NT® 4.0 or Windows® 2000, but it is not supported on Windows 98.

#### **Remote Agent Activation and Push Subscriptions**

When Distributor and Subscriber servers have a reliable, continuous connection, push subscriptions allow centralized subscription management. Push subscriptions offer an advantage for organizations that want to control who is allowed to subscribe to publications and when. Push subscriptions are also helpful for circumstances in which the Subscriber needs updates sent to them as soon as they occur.

For push subscriptions, the Distribution Agent (used in snapshot replication or

transactional replication) or the Merge Agent (used in merge replication) runs on the Distributor; however, the Distributor can become overloaded as the number of push subscriptions increases.

Remote agent activation allows you to offload agent activity to the Subscriber, which reduces the amount of processing on the Distributor. Using DCOM, you can activate the agent remotely and increase the number of push subscriptions the Distributor can handle.

Using DCOM for remote agent activation with push subscriptions, the Distributor first establishes a connection to the Subscriber. After the connection is made, SQL Server Agent on the Distributor uses DCOM to activate the Distribution Agent or the Merge Agent on the Subscriber.

### **Remote Agent Activation and Pull Subscriptions**

Pull subscriptions offer the ability to manage subscription synchronization locally. This is important for:

- Anyone who travels and needs to connect and synchronize data on demand.
- Remote offices that need to manage subscription synchronization because they do not have a reliable, continuous connection to the Publisher or Distributor.

For pull subscriptions, the Distribution Agent or the Merge Agent runs on the Subscriber. You can reduce processing at the Subscriber by offloading the Distribution Agent or the Merge Agent activity to the Distributor and using DCOM to activate the agent.

Using DCOM for remote agent activation with pull subscriptions, the Subscriber first establishes a connection to the Distributor. After the connection is made, SQL Server Agent on the Subscriber uses DCOM to activate either the Distribution Agent or the Merge Agent on the Distributor.

#### **Subscription Security Requirements**

When you create a subscription, the Distribution Agent or the Merge Agent runs

under the security context of SQL Server Agent. Using the security context of SQL Server Agent, the Distribution Agent establishes a connection to the Subscriber and to the Distributor, and when required, to the snapshot folder. The Merge Agent establishes a connection to the Subscriber, the Distributor, the Publisher, and when required, to the snapshot folder. You can view the security requirements for a subscription as if SQL Server Agent were making all of the connections.

After an agent is activated on a remote computer, the agent will be run under the security context as configured through DCOM. When you configure DCOM for remote agent activation, you need to enter a user account that will be used to activate either the Distribution Agent or the Merge Agent. It is recommended that you provide a custom account that is the same as the SQL Server Agent account on the original activating computer.

### **Enabling Remote Agent Activation When Creating Subscriptions**

When creating a push subscription or a pull subscription, you enable remote agent activation by specifying where the agent will run. If the Subscriber is the same server as the Distributor, you will not see the option to run the agent at another server in the Push Subscription or Pull Subscription wizards.

**IMPORTANT** After you specify where the agent should run when creating the subscription, synchronization may fail if you specified that the subscription should be synchronized automatically and you have not configured DCOM for the remote agent activation.

### **Configuring DCOM for Remote Agent Activation**

DCOM handles low-level details of network protocols by extending the Component Object Model (COM) to support communication among objects on different computers, a LAN, a WAN, or the Internet. When configuring DCOM, consider the security that is already in place for SQL Server Agent as well as the type of subscriptions that will be used.

For push subscriptions:

• DCOM must be configured on the Subscriber before you change an existing push subscription or create a new push subscription using remote agent activation.

- You must have administrative privileges on the Subscriber.
- The SQL Server Agent on the Distributor must be allowed to use DCOM on the Subscriber.
- An account or security context needs to be specified through DCOM that will allow the Distribution Agent or the Merge Agent to be run on the Subscriber.

For pull subscriptions:

- DCOM must be configured on the Distributor before you change an existing pull subscription or create a new pull subscription using remote agent activation.
- You must have administrative privileges on the Distributor.
- The SQL Server Agent on the Subscriber must be allowed to use DCOM on the Distributor.
- An account or security context needs to be specified through DCOM that will allow the Distribution Agent or the Merge Agent to be run on the Distributor.

#### To configure DCOM to run the Distribution Agent remotely

# **Replication Alerts**

SQL Server Enterprise Manager and SQL Server Agent provide a way to monitor events, such as replication agent errors, using alerts. SQL Server Agent monitors the Microsoft® Windows NT® 4.0 or Microsoft Windows® 2000 application log, watching for an event that qualifies as one of the defined alerts. If such an event occurs, SQL Server Agent can respond automatically, either by executing a task that you have defined or by sending e-mail or a pager message to a specified operator.

You can select a Distributor and use Replication Monitor to display a list of all of the replication-related alerts on the server.

Microsoft SQL Server<sup>™</sup> 2000 includes a set of predefined alerts for replication. You can configure these alerts to notify operators about the state of replication. Operators can then intervene in the replication process manually or configure an automated response job. Alerts that support an automated response job enter additional information into the **msdb..sysreplicationalerts** system table. The information in **sysreplicationalerts** can be used by a custom Transact-SQL job when responding to the alert.

Message		Condition Causing	Enters Additional Information in
ID	Predefined Alert	the Alert to Fire	sysreplicationalerts
14150	Replication: Agent success	Agent shuts down successfully.	Yes
14151	Replication: Agent failure	Agent shuts down with an error.	Yes
14152	Replication: Agent retry	Agent shuts down after unsuccessfully retrying an operation (agent encounters an error such as server not available, deadlock, connection failure, or	Yes

The following alerts are installed when a computer is configured as a Distributor.

		time-out failure).	
14157	<b>Replication:</b>	Inactive subscription	No
	Subscription	was deleted.	
	cleaned up.		
20574	<b>Replication:</b>	Distribution or Merge	Yes
	Subscriber has	Agent fails data	
	failed data	validation.	
	validation		
20575	<b>Replication:</b>	Distribution or Merge	Yes
	Subscriber has	Agent passes data	
	passed data	validation.	
	validation		
20572	<b>Replication:</b>	Response job	No
	Subscription	'Reinitialize	
	reinitialized after	subscriptions on data	
validation failure		validation failure'	
		reinitializes a	
		subscription	
		successfully.	

# Viewing the Application Log

To view the Microsoft® Windows NT® 4.0 or Windows® 2000 application log, use the Windows NT 4.0 or Windows 2000 Event Viewer. If you are part of the **Windows NT Administrators** group, you can also view remote application logs. The application log contains SQL Server error messages as well as messages for all activities on the computer.

When you use the Windows NT application log, each SQL Server session writes new events to an existing application log; you can filter the log for specific events. Unlike the SQL Server error log, a new application log is not created each time you start SQL Server; however, you can specify how long logged events will be retained.

# Automating a Response to an Alert

Usually, when an alert occurs, the only information you have to help you understand what caused the alert and the appropriate action to take is contained in the alert message itself. Creating jobs to respond to the alert is timeconsuming because you must first parse and analyze the information in the message and then insert the relevant information into Transact-SQL commands. Microsoft® SQL Server<sup>™</sup> 2000 replication makes automating response jobs easier by providing additional information about the alert. This information is stored in the **sysreplicationalerts** system table. In addition to providing detailed information, **sysreplicationalerts** provides that information already parsed in a form easily used by customized programs.

For example, if the **pubs** data at Subscriber A fails the validation check, SQL Server triggers alert message 20574 notifying you of that failure. The message you receive may be:

"Subscriber 'A', subscription to article 'authors' in publication 'pubs' fai

If you create a response job based on the alert message, you must manually parse the Subscriber name, article name, publication name, and error from the message. However, because the Distribution Agent writes that same information in **sysreplicationalerts**, along with details such as the type of agent, time of the alert, publication database, Subscriber database, and type of publication, the response job can directly query the relevant information from the table. Although the exact row cannot be associated with a specific instance of the alert, the table has a **status** column, which can be used to keep track of serviced entries. The entries in this table are maintained for the history retention period.

For example, if you were to create a response job in Transact-SQL that services alert message 20574, you might use the following logic:

declare hc cursor local for select publisher, publisher\_db, publication, j
subscriber\_db, alert\_id from
msdb..sysreplicationalerts where
alert\_error\_code = 20574 and status = 0
for read only

open hc

fetch hc into @publisher, @publisher\_db, @publication, @publication while (@@fetch\_status <> -1)

begin

/\* Do custom work \*/

/\* Update status to 1, which means the alert has been serviced. This proupdate msdb..sysreplicationalerts set status = 1 where alert\_id = @aler fetch hc into @publisher, @publisher\_db, @publication, @publicatio end

close hc

deallocate hc

# **Predefined Response Jobs**

Whenever a computer is configured as a Distributor, the following predefined alert response jobs are installed.

Predefined		
Response Job	<b>Responds to Alert</b>	Action
Reinitialize	<b>Replication:</b>	Reinitializes all subscriptions that
subscriptions on	Subscriber has	have logged a <b>sysreplicationalerts</b>
data validation	failed data	record with <b>alert_error_code</b> =
failure.	validation.	20574.
		If a transactional publication, only articles that failed are reinitialized. If a merge publication, the whole publication is reinitialized.

**Note** The response jobs included in Microsoft® SQL Server<sup>™</sup> 2000 are provided only for the most well known responses and as examples you can use for writing your own response jobs. The provided response jobs are not associated with an alert after they have been installed. You must configure an alert manually to call the appropriate response job.

## See Also

SQLServerAgent Service

Validating Replicated Data

# **Subscription Deactivation and Expiration**

Subscriptions can be deactivated or can expire if they are not synchronized within a specified period of time. The action that occurs depends on the type of replication and the retention period that is exceeded.

# **Snapshot and Transactional Replication Subscriptions**

If a subscription is not synchronized within a specified period of time, there is a possibility the subscription may get deleted or it may be automatically marked deactivated and require reinitialization. Whether it expires and is deleted or gets marked deactivated and requires initialization depends upon whether it exceeds the subscription expiration property of the publication or the maximum transaction retention property of the distribution database as well as whether or not it is an anonymous subscription.

# **Subscription Deactivation**

When a subscription is not synchronized (for example, the Distribution Agent for it has not run or cannot connect to the Subscriber) within the maximum transaction retention period of the distribution database and there are changes in the distribution database waiting to be picked up, the subscription will be marked deactivated by the Distribution Cleanup Agent that runs on the Distributor. The default for maximum transaction retention period is 72 hours for transactional replication and the Distribution Cleanup Agent runs every 10 minutes by default.

If there is no activity at the Publisher, subscriptions will not be deactivated even if they have not been synchronized within the distribution retention period. After a subscription is marked inactive, the Distribution Agent will fail with an error message that informs the user that the subscription has been deactivated and that it needs to be reinitialized. The subscription will then need to be reinitialized and a new snapshot applied at the Subscriber before replication continues for that subscription.

In addition to deactivating subscriptions that have not synchronized within the maximum transaction retention period of the distribution database, the

Distribution Cleanup Agent is also responsible for cleaning up transactions in the distribution database that have been delivered to Subscribers with named subscriptions.

If anonymous subscriptions are used, this agent will clean up only transactions in the distribution database that have exceeded the maximum transaction retention period. The Distribution Cleanup Agent will not clean up transactions in the distribution database before the end of the retention period when anonymous subscriptions are used because it cannot be sure that the Subscribers using anonymous subscriptions have received the transactions stored in the distribution database. If you set the retention period to a high value, the distribution database will grow larger if you are using anonymous subscriptions because of this.

#### To modify a Distributor or add or modify a distribution database

# Validating Replicated Data

Problems encountered during replication often occur because data at the Subscriber is not in synchronization with data at the Publisher. Microsoft® SQL Server<sup>™</sup> 2000 replication can validate the replicated data at a Subscriber as the replication process is occurring to ensure that data at the Subscriber matches data at the Publisher.

You do not need to stop updates to the Publisher and wait for the Subscriber to become fully synchronized before testing that data has been received and applied correctly. You can validate the data in snapshot replication, transactional replication, or merge replication. Validation can be performed for specific subscriptions or for all subscriptions to a publication.

## **How Inline Data Validation Works**

SQL Server validates data by calculating a rowcount and/or a checksum at the Publisher and then comparing those values to the rowcount and/or checksum calculated at the Subscriber. One value is calculated for the entire publication table and one value is calculated for the entire subscription table, but data in **text** or **image** columns is not included in the calculations.

While the calculations are performed, shared locks are placed temporarily on tables for which rowcounts or checksums are being run, but the calculations are completed quickly and the shared locks removed, usually in a matter of seconds.

When validating replicated data, consider the following:

- Is the fact that validation failed really a problem? Some validation failures are explainable, and you may not want to reinitialize.
- If the validation failure is an issue, consider the different options for synchronizing the data, including a full reinitialization, a partial reinitialization a previous state, or manually updating the data so that it is synchronized.

# Validating Replicated Data for Transactional Replication

Validation can be performed on transactional replication, subscriptions that use immediate updating or queued updating, and on horizontal and vertical partitions of data.

You can choose any of the following methods for validation:

- Rowcount only.
- Rowcount and checksum.
- Rowcount and binary checksum (this is available only for Subscribers running Microsoft SQL Server 2000).

You can configure validation using SQL Server Enterprise Manager or Transact-SQL system stored procedures. Regardless of which you use, when you run validation, stored procedures are executed at the Publisher. The stored procedure **sp\_publication\_validation** calls **sp\_article\_validation** for each article that is being validated, and **sp\_article\_validation** calls **sp\_table\_validation** for each table, which then generates the rowcount or checksum calculations. The **sp\_table\_validation** command is posted as a replication command to the Subscriber using the Log Reader Agent and Distribution Agent, and the calculations are then made at the Subscriber.

**Note** Subscribers running SQL Server 6.5 can use rowcount only validation, but not checksum validation. You can validate based on a binary checksum calculation if Subscribers are running SQL Server 2000.

You can validate replicated data on a schedule by creating a Transact-SQL job that calls **sp\_publication\_validation** or **sp\_article\_validation**.

Unless you are a member of the **sysadmin** or **db\_owner** roles, you must have SELECT permissions on all columns of the base table used in the article (even if the article is vertically partitioned) in order to execute sp\_publication\_validation.

# Validation with Checksums

When checksums are used, 32-bit redundancy check (CRC) occurs on a columnby-column basis rather than a CRC on the physical row on the data page. This allows the columns with the table to be in any order physically on the data page, but still compute to the same CRC for the row. Checksum validation can be used when there are row (horizontal) or column (vertical) filters on the publication. Because checksums can require large amounts of processor resources when validating a large data set, you may want to schedule validation to occur when there is the least activity on the servers used in replication.

Subscribers running SQL Server 7.0 use the checksum routines released in SQL Server 7.0, which generate CRC values that are different than those generated with SQL Server 2000. The checksum routines released in SQL Server 7.0 cannot validate vertical partitions, or logical table structures where column offsets differ (due to ALTER TABLE statements that DROP and ADD columns).

## Setting the Rowcount\_only Parameter

Value	Description
0	Execute checksum functionality released with SQL
	Server 7.0.
1 (Default)	Execute a rowcount check only.
2	Execute checksum functionality released with SQL
	Server 2000.

The *@rowcount\_only* parameter is a *smallint* and accepts the following values.

Because Subscribers running SQL Server 7.0 will use this parameter as a **bit** data type, not a **smallint**, SQL Server will interpret the parameter as 'ON'. Setting the parameter to a value of 2 with a Subscriber running SQL Server 7.0 will result in a rowcount only validation at the Subscriber. If you need to run a checksum validation for a Subscriber running SQL Server 7.0, use the value of 0 for this parameter. Subscribers running SQL Server 2000 could use the same value (0), but the checksum functionality would have the SQL Server 7.0 limitations.

To validate transactional data using SQL Server Enterprise Manager

- 1. At the Distributor, expand **Replication Monitor**, expand **Publishers**, and then expand a specific Publisher.
- 2. Right-click a transactional publication, and then click Validate

#### subscriptions.

- 3. Choose whether you want to validate all subscriptions or just specific subscriptions, and if you want to validate specific subscriptions, select those in the text box.
- 4. To choose the type of validation, click **Validation Options**.
- 5. Choose whether you want to compute a fast rowcount based on cached table information, compute an actual row count by querying the tables directly, or compute a fast row count and if differences are found, compute an actual row count.
- 6. You can also choose to enable Compare checksums to validate data, a binary checksum (if the Subscriber is running SQL Server 2000), and you can choose to stop the Distribution Agent after the validation has completed.

# To validate transactional data using Transact-SQL system stored procedures

- To validate all articles in a publication and specify rowcount only (the default) or checksum validation, execute **sp\_publication\_validation**. This will call **sp\_article\_validation** for each article in the publication.
- To validate specific articles and specify rowcount only or checksum validation, execute **sp\_article\_validation**.

## Validation and Immediate Updating

When using inline publication validation (**sp\_publication\_validation**) on immediate updating subscriptions, there is a period of time when a change on the Subscriber will cause the publication validation to fail. This occurs when a data change is made on the Subscriber after a publication validation has been run on the Publisher, but before the publication validation can be performed on the Subscriber.

With transactional replication (without updatable subscriptions), changes can be made only at the Publisher, so changes made to the Publisher after **sp\_publication\_validation** has been executed will be applied at the Subscriber after the validation is run on the Subscriber.

However, when using immediate updating subscriptions, data modifications can be made at the Subscriber. Any changes made at the Subscriber after validation was run on the Publisher are reflected immediately at the Subscriber. Validation will fail because the checksum and rowcount calculations were based on data in the publication table before changes were made at the Subscriber. To avoid this, stop all data modifications at the Subscriber during the validation process.

# Considerations when Validating Replicated Data for Transactional Replication

The following are validation restrictions when using validation for transactional replication:

- Checksum validations are not supported for transformable subscriptions because values are likely to be transformed between Publisher and Subscriber and checksum values would not be the same.
- Rowcount validation is not supported for an article that is configured as a DTS horizontal partition because the filter criteria is saved as part of a DTS package, not in a view at the Publisher like replication filters.
- Validation for replicated data to heterogeneous Subscribers is not supported.

# Validation Failure and Alerts

If validation between data at the Publisher and data at the Subscriber fails, you can configure replication alerts to notify you of the failure (with a message sent through e-mail or to a pager) and you can have the subscriptions reinitialized automatically.

#### To configure automatic reinitialization of subscriptions that fail validation

- 1. At the Distributor, expand **Replication Monitor**, click **Replication Alerts**, right-click the **Replication: Subscription has failed data validation** alert, and then click **Properties**.
- 2. On the **General** tab, select the **Enabled** check box.
- 3. On the **Response** tab, select **Execute job**, and then in the drop down box, click **Reinitialize subscriptions having data validation failures**.
- 4. To send a reinitialize confirmation message to the event log, right-click the **Replication: Subscription reinitialized after validation failure** alert, and click **Properties**.
- 5. On the **General** tab, select the **Enabled** check box.

# Validating Replicated Data for Merge Replication

Using SQL Server Enterprise Manager, you can choose to validate all subscriptions to a merge publication. Using Transact-SQL system stored procedures, you can validate all subscriptions to a merge publication or specified subscriptions.

You can choose any of the following methods for validation:

- Rowcount only
- Checksum

To request validation of replicated data at a merge Subscriber, you can use:

- SQL Server Enterprise Manager, which allows you to validate all subscriptions to a publication.
- The Merge Agent command line or the Merge Agent Command Prompt

Utility specifying the **–Validate** parameter. If the Merge Agent is running in continuous mode, the **–Validate** parameter run at an agent command prompt will conduct validation until the **-ValidationInterval** value is reached. Validation will occur after the merge process is complete.

- The **sp\_validatemergepublication** Transact-SQL system stored procedure. This will conduct a publication-wide validation for which all subscriptions (push, pull, and anonymous) will be validated once each.
- The **sp\_validatemergesubscription** Transact-SQL system stored procedure, which runs validation once on the Merge Agent for the specified subscription.

Running the Merge Agent with the **-Validate** parameter causes SQL Server to temporarily lock the Subscriber tables to prevent further changes. SQL Server then computes either a rowcount or checksum of each replicated table at the Subscriber and at the Publisher. If there is a difference, SQL Server locks the discrepant table at the Publisher and any new data changes are downloaded to the Subscriber. After downloading is complete, SQL Server recalculates the rowcount or checksum at the Subscriber and Publisher and compares them again. After validation is complete, SQL Server removes all locks on Subscriber and Publisher tables.

You can validate your data on a regular schedule by adding **-Validate** to the Merge Agent profile at a specified time. Because inline validation may be time-consuming or may result in undesirable contention between the Publisher and Subscriber, you should schedule validation for a time when Publisher and Subscriber activity is at a minimum.

• In case of merge validation failure, you can respond to the failure by using SQL Server Enterprise Manager to configure the replication alert named **Replication: Subscriber has failed data validation** so that you are notified of the failure or you can reinitialize the subscription to ensure that data at the Subscriber is in synchronization with data at the Publisher. Reinitializing the subscription should be performed with

caution because it can be a resource-intensive process for the Publisher, Distributor and Subscribers, and users may not be able to update data while the initial snapshot is reapplied at Subscribers.

When validating merge replication, another option is to validate data, and if data is not converged, to conduct a partial reinitialization of the subscription. This partial reinitialization will return the Subscriber back to a previous state when data was in synchronization. Using the **Validate** and **Resynchronize Subscription** option in SQL Server Enterprise Manager or **sp\_resyncmergesubscription**, you can resynchronize a merge subscription to a known validation state that you specify. This allows you to force convergence or synchronize the subscription database to a specific point in time, such as the last time there was a successful validation, or to a specified date. When resynchronizing a subscription using this method, the snapshot is not reapplied.

#### To validate merge data using SQL Server Enterprise Manager

- 1. At the Distributor, expand **Replication Monitor**, expand **Publishers**, and then expand a specific Publisher.
- 2. Right-click a merge publication, and then click **Validate all subscriptions**.
- 3. Choose whether you want to validate replicated data using rowcounts only, rowcounts and checksums, or rowcounts and comparing binary checksums (all Subscribers must be running SQL Server 2000 to use this option). Validation will occur the next time the Merge Agent runs with results displayed in Replication Monitor.

#### To validate and resynchronize subscriptions

- Expand **Replication Monitor**, expand the **Publishers** folder, and then expand a registered Publisher. Right-click a publication, and then click **Validate and Resynchronize Subscriptions**.
- Execute **sp\_resyncmergesubscription** at the Publisher on the publication database or at the Subscriber on the subscription database.

#### To validate merge data using Transact-SQL system stored procedures

- To mark all named and anonymous subscriptions for validation the next time the Merge Agent runs, execute **sp\_validatemergepublication**.
- To mark specific subscriptions for validation, execute **sp\_validatemergesubscription** or **sp\_validatemergepullsubscription**.

To validate merge data using a Merge Agent command line parameter

- 1. At the Distributor, expand **Replication Monitor**, click the **Merge Agents** folder, right-click an agent, and then click **Agent Properties**.
- 2. On the **Steps** tab, double click the **Run agent** step.
- 3. In the command text box, type **–validate** and specify 1 for rowcountonly validation, 2 for rowcount and checksum validation. Validation will occur the next time the Merge Agent runs and success or failure messages are logged in the Merge Agent History.
- 4. If you want to schedule validation, set the **–ValidateInterval** parameter on the Merge Agent command line to the number of minutes when you want the validation to occur (the default is to validate every 60 minutes).

# **Replication and Heterogeneous Data Sources**

Microsoft® SQL Server<sup>™</sup> 2000 offers the ability to replicate data to any heterogeneous data source that provides a 32-bit ODBC or OLE DB driver on Microsoft Windows® 2000, Microsoft Windows NT® Server 4.0, or Windows 98 operating systems. Additionally, SQL Server 2000 can receive copies of data replicated from Microsoft Access, Microsoft Exchange, Oracle, DB2 Universal, DB2/MVS, and DB2 AS400.

## **Heterogeneous Subscribers**

Publishing to heterogeneous data sources allows corporations that have acquired different databases to continue providing SQL Server 2000 to individuals or offices using those databases.

The simplest way to publish data to a heterogeneous data source is by using OLE DB or ODBC and creating a push subscription from the Publisher to the OLE DB or ODBC Subscriber.

SQL Server 2000 supports replication between different versions of SQL Server and it supports replication to Subscribers running Microsoft SQL Server 2000 Windows CE Edition (SQL Server CE). For more information, see <u>Replication</u> <u>Between Different Versions of SQL Server</u> and <u>Replication with SQL Server for</u> <u>Windows CE</u>.

## **Heterogeneous** Publishers

SQL Server 2000 can subscribe to snapshot or transactional data replicated from Oracle, DB2, Access, and other data sources. This allows companies that are planning to deploy large databases or a data warehouse with SQL Server, or Internet and intranet applications, to gain access to various sources of data. That data can then be consolidated in SQL Server 2000 using replication, and placed into a data mart, data warehouse, or multidimensional database designed for SQL Server Analysis Services.

To implement snapshot or transactional replication published by heterogeneous data sources to your SQL Server 2000 applications, configure SQL Server with third-party software or using applications built with SQL-DMO and the

Replication Distributor Interface.

For more information, see <u>Programming Replication from Heterogeneous Data</u> <u>Sources</u>.

# **Heterogeneous Subscribers**

Microsoft® SQL Server<sup>™</sup> 2000 supports publishing to heterogeneous data sources that provide 32-bit ODBC or OLE DB drivers on Microsoft Windows® 2000, Microsoft Windows NT® 4.0 and Microsoft Windows 98. Heterogeneous Subscribers to SQL Server include:

- Microsoft Access databases.
- Oracle databases.
- Other databases on heterogeneous Subscribers that comply with SQL Server ODBC or OLE DB Subscriber requirements.

The simplest way to publish data to a heterogeneous Subscriber is by using ODBC and creating a push subscription from the Publisher to the ODBC Subscriber. As an alternative, you can create a publication and then create an application with an embedded distribution control. The embedded control implements the pull subscription from the Subscriber to the Publisher. For ODBC Subscribers, the subscribing database has no administrative capabilities regarding the replication being performed.

## **ODBC/OLE DB Driver Support**

ODBC drivers and OLE DB providers for various heterogeneous data sources are included on the SQL Server 2000 compact disc.

## **Stored Procedures That Support Replication to Heterogeneous Subscribers**

SQL Server 2000 provides the following stored procedures and extended stored procedures to support replication to ODBC Subscribers.

Description
Reports all defined ODBC DSNs for a server running

	under a specific Windows NT 4.0 or Windows 2000 user account.
sp_dsninfo	Retrieves ODBC DSN information from the replication Distributor associated with the current server, if replication is installed.

**Note** SQL Server Enterprise Manager (the recommended tool) uses these stored procedures automatically to set up replication to ODBC Subscribers. Use these stored procedures directly only if you are not using SQL Server Enterprise Manager.

## **Replication Restrictions for Heterogeneous Subscribers**

The following restrictions apply to replication to heterogeneous Subscribers:

- Tables replicated to heterogeneous Subscribers will adopt the table naming conventions of the heterogeneous data source.
- Schema files that create tables at the Subscriber do not include quotation marks around table names, and the new table name is dependent on the behavior of the heterogeneous Subscriber on which they are created. For example, if you have a Subscriber running Oracle, and a table is created on Oracle without quotation marks around the table name, it will default to an uppercase table name on the Oracle server. If you specify the name Shipper in the article properties, it will become SHIPPER on the Oracle Subscriber.
- Transactions applied to the heterogeneous Subscriber using the Distribution Agent do have quotation marks around table names.
- Batched statements to ODBC Subscribers are not supported (because the distribution task **commit batch size** option is ignored).
- The ODBC DSN must conform to SQL Server 2000 naming

conventions (because the DSN is stored in the **sysservers** table).

- The publication option to truncate before synchronization is not supported if the ODBC DSN is not a SQL Server DSN. ODBC Subscribers are not allowed to subscribe to publications that have this option selected.
- The quoted identifier character on the target server (as reported by the ODBC driver) is used.
- The character format bulk copy method must be selected for synchronization (using the Create Publication Wizard on the publication property dialog box). ODBC Subscribers cannot subscribe to publications that have selected the native format bulk copy method for synchronization.
- Only NULL, NOT NULL, IDENTITY, and the constraint PRIMARY KEY for CREATE TABLE are supported for all heterogeneous Subscribers. Therefore, SQL Server 2000 does not support adding articles to a publication after a subscription has been created for a heterogeneous Subscriber. Each time an article is added or deleted from the publication, the subscription must be reinitialized.

# **Access Subscribers**

Microsoft® SQL Server<sup>™</sup> 2000 includes an ODBC driver and OLE DB provider that supports Microsoft Access 97 or Microsoft Access 2000 subscriptions to SQL Server publications. SQL Server 2000 Setup installs the driver and provider automatically.

To replicate to Access Subscribers, you must assign the MSSQLServer service the same domain user account assigned by SQL Server Agent, for the service to have the necessary permissions to connect to an .mdb file over the network. Use the Services application in Control Panel to do this.

**Note** When you register a Access DSN on a remote server, supply a UNC path (not a redirected drive letter).

**IMPORTANT** If you do not enable heterogeneous Subscribers and you create the subscription database in the Create Publication Wizard, the schema will be published to the Subscriber, but the data will not be, and you will not receive an error. To enable heterogeneous Subscribers, on the Specify Subscriber Types page of the Create Publication Wizard, select **Heterogeneous data sources, such as Oracle or Microsoft Access; devices running SQL Server CE; or servers running earlier versions of SQL Server**.

# Data Type Mapping to Jet-SQL 3.51 (Access 8) for Transactional Replication

The following table maps data types for transactional replication to Access Subscribers When you replicate to ODBC Subscribers, the distribution task maps SQL Server 2000 data types to the closest data type on the target database.

SQL Server 2000 data type	Access Jet-SQL 3.51 data type
binary(n)	LONGBINARY
bit	BIT
char(n)	LONGTEXT
datetime	CHAR (23)
decimal	CHAR (30)
float	DOUBLE

image	LONGBINARY
int	LONG
int	LONG
money	CHAR (25)
nchar(n)	LONGTEXT
ntext	LONGTEXT
numeric	CHAR (30)
nvarchar(n)	LONGTEXT
real	SINGLE
smalldatetime	DATETIME
smallint	SHORT
smallmoney	DOUBLE
text	LONGTEXT
timestamp	BINARY (8)
tinyint	BYTE
uniqueidentifier	CHAR (36)
varbinary(n)	LONGBINARY
varchar(n)	LONGTEXT

## Data Type Mapping to Jet-SQL 4.0 for Transactional Replication

The following table maps data types for transactional replication to Access Subscribers. When you replicate to ODBC Subscribers, the distribution task maps SQL Server 2000 data types to the closest data type on the target database.

**Note** The data type mapping from SQL Server to Jet-SQL 4.0 is the same for snapshot replication, transactional replication, and merge replication.

SQL Server 2000 data type	Microsoft Jet-SQL 4.0 data type
binary(n)	BINARY ( <i>n</i> )
Bit	BIT
char(n)	CHAR (n)
datetime	DATETIME
decimal	DECIMAL
float	FLOAT

image	IMAGE
int	INT
money	CURRENCY
nchar(n)	NCHAR ( <i>n</i> )
numeric	NUMERIC
nvarchar(n)	NCHAR VARYING ( <i>n</i> )
real	REAL
smalldatetime	DATETIME
smallint	SMALLINT
smallmoney	CURRENCY
text	LONGTEXT
timestamp	BINARY
tinyint	BYTE
uniqueidentifier	GUID
varbinary(n)	VARBINARY (n)
varchar(n)	VARCHAR (n)

# **Oracle Subscribers**

Microsoft® SQL Server<sup>™</sup> 2000 includes an ODBC driver and OLE DB provider that support Oracle subscriptions to SQL Server publications on Intel computers. SQL Server 2000 Setup installs the driver automatically.

**Note** To replicate to Oracle ODBC and OLE DB Subscribers, you must also obtain the appropriate Oracle SQL\*Net driver from Oracle or from your software vendor. You must then install the driver on the Publisher and the Distributor.

## **Replication Restrictions for Oracle Subscribers**

The following restrictions apply when replicating to an Oracle ODBC Subscriber:

- Replication of tables that have names with spaces will not be created on the Oracle subscriber. Replication will fail with Oracle error ORA-00903: invalid table name.
- The **date** data type is a small **datetime** (the range is 4712 B.C. to 4712 A.D.).

If you are replicating to Oracle, verify that SQL Server **datetime** entries in a replicated column are within this range.

- A replicated table can have only one column of either **text** or **image** data type, which is mapped to long raw.
- The **datetime** data type is mapped to char4.
- The SQL Server 2000 ranges for **float** and **real** data types are different from the Oracle ranges.

The following table maps data types for replication to Oracle Subscribers.



SQL Server 2000 data type	Oracle data type
bigint	NUMBER
binary	LONG RAW NOT NULL
bit	NUMBER (1, 0)
char	VARCHAR2 (900) NOT NULL
datetime	DATE
decimal	NUMBER (255, 3) NOT NULL
float	FLOAT NOT NULL
image	LONG RAW
int	NUMBER (255, 3) NOT NULL
money	NUMBER (255, 3) NOT NULL
nchar	VARCHAR2 (2000) NOT NULL
ntext	LONG
numeric	NUMBER (255, 3) NOT NULL
nvarchar	VARCHAR2 (2000) NOT NULL
real	FLOAT NOT NULL
smallint	NUMBER (255, 3) NOT NULL
smalldatetime	DATE NOT NULL
smallmoney	NUMBER (255, 3) NOT NULL
sql_variant	LONG
sysname	CHAR(255)
text	LONG
timestamp	RAW (255)
tinyint	NUMBER (255, 3) NOT NULL

# **Oracle Data Type Definitions**

The following table lists the Oracle data type definitions.

Oracle data type	Definition
CHAR	<=2000
DATE	Jan 1, 4712 B.C. to Dec 31, 4712
	A.D.

DECIMAL	Same as Number
FLOAT	Same as Number
INTEGER	Same as Number
LONG	<=2GB
LONG RAW	Raw data; Same as Long
LONG VARCHAR	Same as Long
NUMBER	1.0E-130 to 9.99E125
SMALLINT	Same as Number
RAW	Raw Binary Data <=255 bytes
ROWID	Unique Value
VARCHAR2	<=4000 bytes
VARCHAR	Same as Varchar2
BLOB	Binary Large Object <=4GB
СОВ	Char Large Object <=4GB
NCLOB	Same as Clob (for multibyte)
BFILE	Pointer to binary operating file

# **IBM DB2/AS400 Subscribers**

IBM DB2/AS400 subscriptions to Microsoft® SQL Server<sup>™</sup> 2000 publications are supported through the OLE DB provider and ODBC driver that are included with Microsoft Host Integration Server 2000.

The following table maps SQL Server 2000 data types to IBM DB2/AS400 data types. When you replicate to OLE DB or ODBC Subscribers, the distribution task maps SQL Server 2000 data types to the closest data type on the target database.

SQL Server 2000 data type	DB2/AS400 data type
binary(n)	CHAR(8000) FOR BIT DATA
bit	SMALLINT
<b>char(</b> <i>n</i> <b>)</b>	CHAR (8000)
datetime	TIMESTAMP
decimal	DECIMAL
double precision	DOUBLE
float	FLOAT
image	VARCHAR(32739) FOR BIT DATA
int	INTEGER NOT NULL
money	DECIMAL (19, 4)
numeric	NUMERIC
real	REAL
smalldatetime	TIMESTAMP NOT NULL
smallint	SMALLINT
smallmoney	DECIMAL (10, 4) NOT NULL,
text	VARCHAR (32739)
timestamp	CHAR(8) FOR BIT DATA)
tinyint	SMALLINT NOT NULL
uniqueidentifier	CHAR (36)
<b>varbinary(</b> n <b>)</b>	VARCHAR(8000) FOR BIT DATA
	NOT NULL
varchar(n)	VARCHAR (8000) NOT NULL

# **IBM DB2/AS400 Data Type Definitions**

The following table lists the IBM DB2/AS400 data type definitions.

DB2/AS400 data type	Definition
INT	9
SMALLINT	4
FLOAT	<=53
NUMERIC	1 - 31 digits
DECIMAL	1 - 31 digits
CHAR	<=32766
VARCHAR	<=32740
LONG VARCHAR	Determined by space available in row
TIMESTAMP	Gregorian
GRAPHIC	<=16383
VARGRAPHIC	<=16370
LONG VARGRAPHIC	Determined by space available in row
REAL	8,7
DOUBLE	17,16

# **IBM DB2/MVS Subscribers**

IBM DB2/MVS subscriptions to Microsoft® SQL Server<sup>™</sup> 2000 publications are supported through the OLE DB provider and ODBC driver that are included with Microsoft Host Integration Server 2000.

The following table maps SQL Server 2000 data types to IBM DB2/MVS data types. When you replicate to OLE DB or ODBC Subscribers, the distribution task maps SQL Server 2000 data types to the closest data type on the target database.

SQL Server 2000 data type	DB2/MVS data type
binary(n)	CHAR(254) FOR BIT DATA NOT
	NULL
bit	SMALLINT
<b>char(</b> <i>n</i> <b>)</b>	CHAR (254) NOT NULL
datetime	TIMESTAMP NOT NULL
decimal	DECIMAL (31, 3) NOT NULL
double precision	DOUBLE
float	FLOAT NOT NULL
image	VARCHAR(4045) FOR BIT DATA
int	INTEGER NOT NULL
money	DECIMAL (19, 4) NOT NULL
nchar(n)	VARCHAR (900) NOT NULL
numeric	NUMERIC (31, 3) NOT NULL
real	REAL NOT NULL
smalldatetime	TIMESTAMP NOT NULL
smallint	SMALLINT NOT NULL
smallmoney	DECIMAL (10, 4) NOT NULL
text	VARCHAR (4045)
timestamp	CHAR(8) FOR BIT DATA
tinyint	SMALLINT NOT NULL
uniqueidentifier	CHAR (38)
varbinary(n)	VARCHAR(4045) FOR BIT DATA

	NOT NULL
varchar(n)	VARCHAR (4045) NOT NULL

## **Other Heterogeneous Subscribers**

This section includes the data type mappings for Subscribers running DB2/NT or DB2/6000 as well as the driver types needed for ODBC Subscribers.

## IBM DB2/NT

IBM DB2/NT subscriptions to Microsoft<sup>®</sup> SQL Server<sup>™</sup> 2000 publications are supported through the OLE DB provider and ODBC driver that are included with Microsoft Host Integration Server 2000.

The following table maps SQL Server 2000 data types to IBM DB2/NT data types. When you replicate to OLE DB or ODBC Subscribers, the distribution task maps SQL Server 2000 data types to the closest data type on the target database.

SQL Server 2000 data type	IBM DB2/NT data type
binary(n)	CHAR(254) FOR BIT DATA NOT
	NULL
bit	SMALLINT
<b>char(</b> <i>n</i> <b>)</b>	CHAR (254) NOT NULL
datetime	TIMESTAMP
decimal	DECIMAL
double precision	DOUBLE
float	FLOAT
image	VARCHAR(4000) FOR BIT DATA
int	INTEGER NOT NULL
money	DECIMAL(19, 4)
numeric	NUMERIC
real	REAL
smalldatetime	TIMESTAMP NOT NULL
smallmoney	DECIMAL(10, 4)
text	VARCHAR (4000)
timestamp	CHAR(8) FOR BIT DATA)

tinyint	SMALLINT NOT NULL
uniqueidentifier	CHAR (38)
varbinary(n)	VARCHAR(4000) FOR BIT DATA NOT NULL
varchar(n)	VARCHAR (4000) NOT NULL

#### **IBM DB2/6000**

IBM DB2/6000 subscriptions to MSQL Server 2000 publications are supported through the OLE DB provider and ODBC driver that are included with Microsoft Host Integration Server 2000.

The following table maps SQL Server 2000 data types to IBM DB2/6000 data types. When you replicate to OLE DB or ODBC Subscribers, the distribution task maps SQL Server 2000 data types to the closest data type on the target database.

SQL Server 2000 data type	IBM DB2/6000 data type
binary(n)	CHAR(254) FOR BIT DATA NOT
	NULL
Bit	SMALLINT
char(n)	CHAR (254) NOT NULL
Datetime	TIMESTAMP
Decimal	NUMERIC (28, 14) NOT NULL
Float	INTEGER NOT NULL
Image	VARCHAR(4000) FOR BIT DATA
Int	INTEGER(10) NOT NULL
Money	DECIMAL(19, 4)
Numeric	NUMERIC
Real	REAL
Smalldatetime	TIMESTAMP NOT NULL
Smallint	SMALLINT
Smallmoney	DECIMAL (10, 4) NOT NULL,
Text	VARCHAR (4000)
Timestamp	CHAR(8) FOR BIT DATA

Tinyint	SMALLINT NOT NULL
Uniqueidentifier	CHAR (38)
varbinary(n)	VARCHAR(4000) FOR BIT DATA NOT NULL
varchar(n)	VARCHAR (4000) NOT NULL

#### **ODBC Driver and OLE DB Provider Support**

ODBC drivers and OLE DB providers for various heterogeneous data sources are included on the SQL Server 2000 compact disc.

Drivers for other ODBC Subscriber types must conform to the SQL Server 2000 replication requirements for generic ODBC Subscribers. The ODBC driver:

- Must be ODBC level-1 compliant.
- Must be 32-bit, thread-safe, and for the processor architecture (Intel or Alpha) on which the distribution process run.
- Must be transaction capable.
- Must support the Data Definition Language (DDL).
- Cannot be read-only.
- Must support long table names such as **MSreplication\_subscriptions**.

#### **Replicating Using OLE DB Interfaces**

SQL Server 2000 replication can use OLE DB interfaces to execute SQL statements at Subscribers using the **ICommand** interface. OLE DB providers must support these objects for transactional replication:

• **DataSource** object

- Session object
- Command object
- **Rowset** object
- Error object

#### **DataSource Object Interfaces**

The following interfaces are required to connect to a data source:

- IDBInitialize
- IDBCreateSession
- IDBProperties

If the provider supports the **IDBInfo** interface, SQL Server 2000 uses the interface to retrieve information such as the quoted identifier character, maximum SQL statement length, and maximum number of characters in table and column names.

#### **Session Object Interfaces**

The following interfaces are required:

- IDBCreateCommand
- ITransaction
- ITransactionLocal

• IDBSchemaRowset

## **Command Object Interfaces**

The following interfaces are required:

- ICommand
- ICommandProperties
- ICommandText
- ICommandPrepare
- IColumnsInfo
- IAccessor
- ICommandWithParameters

**IAccessor** is necessary to create parameter accessors. If the provider supports **IColumnRowset**, SQL Server 2000 uses that interface to determine whether a column is an identity column.

## **Rowset Object Interfaces**

The following interfaces are required:

- IRowset
- IAccessor
- IColumnsInfo

An application should open a rowset on a replicated table that is created in the subscribing database. **IColumnsInfo** and **IAccessor** are needed to access data in the rowset.

## **Error Object Interfaces**

Use the following interfaces to manage errors:

- IErrorRecords
- IErrorInfo

Use **ISQLErrorInfo** if it is supported by the OLE DB provider.

For more information about the OLE DB provider, see the documentation supplied with your OLE DB provider.

**Note** The primary source of information regarding the use of OLE DB is the *OLE DB Programmer's Reference Version 2.0* available with the OLE DB Software Development Kit (SDK).

## **Implementing Merge Replication to Access Subscribers**

When using releases of Microsoft® Access later than Access 8, you have a choice between using the SQL Server 2000 Desktop Engine or Microsoft Jet as the database engine and data storage for your Access database. Desktop Engine is a data store based on Microsoft SQL Server<sup>™</sup> 2000 technology, but it is designed and optimized for use on smaller computer systems, such as a one computer or small workgroup server. Because Desktop Engine is based on the same database engine as SQL Server, most Access projects or client/server applications run on either Desktop Engine or SQL Server Standard or Enterprise Edition unchanged. However, unlike other editions of SQL Server, Desktop Engine has a 2 gigabyte database size limit, it does not support symmetrical multiprocessing (SMP), and it cannot be a Publisher for a transactional publications).

If you select Desktop Engine or SQL Server as the database engine for your application, there are no further steps required to replicate between a SQL Server Publisher and an Access Subscriber. The computer running Access appears in SQL Server Enterprise Manager as simply another server.

If you select Microsoft Jet as the database engine for your Access application, you must enable the Jet version 4.0 database as a Subscriber. To do so, you must configure SQL Server to use an OLE DB connection to the database for each Jet Subscriber. The easiest way to do this is through SQL Server Enterprise Manager; however, you can also add a Jet database as a linked server programmatically by executing **sp\_addlinkedserver**.

Replication to Access Subscribers is subject to the following restrictions:

- Microsoft Jet 4.0 does not support case-sensitive sort orders. Do not use an instance of SQL Server with a case-sensitive sort order installed to create publications for Jet 4.0 Subscribers.
- Microsoft Jet 4.0 does not support push subscriptions from Publishers running on DEC Alpha servers to Jet 4.0 Subscribers running on other platforms. Instead of creating a push subscription at the DEC Alpha

Publisher, create a pull subscription at the Jet 4.0 Subscriber.

- SQL Server does not support known pull subscriptions but does support anonymous pull subscriptions from Jet 4.0 Subscribers. This functionality is implemented using the Microsoft ActiveX® replication controls.
- You cannot replicate both merge and transactional publications from the same publication database to a Jet Subscriber.
- When running the Merge Agent with the **-validate** parameter, only rowcount validation is supported. You cannot use checksum validation when validating replicated data to a Jet Subscriber.
- Column names cannot be the same names as those columns used during Jet replication. Reserved column names include: s\_Generation, s\_GUID, s\_Lineage and s\_ColLineage.

# Data Type Mapping to Jet-SQL 4.0 for Merge Replication

The following table maps data types for merge replication to Microsoft® Access Subscribers. When you replicate to ODBC Subscribers, the distribution task maps Microsoft SQL Server<sup>™</sup> 2000 data types to the closest data type on the target database.

**Note** The data type mapping from SQL Server 2000 to Jet-SQL 4.0 is the same for snapshot replication, transactional replication, and merge replication.

SQL Server 2000 data type	Microsoft Jet-SQL 4.0 data type
bigint	DECIMAL
binary(n)	BINARY ( <i>n</i> )
bit	BIT
<b>char(</b> <i>n</i> <b>)</b>	CHAR (n)
datetime	DATETIME
decimal	DECIMAL
float	FLOAT
image	IMAGE
int	INT
money	CURRENCY
nchar(n)	NCHAR (n)
ntext	LONGTEXT
numeric	DECIMAL
nvarchar(n)	NCHAR VARYING (n)
real	REAL
smalldatetime	DATETIME
smallint	SMALLINT
smallmoney	CURRENCY
text	LONGTEXT
timestamp	BINARY

tinyint	BYTE
uniqueidentifier	GUID
varbinary(n)	VARBINARY (n)
varchar(n)	VARCHAR (n)

## **Heterogeneous** Publishers

Microsoft® SQL Server<sup>™</sup> 2000 can subscribe to snapshot or transactional data published from Oracle, DB2, Access, and other data sources. This allows organizations that are planning to deploy large databases or a data warehouse with SQL Server, or Internet and intranet applications, to gain access to various sources of data. That data can then be consolidated in SQL Server 2000 using replication, and then placed into a data mart, data warehouse, or multidimensional database, or the data can then be replicated to other data sources.

Methods for implementing replication published by heterogeneous data sources to your SQL Server 2000 applications include:

- Building applications with SQL-DMO and the Replication Distributor Interface
- Using third-party tools

Microsoft SQL Server 2000 provides a programming framework that enables heterogeneous data sources to become Publishers of snapshot and transactional publications within the SQL Server 2000 replication framework. You can use the Replication Distributor Interface with programmable SQL-DMO objects and third-party tools to publish data incrementally from heterogeneous Publishers.

Using third-party tools, you can configure Oracle, DB2, and other data sources as a merge or incremental Publisher for SQL Server Subscribers, which can then propagate data to other SQL Server or heterogeneous data sources.

By integrating with SQL Server 2000 replication, heterogeneous applications can inherit a full set of replication features, such as:

- Remote store-and-forward databases and Distribution Agents.
- Heterogeneous Subscribers, including Microsoft Access, Oracle, and DB2.

- Pull subscriptions.
- Anonymous subscriptions.
- Internet subscriptions.
- Subscriptions on computers running Microsoft Windows® 98.
- Stand-alone and embeddable Distribution Agents.
- Monitoring and troubleshooting tools using SQL Server Enterprise Manager.
- Replication agent scheduling using SQL Server Agent.
- Alerts and notifications.
- Performance monitoring.

The programming framework for transactional and snapshot replication from heterogeneous data sources includes:

- Programmable SQL-DMO replication objects for administering and monitoring replication.
- The Replication Distributor Interface for storing replicated transactions from a heterogeneous Publisher.
- A Distribution Agent to forward the transactions to Subscribers.
- SQL Server Enterprise Manager to administer and monitor replication

graphically.

Using Data Transformation Services (DTS), heterogeneous data sources can be used to create snapshot replication publications.

## See Also

Programming Replication from Heterogeneous Data Sources

# **Replication Security**

Microsoft<sup>®</sup> SQL Server<sup>™</sup> 2000 replication uses a combination of security methods to protect the data and business logic in your application.

Security	Description
Role Requirements	By mapping user logins to specific SQL Server 2000 roles, SQL Server 2000 allows users to perform only those replication and database activities authorized for that role. Replication grants certain permission to the <b>sysadmin</b> fixed server role, the <b>db_owner</b> fixed database role, the current login, and the <b>public</b> role.
Connecting to the Distributor	SQL Server 2000 provides a secure administrative link between the Distributor and Publisher. Publishers can be treated as trusted or nontrusted.
Snapshot Folder Security	With alternate snapshot locations, you can save your snapshot files to a location other than at the Distributor (for example, a network share, an FTP site, or removable media). When saving snapshots, ensure that replication agents have proper permission to write and read the snapshot files.
Publication Access Lists	Publication access lists (PALs) allow you to determine which logins have access to publications. SQL Server 2000 creates the PAL with default logins, but you can add or delete logins from the list.
Agent Login Security	SQL Server 2000 requires each user to supply a valid login account to connect to the server. Replication agents are required to use valid logins when connecting to Publishers, Distributors, and Subscribers. However, agents can also use different logins and security modes when connecting to different servers simultaneously.
Password	Passwords used in SQL Server 2000 replication are
Encryption	encrypted automatically for greater security.
Security and Replication	Filtering replicated data can be used to increase data security, and there are additional security

Options	considerations when using dynamic snapshots, immediate updating, and queued updating.
Security and	Different types of replication over the Internet have
Replication Over	different security levels. Additionally, when
the Internet	transferring replication files using FTP sites,
	precautions must be taken to secure the site and still
	make it accessible to replication agents.

## **Role Requirements**

Microsoft® SQL Server<sup>™</sup> 2000 replication restricts the specific actions that a user can perform based on the role mapped to the user's login. Replication has granted certain permissions to the **sysadmin** server role, the **db\_owner** database role, and the logins in the publication access list (PAL).

Replication administration	Membership requirement
Enable, modify or drop a Distributor.	<b>sysadmin</b> server role.
Enable, modify, or drop a Publisher.	<b>sysadmin</b> server role.
Enable, modify, or drop a Subscriber.	<b>sysadmin</b> server role.
Enable a database for replication.	<b>sysadmin</b> server role.
Create or drop a publication.	<b>sysadmin</b> server role or <b>db_owner</b> database role.
Modify publication properties.	<b>sysadmin</b> server role or <b>db_owner</b> database role. If the login is in the PAL, a user can view the publication properties as read-only even if the user is not a member of the <b>sysadmin</b> or <b>db_owner</b> roles.
Create or delete a push subscription.	<b>sysadmin</b> server role or <b>db_owner</b> database role.
Create a pull subscription.	<b>sysadmin</b> server role or <b>db_owner</b> database role or any login in the PAL.
Delete a pull subscription.	<b>sysadmin</b> or <b>db_owner</b> database role, or the creating login of a pull subscription.
Update a PAL.	<b>sysadmin</b> server role or <b>db_owner</b> database role.
Enable snapshots for FTP downloading using the Internet.	<b>sysadmin</b> server role or <b>db_owner</b> database role.

These tables summarize the requirements for common replication actions.

View replication activity, errors and	<b>replmonitor</b> role.
history using Replication Monitor.	
A user cannot modify agent	
profiles, schedules, and so on,	
unless the user is a member of the	
<b>sysadmin</b> server role.	

Replication agents	Membership requirement
Configure agent profile.	<b>sysadmin</b> server role.
Monitor replication agents.	<b>sysadmin</b> server role.
At the Publisher, logins for	For pull subscriptions, login must be in
Snapshot Agents, Log Reader	the publication access list of the
Agents, and Merge Agents.	referenced publication. For push
	subscriptions, login must be member
	of <b>db_owner</b> (includes <b>sysadmin</b> ) in
	the publication database.
At the Distributor, logins for	For pull subscriptions, login must be in
Snapshot Agents, Log Reader	the publication access list of the
Agents, Distribution Agents, and	referenced publication or <b>db_owner</b>
Merge Agents.	database role on the distribution
	database. For push subscriptions, login
	must be member of <b>db_owner</b>
	(includes <b>sysadmin</b> ) in the distribution
	database.
Distribution Agents and Merge	For both push and pull subscriptions,
Agents logging into the Subscriber.	the login must be a member of
	<b>db_owner</b> (includes <b>sysadmin</b> ) in the
	subscription database.
Replication agents	Membership requirement
Configure agent profile.	<b>sysadmin</b> server role.

Replication tasks	Membership requirement
1	<b>sysadmin</b> server role or <b>db_owner</b> database role on the distribution

	database.
Schedule jobs.	<b>sysadmin</b> server role or <b>db_owner</b> database role on the <b>msdb</b> database.
Merge data during merge replication.	The merge process requires an entry for the Publisher in the <b>sysservers</b> table on the Subscriber. If the entry does not exist, SQL Server will attempt to add this entry. If the login used by the Merge Agent does not have access to add the entry (such as <b>db_owner</b> of the subscription database), an error will be returned.

## **Connecting to the Distributor**

The Distributor can be the same server as the Publisher (local Distributor), or it can be a separate server from the Publisher (remote Distributor). When using remote Distributors, you can configure the security necessary when the Publisher and Distributor connect.

The connection between a Publisher and a remote Distributor is a hybrid of a linked server and remote server. The connection uses the login **distributor\_admin**. At the remote Distributor, the Publisher can be configured to be either trusted (no password is required for the **distributor\_admin** login) or non-trusted (a password is required).

It is recommended that you use a non-trusted connection for the Publisher connection to the Distributor, requiring a **distributor\_admin** password. This increases security at the Distributor by restricting access. Members of the **sysadmin** or **db\_owner** roles who want to use a Distributor must know the administrative link password. An incorrect **distributor\_admin** password at the Publisher causes the configuration of replication at the Publisher to fail.

**WARNING** Do not change the password for the **distributor\_admin** manually. Always use either **sp\_changedistributor\_password** or the **Distributor** tab of the **Publisher and Distributor Properties** in SQL Server Enterprise Manager because password changes are then applied to local publications automatically. Changing the **distributor\_admin** password manually causes publications using a local Distributor to fail.

#### To add or change a password on a Distributor

## **Snapshot Folder Security**

Alternate snapshot locations enable you to store snapshot files in a location other than or in addition to the default location, which is often located on the Distributor. Alternate locations can be on another server, on a network share, or on removable media (such as CD-ROMs or removable disks).

When specifying the snapshot location on a network share, it is recommended that you dedicate the share for snapshot storage and files that have the same security standards. Next, give the replication agents Write permission on the share and in the snapshot location and appropriate folders so they can write the snapshot files there.

Subscribers that need to access the snapshot files will need Read permission to the snapshot location and appropriate folders. If the snapshot folder is not shared for the appropriate Subscribers, the replication agents cannot access the folder and replication fails.

On a Distributor running the Microsoft® Windows NT® 4.0 or Microsoft Windows® 2000 operating system, the snapshot folder defaults to using the <drive>\$ share and a path of \\<computer>\<drive>\$\Mssql\Repldata\. On a Distributor running the Microsoft Windows 98 operating system, the snapshot folder defaults to using the <drive> without a share and a path of <drive>:\Mssql\Repldata. You can, however, save the snapshot files to a location other than the default.

If your application requires the ability to create pull subscriptions on a server running the Windows 98 operating system, you must change the snapshot folder to a network path accessible by replication agents running at the Publisher and Subscribers. You can change the local path to a network path by sharing the folder manually.

#### See Also

Alternate Snapshot Locations

Security and Replication Over the Internet

# **Publication Access Lists**

When you create a publication, Microsoft® SQL Server<sup>™</sup> 2000 creates a publication access list (PAL) for the publication. The PAL contains a list of logins that are granted access to the publication. The logins included in the PAL are members in the **sysadmin** fixed server role and the current login.

The PAL functions similarly to a Microsoft Windows® 2000 access control list. When a user or replication agent attempts to log in to a Publisher, SQL Server 2000 first checks to see if the login is in the PAL. If you must further expand or restrict access to a publication, you can add or delete logins in the PAL using SQL Server Enterprise Manager or the **sp\_grant\_publication\_access** and **sp\_revoke\_publication\_access** stored procedures.

A snapshot, transactional, or merge publication may be secured with a PAL through SQL Server Enterprise Manager or programmatically.

**Note** A replication agent login for the Publisher and Distributor must exist in the PAL before it can access the publication. The user login must also exist in the publication database or the database must allow guest users. If you are using a remote Distributor, the logins must exist at both the Publisher and the Distributor before it can be added to the PAL. Because the replication agents run under SQL Server Agent, the account under which SQL Server Agent runs on a Windows platform must be in the PAL.

If you have a large number of user logins to add to the PAL, consider making them all members of a single Windows 2000 group and then adding the Windows 2000 group to the PAL.

#### To grant or revoke access to a publication

# **Agent Login Security**

Replication implements login security by requiring a user to have a valid login account and password to connect to a Publisher, Distributor, or Subscriber. Replication agents run under SQL Server Agent and use the associated logins and passwords to connect to the various replication objects and to perform their roles in the synchronization process.

On the Microsoft® Windows® 98 operating system, SQL Server Agent and the replication agents run under the security account of the user logging on to Windows. On the Microsoft Windows NT® 4.0 and Windows 2000 operating system, replication agents run under the login or security context of the SQLServerAgent service. Each agent connects to one or more servers and must have a valid login to complete the connection.

## **Applying a Snapshot**

When applying a snapshot, the agents must have the following capabilities:

• The Snapshot Agent connects to the publication database on the Publisher and to the distribution database on the Distributor. The Snapshot Agent also writes to the snapshot folder when storing the snapshot files.

## **Transactional Replication**

The agents used in transactional replication must have the following capabilities:

- The Log Reader Agent connects to the publication database at the Publisher and to the distribution database at the Distributor.
- With a push subscription, the Distribution Agent is, by default, located on the Distributor and connects first to the distribution database on the Distributor. While connected to the Distributor, the Distribution Agent connects to the subscription database at the Subscriber. The Distribution Agent also reads from the snapshot folder when applying the snapshot files.

• With a pull subscription, the Distribution Agent is, by default, located on the Subscriber and connects first to the subscription database on the Subscriber. While connected to the Subscriber, the Distribution Agent connects to the distribution database at the Distributor. The Distribution Agent also reads from the snapshot folder when applying the snapshot files.

## **Merge Replication**

The agents used in merge replication must have the following capabilities:

- With a push subscription, the Merge Agent is located on the Distributor and connects first to the distribution database on the Distributor. While connected to the Distributor, the Merge Agent connects to the subscription database at the Subscriber and then to the publication database at the Publisher. The Merge Agent also reads from the snapshot folder when applying the snapshot files.
- With a pull subscription, the Merge Agent is located on the Subscriber and connects first to the subscription database on the Subscriber. While connected to the Subscriber, the Merge Agent connects to the distribution database at the Distributor and then to the publication database at the Publisher. The Merge Agent also reads from the snapshot folder when applying the snapshot files.
- Merge replication requires an entry for the Publisher in the **sysservers** table at the Subscriber. If the entry does not exist, either SQL Server will attempt to add the entry when you create a merge publication or the Merge Agent will attempt to add the entry. If the login used does not have sufficient access to add the entry in **sysservers**, an error will be returned.

**Note** For an agent that holds simultaneous connections, Microsoft SQL Server<sup>™</sup> allows you to configure the login for each connection independently. For example, if the Snapshot Agent connects to the Publisher and to the

Distributor, each connection can use a different login.

# **Security and Replication Options**

## **Filtering Published Data**

Filtering published data allows you to restrict access to data and allows you to specify the data that is available at the Subscriber. You can filter data horizontally or vertically with any type of replication so partitions based on user requirements and needs can be published to Subscribers.

Additionally, dynamic filters can be used with merge replication and custom data partitions can be created with transactional replication to filter rows based on values retrieved from the Subscriber. For example, using the SUSER\_SNAME function in a merge replication dynamic filter, you can propagate just the rows that relate to the value at the Subscriber retrieved by SUSER\_SNAME.

For more information, see <u>Filtering Published Data</u>.

## **Dynamic Snapshots**

Dynamic snapshots provide a performance advantage when applying the snapshot of a merge publication with dynamic filters. By using Microsoft® SQL Server<sup>™</sup> 2000 bulk copy files to apply data to a specific Subscriber instead of a series of INSERT statements, you will improve the performance when applying the initial snapshot for dynamically filtered merge publications.

The following security considerations must be met to use dynamic snapshots:

- SQL Server on the Publisher must be running under mixed security mode.
- The login specified as the value of the Publisher login must be in the publication access list (PAL), or be a member of the publication database sysadmin role or db\_owner group. This login can be specified in the Create Dynamic Snapshot Job Wizard or by using the DynamicFilterLogin parameter of the Snapshot Agent.
- Because SQL Server adds and drops temporary logins in the Snapshot

Agent, the Publisher login of the Snapshot Agent must be a member of the **securityadmin** server role and be a member of the **db\_owner** group on the publication database to generate dynamic snapshots.

• Dynamic filter logins specified for dynamic snapshot generation must be members of the corresponding PAL.

For more information, see **Dynamic Snapshots**.

## **Immediate Updating and Queued Updating**

The immediate updating option supports either dynamic remote procedure call (RPC) mode or static RPC mode for the two-phase commit protocol (2PC) connection from the synchronization triggers back to the Publisher.

In dynamic RPC mode, synchronization triggers connect dynamically to the Publisher, using a supplied server name, login, and password. This mode offers increased security for users who do not want a statically defined linked server/remote server connection from a Subscriber to Publisher. It is also easier to use when setting up push subscriptions because the Publisher does not have to be predefined at the Subscriber.

In static RPC mode, synchronization triggers connect to the Publisher over a statically defined server name defined as a linked server or remote server in the **sysservers** table. This entry is added by an administrator at the Subscriber. The configuration mode is set automatically when creating push or pull subscriptions.

• The immediate updating subscription connection to the Publisher (controlled by **sp\_link\_publication**) can use security mode 0 for SQL Server Authentication or 2 for linked server definition to create login mappings. The publication access list (PAL) must include at least one SQL Server Authentication account unless you use security mode 2 and configure delegation (it is possible to set up Windows Authentication in mode 2 by configuring delegation). You can make connections to the Publisher under Windows user accounts invoking the INSERT, UPDATE, and DELETE triggers at the Subscriber using delegation. To set up delegation, see <u>sp\_addlinkedsrvlogin</u>.

- When setting up a push subscription using the Push Subscription Wizard in SQL Server Enterprise Manager or the **sp\_addsubscription** stored procedure, the default configuration uses dynamic RPC at the Subscriber. The dynamic RPC defaults to using the **sa** login with no password. This is done to avoid sending logins or passwords over the network, and can be changed at the Subscriber using **sp\_link\_publication**.
- When setting up a pull subscription using the Pull Subscription Wizard in SQL Server Enterprise Manager, you choose the desired configuration mode. If you choose static RPC, the server name must already exist. If you choose dynamic RPC, you must supply a login and password that the synchronization triggers will use to connect to the Publisher.
- When setting up a pull subscription using stored procedures, you must explicitly call **sp\_link\_publication** after calling **sp\_addpullsubscription** at the Subscriber.

When using dynamic RPCs, Microsoft® SQL Server<sup>™</sup> 2000 handles login and password forwarding by adding a replication command to the distribution database to call **sp\_addsynctriggers** at the Subscriber. When executed at the Subscriber, **sp\_addsynctriggers** creates immediate updating triggers and configures the linked server connection.

When executed, the immediate updating stored procedures at the Subscriber check the PAL at the Publisher to ensure that the user account executing the RPC has permissions to update the data in the publication.

# Security and Replication Over the Internet

Different types of replication over the Internet have different security levels. Additionally, when transferring replication files using FTP sites, precautions must be taken to secure the site and still make it accessible to replication agents.

## Virtual Private Network

Using a Virtual Private Network (VPN) is the most secure option for implementing replication over the Internet. VPNs include client software so that computers connect over the Internet (or in special cases, even an intranet) to software in a dedicated computer or a server. Optionally, encryption at both ends as well as user authentication methods keep data safe. The VPN connection over the Internet logically operates as a Wide Area Network (WAN) link between the sites.

A VPN connects the components of one network over another network. This is achieved by allowing the user to tunnel through the Internet or another public network (using a protocol such as Microsoft Point-to-Point Tunneling Protocol (PPTP) available with the Microsoft® Windows NT® version 4.0 or Microsoft Windows® 2000 operating system, or Layer Two Tunneling Protocol (L2TP) available with the Windows 2000 operating system). This process provides the same security and features previously available only in a private network.

For more information, see Virtual Private Networks in the Windows 2000 documentation or <u>Publishing Data Over the Internet Using VPN</u>.

## **Microsoft Proxy Server**

Integrating Microsoft SQL Server<sup>™</sup> 2000 replication with Microsoft Proxy Server allows for replication over the Internet with security configured on the Microsoft Windows NT version 4.0 or Microsoft Windows 2000 Server operating systems, Proxy Server, and SQL Server 2000.

For replicating data over the Internet when a firewall is present, configuring replication with Microsoft Proxy Server provides security so that so that unauthorized Internet users cannot gain access to internal network resources, and the Subscriber can connect to a port on the Proxy Server that limits Subscriber access only to the services where permission is been granted.

For more information, search for the "Configuring Proxy Server for SQL Server Replication" white paper at <u>Microsoft Web site</u>.

## **TCP/IP and File Transfer Protocol**

For replication over the Internet where a firewall is not a concern, or for transferring snapshot files, you can use TCP/IP and File Transfer Protocol (FTP).

If you use FTP to download the snapshot files, define the FTP site without Write access. Although this is the default setting for many services, confirm that the setting has not been changed after installation.

**CAUTION** When a Subscriber completes applies the initial snapshot files from an FTP site, the files transmitted using FTP are left on the Subscriber disk. The files are visible to at least all other logins that can access the computer. The files are accessible to any users logged into the same computer. To prevent this, set the cache retention settings low and/or purge Microsoft Internet Explorer cache after applying snapshots.

For more information, see <u>Publishing Data Over the Internet Using TCP/IP and</u> <u>FTP</u>.

# **Enhancing Replication Performance**

You can enhance the general performance for all types of replication in your application and on your network by:

- Setting a minimum amount of memory allocated to Microsoft® SQL Server<sup>™</sup> 2000.
- Using a separate disk drive for the transaction log for all databases involved in replication.
- Consider adding memory to servers used in replication.
- Using multiprocessor computers.
- Setting a fixed size for the distribution database.
- Publishing only the amount of data required.
- Running the Snapshot Agent only when necessary and at off-peak times.
- Placing the snapshot folder on a drive not used to store database or log files.
- Using a single snapshot folder per publication.
- Consider using compressed snapshot files.
- Reducing the distribution frequency when replicating to numerous

Subscribers.

- Consider use of pull or anonymous subscriptions.
- Reduce the verbose level of replication agents to '0' except during initial testing, monitoring, or debugging.
- Run agents continuously instead of on very frequent schedules.
- Consider using the –UseInprocLoader agent property.

## Set a Minimum Amount of Memory Allocated to SQL Server

By default, SQL Server 2000 changes its memory requirements dynamically based on available system resources. To avoid low memory availability during replication activities, use the **min server memory** option to set the minimum available memory. If the server is a remote Distributor or a combined Publisher and Distributor, you must assign it at least 16 megabytes (MB) of memory. For more information, see <u>Server Memory Options</u>.

# Use a Separate Disk Drive for All Databases Involved in Replication

This applies to the publication database, the distribution database, and the subscription database. You can decrease the time it takes to write transactions by storing the log files on a disk drive different than the one used to store the database. You can mirror that drive, using a Redundant Array of Inexpensive Disks (RAID)-1, if you require fault tolerance. Use RAID 0 or 0+1 (depending on your need for fault tolerance) for other database files. This is a good practice regardless of whether or not replication is being used. For more information, see RAID Levels and SQL Server.

## **Consider Adding Memory to Servers Used in Replication**

If you need to improve replication performance, consider adding memory to the

servers used in replication. For example, if the computer is configured with 64 megabytes (MB) of memory, consider increasing the memory to 128 MB or more. You can use the **sp\_configure** stored procedure to assign additional memory to Microsoft® SQL Server<sup>™</sup> 2000.

## **Use Multiprocessor Computers**

SQL Server 2000 replication agents can take advantage of additional processors on the server. If you are running at high CPU usage, consider installing a faster CPU or multiple CPUs (symmetric multiprocessing).

## **Publish Only the Amount of Data Required**

Because replication is easy to set up, there is a tendency to publish more data than is actually required. This can consume additional resources within the distribution databases and snapshot files, and can lower the throughput for required data. Avoid publishing unnecessary tables and consider updating publications less frequently.

## **Run the Snapshot Agent Only When Necessary and at Off-Peak** Times

The Snapshot Agent bulk copies data from the published table on the Publisher to a file in the snapshot folder on the Distributor. In SQL Server 2000, the process of generating a snapshot for transactional replication no longer holds table locks on the published tables. Similarly, for merge replication in SQL Server 2000, concurrency is improved and lock duration is reduced when a snapshot is being generated. Although this reduces the impact on concurrently connected users, generating a snapshot is still a resource intensive process and is best scheduled during off-peak times.

## Place the Snapshot Folder on a Drive that Does Not Store Database or Log Files

Similarly, the Snapshot Agent will perform a sequential write of data to the snapshot folder when generating the snapshot for any publication type. Because the snapshot agent always copies a complete copy of the data in the publication to disk when replicating changes, placing the snapshot folder on a separate drive

from any database or log files reduces contention among the disks and helps the snapshot process complete faster.

## **Using a Single Snapshot Folder Per Publication**

When specifying the publication properties related to snapshot location, you can choose to generate snapshot files to the default snapshot folder, to an alternate snapshot folder, or to both. Generating snapshot files in both locations requires additional processing when the Snapshot Agent runs. This takes more time than generating the snapshot files to a single location for the publication.

For more information, see <u>Alternate Snapshot Locations</u>.

## **Consider Using Compressed Snapshots**

Compressing snapshot files in the alternate snapshot folder can reduce snapshot disk storage requirements and, in some cases, improve the performance of transferring snapshot files across the network when they are used for replication over the Internet. However, compressing the snapshot requires additional processing by the Snapshot Agent when generating the snapshot files, and by the merge agent when applying the snapshot files. This may slow down snapshot generation and increase the time it takes to apply a snapshot in some cases. Consider these tradeoffs carefully when using compressed snapshots.

For more information, see <u>Compressed Snapshot Files</u>.

## **Reduce the Distribution Frequency When Replicating to Numerous Subscribers**

A single Distributor can distribute transactions to a larger number of Subscribers if the Distribution and Merge Agents associated with each Subscriber are scheduled to run less frequently. Stagger when the Distribution Agents or Merge Agents are initially run so they do not all attempt to start simultaneously the first time they are started. If the agents are running on a scheduled basis, the schedules are set by default so that the agents are not running at the same time for regular synchronizations.

## **Consider Pull or Anonymous Subscriptions**

The Distribution or Merge Agent runs on the Distributor for push subscriptions, and on Subscribers for pull or anonymous subscriptions. Using pull or anonymous subscriptions can increase performance by moving Distribution or Merge Agent processing from the Distributor to Subscribers.

You can also offload agent processing by using Remote Agent Activation. Agent processing can be moved to the Subscriber for push subscriptions and to the Distributor for pull subscriptions. Administration of the agent still takes place at the Distributor for push subscriptions and at the Subscriber for pull subscriptions. For more information, see <u>Remote Agent Activation</u>.

Anonymous subscriptions, which are especially useful for Internet applications, do not require that information about the Subscriber be stored in the distribution database at the Distributor for transactional replication and reduces the storage of information about the Subscriber in the publishing database for merge replication. This reduces the resource demands on the Publisher and Distributor because they do not have to maintain information about anonymous Subscribers.

Anonymous subscriptions are a special category of pull subscriptions. In regular pull subscriptions, the Distribution or Merge Agent runs at the Subscriber (thereby reducing the resource demands on the Distributor), but still stores information at the Publisher. When a publication supports anonymous subscriptions, the publication is configured to always have a snapshot ready for new Subscribers.

For transactional replication, this means that every time the Snapshot Agent runs, a new snapshot will be generated. Typically, a snapshot is not generated if there are no new Subscribers waiting for a snapshot or no Subscriber needs to be reinitialized at the time the Snapshot Agent is run. So while anonymous Subscribers can reduce the resource demands at the Distributor, the tradeoff is that a snapshot is generated more often. With merge replication, a new snapshot is always generated when the Snapshot Agent runs regardless of the type of subscriptions supported by the publication.

## Additional Indexes at the Subscriber

If a subscription database needs to be used for decision support analysis and you add a lot of indexes to support these queries, you should note that these additional indexes may significantly reduce the throughput with which changes

can be applied to the Subscriber by the Distribution Agent or Merge Agent. In some cases, where you are mostly aggregating the data at the Subscriber, it may be more efficient to create an indexed view at the Publisher and publish it as a table to the Subscriber using transactional replication. For more information, see <u>Indexed Views</u>.

## Application Logic in Triggers at the Subscriber

Similarly, additional business logic in user defined triggers at the Subscriber may also slow down the replication of changes to the Subscriber. For transactional replication, it can be more efficient to include this logic in custom stored procedures used to apply the replicated commands. For more information, see <u>Using Custom Stored Procedures in Articles</u>.

## **Use Horizontal Partitioning Judiciously**

When a transactional publication is set up with an article(s) that is horizontally partitioned, the log reader has to apply the filter to each row affected by an update to the table as it scans the transactions log. The throughput of the log reader will therefore be affected. If achieving maximum throughput is key, you should consider using DTS custom partitions to do custom horizontal partitions . That allows the log reader agent to move transactions out of the published database's log as quickly as possible. Instead of affecting all Subscribers with the overhead of filtering the data, only the subscriber that chooses to use a DTS package to filter the data is affected.

Similarly, merge replication must evaluate changed or deleted rows to determine every time you synchronize changes to determine which Subscribers should receive those rows. When horizontal partitioning is employed to reduce the subset of data required at a Subscriber, this processing is more complex and can be slower than when you publish all rows in a table. Consider carefully the tradeoff between reduced storage requirements at each subscriber and the need for achieving maximum throughput.

## Use a Fast Network

The propagation of changes to the Subscriber can be significantly enhanced by using a very fast network of 100 Mbps or faster.

## **Reduce the Verbose Level of Replication Agents**

Reduce the **–HistoryVerboseLevel** parameter and/or the **–OutputVerboseLevel** parameter of the Distribution Agents or Merge Agents to the lowest value. This will reduce the amount of new rows inserted to track agent history and output. Instead, previous history messages with the same status will be updated to the new history information. Changing this agent parameter can yield a significant performance gain of up to or over 10 to 15 percent.

However, you should increase the –HistoryVerboseLevel for testing, monitoring, and debugging so that you have as much history information about agent activity as possible.

## **Run Agents Continuously Instead of on Very Frequent Schedules**

Setting the agents to run continuously rather than creating frequent schedules (such as every minute) will improve replication performance. When you set the Distribution Agent or Merge Agent to run continuously, whenever changes occur, they will be immediately propagated to the other servers that are connected in the topology. Because the agent is continuously running, it does not have to start and stop which causes more work for the server where the agent is running.

## Consider Using the –UseInprocLoader Agent Property

The **–UseInprocLoader** agent property improves performance of the initial snapshot for snapshot replication, transactional replication, and merge replication.

When you apply this property to either the Distribution Agent (for snapshot replication or transactional replication) or the Merge Agent (for merge replication), the agent will use the in-process BULK INSERT command when applying snapshot files to the Subscriber.

The **–UseInprocLoader** property cannot be used with character mode **bcp**, and it cannot be used by OLE DB or ODBC Subscribers.

**IMPORTANT** When using the **–UseInprocLoader** property, the SQL Server 2000 account under which the Subscriber is running must have read permissions on the directory where the snapshot .bcp data files are located. When the –

**UseInprocLoader** property is not used, the agent (for heterogeneous Subscribers) or the ODBC driver loaded by the agent (for SQL Server 2000 Subscribers) reads from the files, so the security context of the Subscriber SQL Server 2000 account is not used.

## **Enhancing Snapshot Replication Performance**

You can enhance the performance of snapshot replication in your application and on your network by:

- Using a quality disk subsystem.
- Using a single snapshot folder per publication.
- Using compressed snapshots.
- Using native **bcp**.

#### Use a Quality Disk Subsystem

Because snapshot replication bulk copies a complete copy of the publication, it writes the entire publication to the snapshot folder. The faster the disk subsystem can read and write data to the disk(s), the faster the snapshot is completed.

#### **Using a Single Snapshot Folder Per Publication**

When specifying the publication properties related to snapshot location, you can choose to generate snapshot files to the default snapshot folder, an alternate snapshot folder, or both. Generating snapshot files in both locations requires additional processing when the Snapshot Agent runs. This takes more time than generating the snapshot files to a single location for the publication.

For more information, see <u>Alternate Snapshot Locations</u>.

#### **Consider Using Compressed Snapshots**

Compressing snapshot files in the alternate snapshot folder can reduce snapshot disk storage requirements and, in some cases, improve the performance of transferring snapshot files across the network when they are used for replication over the Internet. However, compressing the snapshot requires additional

processing by the Snapshot Agent when generating the snapshot files and by the merge agent when applying the snapshot files. This may slow down snapshot generation and increase the time it takes to apply a snapshot in some cases. Consider these tradeoffs carefully when using compressed snapshots.

For more information, see <u>Compressed Snapshot Files</u>.

## **Consider Using Native bcp**

When you are not using ODBC or OLE DB Subscriber or using transformable subscriptions and you have a large volume of data, consider using native **bcp** mode to apply snapshot files to Subscribers. Storing information in native format is useful when information must be copied from one instance of Microsoft® SQL Server<sup>TM</sup> to another.

# Enhancing Snapshot and Transactional Replication Performance

You can enhance the performance of snapshot or transactional replication in your application and on your network by:

- Configuring the Distributor on a dedicated server.
- Increasing memory on the Distributor.
- Subscribing to all articles in a publication.
- Using stored procedure replication when a large number of rows are affected.
- Minimizing the retention period for transactions and history.

## **Configure the Distributor on a Dedicated Server**

You can reduce processing overhead on the publishing server by configuring a computer dedicated to the distribution process. This may result in performance gains for both the Publisher and the Distributor.

## **Increase Memory on the Distributor**

In addition to the benefits of maintaining a dedicated Distributor, you can realize additional performance gains by increasing the amount of memory on the Distributor. This is especially true if the Distributor is supporting replication to a large number of Subscribers. For example, if the computer is configured with 64 megabytes (MB) of memory, consider increasing the memory to 128 MB or more. You can use the **sp\_configure** stored procedure to assign additional memory to Microsoft® SQL Server<sup>™</sup> 2000.

## Subscribe to All Articles in a Publication

By default, a subscription includes all the articles in a publication. By not having to exclude any articles from a publication, the Distribution Agent can use an optimal query during synchronization.

# Use Stored Procedure Replication When a Large Number of Rows are Affected

If a single set update/delete at the Publisher affects a very large number of rows, the change to each row affected by the update is logged individually in the transaction log of the database. The log reader will propagate these as individual updates (within a single transaction) and when the Distribution Agent applies the changes it can take much longer than the original update at the Publisher.

If you have batch updates that occasionally affect a large number of rows at the Subscriber, you should consider updating the published table using a stored procedure and publish the execution of the stored procedure. Instead on a sending an update/delete for every row affected by the update/delete, the Distribution Agent will execute the same procedure at the subscriber with the same parameter values. This is faster by a large magnitude compared to sending the update/delete as individual row changes. For more information see <u>Publishing Stored Procedure Execution</u>.

## **Use Custom Stored Procedures to Update Subscribers**

By default when a Subscriber is set up for transactional replication, the process of applying a snapshot to a Subscriber, in addition to creating the table(s) and populating them, will also create a set of stored procedures at the Subscriber (for INSERT, UPDATE and DELETE).

Subsequently when changes are made to a published table, the log reader will construct a stored procedure call instead of SQL statements representing the change. The distribution agent then executes this while applying changes to a Subscriber. This is much more efficient than SQL statements over which it provides significant performance gains.

These stored procedures can be further customized, which is generally better than adding Subscriber-specific logic in triggers (for actions such as maintaining aggregate tables).

For more information, see <u>Using Custom Stored Procedures in Articles</u>.

## Minimize the Retention Period for Transactions and History

You can reduce the amount of disk space used on the Distributor by minimizing the amount of time that replicated transactions and history are stored in the distribution database after they have been delivered to Subscribers.

# **Reduce Unnecessary Reinitialization or Expiration of Subscriptions**

If a Subscriber does not synchronize for a long time there is a possibility the subscription may get dropped or it may be automatically marked deactivated and require reinitialization. Whether it expires and is dropped or gets marked deactivated and requires initialization depends upon whether it exceeds the Subscription Expiration property of the publication or the Maximum Transaction Retention property of the distribution database as well as whether or not it is an Anonymous subscriber.

If you do not want your subscriptions to expire, you should set the publication retention to "0". If you do not want your subscriptions to be deactivated you should set the Maximum Distribution Retention period to a higher value than the default of 72 hours taking into consideration the effect it may have on the size of the distribution database. For more information, see <u>Subscription Deactivation</u> and <u>Expiration</u>.

## Use a Quality Disk Subsystem

Because snapshot replication copies a complete copy of data in the publication, it writes data for the entire publication to the snapshot folder. The faster the disk subsystem can read and write data to the disk(s), the faster the snapshot is completed.

## **Enhancing Transactional Replication Performance**

You can enhance the performance of transactional replication in your application and on your network by:

- Increasing the Log Reader Agent read batch size.
- Minimizing the log history and retention period.
- Optimizing your database design to include replication considerations.
- Using custom stored procedures for inserts, updates, and deletes at Subscribers.
- Avoiding horizontal filtering.

#### **Increase the Log Reader Agent Read Batch Size**

The Log Reader Agent and Distribution Agents support batch sizes for transaction read and commit operations. Batch sizes default to 500 transactions. When a large number of transactions are written to a publishing database but only a small subset of those are marked for replication, you should use the **- ReadBatchSize** parameter to increase the read batch size of the log reader. The Log Reader Agent reads the specific number of transactions from the log, whether or not they are marked for replication. For more information, see <u>Replication Log Reader Agent Utility</u>.

#### **Minimize the Log History and Retention Period**

You can reduce the amount of disk space used on the Distributor by minimizing the amount of time for log history and transaction retention. For more information, see <u>Transactional Replication</u>.

## **Optimize Your Database Design to Include Replication**

## Considerations

Horizontal partitions can inhibit replication performance. Consider database design options that reduce the need to filter rows when defining articles in a publication. Alternatively, consider using custom stored procedures that can delete unnecessary rows at the Subscriber or using custom data partitions with transformable subscriptions (for more information, see <u>Using Transformable</u> <u>Subscriptions to Create Custom Data Partitions</u>).

# Use Custom Stored Procedures for Inserts, Updates, and Deletes at Subscribers

When Microsoft® SQL Server<sup>™</sup> 2000 applies transactions at a Subscriber, by default it overrides the INSERT, UPDATE, and DELETE statements from the transaction log with custom stored procedures. For example, instead of applying the INSERT statement read from the transaction log, the Distribution Agent can run a stored procedure at the Subscriber to perform the same action. These stored procedures can be further customized, which is generally better than adding Subscriber-specific logic in triggers (for actions such as maintaining aggregate tables).

## **Avoid Horizontal Filtering**

The criteria set for a horizontal filter are evaluated one time for each row marked for replication in the publication database log. This determines whether the row should be moved to the distribution database. For applications that require maximum data throughput, horizontal filtering of articles may not be the best choice for minimizing the rows delivered to each Subscriber. Instead, developing natural partitions of the table may be a better choice using custom data partitions with transformable subscriptions (for more information, see <u>Using</u> <u>Transformable Subscriptions to Create Custom Data Partitions</u>).

## **Enhancing Merge Replication Performance**

You can enhance the performance of merge replication in your application and on your network by:

- Using indexes on columns used in subset and join filters.
- Creating a ROWGUIDCOL column prior to generating the initial snapshot.
- Using native mode **bcp** whenever possible.
- Increasing the batch sizes processed by the Merge Agent.
- Using pull and anonymous subscriptions when there are a large number of Subscribers.
- Limiting the use of **text** and **image** columns.
- Considering over-normalization of tables containing **text** and **image** columns.
- Using static rather than dynamic partitions when possible.
- Using dynamic snapshots for dynamically filtered publications.
- Limiting complexity of subset filter clauses.
- Reducing publication retention settings.

- Selecting column-level tracking when bandwidth is limited.
- Optimizing synchronization when partitioning data.
- Controlling article processing order if using triggers for referential integrity.
- Using global subscriptions.
- Occasionally re-indexing merge replication system tables.
- Not overusing join filters.
- Modifying database design.
- Limiting or controlling simultaneous agent processing.
- Considering Reinitialization of the subscription.

#### Using Indexes on Columns Used in Subset and Join Filters

When you use a filter on a published article, create an index on each of the columns that is used in the filter's WHERE clause. Without an index, Microsoft® SQL Server<sup>™</sup> 2000 has to read each row in the table to determine whether the row should be included in the article (that is, in the horizontal partition of the table). With an index, SQL Server 2000 can quickly locate which rows should be included. The fastest processing takes place if SQL Server 2000 can fully resolve the WHERE clause of the filter from just the index.

Indexing all the columns used in JOIN filters is also important. Each time the Merge Agent runs, it searches the base table to determine which rows in the base table and which rows in related tables are included in the article. Creating an index on the JOIN columns saves SQL Server 2000 from having to read each

row in the table every time the Merge Agent runs.

For more information, see <u>Filtering Published Data</u>.

## **Create a ROWGUIDCOL Column Prior to Generating the Initial Snapshot**

By creating a column that can be used to help track changes during merge replication, you will avoid the sometimes significant time (and disk and log) decrease in performance that occurs from waiting for the Snapshot Agent to alter the tables for you.

Merge replication requires that each published table have a ROWGUIDCOL column. If a ROWGUIDCOL column does not exist in the table before the Snapshot Agent creates the initial snapshot files, the agent must first add and populate the ROWGUIDCOL column. To gain a performance advantage when generating snapshots during merge replication, create the ROWGUIDCOL column on each table before publishing using merge replication. The column can have any name (rowguid is used by the Snapshot Agent by default), but must contain the following data type characteristics:

- The data type as UNIQUEIDENTIFIER.
- The default as NEWID().
- The ROWGUIDCOL property.
- A unique index on the column.

The ROWGUIDCOL column is used frequently in merge replication during tracking and synchronization of changes made at the Publisher and at Subscribers.

## Increase the Batch Sizes Processed by the Merge Agent

By default, the Merge Agent processes 100 generations in each batch uploaded and downloaded between the Publisher and Subscriber. If you make frequent updates to a single table and update a large number of rows in a single transaction, consider increasing the number of generations in each batch. You can set the **-DownloadGenerationsPerBatch** and **- UploadGenerationsPerBatch** parameters in the Merge Agent profile.

# Use Pull and Anonymous Subscriptions When There Are a Large Number of Subscribers

A pull subscription moves the Distribution Agent from the Distributor to the Subscriber. Relocating the Distribution Agent reduces the amount of processing the Distributor must do for each pull subscription and shifts the processing overhead to the Subscriber. By creating pull subscriptions instead of push subscriptions, you free up more processing capacity at the Distributor for performing other replication or application tasks. By creating anonymous subscriptions, you can further reduce the resource demands on the Distributor because no meta data has to be stored about the Subscriber.

## Use Native Mode bcp Whenever Possible

When you create a publication, you have the choice of specifying that one or more Subscribers will be Microsoft Jet 4.0 (Microsoft Access) or SQL Server for Windows® CE databases. Enabling support for these types of Subscribers causes the Snapshot Agent to store the snapshot files in character format instead of native SQL Server 2000 format. Because it takes additional processing time and storage space for SQL Server 2000 to process and store character format files, do not enable SQL Server for Windows CE or Access Subscribers unless you are sure that you will actually have such Subscribers.

## Limit the Use of text and image Columns

**text** and **image** columns require more storage space and processing than other column data types. Do not include **text** and **image** columns in articles unless absolutely necessary for your application.

## **Consider Over-normalizing Tables Containing text and image Columns**

When synchronization occurs, the Merge Agent may need to read and transfer

the entire data row from a Publisher or Subscriber. If the row contains **text** and **image** columns this process can require additional memory allocation and negatively impact performance even though these columns may not have been updated. To reduce the likelihood that this performance impact will occur, consider putting **text** and **image** columns in a separate table using a one-to-one relationship to the rest of the row data.

# **Reducing Use of Horizontal Filtering**

When subset filters or join filters are used to filter the data in a publication, the Merge Agent must determine if rows need to be added to or removed from a subscription database. While this can decrease the amount of data that must be transferred to each Subscriber, it can increase the amount of processing required at the Publisher during each synchronization. If data is not horizontally filtered, all data changes must be sent to each Subscriber and the Merge Agent will begin sending data to Subscribers quickly without having to first evaluate the filter criteria for the publication.

Publish unrelated tables in separate publications if some tables receive a lot of activity and others do not.

#### Use Static Rather Than Dynamic Partitions When Possible

Dynamic filters and partitions are a powerful feature of SQL Server 2000 replication. However, even with indexes on the filtered columns, SQL Server 2000 must still read each row in the dynamic partition and compare it to the filtered value. Using static filters and partitions reduces the processing time required to complete the merge process.

#### **Using Dynamic Snapshots for Dynamic Filtered Publications**

When dynamic filters are used to partition a publication, the Snapshot Agent cannot pre-determine the data required for a Subscriber. As such, the Merge Agent must request inserts for all data specific to its partition after it applies the schema files from the snapshot folder. Processing the initial snapshot for a large volume of data using inserts can be significantly slower than processing the same data using the SQL Server bulk copy utility.

With SQL Server 2000, the Snapshot Agent can be instructed to generate bcp

files specific to each subscriber by creating a dynamic snapshot job. Dynamic snapshots will generate bcp files as though a static filter had been applied to the publication. While this requires running the Snapshot Agent in a special mode once for each partition of data to be generated for Subscribers, it can dramatically improve the time it takes the Merge Agent to apply the data when processing the snapshot files.

For more information, see **Dynamic Snapshots**.

# Limiting Complexity of Subset Filter Clauses

When using subset filter clauses to horizontally partition data in a publication, limit the complexity of the filter criteria. The subset filter clause will be evaluated frequently to determine which changed rows of published data should be synchronized with each Subscriber. Limiting the complexity of the filtering criteria will help improve performance when the merge agent is evaluating row changes to send to Subscribers. Avoid using sub-selects within merge subset filter clauses. Instead, consider using join filters, which are generally more efficient when used to partition data in one table based on the subset filter clause in another table.

**Note** Do not overuse join filters. Join filters with dozens or more tables will impact performance.

For more information, see Filtering Published Data.

# **Reducing Publication Retention Settings**

Publication retention determines how long a Subscriber can go without synchronizing incremental changes before that Subscriber is considered to be out of synchronization and requires a new snapshot from the Publisher.

This setting also controls how long some merge tracking meta data is maintained in the publication and subscription databases. You can control the growth of merge tracking meta data and, in some cases, see improved performance while synchronizing changes, if you reduce the publication retention period setting. Select a publication retention setting that is adequate to support Subscribers working offline for extended periods of time.

#### Selecting Column-level Tracking When Bandwidth is Limited

While business application needs generally drive the choice selection of row- or column-level tracking for merge publications, there can be a performance benefit to selecting column-level tracking when bandwidth availability is low. Column-level tracking of data changes allows the Merge Agent to send only the changed columns and **rowguidcol** property across the network for changed rows. Conversely, the Merge Agent will always send the entire row when row-level tracking is used. Sending only the changed columns can provide better performance across a network with limited bandwidth when an application frequently changes only a few columns in a table that has many columns.

#### **Optimizing Synchronization When Partitioning Data**

Selecting the @keep\_partition\_changes option when adding an article to a merge publication can significantly reduce the amount of time it takes the Merge Agent to determine whether recently changed rows should be sent to a Subscriber.

In cases where an application updates a column used in a subset filter or join filter, the Merge Agent must do additional work to determine if that row change requires that rows be added to or removed from the partition for each Subscriber as they synchronize. By maintaining some additional data about the changed rows in the publication database, the Merge Agent can more quickly determine which partition-related row changes are relevant to each Subscriber.

**CAUTION** Choosing to maintain this additional information at the Publisher will result in an increase in the storage requirements for the merge replication tracking system tables in the publication database. However, if UPDATES to columns included in partitions are not atypical, the performance gains are usually worth maintaining the additional information.

For more information, see Optimizing Synchronization.

# **Controlling Article Processing Order If Using Triggers for Referential Integrity**

When publishing tables related to one another via declared foreign key constraints or constraints enforced via triggers, the Merge Agent will need to apply changes to related rows in the correct order to propagate all changes. If

you are using declared referential integrity, SQL Server will process articles in order based on the relationships. By processing articles in the optimal order based on the action being performed (for example, inserting parent rows before related child rows), the Merge Agent can avoid additional retry operations when processing articles during synchronizing.

If declared referential integrity is not used, the Merge Agent will, by default, process articles in the order they are added to a publication via stored procedures or SQLDMO – article order cannot be controlled through SQL Server Enterprise Manager.

If triggers are used to enforce referential integrity, the Merge Agent will not recognize this as declared referential integrity, and you need to be aware of the processing order of the articles.

# **Using Global Subscriptions**

When synchronizing changes for a local or anonymous merge Subscriber, the Publisher must also synchronize additional system tracking data that would otherwise be unnecessary with global subscriptions. Using global subscriptions may improve synchronization performance in cases where subscribers make frequent updates.

#### **Occasionally Re-index Merge Replication System Tables.**

As part of maintenance for merge replication, occasionally check the growth of the system tables associated with merge replication: MSmerge\_contents, MSmerge\_genhistory, and MSmerge\_tombstone. Periodically re-index these tables by running DBCC Transact-SQL commands. To re-index these system tables, execute the following commands on the publication database:

- DBCC DBREINDEX ('MSmerge\_contents')
- DBCC DBREINDEX ('MSmerge\_genhistory')
- DBCC DBREINDEX ('MSmerge\_tombstone')

Additionally, you should minimize the size of the merge system tables

(specifically MSmerge\_history) by using **sp\_mergecleanupmetadata**. For more information, see <u>How Merge Replication Works</u>.

#### **Not Overusing Join Filters**

Join filters with several tables (such as dozens or hundreds of tables) will seriously impact performance during merge processing. It is recommended that if you are generating join filters of five or more tables that you consider other solutions. Another strategy might be to avoid filtering tables that are primarily lookup tables, smaller tables, and tables that are not subject to change. Make those tables part of the publication in their entirety. It is recommended that you use join filters only between tables for which it is important they carefully partition among Subscribers.

#### **Modify Database Design**

The design of the database ultimately determines the complexity and processing resource requirements of the queries used by merge replication, which affects merge performance. A poor database design or a database design that does fit with the publication (or filtering) needs of a merge publication may require some structural changes to the database to improve merge performance. Specifically, adding columns or tables to support dynamic partitioning logic more efficiently, and making sure that the columns used in the filtering expressions can take advantage of indexes. Generic 'optimizing queries for index usage' rules apply. If you generically mention to use indexes on all filtering columns, this may actually be counter-productive in terms of index maintenance if the index is not used by the query optimizer, because the data is not very unique or the expression cannot use indexes. Sometimes changing the filtering expressions will allow an existing index to be used where it was not before.

#### Limit or Control Simultaneous Agent Processing

Limit or control the number of multiple simultaneous Snapshot Agent or Merge Agent processes, especially with large data sets, complex partitioning logic, and large volumes of merged changes. The @max\_concurrent\_merge and @max\_concurrent\_dynamic\_snapshots parameters for sp\_addmergepublication can help with this.

#### **Consider Reinitializing the Subscription**

When large amounts of changes need to be sent to subscribers, reinitializing them with a new snapshot may be faster than using merge to move the individual changes.

#### See Also

Agent Profiles Creating an Index Creating and Modifying Identifier Columns Data Types and Table Structures Dynamic Filters Planning for Replication Replication

# **Backing Up and Restoring Replication Databases**

In addition to the regular backup and restore guidelines and procedures for Microsoft® SQL Server<sup>™</sup> 2000, additional considerations for backing up and restoring the databases are involved in replication.

The considerations for backing up databases used in snapshot replication, transactional replication, or merge replication vary according to the role the server performs in replication and where the failure occurs in the replication topology.

To restore replication, back up some or all of the following regularly:

- Publisher
- Distributor
- Subscriber(s)

Your backup strategy will depend on your needs for restoring a replicated environment quickly, and on the degree of complexity you can tolerate in your backup plan. You only need to back up all databases if you want to restore any replica immediately from backup while minimizing the likelihood of data loss.

Maintaining a regular backup of the Publisher databases, and leveraging the SQL Server replication built-in ability to reinitialize one or more subscriptions ondemand provides a simple recovery strategy. This strategy can be used to support a large enterprise of mobile, occasionally connected Subscribers that otherwise would not typically participate in regular backup management at each node in the topology. You could further limit regular backups to your publication databases and rely on SQL Server replication scripting to provide a method for reestablishing replication if you need to restore the entire replication environment.

Another strategy includes backing up only the Publisher and the Distributor as long as the Publisher and Distributor are synchronized. This strategy allows you to restore a replication environment completely. Backing up a Subscriber is optional but can reduce the time it takes to recover from a failure of the Subscriber.

Basic backup plans can result in a longer time to restore the replication environment. If your application requires that you restore replication immediately, you may want to consider more complex backup and recovery strategies described later in this section.

In most situations, the publications and distribution databases should be backed up after adding or changing replication objects such as articles and subscriptions, or after a schema change is made that affects replication. If the distribution database is restored to a version that is before such a change, the publication database will have to be restored to a version before that change as well.

As part of any backup strategy, always keep a current script of your replication settings in a safe location. This should be done in addition to regular backups of the Publisher, Distributor and the Subscribers. In the event of a total server failure or the need to set up a test environment, you can modify the script by changing the server name references and using the script to help recover replication with the previous settings.

You should also script the enabling and disabling of replication. These scripts are part of the backup of the Publisher or Distributor.

For more information about generating SQL scripts for setting up or disabling replication, see <u>Scripting Replication</u>.

#### **Backing Up the Publisher**

Publication databases are the primary, or central source, of data in a replication topology; therefore, even the most basic recovery plan should include regular backups at the Publisher. Backing up the Publisher requires you to back up the publication database regularly on the server where the Publisher is located. Back up the publication database and then make transaction log backups and/or differential database backups. You can also back up the **master** and **msdb** system databases to protect against total loss of the system and not just the publication database. If you are shipping transaction logs to a warm standby server, back up the **msdb** system database regularly (which is required if log shipping is used).

#### **Backing Up the Distributor**

Backing up the Distributor involves backing up the distribution database, the **msdb** database, and the **master** system database. This allows you to recover from almost any type of failure without having to re-create publications or reconfigure replication.

Backing up the Distributor preserves the snapshot of the publication as well as the history, error, and replication agent information for your application. It allows you to recover faster in the event of a Publisher or Distributor failure because there is no need to re-establish replication. Particularly for transactional replication, this strategy requires coordination between backing up the publication database and the distribution database. SQL Server 2000 handles this coordination automatically. Back up the distribution database, and then make transaction log backups and differential database backups.

For more information, see <u>Strategies for Backing Up and Restoring</u> <u>Transactional Replication</u>.

#### **Backing Up the Subscriber**

A comprehensive backup recovery strategy may rely on reinitialization of subscriptions in the event that recovery is required, or may include regular backups of each subscription database and relevant system databases at the Subscriber. Backing up the Subscriber involves backing up the subscription database and, optionally, the **msdb** and **master** system databases. The **msdb** and **master** databases need to be backed up only if it is a Subscriber that uses pull subscriptions and only if there is a need to be able to restore after a total system loss.

Backup the subscriptions database and then make transaction log backups and incremental database backups.

**Note** Backing up each Subscriber is not required to reestablish replication after a failure. Under most circumstances, backing up the Publisher and Distributor regularly should be sufficient. If the cost of reinitializing a Subscriber is significantly greater than the cost of restoring it from a backup, and the complexity of managing backups among the replicas within the enterprise is manageable, you should consider backing up the Subscriber.

See Also

Validating Replicated Data

Replication

# Strategies for Backing Up and Restoring Snapshot Replication

Snapshot replication is best used as a method for replicating data that changes infrequently or where the most up-to-date values (low latency) are not a requirement. When synchronization occurs, the entire snapshot is generated and sent to Subscribers.

Because snapshot replication propagates changes by generating and delivering a complete snapshot for the publication, it is not necessary to back up the publication database as frequently as it is backed up in transactional replication or merge replication. The publication database needs to be backed up when changes are made to existing publication properties or when new publications are added.

When you back up the Publisher, also back up the Distributor. While the backups are in progress, no new snapshot publications or subscriptions should be added. This ensures that when the Publisher and Distributor are restored, they both will both contain the same information.

#### **Backing Up and Restoring the Publication Database**

The Log Reader Agent is less important in back up and restoration than it is in transactional replication. The publication database needs to be backed up only when changes are made to existing publications (such as an article added or deleted, or schema changes on the publication database that affect the publication), or new publications are added.

#### **Backing Up and Restoring the Distribution Database**

Before backing up the distribution database, it is recommended that you run the Distribution Cleanup Task to make sure any unnecessary information is cleaned up and does not add to the time it takes to back up the distribution database.

The distribution database should be backed up at the same time as the publication database. During the back up, do not add new snapshot publications or subscriptions.

#### **Backing Up and Restoring the msdb Database**

The **msdb** database contains the job definitions for replication agents that are run under the control of SQL Server Agent. To provide additional security against a total system failure, the **msdb** database on the Publisher, Distributor, and Subscribers (that use pull subscriptions) must be backed up periodically whenever a subscription is dropped, whenever a change is made to a replication agent, or when a new Publisher is added to the Distributor.

#### **Backing Up and Restoring the master Database**

When a new Subscriber is added, an entry for the Subscriber is added to the **sysservers** table in the **master** database on the Publisher. When a new Publisher is added to a Distributor, an entry for the Publisher is added to the **sysservers** table in the **master** database on the Distributor.

To restore replication after the loss of the Publisher or Distributor, back up the **master** database on the Publisher and Distributor each time a new Subscriber or Publisher is added (respectively).

Replication

# Strategies for Backing Up and Restoring Transactional Replication

Microsoft® SQL Server<sup>™</sup> 2000 allows you to restore transactional replication databases without reinitializing subscriptions or disabling and reconfiguring publishing and distribution. You can set up replication to work with log shipping, enabling you to use a warm standby server without reconfiguring replication.

Recovering transactional replication from a loss of the publication database or distribution database, without having to reinitialize subscriptions or reconfigure replication, requires the publication database and the distribution database be restored to a consistent point in time. In SQL Server version 7.0 and earlier, this had to be ensured manually by backing up the publication database and distribution database simultaneously, and at the same time ensuring no changes were being made to the databases while the backup was in progress. SQL Server 2000 automatically handles the coordination of the backups of the two databases.

To ensure that you can restore the Publisher or Distributor at any time, SQL Server 2000 requires the replication database option **sync with backup** be set to **true** on the publication database and on the distribution database. If you use this option, you will need to back up the publication database and distribution databases (usually you would back up the transaction log or make differential backups) frequently because the frequency of backups determines the latency with which replication delivers changes to Subscribers.

**IMPORTANT** Only SQL Server 2000 Publishers support this option. If the distribution database is set to **sync with backup**, Publishers running SQL Server 7.0 and earlier and using that distribution database will be treated as if the option is not set.

#### **Backing Up and Restoring the Publication Database**

Usually the Log Reader Agent runs in continuous mode, monitoring the log for data changes, which it immediately propagates to the distribution database (typically within a few seconds). In addition, because backups of the publication database usually occur on a scheduled basis, the Log Reader Agent may be transferring transactions faster than they are being backed up. If the Publisher fails and is restored, the distribution database may already have transactions that will not exist in the restored publication database because those transactions were not backed up.

Setting the **sync with backup** option on the publication database ensures that the Log Reader Agent will not propagate any transactions to the distribution database that have not been backed up at the Publisher. This ensures that the last backup can be restored without any possibility of the distribution database having transactions that the restored publication database does not have.

Synchronizing the Log Reader Agent with backing up the publication database means that replication latency (the time it takes for changes at the Publisher to be delivered to the Subscriber), which can often be as low as a few seconds, is now constrained to be equal to the frequency of backups at the Publisher. For example, if you are backing up the transaction log of the publication database every five minutes, replication latency could be as much as five minutes plus the time it takes to complete the backup. On the average, it will be less than five minutes, but more than typical transactional replication latency, which can be tens of seconds. If you synchronize the Log Reader Agent with the backup, it is recommended that you back up the publication database (database backup followed by log and/or differential database backups) as frequently as possible to reduce the time it takes for changes to appear at Subscribers.

#### To synchronize the publication database to a backup

• Execute sp\_replicationdboption '<publicationdatabasename>', 'sync with backup', 'true'.

**Note** If you change the **sync with backup** option to **false**, the truncation point of the publication database will be updated after the Log Reader Agent runs, or after an interval if the Log Reader Agent is running continuously. The maximum interval is controlled by the **–MessageInterval** agent parameter with a default of 30 seconds.

To determine if the **sync with backup** option has been set on a publication database, use the **IsSyncWithBackup** property of the **DatabasePropertyex()** intrinsic function. You can also run the system stored procedure sp\_helpdb to check if this option has been set.

If the increase in replication latency is not acceptable, do not to set the **sync with** 

**backup** option on the publication database. If the publication database fails, it will be possible for the distribution database to have transactions that the restored publication database does not have, and it is not guaranteed that the Subscriber will be in synchronization with the Publisher.

# **Restoring the Publication Database When the sync with backup Option is False**

If you do not set the **sync with backup** option and allow the distribution database to have transactions that the restored publication database does not have, it is possible to restore a publication database from backup and for replication to continue, but the Subscriber and Publisher may no longer be in synchronization. To accomplish this:

- 1. Restore the publication database. At this point, you will get an error from the Log Reader Agent because it will detect that the Distributor is ahead of the Publisher.
- 2. Run **sp\_replrestart** in the publication database with no parameters. This forces replication to continue even if the Distributor and some Subscribers may now have data that the Publisher no longer has.
- 3. Ensure that the Distribution Agents, which could now deliver duplicate rows to Subscribers, can continue despite these failures. Choose the **–SkipError** Distribution Agent profile, or you can manually add the **–SkipError** parameter to the runtime parameters of the Distribution Agents and supply the errors you want the Distribution Agents to ignore. For more information, see <u>Distribution Agent Profile</u>.

**CAUTION** This method can lead to inconsistencies between data at the Publisher and data at the Subscribers.

#### **Backing Up and Restoring the Distribution Database**

The distribution database can be restored to the last backup without reconfiguring replication or reinitializing subscriptions. Usually, the Log Reader

Agent connects to the publication database, scans the log, retrieves the next set of N transactions that need to be replicated, propagates them to the distribution database, and then indicates to the publication database that the transactions have been successfully committed at the distribution database.

At this point, the publication database can truncate the part of the log that contains these transactions (provided they have been backed up). If the distribution database fails at this point and is restored to a previous backup, it will not be possible for the Log Reader Agent to deliver the missing transactions because the part of the log containing them may have been truncated.

Setting the **sync with backup** option on the distribution database ensures that the log of the publication database will not be truncated beyond the point up to which all transactions have been propagated to the distribution database. It also ensures that the distribution database with the new transactions has been backed up. The distribution database can be restored to the last backup and the Log Reader Agent will be able to deliver transactions that the restored distribution database is now missing. Replication will continue unaffected.

**IMPORTANT** To backup the distribution database more frequently by backing up the transaction logs and setting the **sync with backup** option, you must set the **trunc. log on chkpt.** option of **sp\_dboption** to **false** on the distribution database.

Unlike the publication database, setting **the sync with backup** option on the distribution database has no effect on replication latency, but it will delay the truncation of the log on the publication database until the corresponding transactions in the distribution database have been backed up. The **sync with backup** option is available only if the Publisher and Distributor are running SQL Server 2000.

#### To synchronize the distribution database to a backup

• Execute sp\_replicationdboption '<distributiondatabasename>', 'sync with backup', 'true'

To determine if the **sync with backup** option has been set on a distribution database, use the **IsSyncWithBackup** property of the **databaseproperty()** intrinsic function. You can also run the system stored procedure **sp\_helpdb** to check if this option has been set.

#### **Backing Up and Restoring a Subscription Database**

To restore the Subscriber to the last backup without any need to reinitialize the subscriptions, ensure that the minimum transaction retention period at the Distributor is greater than the frequency of the backup interval at the Subscriber. This guarantees that when you restore a Subscriber, all the transactions necessary for the Subscriber to catch up will still be available in the distribution database. When you restore a Subscriber, the Distribution Agent delivers any transactions the Subscriber is missing. By default, the minimum transaction retention period is set to **0**, and under most circumstances a transaction that has been delivered to all Subscribers will be deleted.

#### To set the minimum transaction retention period of the Distributor

- 1. In SQL Server Enterprise Manager, expand a server group, expand the Distributor, right-click the **Replication** folder, and then click **Configure Publishing, Subscribers, and Distribution**.
- 2. To modify the distribution database, click the **Properties** button for the distribution database to change the transaction retention period.

**Note** It is not necessary to back up the Subscribers to restore transactional replication; however, it is essential that you back up the Publisher and Distributor. If the cost of generating a snapshot and reinitializing the subscription is less than the time it would take to restore the subscription database from a backup, there is no need to back up the subscription database.

#### **Backing Up and Restoring the msdb Database**

The **msdb** database contains the job definitions for replication agents that are run under the control of SQL Server Agent. To provide additional security against a system failure, the **msdb** database on the Distributor and Subscribers that use pull subscriptions must be backed up periodically, whenever a subscription is dropped or a new one added, or whenever a change is made to a replication agent.

The **msdb** database may also contain Data Transformation Services (DTS) package definitions used in replication if transformable subscriptions are used with any transactional or snapshot publications. To provide assurance against

system failure, the **msdb** database on the Distributor and on the Subscribers that use transformable subscriptions must be backed up periodically as well. These operations should be performed any time the DTS package associated with a publication is modified or each time a subscription is dropped or a new one is added. This ensures that the most up-to-date definitions can be recovered.Backing up and restoring the **msdb** database allows you to restore replication after a complete loss of the Distributor or Subscriber.

#### **Backing Up and Restoring the master Database**

When a new Subscriber is added, an entry for the Subscriber is added to the **sysservers** table in the **master** database on the Publisher. Back up the **master** database at the Publisher and after a Subscriber is added or after an entry for the Publisher is added to the **sysservers** table in the **master** database on the Distributor.

To restore replication after loss of the Publisher or Distributor, back up the **master** database on the Publisher and Distributor each time a new Subscriber or Publisher is added (respectively).

Replication

# **Transactional Replication and Log Shipping**

Microsoft® SQL Server<sup>™</sup> 2000 transactional replication can be configured to work with log shipping to provide a warm standby server recovery option if the Publisher fails.

You must be running Microsoft SQL Server 2000 Enterprise Edition to use log shipping. There are two modes for replication and log shipping working together: synchronous and semi-synchronous.

#### Synchronous Mode

In synchronous mode, the **sync with backup** option is set on the publication database. This causes the Log Reader Agent to synchronize with the publication database backup. In this mode, the Log Reader Agent does not propagate any transactions from the Publisher to the distribution database if they have not been backed up. This ensures that no Subscriber will get ahead of the Distributor; however, this also means that replication latency (the time it takes changes made at the Publisher to appear at the Subscriber), which can usually be as low as a few seconds, is now constrained to be greater than or equal to the log shipping interval. Typically, this is between two and ten minutes.

The advantage of using synchronous mode is that after failing over to the new Publisher, all replication servers are in synchronization.

#### To configure replication to work with log shipping in synchronous mode

1. On the publication database, execute sp\_replicationdboption '<publicationdatabasename>', 'sync with backup', 'true'.

When this option is set, the Log Reader Agent will not process the transaction until it is backed up through either database backup or log backup.

- 2. Set up log shipping for the publication database.
- 3. When the Publisher fails, restore the last log of the database using the KEEP\_REPLICATION option with RESTORE LOG. This will keep

all the replication settings.

4. Rename the warm standby server to the name of the original Publisher. Replication will continue to distribute data changes to Subscribers.

#### Semi-Synchronous Mode

If the increased latency that occurs in synchronous mode is unacceptable, and the possibility that the warm standby Publisher and the Subscribers are not synchronized is acceptable, use semi-synchronous mode.

The warm standby Publisher and the Subscribers may not be synchronized because the performance of the Log Reader Agent and the backups are not synchronized. This allows transactions that may not have been backed up on the Publisher and shipped to the warm standby to be propagated to the Distributor and then to Subscribers. Although the Publisher and the Subscribers are now out of synchronization, you can restart replication.

# To configure replication to work with log shipping in semi-synchronous mode

- 1. Set up log shipping for the publication database.
- 2. When the Publisher fails, restore the last log of the database using the KEEP\_REPLICATION option with RESTORE LOG. This will keep all replication settings.
- 3. Rename the warm standby server to the name of the original Publisher. You may receive an error message from the Log Reader Agent that the publication database and the distribution database are not synchronized.
- 4. Execute **sp\_replrestart**. This stored procedure can be used to force the Log Reader Agent to ignore all the previous replicated transactions in the publication database log. Transactions applied after the completion

of the stored procedure will be processed by the Log Reader Agent. You can restart the Log Reader Agent after the stored procedure executes successfully.

**IMPORTANT** The **sp\_replrestart** system stored procedure should be used only with log shipping. It can also be used under controlled circumstances if you need to restore the publication database are you are not using the **sync with backup** option. This option should be used only when the Log Reader fails to process replicated transactions in the publication database log and there are no other ways to resolve the problem.

5. Set the profile of the Distribution Agent to the Skip Error profile because lost transactions (some of which have already been replicated to the Subscribers) may be reapplied at the Publisher. Replication

# Strategies for Backing Up and Restoring Merge Replication

Microsoft® SQL Server<sup>™</sup> 2000 allows you to restore replicated databases without reinitializing subscriptions or disabling and reconfiguring Publishers or Subscribers. With merge replication, you can use the latest data stored at other sites to resynchronize a server with changes that may not have been preserved in a recent backup. You can also configure replication to work with log shipping, enabling you to use a warm standby server without reconfiguring replication.

Because merge replication stores change tracking meta data directly in your publication and subscription databases, there is no general requirement that you restore the publication database and distribution database to a consistent point in time. When you back up or restore a publication or subscription database, you also back up or restore the system meta data used to track replicated changes to a point in time consistent with your replicated data.

Merge replication ensures data convergence among all replicas in your topology. When it is necessary to restore a backup of a database, there are generally multiple options for recovery depending on the role of the database requiring a restore.

As part of any recovery strategy, always keep a current script of your replication settings in a safe location. In the event of server failure or the need to set up a test environment, you can modify the script by changing server name references, and it can be used to help recover your replication settings. In addition to scripting your current replication settings, you should script the enabling and disabling of replication.

#### **Backing Up and Restoring the Publication Database**

When restoring a publication database, you may want to reinitialize all subscriptions to any restored publications. You may also want to synchronize immediately with a Subscriber that has the latest data. Reinitializing all subscriptions provides a convenient mechanism to reset all replicas of the publication database to a state consistent with the restored publication database. Alternatively, you may want to synchronize your publication database immediately with a subscription database that has the latest data, and attempt to recover any changes synchronized with that replica but not included in the most recent publication database backup of publication database transaction log backup.

For example, suppose a publication database is backed up, changes are made in the publication database, a subscription database is synchronized with the publication database, and then the publication database is restored from backup. There are two choices for restoring the database:

- Synchronize the publication database with the subscription database and all changes made previously in the publication database, but not represented in the restored backup, will be uploaded from the subscription database to the publication database.
- Reinitialize all subscriptions to the publications in the publication database.

You may want to reinitialize all subscriptions if you are restoring a publication database to an earlier point in time as a mechanism to recover from an erroneously performed batch data operation, or if you are recovering your publication database to an earlier state. Reinitializing all subscriptions extends the recovery to an earlier state to all replicas within the enterprise. If you choose this option, it is recommended that you generate a new snapshot for delivery to reinitialized Subscribers immediately after restoring your publication database.

Performing replication configuration or maintenance activities in the publication database, synchronizing those changes with subscription databases, and then restoring the publication database to a state prior to the configuration changes may require a reinitialization of all subscriptions to effected publications in the restored publication database. Subscription databases are expected to have the same publication definition represented in the corresponding publication database whenever synchronization occurs.

It is recommended that you back up the publication database (either incremental or full backup) whenever changes are made to a replicated objects schema (for example, adding or dropping a column) or to a publication property even though you may have regularly scheduled database and log backups to be performed on a regular schedule. A description of some common actions affecting replication configuration or replicated object schema are described later in this topic.

# **Backing Up and Restoring the Distribution Database**

When restoring a publication database that contains only merge publications, it is not always necessary to restore the corresponding distribution database to a consistent point in time. The distribution database has a limited role in merge replication as the common store for synchronization history and error tracking information. It does not store any data used in change tracking and it does not provide temporary storage of merge replication changes to be forwarded to subscription databases. In most cases, it is not necessary to restore a distribution database when restoring a publication database backup for merge publications. The exception is when any database maintenance activity has been performed in the publication database or distribution database that affects replication configuration or replicated object schema.

# **Backing Up and Restoring a Subscription Database**

Similar to backing up a publication database, when a subscription database is restored, you are restoring replication change tracking data to a state consistent with the replicated data. Synchronizing the subscription database with each of its publications following a restore results in the Merge Agent downloading any changes that the subscription database backup has not yet received from the various publication databases for which it has Subscribers. A reinitialization of the subscription database is generally not required, and only the data changes since the backup was taken are synchronized between the publication database and the subscription database.

To restore a subscription database without any need to reinitialize its subscriptions, ensure that the restored database backup represents the subscription database in a state in which it has synchronized all subscriptions within the defined publication retention period. Restoring a database (and transaction logs) to a point in time prior to the retention period of the publication will require that the subscriptions in the subscription database be reinitialized. For more information about retention periods, see <u>Subscription Deactivation and Expiration</u>.

# **Backing Up and Restoring a Republishing Database**

When a database subscribes to data from a Publisher and in turn publishes that same data to other subscription databases, it is referred to as a republishing database. When restoring a republishing database, follow the guidelines described in the Backing Up and Restoring a Publication Database and Backing Up and Restoring a Subscription Database sections in this topic.

#### Backing Up and Restoring the msdb System Database

The **msdb** database at the Publisher contains the job definitions for replication agents that are run under the control of SQL Server Agent. The **msdb** database at the Distributor contains the job schedule, steps, alerts, and other job components for all Snapshot Agents, agents used with push subscriptions, and miscellaneous replication agents. The **msdb** database at each Subscriber contains similar job information for all pull subscription agents. The **msdb** database at the Distributor also contains the agent profile information for all replication agents.

To provide improved recovery options if you need to restore one or more replicated databases, the **msdb** database should be backed up periodically. In additionensure that an accurate backup of the **msdb** database is taken whenever any database maintenance activity has been performed in the publication database, distribution database, or subscription database that affects replication configuration (especially agent profiles or agent properties) .Backing up the **msdb** database is necessary in the event you want to restore replication after the loss a Distributor or Subscriber.

#### **Backing Up and Restoring the master Database**

It is not generally necessary to back up the **master** database on a regular basis; however, similar to backing up the **msdb** database, the **master** database is involved in storing limited configuration information regarding the replicated databases on any instance of SQL Server. For example, when a server is enabled as a Distributor, Publisher, or Subscriber, the **sysservers** table in the **master** database on the Distributor is updated. To restore replication after the loss of a **master** database on a Publisher or Distributor, back up the **master** database on the Publisher and Distributor. It is recommended that you back up the **master** database periodically and when any database maintenance activity has been performed in the publication database, distribution database, or subscription database that effects replication configuration (especially changes to enabled Publishers or Subscribers).

#### Merge Replication, Log Shipping, and Alternate Synchronization Partners

Microsoft® SQL Server<sup>™</sup> 2000 merge replication can be configured to work with log shipping to provide a warm standby server recovery option if the Publisher fails. Merge replication also allows Subscribers to synchronize with an alternate Publisher in the event the Publisher at which their subscriptions originated is unavailable.

Alternate synchronization partners can be used with any edition of SQL Server 2000 that supports merge replication; however, you must be running Microsoft SQL Server 2000 Enterprise Edition to use log shipping. Because merge replication tracks changes directly in the publication database, merge replication works with log shipping in a semi-synchronous mode only.

#### Semi-Synchronous Mode

In semi-synchronous mode, there is a possibility that the warm standby Publisher and its Subscribers may not be synchronized at the point of failover if any changes synchronized between the primary Publisher and its Subscribers have not yet been transferred using log shipping to the warm standby Publisher at the point of failover. When restoring a publication database from backup, you may want to reinitialize all subscriptions to publications following a failover, or you may elect to synchronize immediately with a Subscriber that has the latest data. Typically, you can synchronize immediately, and use log shipping to help provide continuous synchronization of updatable replicas if the primary Publication server fails.

#### **Alternate Synchronization Partners**

Similar to log shipping, using alternate synchronization partners during merge replication is an option that supports continuous synchronization in the event of a failure of the primary Publisher. Log shipping can be used to send all changes, including schema changes, user modifications, and database maintenance activities, to a warm standby Publisher. Specifying an alternate synchronization partner for publications defined at a Publisher provides a method to synchronize data changes to replicated tables with servers other than the Publisher at which a subscription originated. Synchronizing with alternate synchronization partners provides the ability for a Subscriber to synchronize data even if the primary Publisher is unavailable. For more information, see <u>Alternate Synchronization</u> <u>Partners</u>.

#### **Common Actions Requiring an Updated Backup**

In addition to regularly scheduled backups, it is recommended that you update backups of the publication, distribution, subscription, **msdb**, and **master** databases after making modifications to your replication schema or topology.

Backup the publication database after:

- Creating new publications.
- Altering any publication property including filtering.
- Adding articles to an existing publication.
- Performing a Publication-wide reinitialization of subscriptions.
- Altering any published table using a replication schema change.
- Performing on-demand script replication.
- Cleaning up merge meta data (running **sp\_mergecleanupmetadata**).
- Changing any article property including changing the selected article resolver.
- Dropping any publications.

- Dropping any articles.
- Disabling replication.

Backup the distribution database after:

- Creating or modifying replication agent profiles.
- Modifying replication agent profile parameters.
- Changing the replication agent properties (including schedules) for any push subscriptions.

Backup the subscription database after:

- Changing any subscription property.
- Changing the priority for a subscription at the Publisher.
- Dropping any subscriptions.
- Disabling replication.

Backup the **msdb** system database after:

- Enabling or disabling replication.
- Adding or dropping a distribution database (at the Distributor).
- Enabling or disabling a database for publishing (at the Publisher).
- Creating or modifying replication agent profiles (at the Distributor).

- Modifying any replication agent profile parameters (at the Distributor).
- Changing the replication agent properties (including schedules) for any push subscriptions (at the Distributor).
- Changing the replication agent properties (including schedules) for any pull subscriptions (at the Subscriber).

Backup the **master** system database after:

- Enabling or disabling replication.
- Adding or dropping a distribution database (at the Distributor).
- Enabling or disabling a database for publishing (at the Publisher).
- Adding the first or dropping the last publication in any database (at the Publisher).
- Adding the first or dropping the last subscription in any database (at the Subscriber).
- Enabling or disabling a Publisher at a Distribution Publisher (at the Publisher and Distributor).
- Enabling or disabling a Subscriber at a Distribution Publisher (at the Subscriber and Distributor).

Replication

# **Restoring Backups of Replicated Databases to the Same Server and Database**

When you create a backup of a database, Microsoft® SQL Server<sup>™</sup> 2000 makes a copy of all user tables and system tables (including **sysobjects**) in the current database. It also makes a complete copy of the log file(s) for the current database, including everything past the last log read transaction.

When you restore a database to the same server and database from which it was backed up, SQL Server 2000 does a full restore of the database and log. SQL Server then reads the **master.dbo.sysdatabases.category** column for the restored database to determine if any replication settings stored in the target database should be preserved.

#### **Publication Databases**

For both transactional and merge publication databases, replication is preserved if the **sysdatabases.category** column is set to indicate the database is enabled for publishing. For transactional and snapshot publishing databases, the category **bit** is set to **1**. For merge publishing databases, the category **bit** is set to **4**.

In most cases, restoring a backup to the same server and database from which it was created will preserve your replication settings. If the failure you are recovering from required you to completely re-create the database you are restoring into, run **sp\_replicationdboption** or enable the database for transactional and merge publishing before restoring your backup.

**CAUTION** If you do not run **sp\_replicationdboption**, your replication settings will be lost during the restore operation.

#### **Distribution Databases**

A single Distributor can store many distribution databases: up to one per Publisher served by the Distributor. It is important that when a publishing database is backed up, a coordinated backup of its associated distribution database is created. A coordinated restore of both databases is often required to preserve transactional integrity in your replication scenario. You may want to consider including a coordinated backup and restore of your replication working directory associated with the publishing database. This can reduce the amount of time required to resynchronize Subscribers in snapshot and transactional replication scenarios.

Similar to publishing databases, distribution databases cannot be restored to any location. Because of several database and server name dependencies among replication Publishers, Distributors, and Subscribers, you must restore to the same server and database you created the backup from to ensure proper resumption of replicated data flow. You should restore a distribution database only when you are restoring a publishing database, and always to the same server and database. After restoring the distribution database, review the replication agent profiles to confirm they are set as required by the application.

#### **Subscription Databases**

For transactional replication, subscription databases contain the table **MSreplication\_subscriptions**, which stores data indicating the last transaction received at the Subscriber. This table is included automatically when a subscribing database is backed up.

After a restore or attach of a transactional subscription database, you should run **sp\_vupgrade\_subscription\_tables** to ensure that all required objects are created and are the correct version. If you do not run

**sp\_vupgrade\_subscription\_tables**, objects necessary for replication may not exist in the subscription database.

Merge subscription databases are internally tracked as a type of publishing database as well. For this reason, the same considerations taken when planning for backup and restore of merge publishing databases should also be applied when working with merge subscribing databases.

#### See Also

MSreplication\_subscriptions sp\_replicationdboption sysdatabases <u>sysobjects</u>

Replication

# **Restoring Backups of Replicated Databases to a Different Server or Database**

When you restore a backup of a replicated database to a server or database other than the one on which it was created, your replication settings cannot be preserved. For publishing databases and merge subscribing databases, a full restore of the database and logs is followed by an automatic removal of replication meta data from the database when the database or server you restore to differs from the one on which the backup was created. If necessary, you can use this approach to recover your data to another server or database and then set up a new replication topology including the restored database.