Replication Programming
Getting Started with Replication Programming

Replication programming contains the following sections.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introducing Replication Programming</strong></td>
<td>Describes the benefits and planning required to program replication.</td>
</tr>
<tr>
<td><strong>Developing Replication Applications Using ActiveX Controls</strong></td>
<td>Describes the Microsoft® ActiveX® controls, with code samples for each control showing various pieces of functionality.</td>
</tr>
<tr>
<td><strong>Replication ActiveX Control Reference</strong></td>
<td>Describes the objects, collections, properties, methods, events, and constants available to the ActiveX controls.</td>
</tr>
<tr>
<td><strong>Developing Replication Merge Conflict Resolvers Through a Custom Resolver</strong></td>
<td>Provides an overview of the methods available for creating a merge conflict resolver</td>
</tr>
<tr>
<td><strong>Programming Replication from Heterogeneous Data Sources</strong></td>
<td>Provides an overview of the methods available for making a heterogeneous data source a Publisher.</td>
</tr>
<tr>
<td><strong>Replication Distributor Interface Reference</strong></td>
<td>Describes the objects, properties, methods, and interface structures available for the Replication Distributor Interface.</td>
</tr>
<tr>
<td><strong>Replication Programming Samples</strong></td>
<td>Describes the replication samples that are included with Microsoft SQL Server™ 2000.</td>
</tr>
</tbody>
</table>

For more information about replication applications that use SQL-DMO, see Developing SQL-DMO Applications and SQL-DMO Reference.
Replication Programming
Introducing Replication Programming

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, when used within custom applications that employ Microsoft Visual Basic® or Microsoft Visual C++®, provide programmable controls for administering and controlling the Snapshot Agent, the Distribution Agent, and the Merge Agent. These controls can be used to program activities needed to operate replication. Using ActiveX replication controls, you can build custom applications to configure and administer replication. Benefits include:
  - Installing a smaller portion of Microsoft SQL Server™ 2000. Because the client has no dependency on the full SQL Server 2000 installation, the user only needs to install Microsoft SQL Server 2000 Desktop Engine.
  - Controlling when replication activity occurs. 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 so that whenever users click the button, the merge ActiveX control connects to the Publisher and activates the Merge Agent for the specified publication, which then merges and synchronizes the data. For more information, see Programming Replication with ActiveX Controls.

- SQL Distributed Management Objects (SQL-DMO) allow you to create custom applications using Visual Basic or C++, which allows you to configure, implement, or maintain your replication topology. SQL-DMO can be used to program replication administration such as configuring distribution and creating subscriptions. The SQL-DMO objects can be used in programming languages such as Visual C++ or Visual Basic.
The Replication Distributor Interface allows you to replicate data from heterogeneous data sources such as Microsoft Access or Oracle. The Replication Distributor Interface is an OLE DB service provider that allows heterogeneous data sources to publish data to SQL Server Subscribers using snapshot replication or read-only transactional replication. It can be used to develop a custom replication application based on proprietary data sources. For more information, see Programming Replication from Heterogeneous Data Sources.

Transact-SQL system stored procedures allow you 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). In most cases, you are better served by using the programming interfaces SQL-DMO and ActiveX replication controls for programming replication rather than writing direct calls to the system stored procedures.
Replication Programming
Benefits of Programming Replication

Programming replication allows you to create custom applications with which you can configure and maintain a replication topology. Some benefits of using custom applications include:

- Making it easier for mobile or occasionally connected users to modify data offline and propagate those changes to other locations when they reconnect to the network. The users can enter their changes to the data, and then when connected to the network, use an application to upload and download data changes. The users do not need to have knowledge of Microsoft® SQL Server™ or replication to perform these actions.

- Allowing you to program heterogeneous data sources, such as Publishers of data, to SQL Server Subscribers. The Replication Distributor Interface can be programmed to support heterogeneous Publishers for snapshot replication and transactional replication.

- Allowing you to save the initial snapshot on removable media (such as a CD-ROM) and apply the snapshot at the Subscriber from the media, rather than applying the initial snapshot over a slow link.

- Simplifying replication by separating functionality into smaller, reusable pieces that manage and administer the replication process easily from a central location.

- Allowing heterogeneous applications to inherit replication features and act as Publishers so SQL Server Subscribers can gain access to data that is stored on a variety of data sources, including legacy data sources and proprietary data sources.

- Storing replication SQL statements, scripts, and .bcp files (in addition to transactions), and forwarding them to the Subscriber.
Writing custom applications to resolve merge conflicts that can occur when the same data is modified at multiple places allow the developer to implement 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®. By using merge replication custom conflict resolvers, you can resolve unique business conflicts by writing scripts that can handle any logic required to resolve complex conflict scenarios. For more information, see Merge Replication Conflict Detection and Resolution.
Replication Programming
Planning for Replication Programming

When planning to program an application used in replication, decide what replication topology you will use, which replication actions need to be performed programmatically, and which actions will be performed using other Microsoft® SQL Server™ 2000 tools.

For the actions that are being controlled programmatically, determine what functionality in your business application will be performed only one time (such as creating the databases, configuring a Publisher, or creating a publication), and what functionality will be performed repeatedly (such as creating subscriptions, synchronizing data between the Publisher and Subscribers, and validating replicated data).

Example

See Also

Planning for Application Development
Replication Programming
Developing Replication Applications Using ActiveX Controls

Microsoft® SQL Server™ 2000 provides Microsoft ActiveX® controls that allow custom applications to embed replication functionality. These controls support synchronization and limited administration of push, pull, and anonymous subscriptions. In addition, these controls can be programmed to add, copy, and delete both pull and anonymous subscriptions; create or attach subscription databases; and create new subscriptions to be synchronized. The activity of these controls can be monitored using Replication Monitor in SQL Server Enterprise Manager.

Software developers can use ActiveX replication controls like any standard built-in control. They have been implemented as in-process components and do not have visible user interfaces. The ActiveX replication controls provided are:

- SQL Snapshot control
- SQL Distribution control
- SQL Merge control

These controls are comparable to the replication agents of the same name. The SQL Snapshot control has functions similar to the Snapshot Agent, the SQL Distribution control has functions similar to the Distribution Agent, and the SQL Merge control has functions similar to the Merge Agent.

The Microsoft SQL Server CD-ROM ships with some ActiveX replication control samples. For more information, see Replication ActiveX Control Samples.

See Also

Replication ActiveX Control Reference
Replication Programming
Requirements for Using Replication ActiveX Controls in Development Environments

This section explains how to use the SQL Distribution control, SQL Merge control, and SQL Snapshot control in Microsoft® Visual Basic® and Microsoft Visual C++® development environments.

Microsoft Visual Basic

To use one or more Microsoft ActiveX® replication controls in a Microsoft Visual Basic program: on the Project/References menu in the Visual Basic Development Environment, in the References dialog box, select the .dll references for the controls you plan to use.

<table>
<thead>
<tr>
<th>Component</th>
<th>Reference</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Distribution</td>
<td>Microsoft SQL Distribution Control 8.0</td>
<td>sqldistx.dll</td>
</tr>
<tr>
<td>Control</td>
<td>SQL Merge Control</td>
<td>sqlmergx.dll</td>
</tr>
<tr>
<td>SQL Merge Control</td>
<td>Microsoft SQL Merge Control 8.0</td>
<td></td>
</tr>
<tr>
<td>SQL Snapshot Control</td>
<td>Microsoft SQL Snapshot Control 8.0</td>
<td>sqlinitx.dll</td>
</tr>
<tr>
<td>Replication Errors</td>
<td>Microsoft SQL Replication Errors 8.0</td>
<td>replerrx.dll</td>
</tr>
</tbody>
</table>

If Microsoft SQL Server™ 2000 was installed to the default locations, these component .dlls are located in C:\Program Files\Microsoft SQL Server\80\COM\.

Microsoft Visual C++

To use the one or more replication controls in a Microsoft Visual C++ program, include the files from the Include Files column of the following table in the appropriate source files of your project. These files are installed on your computer only if Headers and Libraries was selected in Development Tools during a custom installation of the SQL Server 2000 client tools.

Reference the files from the Libraries column of the table in your project or NMAKE file. These files are installed automatically during a typical installation.
of the SQL Server 2000 client tools.

<table>
<thead>
<tr>
<th>Component</th>
<th>Include Files</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Distribution Control</td>
<td>sqldistx.h</td>
<td>sqldistx.dll</td>
</tr>
<tr>
<td></td>
<td>repldstx.c</td>
<td></td>
</tr>
<tr>
<td>SQL Merge Control</td>
<td>sqlmergx.h</td>
<td>sqlmergx.dll</td>
</tr>
<tr>
<td></td>
<td>replmrgx.c</td>
<td></td>
</tr>
<tr>
<td>SQL Snapshot Control</td>
<td>sqlinitx.h</td>
<td>sqlinitx.dll</td>
</tr>
<tr>
<td></td>
<td>replinix.c</td>
<td></td>
</tr>
<tr>
<td>Replication Errors</td>
<td>replerrx.h</td>
<td>replerrx.dll</td>
</tr>
</tbody>
</table>

If SQL Server 2000 was installed to the default locations, the files in the **Include Files** column are located in C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Include\.

The files in the **Libraries** column are located in C:\Program Files\Microsoft SQL Server\80\COM\.

After the controls are referenced, they can be included in the Components toolbar; however, these controls are not user interface controls. Drawing the controls on the form will not instantiate them. The controls need to be instantiated with the **NEW** keyword. In addition, a variable of the ActiveX object's class can be declared using the ** WithEvents** keyword. This enables the program to receive callbacks from the agents, and the application can cancel the ActiveX object in this callback function. The callback is also available with the **ReplErr** object, although it is a notify callback and not a status callback.

**See Also**

[Replication ActiveX Control Reference](#)
Replication Programming
Requirements for Deploying Replication ActiveX Controls

When deploying the Microsoft® ActiveX® replication controls independently of Microsoft SQL Server™ 2000, you must include additional files in the installation kit that you use to distribute your application. If you deploy your application using these ActiveX replication controls on a computer where SQL Server 2000 Desktop Engine or SQL Server 2000 client software is also installed, these files will already be present.

If you use Microsoft Visual Studio® and the Deployment Wizard to prepare your installation kit, the wizard will recognize that the files from the Referenced Libraries column in the following table are needed, and will include them in the installation kit. The wizard will also indicate that dependency information for these files is not available. Include the files from the Dependent Libraries column of the table in your installation kit (click the Add button on the Included Files page of the wizard). The paths shown in this table assume SQL Server 2000 was installed to the default locations.

<table>
<thead>
<tr>
<th>Replication Control</th>
<th>Referenced Libraries</th>
<th>Dependent Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Distribution Control</td>
<td>Sqldistx.dll</td>
<td>• C:\Program Files\Microsoft SQL Server\80\COM\Rdistcom.dll</td>
</tr>
<tr>
<td></td>
<td>Replerrx.dll</td>
<td>• C:\Program Files\Microsoft SQL Server\MSSQL\Binn\Sqlrepss.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• $(WinSysPath)\Sqlwoa.dll</td>
</tr>
<tr>
<td>SQL Merge Control</td>
<td>Sqlmergx.dll</td>
<td>• C:\Program Files\Microsoft SQL Server\80\COM\Replprov.dll</td>
</tr>
<tr>
<td></td>
<td>Replerrx.dll</td>
<td>• C:\Program Files\Microsoft SQL Server\80\COM\Replprov.dll</td>
</tr>
<tr>
<td>SQL Snapshot Control</td>
<td>Sqliinitx.dll</td>
<td>Replerrx.dll</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C:\Program Files\Microsoft SQL Server\80\COM\Rinitcom.dll</td>
</tr>
</tbody>
</table>

- Server\80\COM\Replrec.dll
- $(WinSysPath)\Sqlwoa.dll
- $(WinSysPath)\Sqlwoa.dll

**See Also**

[Replication ActiveX Control Reference](#)
Replication Programming
Programming the SQL Snapshot ActiveX Control

The SQL Snapshot control is implemented as a Microsoft® ActiveX® in-process component. It provides a way to create snapshots, and is used by all types of replication. The control is comparable to the Snapshot Agent. Its primary class, the SQLSnapshot object, creates a snapshot of the specified publication on the specified Distributor.

Dynamic snapshots are supported. There are properties to provide values for the HOST_NAME() and SUSER_SNAME() functions when they appear in the filter criteria of dynamic publications, and to specify the location where the dynamic snapshots are written.

Instantiating the SQL Snapshot Control

This example is a Microsoft® Visual Basic® Sub procedure that creates a snapshot from a publication named FullSnapPublication using the database UE_PublisherDB on Publisher UE_PUBLISHER and saves it on Distributor UE_DISTRIBUTOR. Windows Authentication is used for both the Publisher and Distributor connections. The example shows the snapshot code for a snapshot publication. The code for a merge publication requires the ReplicationType property to be set.

Sub Main()
    Dim oSnapCtl As SQLINITXLib.SQLSnapshot

    Set oSnapCtl = New SQLINITXLib.SQLSnapshot
    oSnapCtl.Publisher = "UE_PUBLISHER"
    oSnapCtl.PublisherDatabase = "UE_PublisherDB"
    oSnapCtl.PublisherSecurityMode = NT_AUTHENTICATION
    oSnapCtl.Publication = "FullSnapPublication"
    oSnapCtl.Distributor = "UE_DISTRIBUTOR"
    oSnapCtl.DistributorSecurityMode = NT_AUTHENTICATION
    oSnapCtl.Initialize
    oSnapCtl.Run
oSnapCtl.Terminate
End Sub

**Note** Include the call to the **Terminate** method to close connections and release allocated memory.

**See Also**

[SQLSnapshot Object](#)
Replication Programming
Programming the SQL Distribution ActiveX Control

The SQL Distribution control is implemented as a Microsoft® ActiveX® in-process component. It provides a way to move schema and data to Subscribers for snapshot replication and to control the synchronization of subscriptions for transactional replication. The control is comparable to the Distribution Agent. Its primary class, the SQLDistribution object, includes the ability to:

- Add, drop, copy, and register subscriptions for synchronization using Windows Synchronization Manager. For more information, see Common SQL Distribution Control and SQL Merge Control Functionality.

- Specify an alternate snapshot folder from which the initial snapshot for a subscription can be applied.

- Specify a working directory to which snapshot files are copied when File Transfer Protocol (FTP) is used to retrieve the initial snapshot.

- Generate a specially formatted subscription file (usually an .msf) in a specified location. These files can be attached to create a synchronized subscription that is registered at the Publisher as part of an attachable subscription database.

- Specify a Data Transformation Services (DTS) package that transforms command rowsets before applying them to a Subscriber.

- Set the Subscriber options to immediate updating or queued updating for changes made at the Subscriber.

- Synchronize push, pull, or anonymous subscriptions in a transactional publication.
For more information, see SQLDistribution Object.

**Instantiating the SQL Distribution Control**

This code segment demonstrates how a Microsoft Visual Basic® program configures the SQLDistribution object to synchronize data with an anonymous SQL Server Subscriber.

In the example, Distributor properties are not set, so the Publisher is also the Distributor. The SubscriberDatabase must already exist when this code is executed. Replace the fields in italic with appropriate values.

'SQLDistribution control declaration.
Private mobjDistr As SQLDISTXLib.SQLDistribution
...

Set mobjDistr = New SQLDISTXLib.SQLDistribution

With mobjDistr
  'Set up the Publisher
  .Publisher = "PublisherServer"
  .PublisherDatabase = "PublisherDatabase"
  .Publication = "PublicationName"
  .PublisherSecurityMode = NT_AUTHENTICATION

  'Set up the Subscriber.
  .Subscriber = "SubscriberServer"
  .SubscriberDatabase = "SubscriberDatabase"
  .SubscriberDatasourceType = SQL_SERVER
  .SubscriberSecurityMode = NT_AUTHENTICATION

  'Set up the Subscription.
  .SubscriptionType = ANONYMOUS
  .SynchronizationType = AUTOMATIC
'Synchronize the data.
.Initialize
.Run
.Terminate
End With

Note When adding an anonymous subscription, you do not have to call the AddSubscription method of the SQLDistribution object. If the anonymous subscription does not exist, it will be added automatically when the first synchronization is performed, as in the previous sample.

When using a merge publication, the same example can be used for programming anonymous subscriptions with the SQLMerge object. Replace references to the SQLDistribution object with references to the SQLMerge object.

Creating Anonymous Internet Subscriptions Sample

ActiveX replication controls can be programmed to synchronize data over the Internet. After the Publisher and Distributor are configured for publishing over the Internet and a publication enabled for anonymous subscriptions is created, an application using an ActiveX replication control can synchronize with the publication data. FTP is used for the snapshot download, both during the initial application and when the subscription is re-initialized. All other synchronizations use SQL packets over TCP/IP to transfer data between Publisher and Subscriber.

Examples

Distribution Control Using IP Address

This code segment demonstrates how a Visual Basic program configures the SQL Distribution control to create a Subscriber database and synchronize data with an anonymous SQL Server Subscriber over the Internet. The Distributor and Publisher are reached using an IP address.

In this example, no distributor properties are set, so the Publisher is also used as the Distributor. Replace the fields in italic with appropriate values. The port
numbers shown are the numbers that are typically used, but they may also need to be changed.'SQLDistribution control declaration.

Private mobjDistr As SQLDISTXLib.SQLDistribution

Set mobjDistr = New SQLDISTXLib.SQLDistribution

With mobjDistr
  'Set up the Publisher.
  .Publisher = "PublisherServer"
  .PublisherDatabase = "PublisherDatabase"
  .Publication = "PublicationName"
  .PublisherSecurityMode = DB_AUTHENTICATION
  .PublisherLogin = "PublisherUserID"
  .PublisherPassword = "PublisherPassword"
  .PublisherAddress = "157.56.17.27,1433"
  .PublisherNetwork = TCPIP_SOCKETS

  'Set up the Subscriber.
  .Subscriber = "SubscriberServer"
  .SubscriberDatabase = "SubscriberDatabase"
  .SubscriberDatasourceType = SQL_SERVER
  .SubscriberSecurityMode = NT_AUTHENTICATION

  'Set up the Subscription.
  .SubscriptionType = ANONYMOUS
  .SynchronizationType = AUTOMATIC

  'Synchronize subscription.
  .Initialize
  .Run
  .Terminate
End With
When using a merge publication, the same example can be used for programming anonymous subscriptions with the SQL Merge control. Replace references to the SQLDistribution object with references to the SQLMerge object.

**Creating Pull Subscriptions to an ODBC Data Source Sample**

The SQL Distribution control can be programmed to synchronize Publishers with ODBC data sources. This feature is one method by which applications can synchronize subscriptions to heterogeneous data sources.

Before using the SQL Distribution control, an ODBC data source name (DSN) must be created at the client computer on which the application hosting the SQL Distribution control is running. If a pull subscription is to be created, the ODBC DSN must also be created at the Distributor if it is a different computer from the client. The DSN at the Distributor must be configured as a Subscriber (for example, by using the `sp_addsubscriber` stored procedure).

This code segment demonstrates how a Visual Basic program configures the SQL Distribution control to create a pull subscription and synchronize data with an ODBC Subscriber. If an anonymous subscription is used, it is not necessary to call AddSubscription or to create the DSN at the Distributor. Replace the fields in italic with appropriate values.

' SQLDistribution object declaration.
Private mobjDistr As SQLDISTXLib.SQLDistribution

'Configure the control for an ODBC subscription.
Set mobjDistr = New SQLDISTXLib.SQLDistribution

With mobjDistr
  'Set up the Publisher.
  .Publisher = "PublisherServer"
  .PublisherDatabase = "PublisherDatabase"
  .Publication = "PublicationName"
  .PublisherSecurityMode = NT_AUTHENTICATION
'Set up the Distributor.
.Distributor = "DistributorServer"
.DistributorSecurityMode = NT_AUTHENTICATION

'Set up the Subscriber.
.Subscriber = "The_ODBC_DSN"
.SubscriberDatasourceType = ODBC_DSN
.SubscriberSecurityMode = DB_AUTHENTICATION

'Subscriber login information needed only if the
' ODBC DSN does not contain login information.
.SubscriberLogin = "SubscriberUserID"
.SubscriberPassword = "SubscriberPassword"

'Set up the subscription.
.SubscriptionName = "SubscriptionName"
.SubscriptionType = PULL
.SynchronizationType = AUTOMATIC

'Create and synchronize the subscription.
.AddSubscription EXISTING_DATABASE, NONE
.Initialize
.Run
.Terminate
End With

Creating Pull Transactional Subscriptions to an OLE DB Data Source Sample

You can program the SQL Distribution control to synchronize data with OLE DB data sources. This feature is one method by which applications can synchronize subscriptions to heterogeneous data sources.
Before running the control, a linked server to the OLE DB database must be created at the Distributor. This can be done in SQL Server Enterprise Manager or by using the `sp_addlinkedserver` stored procedure. A heterogeneous Subscriber using the linked server must be created at the Publisher (for example, by using the `sp_addsubscriber` stored procedure).

SQL Server 2000 does not support anonymous subscriptions using OLEDB_DATASOURCE with the `SubscriberDatasourceType` property. You can create OLE DB anonymous subscriptions to instances of SQL Server by setting `SubscriberDatasourceType` to SQL_SERVER.

This code segment demonstrates how a Visual Basic program can configure the SQL Distribution control to create a pull subscription and synchronize data with an OLE DB Subscriber. Replace the fields in italic with appropriate values.

' Declare the SQLDistribution object.
Private mobjDistr As SQLDISTXLib.SQLDistribution

'Configure the control for an OLE DB subscription.
Set mobjDistr = New SQLDISTXLib.SQLDistribution

With mobjDistr
  'Set up the Publisher.
  .Publisher = "PublisherServer"
  .PublisherDatabase = "PublisherDatabase"
  .Publication = "PublicationName"
  .PublisherSecurityMode = NT_AUTHENTICATION

  'Set up the Subscriber.
  .Subscriber = "LinkedServerName"
  .SubscriberDatasourceType = OLEDB_DATASOURCE
  .SubscriberSecurityMode = DB_AUTHENTICATION

  'Subscriber login information is needed only if the OLE DB ' linked server definition does not include login information.
.SubscriberLogin = "SubscriberUserID"
.SubscriberPassword = "SubscriberPassword"

'Set up the subscription.
.SubscriptionName = "SubscriptionName"
.SubscriptionType = PULL
.SynchronizationType = AUTOMATIC

'Create and synchronize the subscription.
.AddSubscription EXISTING_DATABASE, NONE
.Initialize
.Run
.Terminate
End With

See Also

Replication and Heterogeneous Data Sources
sp_addlinkedserver
sp_addsubscriber
SubscriberDatasourceType Property
Replication Programming
Programming the SQL Merge ActiveX Control

The SQL Merge control is implemented as a Microsoft® ActiveX® in-process component. It provides a way to synchronize data in merge subscriptions. It is not used in snapshot replication or transactional replication. Its primary class, the SQLMerge object, provides the functionality of the Merge Agent and supports synchronization of push, pull, or anonymous subscriptions to a merge publication. It also includes the options to:

- Add, drop, copy, and register subscriptions for synchronization using Windows Synchronization Manager. For more information, see Common SQL Distribution Control and SQL Merge Control Functionality.

- Apply the initial snapshots to the Subscriber.

- Merge incremental changes that occurred at the Publisher or Subscribers after the initial snapshot was created.

- Reconcile conflicts according to the rules configured, through a COM component custom conflict resolver, or interactively by setting the UseInteractiveResolver property.

- Specify the direction of the synchronization so that it executes only the upload phase, the download phase, or both the upload and download phases.

- Specify an alternate snapshot folder from which the snapshot for a subscription can be applied.

- Specify a client-side working folder to which snapshot files can be copied using FTP.
• Copy a subscription database by generating a specially formatted subscription file in a specified location. These files can be attached to create a synchronized subscription, registered at the Publisher as part of an attachable subscription database.

• Synchronize with a Publisher other than the one at which its subscription originated. This alternate synchronization partner must contain the same schema and data set as the original Publisher.

• Attach a subscription database by specifying Subscriber-side properties only. The Publisher, PublisherDatabase, Distributor, and Publication properties do not need to be set while adding attachable subscription databases.

**Instantiating the SQL Merge Control**

The SQL Distribution and SQL Merge controls can be used to create a subscription database and a pull subscription, as well as synchronize with the publication data.

This code segment demonstrates how a Microsoft Visual Basic® program can configure the SQLMerge object to create a Subscriber database and subscription using the AddSubscription method, and then synchronizes data with the SQL Server Publisher.

In the example, Microsoft SQL Server™ 2000 generates the subscription name. Replace the fields in italic with appropriate values.

'SQLMerge control declaration.
Private mobjMerge As SQLMERGXLib.SQLMerge

Set mobjMerge = New SQLMERGXLib.SQLMerge

With mobjMerge
   'Set up the Publisher.
   .Publisher = "PublisherServer"
.PublisherDatabase = "PublisherDatabase"
/Publication = "PublicationName"
/PublisherSecurityMode = NT_AUTHENTICATION

'Set up the Distributor.
/Distributor = "DistributorServer"
/DistributorSecurityMode = NT_AUTHENTICATION

'Set up the Subscriber.
/Subscriber = "SubscriberServer"
/SubscriberDatabase = "SubscriberDatabase"
/SubscriberDatasourceType = SQL_SERVER
/SubscriberSecurityMode = DB_AUTHENTICATION
/SubscriberLogin = "SubscriberUserID"
/SubscriberPassword = "SubscriberPassword"

'Set up the subscription.
/SubscriptionType = PULL
/SynchronizationType = AUTOMATIC

'Set up the database and subscription.
/AddSubscription CREATE_DATABASE, NONE

'Synchronize the subscription.
/Initialize
/Run
/Terminate

End With

Note  Subscriptions other than anonymous subscriptions that are added using the SQLDistribution or SQLMerge objects must be added by explicitly calling the AddSubscription method before attempting to initialize and synchronize the subscription for the first time.
Both **SQLDistribution** and **SQLMerge** objects can be used to create a subscription database and a pull subscription, as well as synchronize with the publication data. When using a transactional publication, the previous sample code can be used for creating a Subscriber database and pull subscriptions with the **SQLDistribution** control. Replace references to the **SQLMerge** object with references to the **SQLDistribution** object. For more information about the merge object, see [SQLMerge Object](#).

**Creating Anonymous Internet Subscriptions Sample**

ActiveX replication controls can be programmed to synchronize data over the Internet. After the Publisher and Distributor are configured for publishing over the Internet and a publication enabled for anonymous subscriptions is created, an application using an ActiveX replication control can synchronize with the publication data. FTP is used for the snapshot download, both during the initial application and when the subscription is re-initialized. All other synchronizations use SQL packets over TCP/IP to transfer data between Publisher and Subscriber.

This example demonstrates how a Visual Basic program configures the SQL Merge Control to synchronize data for an anonymous subscription over the Internet. The Distributor and Publisher are reached using a Uniform Resource Locator (URL).

The **SubscriberDatabase** must already exist when this code is executed. Because the FTP information has not been provided, the specification of FILETRANSFERFTP for the **FileTransferType** property causes the control to request the FTP information from the Distributor.

A handler for the **Status** event is included. It displays the most recent status message in a label. Replace the fields in italic with appropriate values.

'SQLMerge control declaration.
Private WithEvents mobjMerge As SQLMERGXLib.SQLMerge

Private Sub RunReplMerge()
'Create SQLMerge control.
Set mobjMerge = New SQLMERGXLib.SQLMerge

With mobjMerge
    'Set up the Publisher.
    .Publisher = "PublisherServer"
    .PublisherAddress = "publisher.company.com"
    .PublisherNetwork = TCPIP_SOCKETS
    .PublisherDatabase = "PublisherDatabase"
    .Publication = "PublicationName"
    .PublisherSecurityMode = DB_AUTHENTICATION
    .PublisherLogin = "PublisherUserID"
    .PublisherPassword = "PublisherPassword"

    'Set up FTP.
    .FileTransferType = FILETRANSFERFTP

    'Set up the Distributor.
    .Distributor = "DistributorServer"
    .DistributorAddress = "distributor.company.com"
    .DistributorNetwork = TCPIP_SOCKETS
    .DistributorSecurityMode = DB_AUTHENTICATION
    .DistributorLogin = "DistributorUserID"
    .DistributorPassword = "DistributorPassword"

    'Set up the Subscriber.
    .Subscriber = "SubscriberServer"
    .SubscriberDatabase = "SubscriberDatabase"
    .SubscriberDatasourceType = SQL_SERVER
    .SubscriberSecurityMode = NT_AUTHENTICATION

    'Set up the subscription.
    .SubscriptionType = ANONYMOUS
.SynchronizationType = AUTOMATIC

'Synchronize the Subscriber.
.Initialize
.Run
.Terminate
End With
Exit Sub
End Sub

Private Function mobjMerge_Status(ByVal Message As String, ByVal Percent As Long) As SQLMERGXLib.STATUS_RETURN_CODE
'Display most recent status message.
    Label1 = Message
    DoEvents
End Function

Note  The URLs publisher.company.com and distributor.company.com need to be resolvable by an external DNS server. If a listening port other than the default port 1433 is used, it must be explicitly coded. For example, if port 1430 is to be used:

    .PublisherAddress = "publisher.company.com,1430"

When using a transactional publication, the same example can be used for synchronizing a Subscriber database with the SQL Distribution control. Replace references to the SQLMerge object with references to the SQLDistribution object.

**Creating and Synchronization Subscriptions to a Jet 4.0 Database**

ActiveX replication controls can be programmed to synchronize data with a Microsoft Jet 4.0 database. This enables applications to synchronize subscriptions to Jet 4.0 databases without having to create an OLE DB data source. The publication must be configured to accept heterogeneous data source
Subscribers. The Subscriber does not need to be configured as a linked server.

This code segment demonstrates how a Visual Basic program configures the SQL Merge control to create a Jet 4.0 database and synchronizes data with it. The **AddSubscription** method need not be used if the database already exists.

**Note** If the publication has not been enabled for heterogeneous Subscribers, the Jet database will be created and the Publisher schema applied, but no data will be copied. No error message or warning is given.

'SQLMerge control declaration.
Private WithEvents mobjMerge As SQLMERGXLib.SQLMerge

'Set the control for a Jet 4.0 database subscription.
Set mobjMerge = New SQLMERGXLib.SQLMerge

With mobjMerge

'Set up the Publisher.
.Publisher = "PublisherServer"
.PublisherDatabase = "PublisherDatabase"
.Publication = "PublicationName"
.PublisherSecurityMode = NT_AUTHENTICATION

'Set up the Subscriber.
.Subscriber = "SubscriberServer"
' **MDBFileSpecification** would be something like C:\ReplDBs\JetPub
.SubscriberDatabasePath = "**MDBFileSpecification**"
.SubscriberDatasourceType = JET4_DATABASE
.SubscriberSecurityMode = DB_AUTHENTICATION
' **JetDatabaseUserID** would be something like "Admin"
.SubscriberLogin = "**JetDatabaseUserID**"
.SubscriberPassword = "**JetDatabasePassword**"
'Set up the subscription.
SubscriptionType = ANONYMOUS
SynchronizationType = AUTOMATIC

'Synchronize the subscription.
.Initialize
.Run
.Terminate
End With

When using a transactional publication, the same example can be used for synchronizing a Subscriber database with the SQL Distribution control. Replace references to the SQLMerge object with references to the SQLDistribution object.

Providing Status and Handling Cancel Requests

ActiveX replication controls provide a Status event that provides status messages and percent complete during Initialize, Run, and other replication control methods. These messages can be displayed in the user interface of the application (for example, a label and a progress bar). The event also supports the ability to cancel the control process.

Assume the application hosts a SQL Merge control, and its visible user interface includes these controls.

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Control Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>lblStatus</td>
</tr>
<tr>
<td>Progress Bar</td>
<td>prgStatus</td>
</tr>
<tr>
<td>Command Button</td>
<td>cmdCancel</td>
</tr>
</tbody>
</table>

The Status event handler updates lblStatus and prgStatus with the status information. When cmdCancel is clicked, a Boolean variable is set. The handler returns a CANCEL notification when it finds the variable set. The DoEvents call
should be included to update the visible controls.

If you use a mechanism similar to this, be sure to inhibit subsequent calls to control methods after the cancel request is received.

This code segment demonstrates how a Visual Basic program displays status information and handles a cancel request. Note that the Status callback might be called again after being cancelled. Some operations cannot be cancelled immediately.

Private WithEvents mobjMerge As SQLMERGXLib.SQLMerge
Private mblnCancel As Boolean

Private Sub cmdCancel_Click()
' Set flag when the Cancel button is clicked.
  mblnCancel = True
End Sub

Private Function mobjMerge_Status(ByVal Message As String, _
    ByVal Percent As Long) As SQLMERGXLib.STATUS_RETURN_CODE
' Display progress and status message.
  lblStatus.Caption = Message
  prgStatus.Value = Percent

  ' Cancel if the button was clicked.
  If mblnCancel Then mobjMerge_Status = CANCEL

  ' Allow screen to update.
  DoEvents
End Function

**Using Error Handling Sample**

ActiveX replication controls provide detailed information about method failures through these mechanisms:
• Returning an error code to the caller of the method. In Microsoft Visual Basic, this is done by raising a runtime error that you can trap in an error handler. In Microsoft Visual C++, the error code is an HRESULT function return value.

• Referencing a SQLReplErrors collection. Each control has an ErrorRecords property that references a SQLReplErrors collection. To retrieve error information, you can enumerate through each SQLReplError object in the collection.

Usually, any error that is returned to the caller is added to the collection. Collection elements that have an error code equal to 0 are supplemental information such as a stored procedure reference with actual arguments substituted, which applies to the previous collection element. The collection is cleared at each new call to a replication control method to ensure that all elements apply to the most recent call.

This example demonstrates how a Visual Basic program reports detailed error information that might be generated by an ActiveX replication control. Most of the code to set up the control is omitted. The error handler displays the error code and description for the raised error and for each error in the errors collection. When the raised error description is the same as the description of one of the collection elements, it is not duplicated in the display. Both the raised error codes and the duplicate collection element error code are displayed.

Private WithEvents mobjMerge As SQLMERGXLib.SQLMerge
Private mobjReplErr As REPLERRXLib.SQLReplError

Private Sub RunReplMerge()
    Dim strPhase As String 'setup/initialize/run/terminate

On Error GoTo ErrorHandler

    Set mobjMerge = New SQLMERGXLib.SQLMerge
With mobjMerge
' Set up the SQL Merge control.
strPhase = "Setup"
.Publisher = "PublisherServer"
.PublisherDatabase = "PublisherDatabase"
.PublisherSecurityMode = NT_AUTHENTICATION
.SubscriberSecurityMode = NT_AUTHENTICATION
'<Remainder of properties set here.>

'Synchronize the subscription.
strPhase = "Initialize"
.Initialize
strPhase = "Run"
.Run
strPhase = "Terminate"
.Terminate
End With
Exit Sub

ErrorHandler:
    Dim blnMsgDupl As Boolean 'True: duplicate found in collection
    Dim strErrMsg As String 'Message buffer

    'Iterate through errors collection.
    For Each mobjReplErr In mobjMerge.ErrorRecords

        'The raised error is the same as the collection element; add error code.
        If Err.Description = mobjReplErr.Description Then
            strErrMsg = strErrMsg & vbCrLf & vbCrLf & _
            mobjReplErr.Description & vbCrLf & _
            "Error " & strErrorNumConv(Err.Number)
            blnMsgDupl = True
        End If
    Next mobjReplErr

'The raised error is not the same as the collection element; add 'description.
Else
    strErrMsg = strErrMsg & vbCrLf & vbCrLf & _
    mobjReplErr.Description
End If

'Append error code from collection.
strErrMsg = strErrMsg & vbCrLf & "Coll. " & _
            strErrorNumConv(mobjReplErr>ErrorNumber)
Next mobjReplErr

'Format message, include raised error if duplicate not found.
strErrMsg = "Error during Merge control " & _
    strPhase & " phase:" & _
    If(blnMsgDupl, "", _
        vbCrLf & vbCrLf & Err.Description & vbCrLf & _
        "Error " & strErrorNumConv(Err.Number)) & _
    strErrMsg
    MsgBox strErrMsg, vbExclamation
End Sub

Private Function strErrorNumConv(ByVal lngErrNum As Long) As String
    'Convert error number into readable forms, 
    'hex, and decimal for the low-order word.
    Dim strErrNums As String

    If lngErrNum < 16 And lngErrNum > -16 Then
        strErrNums = CStr(lngErrNum)
    ElseIf lngErrNum < 65536 And lngErrNum > -65536 Then
        strErrNums = "x" & Hex(lngErrNum) & _
                      " = " & CStr(lngErrNum)
    Else

strErrNums = "x" & Hex(lngErrNum) & 
    
    " = x" & Hex(lngErrNum And -65536) & 
    
    " + " & CStr(lngErrNum And 65535)
End If

strErrorNumConv = "Code: " & strErrNums
End Function

If this example is run as shown, with part of the control setup missing, the following error is displayed.

Error during Merge control Initialize phase:

'The property 'Publication' must be set before initializing the SQL Merge control.
Error Code: x80004005 = x80000000 + 16389
Coll. Code: x7918 = 31000

'The property 'Subscriber' must be set before initializing the SQL Merge control.
Coll. Code: x7918 = 31000

See Also

ErrorRecords Property

SQLReplError Object

SQLReplErrors Collection
Replication Programming
Common SQL Distribution Control and SQL Merge Control Functionality

The SQL Distribution control and SQL Merge control have common functionality. This functionality includes:

- Adding subscriptions.

- Registering subscriptions in Microsoft® Windows Synchronization Manager.

- Copying subscriptions.

- Dropping subscriptions.

Note This functionality applies only to existing pull and anonymous subscriptions. It cannot be used for push subscriptions.

Adding Subscriptions

The SQLDistribution and SQLMerge objects can be configured to add subscriptions using the AddSubscription method. When adding a subscription, you can specify to create the Subscriber database, use an existing database, attach a database .mdf file, register an existing subscription, or attach a subscription .msf file.

To add a pull subscription, the Subscriber must already be defined at the Publisher.

Explicitly adding a subscription for an anonymous Subscriber is optional.

Registering Subscriptions

In addition to adding the subscription, the subscription can be registered in Microsoft Windows Synchronization Manager using the AddSubscription method. After being registered, there is no need to use the Microsoft ActiveX®
replication controls to synchronize the Subscriber because you can perform this operation using the Microsoft Windows Synchronization Manager. The Windows Synchronization Manager can be accessed from the Start menu by clicking Programs, clicking Accessories, then clicking Synchronize.

To register a subscription in Windows Synchronization Manager, specify SYNC_MANAGER (instead of the default NONE) for the second argument of AddSubscription.

**Copying Subscriptions**

The **SQLDistribution** and **SQLMerge** objects can be used to copy a specially prepared database subscription file (typically with an .msf extension) to a Subscriber, attach the subscription, and receive an immediately synchronized subscription at the original Publisher. The **CopySubscription** method creates the .msf file. The **AddSubscription** method with the ATTACH_SUBSCRIPTION option creates the new subscription from the .msf file.

**Dropping Subscriptions**

The **SQLDistribution** and **SQLMerge** objects can be programmed to drop subscriptions using the **DropSubscription** method. When removing the subscription, you can specify if the Subscriber database should be dropped. You can also specify that the subscription be unregistered from Microsoft Windows Synchronization Manager, but not dropped by specifying UNREGISTER_SUBSCRIPTION for the parameter of **DropSubscription**.

**See Also**

- AddSubscription Method
- CopySubscription Method
- DBADDOPTION
- DBDROPOPTION
- DropSubscription Method
- SUBSCRIPTION_HOST
Replication Programming
Programming Replication ActiveX Controls Using VBScript

Using the Microsoft® ActiveX® replication controls with Microsoft Visual Basic® Scripting Edition is supported with the following limitations:

- Named constants are not supported. You must specify the actual value represented by the constant; for example, AddSubscription(0, 1) instead of coding AddSubscription(EXISTING_DATABASE, SYNC_MANAGER).

- The error handler is not supported. You must use the ErrorRecords.Count method to determine if there are failures.

This example demonstrates how an HTML page using Visual Basic Scripting Edition configures the SQL Distribution control to synchronize data for an anonymous subscription over the Internet:

```html
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>New Page 1</title>
</head>

<body>
<p><font face="Papyrus">This is only a test.</font></p>
<p>
<object classid="clsid:08B0B2E6-3FB3-11D3-A4DE-00C04F610189"
</object>
</p>
```
<script LANGUAGE="VBScript">
<!-
Sub window_onload()
  Call DoSync()
End Sub

Sub DoSync()
  Dim oErrorObject
  On Error Resume Next

  ' Configure the control for an anonymous subscription.
  oSQLDistribution.Publisher = "PUBLISHERNAME"
  oSQLDistribution.PublisherDatabase = "PublishedDBName"
  oSQLDistribution.Publication = "PublicationName"
  oSQLDistribution.PublisherSecurityMode = 0 ' DB_AUTHENTICATION
  oSQLDistribution.PublisherLogin = "sa"
  oSQLDistribution.PublisherPassword = ""
  oSQLDistribution.SubscriberDatasourceType = 0 ' SQL_SERVER
  oSQLDistribution.Subscriber = "SUBSCRIBERNAME"
  oSQLDistribution.SubscriberDatabase = "SubscribingDBName"
  oSQLDistribution.SubscriptionType = 2 ' ANONYMOUS
  oSQLDistribution.SubscriberSecurityMode = 0 ' DB_AUTHENTICATION
  oSQLDistribution.SubscriberLogin = "sa"
  oSQLDistribution.SubscriberPassword = ""

  'Configure the control to access Publisher over the Internet using TCP/
  oSQLDistribution.PublisherNetwork = 1 ' TCPIP_SOCKETS

  ' Replace 000.00.00.00 with the correct IP address
  oSQLDistribution.PublisherAddress = "000.00.00.00,1433"
  oSQLDistribution.FileTransferType = 1 ' FILETRANSFERFTP
  ' Synchronize the data.
  oSQLDistribution.Initialize
</script>
If (oSQLDistribution.ErrorRecords.Count > 0) Then  
' Display each error message.  
For Each oErrorObject in oSQLDistribution.ErrorRecords  
  MsgBox oErrorObject.Description, vbCritical, "SQLDist Failure"  
Next  
Else  
oSQLDistribution.Run  
If (oSQLDistribution.ErrorRecords.Count > 0) Then  
  ' Display each error message.  
  For Each oErrorObject in oSQLDistribution.ErrorRecords  
    MsgBox oErrorObject.Description, vbCritical, "SQLDist Failure"  
  Next  
Else  
oSQLDistribution.Terminate  
If (oSQLDistribution.ErrorRecords.Count > 0) Then  
  ' Display each error message.  
  For Each oErrorObject in oSQLDistribution.ErrorRecords  
    MsgBox oErrorObject.Description, vbCritical, "SQLDist Failure"  
  Next  
End If  
End If  
End If  
End Sub

Sub oSQLDistribution_Status(Message, Percent)  
  ' Display message here.  
End Sub  
</script>  
</body>  
</html>

Note  The properties in this example are the same for both the SQL Merge and
SQL Distribution controls. You can replace references to the SQLDistribution object with references to the SQLMerge object.
Replication Programming
Replication ActiveX Control Reference

Microsoft® ActiveX® replication controls provide a means of programmatically controlling Merge Agent, Distribution Agent, and Snapshot Agent activity using a program written in Microsoft Visual Basic®, Microsoft Visual C++®, or other development languages that support COM.

The Microsoft ActiveX replication controls include:

- SQL Snapshot controls
- SQL Distribution controls
- SQL Merge controls

See Also

Developing Replication Applications Using ActiveX Controls
Object Model for ActiveX Controls
Other Replication Control Objects and Collections
Replication ActiveX Control Properties
Replication ActiveX Control Methods
Replication ActiveX Control Events
Replication ActiveX Control Constants
Replication Programming
# Object Model for ActiveX Controls

The table shows the Microsoft® ActiveX® controls and lists and describes their associated objects and collections.

<table>
<thead>
<tr>
<th>ActiveX control</th>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Distribution Control</td>
<td>SQLDistribution Object</td>
<td>Provides the functionality of the Distribution Agent and supports synchronization of push, pull, or anonymous subscriptions to a transactional publication.</td>
</tr>
<tr>
<td>SQL Merge Control</td>
<td>SQLMerge Object</td>
<td>Provides the functionality of the Merge Agent and supports synchronization of push, pull, or anonymous subscriptions to a merge publication.</td>
</tr>
<tr>
<td></td>
<td>SQLReplError Object</td>
<td>Defines an error that occurred during processing by an ActiveX replication control.</td>
</tr>
<tr>
<td></td>
<td>SQLReplErrors Collection</td>
<td>Contains a SQLReplError object for each error that has occurred during the most recent method execution of an ActiveX replication control.</td>
</tr>
<tr>
<td></td>
<td>AlternateSyncPartners Property</td>
<td>Returns a reference to the AlternateSyncPartners collection.</td>
</tr>
<tr>
<td></td>
<td>AlternateSyncPartners Collection</td>
<td>Contains all of the AlternateSyncPartner objects.</td>
</tr>
<tr>
<td></td>
<td>AlternateSyncPartner Object</td>
<td>Defines an alternate synchronization partner that a subscription (referenced by the SQLMerge object) can use if</td>
</tr>
</tbody>
</table>
the primary Publisher is not available, or if use of the alternate synchronization partner is preferable.

<table>
<thead>
<tr>
<th>SQLReplError Object</th>
<th>Defines an error that occurred during processing by an ActiveX replication control.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLReplErrors Collection</td>
<td>Contains a SQLReplError object for each error that has occurred during the most recent method execution of an ActiveX replication control.</td>
</tr>
</tbody>
</table>

SQL Snapshot Control

<table>
<thead>
<tr>
<th>SQLSnapshot Object</th>
<th>Creates a snapshot of the specified publication on the specified Distributor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLReplError Object</td>
<td>Defines an error that occurred during processing by an ActiveX replication control.</td>
</tr>
<tr>
<td>SQLReplErrors Collection</td>
<td>Contains a SQLReplError object for each error that has occurred during the most recent method execution of an ActiveX replication control.</td>
</tr>
</tbody>
</table>

See Also

- How Snapshot Replication Works
- How Transactional Replication Works
Replication Programming
SQLSnapshot Object

The SQL Snapshot control is implemented as a Microsoft® ActiveX® in-process component. It provides an alternate way, in addition to the Snapshot Agent, to create snapshots. The primary class of the SQLSnapshot, the SQLSnapshot object, creates a snapshot of the specified publication on the specified Distributor.

Dynamic snapshots are supported. There are properties to provide values for the HOST_NAME( ) and sp_repl_suser_sname( ) functions when they appear in the filter criteria of dynamic publications, and to specify the location where the dynamic snapshots are written.

Properties

- Distributor Property
- DistributorLogin Property
- DistributorPassword Property
- DistributorSecurityMode Property
- DynamicFilterHostName Property
- DynamicFilterLogin Property
- DynamicSnapshotLocation Property
- ErrorRecords Property
- LoginTimeout Property
- ProfileName Property
- Publication Property
- Publisher Property
- PublisherDatabase Property
- PublisherLogin Property
PublisherPassword Property
PublisherSecurityMode Property
QueryTimeout Property
ReplicationType Property

Methods
Initialize Method
Run Method
Terminate Method

Events
Status Event

Remarks
To refer to the elements of this object from a Microsoft Visual Basic® application, in the Project/References dialog box, select Microsoft SQL Snapshot Control 8.0. In a C++ application, include Replinx.c and Sqlinitx.h, and reference Sqlinitx.dll from the project/NMAKE file.

See Also
Developing Replication Applications Using ActiveX Controls
Replication Programming
**SQLDistribution Object**

The SQL Distribution control is implemented as a Microsoft® ActiveX® in-process component. The SQL Distribution control provides an alternate way, in addition to the Distribution Agent, to control synchronization of transactional replication subscriptions. Its primary class, the `SQLDistribution` object, provides the functionality of the Distribution Agent and supports synchronization of push, pull, or anonymous subscriptions to a transactional publication. The `SQLDistribution` object also includes options to:

- Add or drop subscriptions and register them for synchronization using Windows Synchronization Manager.
- Specify an alternate snapshot folder, from which the initial snapshot for a subscription can be applied.
- Specify a working directory to which snapshot files are copied using FTP.
- Generate a specially formatted subscription file in a specified location. These files can then be attached to create a synchronized subscription registered at the Publisher, as part of the attach-and-go functionality.
- Specify a Data Transformation Services (DTS) package that transforms command rowsets before applying them to a Subscriber.
- Set the Subscriber update mode to use immediate or queued updating for changes made at the Subscriber.

**Properties**

- [AltSnapshotFolder Property](#)
- [Distributor Property](#)
DistributorAddress Property
DistributorLogin Property
DistributorNetwork Property
DistributorPassword Property
DistributorSecurityMode Property
DTSPackageFileName Property
DTSPackagePassword Property
ErrorRecords Property
FileTransferType Property
FTPAddress Property
FTPLogin Property
FTPPassword Property
FTPPort Property
LoginTimeout Property
MaxDeliveredTransactions Property
ProfileName Property
Publication Property
Publisher Property
PublisherAddress Property
PublisherDatabase Property
PublisherLogin Property
PublisherNetwork Property
PublisherPassword Property
PublisherSecurityMode Property
QueryTimeout Property
SkipErrors Property
Subscriber Property
SubscriberDatabase Property
SubscriberDatabasePath Property
SubscriberDatasourceType Property
SubscriberLogin Property
SubscriberPassword Property
SubscriberSecurityMode Property
SubscriptionName Property
SubscriptionType Property
SynchronizationType Property
UndeliveredCommands Property
UndeliveredTransactions Property
WorkingDirectory Property

Methods
AddSubscription Method
CopySubscription Method
DropSubscription Method
Initialize Method
ReinitializeSubscription Method
Run Method
SetFailoverMode Method
Terminate Method

Events
**Remarks**

To refer to the elements of this object from a Microsoft Visual Basic® application, in the **Project/References** dialog box, select **Microsoft SQL Distribution Control 8.0**. In a C++ application, include Repldstx.c and Sqldistx.h, and reference Sqldistx.dll from the project/NMAKE file.

**See Also**

[Developing Replication Applications Using ActiveX Controls](#)
Replication Programming
**SQLMerge Object**

The SQL Merge control is implemented as a Microsoft® ActiveX® in-process component. The SQL Merge control provides an alternate way, in addition to the Merge Agent, to synchronize merge subscriptions. Its primary class, the SQLMerge object, provides the functionality of the Merge Agent and supports synchronization of push, pull, or anonymous subscriptions to a merge publication. The SQLMerge object also includes the options to:

- Add or drop subscriptions and register them for synchronization using Windows Synchronization Manager.

- Specify whether only the upload phase, the download phase, or both phases are run.

- Specify an alternate snapshot folder, from which the initial snapshot for a subscription can be applied.

- Specify a client-side working directory to which snapshot files can be copied using FTP.

- Copy a subscription database by generating a specially formatted subscription file in a specified location. These files can then be attached to create a synchronized subscription registered at the Publisher, as part of the attach-and-go functionality.

- Synchronize with a Publisher other than the one at which its subscription originated. Alternate Publishers must contain the same schema and data set as the original Publisher.

- Access an alternate sync partner when the primary Publisher of the data for the Subscriber is temporarily unavailable, or select an alternate sync partner to use the fastest available connection speed.
• Attach a subscription database by specifying only Subscriber-side properties. The **Publisher**, **PublisherDatabase**, **Distributor**, and **Publication** properties no longer need to be set while adding attach-and-go subscriptions.

• Validate that a subscription has the expected data.

**Properties**

- AlternateSyncPartners Property
- AltSnapshotFolder Property
- Distributor Property
- DistributorAddress Property
- DistributorLogin Property
- DistributorNetwork Property
- DistributorPassword Property
- DistributorSecurityMode Property
- DynamicSnapshotLocation Property
- ErrorRecords Property
- ExchangeType Property
- FileTransferType Property
- FTPAddress Property
- FTPLogin Property
- FTPPassword Property
- FTPPort Property
- HostName Property
LoginTimeout Property
ProfileName Property
Publication Property
Publisher Property
PublisherAddress Property
PublisherChanges Property
PublisherConflicts Property
PublisherDatabase Property
PublisherLogin Property
PublisherNetwork Property
PublisherPassword Property
PublisherSecurityMode Property
QueryTimeout Property
Subscriber Property
SubscriberChanges Property
SubscriberConflicts Property
SubscriberDatabase Property
SubscriberDatabasePath Property
SubscriberDatasourceType Property
SubscriberLogin Property
SubscriberPassword Property
SubscriberSecurityMode Property
SubscriptionName Property
SubscriptionPriority Property
SubscriptionPriorityType Property
**SubscriptionType Property**

**SynchronizationType Property**

**SyncToAlternate Property**

**UseInteractiveResolver Property**

**Validate Property**

**WorkingDirectory Property**

**Methods**

**AddSubscription Method**

**CopySubscription Method**

**DropSubscription Method**

**Initialize Method**

**ReinitializeSubscription Method**

**Run Method**

**Terminate Method**

**Events**

**Status Event**

**Remarks**

To refer to the elements of this object from a Microsoft Visual Basic® application, in the **Project/References** dialog box, select **Microsoft SQL Merge Control 8.0**. In a C/C++ application, include Replmrgx.c and Sqlmergx.h, and reference Sqlmergx.dll from the project/NMAKE file.

**See Also**

[Developing Replication Applications Using ActiveX Controls](#)
Replication Programming
## Other Replication Control Objects and Collections

The table lists the objects and collections that are used to implement features of the Microsoft® ActiveX® replication controls.

<table>
<thead>
<tr>
<th>Object/Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternateSyncPartner Object</td>
<td>Defines an alternate Publisher that a subscription can use.</td>
</tr>
<tr>
<td>AlternateSyncPartners Collection</td>
<td>Contains the alternate Publishers that a subscription can use.</td>
</tr>
<tr>
<td>SQLReplError Object</td>
<td>Defines an error that occurred during processing by a replication control.</td>
</tr>
<tr>
<td>SQLReplErrors Collection</td>
<td>Contains the error objects for a replication control.</td>
</tr>
</tbody>
</table>
Replication Programming
AlternateSyncPartner Object

An *AlternateSyncPartner* object defines an alternate Publisher that a subscription referenced by a *SQLMerge* object can use if the primary Publisher is not available, or if use of the alternate Publisher is preferable (for example, it is reachable by a faster or more reliable network connection).

**Properties**

- Distributor Property
- FriendlyName Property
- Publication Property
- Publisher Property
- PublisherDatabase Property

**Remarks**

To select an alternate synchronization partner, the *Publisher*, *PublisherDatabase*, *Publication*, and *Distributor* properties of the *SQLMerge* object should be set to the corresponding property values of the *AlternateSyncPartner* object before the Run method of the *SQLMerge* object is called.

**See Also**

- *AlternateSyncPartners Collection*
- *SQLMerge Object*
Replication Programming
AlternateSyncPartners Collection

The **AlternateSyncPartners** collection contains the **AlternateSyncPartner** objects, which define the alternate Publishers that a subscription referenced by a **SQLMerge** object can use if the primary Publisher is not available, or if use of the alternate Publisher is preferable.

**Applies To**

[SQLMerge Object](#)

**Properties**

[Count Property](#)

**Remarks**

The **AlternateSyncPartners** property of a **SQLMerge** object is used to return a reference to the **AlternateSyncPartners** collection.

**See Also**

[AlternateSyncPartner Object](#)

[AlternateSyncPartners Property](#)
Replication Programming
SQLReplError Object

A SQLReplError object defines an error that occurred during processing by a Microsoft® ActiveX® replication control.

Properties

Description Property

ErrorNumber Property

ErrorNumberString Property

Source Property

SourceType Property

Remarks

SQLReplError objects are referenced from the SQLReplErrors collection. A reference to this collection is obtained from the ErrorRecords property of replication ActiveX controls.

To refer to the elements of this object from a Microsoft Visual Basic® application, in the Project/References dialog box, select Microsoft SQL Replication Errors 8.0. In a C++ application, include Replerrx.h and reference Replerrx.dll from the project/NMAKE file.

An interface ISQLReplError is available directly from the replication ActiveX control components; however, use of the SQLReplError object is recommended.

See Also

SQLDistribution Object

SQLMerge Object

SQLReplErrors Collection

SQLSnapshot Object
Replication Programming
**SQLReplErrors Collection**

The **SQLReplErrors** collection contains an **SQLReplError** object for each error that has occurred during the most recent method execution of a Microsoft® ActiveX® replication control.

### Applies To

- **SQLDistribution Object**
- **SQLMerge Object**
- **SQLSnapshot Object**

### Properties

- **Count Property**

### Methods

- **Add Method**
- **AddReplError Method**

### Events

- **Notify Event**

### Remarks

A reference to the **SQLReplErrors** collection is obtained from the **ErrorRecords** property of replication ActiveX controls. The collection is cleared before the execution of each control method.

To refer to the elements of this collection from a Microsoft Visual Basic® application, in the **Project/References** dialog box, select **Microsoft SQL Replication Errors 8.0**. In a C++ application, include Replerrx.h and reference Replerrx.dll from the project/NMAKE file.
An interface **ISQLReplErrors** is available directly from the replication ActiveX control components; however, using the **SQLReplErrors** collection is recommended.

**See Also**

[ErrorRecords Property](#)

[SQLReplError Object](#)
Replication Programming
Replication ActiveX Control Properties

The topics in this section define the properties of the Microsoft® ActiveX® replication control objects and collections.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternateSyncPartners Property</td>
<td>PublisherChanges Property</td>
</tr>
<tr>
<td>AltSnapshotFolder Property</td>
<td>PublisherConflicts Property</td>
</tr>
<tr>
<td>Count Property</td>
<td>PublisherDatabase Property</td>
</tr>
<tr>
<td>Description Property</td>
<td>PublisherLogin Property</td>
</tr>
<tr>
<td>Distributor Property</td>
<td>PublisherNetwork Property</td>
</tr>
<tr>
<td>DistributorAddress Property</td>
<td>PublisherPassword Property</td>
</tr>
<tr>
<td>DistributorLogin Property</td>
<td>PublisherRPCLogin Property</td>
</tr>
<tr>
<td>DistributorNetwork Property</td>
<td>PublisherRPCPassword Property</td>
</tr>
<tr>
<td>DistributorPassword Property</td>
<td>PublisherRPCSecurityMode Property</td>
</tr>
<tr>
<td>DistributorSecurityMode Property</td>
<td>PublisherSecurityMode Property</td>
</tr>
<tr>
<td>DTSPackageFileName Property</td>
<td>QueryTimeout Property</td>
</tr>
<tr>
<td>DTSPackagePassword Property</td>
<td>ReplicationType Property</td>
</tr>
<tr>
<td>DynamicFilterHostName Property</td>
<td>SkipErrors Property</td>
</tr>
<tr>
<td>DynamicFilterLogin Property</td>
<td>Source Property</td>
</tr>
<tr>
<td>DynamicSnapshotLocation Property</td>
<td>SourceType Property</td>
</tr>
<tr>
<td>ErrorNumber Property</td>
<td>Subscriber Property</td>
</tr>
<tr>
<td>ErrorNumberString Property</td>
<td>SubscriberChanges Property</td>
</tr>
<tr>
<td>ErrorRecords Property</td>
<td>SubscriberConflicts Property</td>
</tr>
<tr>
<td>ExchangeType Property</td>
<td>SubscriberDatabase Property</td>
</tr>
<tr>
<td>FileTransferType Property</td>
<td>SubscriberDatabasePath Property</td>
</tr>
<tr>
<td>FriendlyName Property</td>
<td>SubscriberDatasourcePath Property</td>
</tr>
<tr>
<td>FTPAddress Property</td>
<td>SubscriberLogin Property</td>
</tr>
<tr>
<td>FTPLogin Property</td>
<td>SubscriberPassword Property</td>
</tr>
<tr>
<td>FTPPassword Property</td>
<td>SubscriberSecurityMode Property</td>
</tr>
<tr>
<td>FTPPort Property</td>
<td>SubscriptionName Property</td>
</tr>
<tr>
<td>Property</td>
<td>Property</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>HostName Property</td>
<td>SubscriptionPriority Property</td>
</tr>
<tr>
<td>LoginTimeout Property</td>
<td>SubscriptionPriorityType Property</td>
</tr>
<tr>
<td>MaxDeliveredTransactions Property</td>
<td>SubscriptionType Property</td>
</tr>
<tr>
<td>ProfileName Property</td>
<td>SynchronizationType Property</td>
</tr>
<tr>
<td>Publication Property</td>
<td>SyncToAlternate Property</td>
</tr>
<tr>
<td>Publisher Property</td>
<td>UndeliveredCommands Property</td>
</tr>
<tr>
<td>PublisherAddress Property</td>
<td>UndeliveredTransactions Property</td>
</tr>
<tr>
<td>UseInteractiveResolver Property</td>
<td>Validate Property</td>
</tr>
<tr>
<td>WorkingDirectory Property</td>
<td></td>
</tr>
</tbody>
</table>
Replication Programming
AlternateSyncPartners Property

The AlternateSyncPartners property returns a reference to the AlternateSyncPartners collection maintained by the SQLMerge object.

Applies To

SQLMerge Object

Syntax

object.AlternateSyncPartners

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLMerge object.</td>
</tr>
</tbody>
</table>

Remarks

The AlternateSyncPartners collection contains the AlternateSyncPartner objects, which define the alternate Publishers that a subscription referenced by a SQLMerge object can employ if the primary Publisher is not available, or if the use of the alternate Publisher is preferable.

The alternate Publishers must be defined in the publication for the AlternateSyncPartners collection to be populated.

Data Type

AlternateSyncPartners Collection

Modifiable

Read-only

Prototype (C/C++)

HRESULT AlternateSyncPartners(IAlternateSyncPartners** pVal);
Replication Programming
**AltSnapshotFolder Property**

The **AltSnapshotFolder** property returns or sets the path to the folder that contains the initial snapshot for a subscription.

**Applies To**

- **SQLDistribution Object**
- **SQLMerge Object**

**Syntax**

\[
\text{object.AltSnapshotFolder} = \text{path}
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>path</td>
<td>Path to alternate snapshot folder.</td>
</tr>
</tbody>
</table>

**Remarks**

The snapshot folder path must include a UNC or FTP designator, for example:

```plaintext
objMerge.AltSnapshotFolder = "file:\Distributor\Backup\Snapshots"
objDistr.AltSnapshotFolder = 
"ftp://distributor.company.com/backup/snapshots,1433"
```

It is recommended you use the **sp_copysnapshot** stored procedure to copy a subscription manually.

**Data Type**

String

**Modifiable**
Read/write

**Prototype (C/C++)**

HRESULT AltSnapshotFolder(BSTR pVal);
HRESULT AltSnapshotFolder(BSTR* pVal);

**See Also**

[How to Browse and Copy Snapshot Files (Transact-SQL)](http://example.com)
Replication Programming
**Count Property**

The **Count** property specifies the number of objects in a collection.

**Applies To**
- **AlternateSyncPartners Collection**
- **SQLReplErrors Collection**

**Syntax**

```
object.Count
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
</tbody>
</table>

**Data Type**

Long

**Modifiable**

Read-only

**Prototype (C/C++)**

```
HRESULT Count(long* cMembers);
```
Replication Programming
Description Property

The `Description` property returns a string that describes the error defined by a `SQLReplError` object.

Applies To

`SQLReplError Object`

Syntax

`object.Description`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <code>SQLReplError</code> object.</td>
</tr>
</tbody>
</table>

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT Description(BSTR pVal);

HRESULT Description(BSTR* pVal);

See Also

`ErrorNumber Property`

`ErrorNumberString Property`

`Source Property`

`SourceType Property`
Replication Programming
Distributor Property

The **Distributor** property sets or returns the name of the Distributor server.

**Applies To**

- AlternateSyncPartner Object
- SQLDistribution Object
- SQLMerge Object
- SQLSnapshot Object

**Syntax**

```
object.Distributor [ = name ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the Distributor used by the Publisher.</td>
</tr>
</tbody>
</table>

**Remarks**

**Distributor** is a required property for the **SQLSnapshot** object but optional for the other controls.

The Distributor is the instance of Microsoft® SQL Server™ on which the snapshots are stored by default, and on which replication history and statistics are logged. For the SQL Distribution control, it is also the location of the store-and-forward database that contains the replicated transactions.

If the **DistributorNetwork** and **DistributorAddress** properties are specified, they will be used instead of the **Distributor** property when connecting to the Distributor.

If Distributor connection properties are not specified, it is assumed the Publisher and Distributor are on the same instance of SQL Server, and Publisher
connection properties will be used when connecting to the Distributor.

**Data Type**
String

**Modifiable**
Read/write

**Prototype (C/C++)**
HRESULT Distributor(BSTR pVal);
HRESULT Distributor(BSTR* pVal);

**See Also**

[DistributorAddress Property](#)
[DistributorNetwork Property](#)
Replication Programming
**DistributorAddress Property**

The **DistributorAddress** property specifies the network address used for connecting to the Distributor when the **DistributorNetwork** property is specified.

**Applies To**

- SQLDistribution Object
- SQLMerge Object

**Syntax**

```
object.DistributorAddress [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Network address used when connecting to the Distributor.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a required property only when **DistributorNetwork** is set to a value other than DEFAULT_NETWORK.

This property is useful when configuring the control to connect to the Distributor without having to use SQL Server Client Network Utility.

For example, if the Distributor is accessed over the Internet, **DistributorNetwork** can be set to TCP/IP_SOCKETS and **DistributorAddress** can be set to a specific IP address.

If the **DistributorNetwork** is TCP/IP_SOCKETS or MULTI_PROTOCOL using TCP/IP, the value is in the form of:

'IP address,socket' (i.e. '111.11.11.11,1433' )
If the Distributor connection properties are not specified, it is assumed that the Publisher and Distributor are the same instance of Microsoft® SQL Server™ and the Publisher connection properties will be used when connecting to the Distributor.

**Data Type**
String

**Modifiable**
Read/write

**Prototype C/C++**
HRESULT get_DistributorAddress(BSTR *pVal);
HRESULT put_DistributorAddress(BSTR newVal);

**See Also**
[Distributor Property](#)
[DistributorNetwork Property](#)
Replication Programming
DistributorLogin Property

The DistributorLogin property specifies the login name used when connecting to the Distributor.

Applies To

SQLDistribution Object
SQLMerge Object
SQLSnapshot Object

Syntax

object.DistributorLogin [= name]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>name</td>
<td>Name used to log in to the Distributor.</td>
</tr>
</tbody>
</table>

Remarks

This is a required property if DistributorSecurityMode is set to DB_AUTHENTICATION.

If the Distributor connection properties are not specified, it is assumed that the Publisher and Distributor are the same instance of Microsoft® SQL Server™, and that the Publisher connection properties are used when connecting to the Distributor.

Data Type

String

Modifiable
Read/write

**Prototype C/C++**

HRESULT get_DistributorLogin(BSTR *pVal);

HRESULT put_DistributorLogin(BSTR newVal);

**See Also**

[DistributorPassword Property](#)

[DistributorSecurityMode Property](#)
Replication Programming
**DistributorNetwork Property**

The **DistributorNetwork** property specifies the client Net-Library used when connecting to the Distributor.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

```
object.DistributorNetwork [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>NETWORK_TYPE constant that specifies the client Net-Library used when connecting to the Distributor.</td>
</tr>
</tbody>
</table>

**Remarks**

This option is useful when configuring the control to connect to the Distributor without having to use SQL Server Client Network Utility.

If the value is not DEFAULT_NETWORK (default), the **DistributorAddress** property must be specified.

**Data Type**

*NETWORK_TYPE*

**Modifiable**

Read/write
Prototype C/C++

HRESULT get_DistributorNetwork(NETWORK_TYPE *pVal);
HRESULT put_DistributorNetwork(NETWORK_TYPE newVal);

See Also

DistributorAddress Property
Replication Programming
DistributorPassword Property

The **DistributorPassword** property sets or returns the login password used when connecting to the Distributor.

**Applies To**

- **SQLDistribution Object**
- **SQLMerge Object**
- **SQLSnapshot Object**

**Syntax**

```object.DistributorPassword [= string]```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>string</code></td>
<td>Password string used when connecting to the Distributor.</td>
</tr>
</tbody>
</table>

**Remarks**

This property is used only when **DistributorSecurityMode** is set to **DB.AUTHENTICATION**. Specify a NULL password as follows:

```     
objMerge.DistributorPassword = "" 
```

If the Distributor connection properties are not specified, it is assumed that the Publisher and Distributor are the same instance of Microsoft® SQL Server™, and that Publisher connection properties will be used when connecting to the Distributor.

The default is no password.

**Data Type**
String

**Modifiable**
Read/write

**Prototype C/C++**

```c
HRESULT get_DistributorPassword(BSTR *pVal);
HRESULT put_DistributorPassword(BSTR newVal);
```

**See Also**

[DistributorLogin Property](#)
[DistributorSecurityMode Property](#)
Replication Programming
**DistributorSecurityMode Property**

The **DistributorSecurityMode** property sets or returns the security mode used when connecting to the Distributor.

**Applies To**

- SQLDistribution Object
- SQLMerge Object
- SQLSnapshot Object

**Syntax**

`object.DistributorSecurityMode [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list</td>
</tr>
<tr>
<td><code>value</code></td>
<td>SECURITY_TYPE constant that specifies the mode of security enforced at the Distributor</td>
</tr>
</tbody>
</table>

**Remarks**

If the value is DB_AUTHENTICATION (default), **DistributorLogin** and **DistributorPassword** will be used when connecting to the Distributor.

NT_AUTHENTICATION is not supported for **DistributorSecurityMode** unless the Distributor uses the Microsoft® Windows NT® 4.0 or Microsoft Windows® 2000 operating system. NT_AUTHENTICATION is not supported for any of **DistributorSecurityMode**, **PublisherSecurityMode**, and **SubscriberSecurityMode** unless the computer on which the replication control is running uses the Windows NT 4.0 or Windows 2000 operating system.

**Data Type**

**SECURITY_TYPE**
Modifiable
Read/write

Prototype C/C++
HRESULT get_DistributorSecurityMode(SECURITY_TYPE *pVal);
HRESULT put_DistributorSecurityMode(SECURITY_TYPE newVal);

See Also

DistributorLogin Property
DistributorPassword Property
Replication Programming
DTSPackageFileName Property

The DTSPackageFileName property returns or sets the name and path of a DTS package used to transform command rowsets before they are applied to a Subscriber.

Applies To

SQLDistribution Object

Syntax

object.DTSPackageFileName [= PackageSpec]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLDistribution object.</td>
</tr>
<tr>
<td>PackageSpec</td>
<td>Fully qualified path to a DTS package.</td>
</tr>
</tbody>
</table>

Remarks

If the DTS package is stored in Microsoft® SQL Server™ or Meta Data Services, the value of this property is the package name. If the package is stored in a file, the property value is the file specification.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT DTSPackageFileName(BSTR pVal);
HRESULT DTSPackageFileName(BSTR* pVal);
See Also

DTSPackagePassword Property
Replication Programming
DTSPackagePassword Property

The **DTSPackagePassword** property returns or sets the **owner** password used to access the DTS package specified by the **DTSPackageFileName** property.

**Applies To**

**SQLDistribution** Object

**Syntax**

```
object.DTSPackagePassword [= string]
```

**Part** | **Description**
---|---
object | Expression that evaluates to a **SQLDistribution** object.
string | Password needed to access the specified DTS package.

**Remarks**

The DTS **owner** password must be provided for this property. The required access cannot be obtained with the **user** password.

This property returns the value to which the property was most recently set. It cannot be used to return the actual **owner** password of the DTS package specified by **DTSPackageFileName**, unless it was set to that value previously.

**Data Type**

String

**Modifiable**

Read/write

**Prototype (C/C++)**

```
HRESULT DTSPackagePassword(BSTR pVal);
```
HRESULT DTSPackagePassword(BSTR* pVal);

**See Also**

[DTSPackageFileName Property](#)
Replication Programming
DynamicFilterHostName Property

The **DynamicFilterHostName** property returns or sets the host name used when creating a dynamic snapshot. This property provides a value used when the publication is dynamically filtered using the **HOST_NAME()** function.

**Applies To**

*SQLSnapshot Object*

**Syntax**

```
oBJECT.DynamicFilterHostName [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <strong>SQLSnapshot</strong> object.</td>
</tr>
<tr>
<td>value</td>
<td>Value to be returned by <strong>HOST_NAME()</strong> in filter clauses when creating a dynamic snapshot.</td>
</tr>
</tbody>
</table>

**Remarks**

Use the **DynamicFilterHostName** property to set a value for **HOST_NAME()** in filtering when a dynamic snapshot is created. For example, if the subset filter clause "rep_id = HOST_NAME()" has been specified for an article, and if you set the **DynamicFilterHostName** property to "FBJones" before calling the **Run** method of the **SQLSnapshot** object, only rows having "FBJones" in the **rep_id** column will be included in the snapshot.

The **DynamicFilterHostName** property applies only to snapshots created for merge publications.

By default, **HOST_NAME()** evaluates to the name of the computer on which the merge control is executing, unless it is overridden by setting the **DynamicFilterHostName** property.

**Data Type**
String

**Modifiable**
Read/write

**Prototype C/C++**

HRESULT get_DynamicFilterHostName(BSTR *pVal);
HRESULT put_DynamicFilterHostName(BSTR newVal);

See Also

HOST_NAME
Replication Programming
**DynamicFilterLogin Property**

The **DynamicFilterLogin** property returns or sets the user name used when creating a dynamic snapshot. This property provides a value used when the publication is dynamically filtered using the **SUSER_SNAME()** function.

**Applies To**

**SQLSnapshot Object**

**Syntax**

```
object.DynamicFilterLogin [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <strong>SQLSnapshot</strong> object.</td>
</tr>
<tr>
<td>value</td>
<td>Value to be returned by <strong>SUSER_NAME( )</strong> in filter clauses when creating a dynamic snapshot.</td>
</tr>
</tbody>
</table>

**Remarks**

Use the **DynamicFilterLogin** property to set a value for **SUSER_SNAME()** in filtering when a dynamic snapshot is created. For example, if the subset filter clause "user_id = SUSER_SNAME()" has been specified for an article, and if you set the **DynamicFilterLogin** property to "rsmith" before calling the **Run** method of the **SQLSnapshot** object, only rows having "rsmith" in the **user_id** column will be included in the snapshot.

The **DynamicFilterLogin** property applies only to snapshots created for merge publications.

**Data Type**

String

**Modifiable**
Read/write

**Prototype C/C++**

HRESULT get_DynamicFilterLogin(BSTR *pVal);
HRESULT put_DynamicFilterLogin(BSTR newVal);

**See Also**

[SUSER_NAME]
Replication Programming
DynamicSnapshotLocation Property

The **DynamicSnapshotLocation** property returns or sets the path to a folder to which the files are to be written when a dynamic snapshot is created.

**Applies To**

**SQLSnapshot Object**

**Syntax**

```plaintext
object.DynamicSnapshotLocation [= path]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <strong>SQLSnapshot</strong> object.</td>
</tr>
<tr>
<td>path</td>
<td>Path of the folder to which the files of a dynamic snapshot are to be written.</td>
</tr>
</tbody>
</table>

**Remarks**

The **DynamicSnapshotLocation** property applies only to snapshots created for Merge publications.

The snapshot folder path must include a UNC or FTP designator, for example:

```plaintext
objMerge.DynamicSnapshotLocation = _
"file:\Distributor\Backup\Snapshots"
objDistr.DynamicSnapshotLocation = _
"ftp://distributor.company.com/backup/snapshots,1433"
```

**Data Type**

String

**Modifiable**

Read/write
Prototype C/C++

HRESULT get_DynamicSnapshotLocation(BSTR *pVal);
HRESULT put_DynamicSnapshotLocation(BSTR newVal);
Replication Programming
**ErrorNumber Property**

The **ErrorNumber** property returns the numeric code that identifies the error.

**Applies To**

SQLReplError Object

**Syntax**

`object.ErrorNumber [ = pVal ]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a SQLReplError object.</td>
</tr>
<tr>
<td><code>pVal</code></td>
<td>Error number.</td>
</tr>
</tbody>
</table>

**Remarks**

The error number is the value assigned by the error source.

**Data Type**

Long

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT ErrorNumber(long pVal);
HRESULT ErrorNumber(long* pVal);
```

**See Also**

Description Property
ErrorNumberString Property
Source Property
SourceType Property
Replication Programming
**ErrorNumberString Property**

The **ErrorNumberString** property returns or sets a string representation of the error number.

**Applies To**

*SQLReplError Object*

**Syntax**

`object.ErrorNumberString [= string]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <code>SQLReplError</code> object.</td>
</tr>
<tr>
<td><code>string</code></td>
<td>String representation of the error number.</td>
</tr>
</tbody>
</table>

**Data Type**

String

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT ErrorNumberString(BSTR pVal);
HRESULT ErrorNumberString(BSTR* pVal);
```

**See Also**

*Description Property*

*ErrorNumber Property*

*Source Property*
Replication Programming
**ErrorRecords Property**

The **ErrorRecords** property returns a reference to the **SQLReplErrors** collection maintained by the replication control.

**Applies To**

- **SQLDistribution Object**
- **SQLMerge Object**
- **SQLSnapshot Object**

**Syntax**

```
object.ErrorRecords
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
</tbody>
</table>

**Remarks**

The **SQLReplErrors** collection is loaded when a control method fails. The error records in the collection are available until the next replication control method is called.

**Data Type**

**SQLReplErrors Collection**

**Modifiable**

Read-only

**Prototype C/C++**

```c
HRESULT ErrorRecords(ISQLReplErrors** pVal);
```
See Also

SQLReplError Object

SQLReplErrors Collection
Replication Programming
ExchangeType Property

The ExchangeType property specifies whether data merges up to the Publisher, down to the Subscriber, or in both directions.

Applies To

SQLMerge Object

Syntax

object.ExchangeType [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLMerge object.</td>
</tr>
<tr>
<td>value</td>
<td>EXCHANGE_TYPE constant that specifies the direction in which data can be merged.</td>
</tr>
</tbody>
</table>

Remarks

The ExchangeType property is typically used first to upload all the changes from several Subscribers to a Publisher, and then to download the changes to the Subscribers only after all the uploads have been completed.

BIDIRECTIONAL is the default. You should use a BIDIRECTIONAL synchronization unless you have a specific reason for separating the UPLOAD and DOWNLOAD phases.

Data Type

EXCHANGE_TYPE

Modifiable

Read/write

Prototype C/C++
HRESULT get_ExchangeType(EXCHANGE_TYPE *pVal);
HRESULT put_ExchangeType(EXCHANGE_TYPE newVal);
Replication Programming
**FileTransferType Property**

The **FileTransferType** property returns or sets how the snapshot file is transferred.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

`object.FileTransferType [ = value ]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value from the <code>FILE_TRANSFER_TYPE</code> enumeration.</td>
</tr>
</tbody>
</table>

**Remarks**

The file transfer options are FTP and UNC (network file copy).

**Data Type**

`FILE_TRANSFER_TYPE`

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT FileTransferType(FILE_TRANSFER_TYPE pVal);
HRESULT FileTransferType(FILE_TRANSFER_TYPE* pVal);
```
Replication Programming
FriendlyName Property

The FriendlyName property returns or sets a display name by which the association of Publisher, publication, and Distributor that makes up an alternate sync partner can be identified.

Applies To

AlternateSyncPartner Object

Syntax

object.FriendlyName [ = name ]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an AlternateSyncPartner object.</td>
</tr>
<tr>
<td>name</td>
<td>Friendly name for alternate sync partner.</td>
</tr>
</tbody>
</table>

Remarks

An AlternateSyncPartner object defines an alternate Publisher that a subscription referenced by the SQLMerge object can use if the primary Publisher is not available.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT FriendlyName(BSTR pVal);
HRESULT FriendlyName(BSTR* pVal);

See Also

SQLMerge Object
Replication Programming
**FTPAddress Property**

The `FTPAddress` property specifies the IP address of the FTP site where publication snapshot files are stored.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

`object.FTPAddress [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Network address of the FTP site.</td>
</tr>
</tbody>
</table>

**Remarks**

`FTPAddress` is an optional property.

If specified, the `FTPLogin`, `FTPPassword`, and `FTPPort` properties are also used.

This option is useful when configuring the control to connect to a Distributor over the Internet, where the Distributor working directory is not directly accessible. Setting this property forces the control to download all publication snapshot files using File Transfer Protocol (FTP) before applying them at the Subscriber.

These properties should not be required when you dynamically determine the FTP login properties by querying the publication. Provided for backward compatibility only: these properties will not be effective when used against Microsoft® SQL Server™ 2000 publications unless the Subscriber is a Microsoft SQL Server version 7.0 Subscriber.
**Note**  The publication must be enabled for the Internet.

**Data Type**
String

**Modifiable**
Read/write

**Prototype C/C++**

HRESULT get_FTPAddress(BSTR *pVal);
HRESULT put_FTPAddress(BSTR newVal);

**See Also**

[ Distributor Property](#)

[ DistributorAddress Property](#)

[Implementing Replication Over the Internet](#)
Replication Programming
**FTPLogin Property**

The **FTPLogin** property specifies the user name of the FTP site where the publication snapshot files are stored.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

`object.FTPLogin [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>FTP user name of the FTP site.</td>
</tr>
</tbody>
</table>

**Remarks**

The default is **anonymous**. These properties should not be required when you dynamically determine the FTP login properties by querying the publication. Provided for backward compatibility only: these properties will not be effective when used against Microsoft® SQL Server™ 2000 publications unless the Subscriber is a Microsoft SQL Server version 7.0 Subscriber.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**
HRESULT get_FTPLogin(BSTR *pVal);
HRESULT put_FTPLogin(BSTR newVal);

See Also

**FTPAddress Property**
**FTPPassword Property**
**FTPPort Property**
Replication Programming
**FTPPassword Property**

The **FTPPassword** property specifies the password of the FTP site where the publication snapshot files are stored.

**Applies To**

- **SQLDistribution Object**
- **SQLMerge Object**

**Syntax**

```
object.FTPPassword [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Password of the FTP site.</td>
</tr>
</tbody>
</table>

**Remarks**

The default is no password. Typically, an e-mail address of the form `user@company.com` is used for FTP passwords.

These properties should not be required when you dynamically determine the FTP login properties by querying the publication. Provided for backward compatibility only: these properties will not be effective when used against Microsoft® SQL Server™ 2000 publications unless the Subscriber is a Microsoft SQL Server version 7.0 Subscriber.

**Data Type**

String

**Modifiable**

Read/write
Prototype C/C++

HRESULT get_FTPPassword(BSTR *pVal);
HRESULT put_FTPPassword(BSTR newVal);

See Also

FTPAddress Property
FTPLogin Property
FTPPort Property
Replication Programming
FTPPort Property

The **FTPPort** property specifies the TCP port number of the FTP site where the publication snapshot files are stored.

**Applies To**

- SQLDistribution Object
- SQLMerge Object

**Syntax**

```
object.FTPPort [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>TCP port number of the FTP site.</td>
</tr>
</tbody>
</table>

**Remarks**

The default value is 21.

These properties should not be required when you dynamically determine the FTP login properties by querying the publication. Provided for backward compatibility only: these properties will not be effective when used against Microsoft® SQL Server™ 2000 publications unless the Subscriber is a Microsoft SQL Server version 7.0 Subscriber.

**Data Type**

Long

**Modifiable**

Read/write
**Prototype C/C++**

HRESULT get_FTPPort(long *pVal);
HRESULT put_FTPPort(long newVal);

**See Also**

[FTPAddress Property](#)
Replication Programming
HostName Property

The **HostName** property returns or sets the host name used when connecting to the Publisher. This property provides a value used when the publication is dynamically filtered using the SQL Server **HOST_NAME( )** function.

**Applies To**

**SQLMerge Object**

**Syntax**

```
object.HostName [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <strong>SQLMerge</strong> object.</td>
</tr>
<tr>
<td>value</td>
<td>Value to be returned by <strong>HOST_NAME( )</strong> in filter clauses.</td>
</tr>
</tbody>
</table>

**Remarks**

Use the **HostName** property to set a value for **HOST_NAME( )** in dynamic filtering. For example, if the subset filter clause "rep_id = HOST_NAME( )" has been specified for an article, and if you set the **HostName** property to "FBJones" before calling the **Run** method of the **SQLMerge** object, only rows having "FBJones" in the **rep_id** column will participate in the replication of that article.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**
HRESULT get_HostName(BSTR *pVal);
HRESULT put_HostName(BSTR newVal);

See Also

HOST_NAME
Replication Programming
LoginTimeout Property

The `LoginTimeout` property specifies the maximum number of seconds to wait for connections to be established.

Applies To

- SQLDistribution Object
- SQLMerge Object
- SQLSnapshot Object

Syntax

```
object>LoginTimeout [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Number of seconds for connections to be established.</td>
</tr>
</tbody>
</table>

Remarks

The default is 15 seconds. A value of 0 specifies an indefinite wait.

Data Type

Integer

Modifiable

Read/write

Prototype C/C++

```
HRESULT get_LoginTimeout(short *pVal);
```
HRESULT put_LoginTimeout(short newVal);
Replication Programming
MaxDeliveredTransactions Property

The MaxDeliveredTransactions property sets or returns the maximum number of transactions to be downloaded to Subscribers during each Run operation.

Applies To

SQLDistribution Object

Syntax

object.MaxDeliveredTransactions [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLDistribution object.</td>
</tr>
<tr>
<td>value</td>
<td>Maximum number of transactions to be downloaded.</td>
</tr>
</tbody>
</table>

Remarks

The default is 0, which means that all available transactions are delivered. Other values can be used to control the number of transactions downloaded during a Run operation.

Data Type

Integer

Modifiable

Read/write

Prototype C/C++

HRESULT get_MaxDeliveredTransactions(long *pVal);
HRESULT put_MaxDeliveredTransactions(long newVal);
See Also

UndeliveredCommands Property
UndeliveredTransactions Property
Replication Programming
ProfileName Property

The ProfileName property returns or sets the name of the agent profile at the Distributor to be used by the replication control.

Applies To

SQLDistribution Object
SQLMerge Object
SQLSnapshot Object

Syntax

object.ProfileName [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Agent profile name.</td>
</tr>
</tbody>
</table>

Remarks

If not specified, the default profile for the agent type is used.

Data Type

String

Modifiable

Read/write

Prototype C/C++

HRESULT get_ProfileName(BSTR *pVal);
HRESULT put_ProfileName(BSTR newVal);

See Also

Agent Profiles

How to create a replication agent profile (Enterprise Manager)
Replication Programming
**Publication Property**

The *Publication* property returns or sets the name of the publication.

**Applies To**

- AlternateSyncPartner Object
- SQLDistribution Object
- SQLMerge Object
- SQLSnapshot Object

**Syntax**

```
object.Publication [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Publication name.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a required property. An error is raised if it is set to an empty string.

If using the SQL Distribution Control to synchronize multiple publications configured to use a shared agent, the value must be ALL. For more information about shared agents, see `sp_addpublication`.

**Data Type**

String

**Modifiable**

Read/write
**Prototype C/C++**

HRESULT get_Publication(BSTR *pVal);
HRESULT put_Publication(BSTR newVal);

**See Also**

[Publisher Property](#)
[PublisherDatabase Property](#)
Replication Programming
Publisher Property

The Publisher property returns or sets the Publisher name where the publication resides.

Applies To

AlternateSyncPartner Object
SQLDistribution Object
SQLMerge Object
SQLSnapshot Object

Syntax

object.Publisher [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Publisher name.</td>
</tr>
</tbody>
</table>

Remarks

If the PublisherNetwork and PublisherAddress properties are specified, they will be used instead of the Publisher property when connecting to the Publisher; otherwise, Publisher is a required property.

Data Type

String

Modifiable

Read/write
**Prototype C/C++**

HRESULT get_Publisher(BSTR *pVal);
HRESULT put_Publisher(BSTR newVal);

**See Also**

Publication Property
PublisherDatabase Property
Replication Programming
PublisherAddress Property

The PublisherAddress property specifies the network address used when connecting to the Publisher when the PublisherNetwork property is specified.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.PublisherAddress [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Network connection string.</td>
</tr>
</tbody>
</table>

Remarks

This is a required property when PublisherNetwork is set to a value other than DEFAULT_NETWORK.

This property is useful when configuring the control to connect to the Publisher without having to use SQL Server Client Network Utility.

For example, if the Publisher is to be accessed over the Internet, PublisherNetwork can be set to TCP/IP_SOCKETS and PublisherAddress can be set to a specific IP address.

If the PublisherNetwork is TCP/IP_SOCKETS or MULTI_PROTOCOL over TCP/IP, the value is in the form of:

'TP address, socket' (i.e. '111.11.11.11,1433")

Data Type
String

**Modifiable**
Read/write

**Prototype C/C++**
HRESULT get_PublisherAddress(BSTR *pVal);
HRESULT put_PublisherAddress(BSTR newVal);

**See Also**

PublisherNetwork Property
Replication Programming
PublisherChanges Property

The **PublisherChanges** property returns the total number of Publisher changes applied to the Subscriber during the last **Run** operation.

**Applies To**

*SQLMerge Object*

**Syntax**

```
object.PublisherChanges [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <strong>SQLMerge</strong> object.</td>
</tr>
<tr>
<td>value</td>
<td>Total number of Publisher rows inserted, deleted, and updated.</td>
</tr>
</tbody>
</table>

**Data Type**

Long

**Modifiable**

Read-only

**Prototype C/C++**

```
HRESULT get_PublisherChanges(long *pVal);
```

**See Also**

*PublisherConflicts Property*

*SubscriberChanges Property*
Replication Programming
**PublisherConflicts Property**

The **PublisherConflicts** property specifies the total number of conflicts that occurred at the Publisher during the last **Run** operation.

**Applies To**

[SQLMerge Object](#)

**Syntax**

```
oBJECT.PublisherConflicts [ = value ]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <strong>SQLMerge</strong> object.</td>
</tr>
<tr>
<td>value</td>
<td>Number of conflicts that occurred.</td>
</tr>
</tbody>
</table>

**Data Type**

Long

**Modifiable**

Read-only

**Prototype C/C++**

```
HRESULT get_PublisherConflicts(long *pVal);
```

**See Also**

[SubscriberConflicts Property](#)

[PublisherChanges Property](#)
Replication Programming
PublisherDatabase Property

The **PublisherDatabase** property returns or sets the name of the publication database.

**Applies To**

- **AlternateSyncPartner Object**
- **SQLDistribution Object**
- **SQLMerge Object**
- **SQLSnapshot Object**

**Syntax**

```
object.PublisherDatabase [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Publication database name.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a required property. An error is raised if it is set to an empty string.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**
HRESULT get_PublisherDatabase(BSTR *pVal);
HRESULT put_PublisherDatabase(BSTR newVal);

**See Also**

Publication Property
Publisher Property
Replication Programming
PublisherLogin Property

The PublisherLogin property sets or returns the login name used when connecting to the Publisher.

Applies To

SQLDistribution Object
SQLMerge Object
SQLSnapshot Object

Syntax

object.PublisherLogin [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>User name used to log in to the Publisher.</td>
</tr>
</tbody>
</table>

Remarks

This is a required property if PublisherSecurityMode is set to DB_AUTHENTICATION.

Data Type

String

Modifiable

Read/write

Prototype C/C++

HRESULT get_PublisherLogin(BSTR *pVal);
HRESULT put_PublisherLogin(BSTR newVal);

See Also

PublisherPassword Property
PublisherSecurityMode Property
Replication Programming
**PublisherNetwork Property**

The **PublisherNetwork** property specifies the client Net-Library used when connecting to the Publisher.

**Applies To**

- **SQLDistribution Object**
- **SQLMerge Object**

**Syntax**

`object.PublisherNetwork [ = value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>NETWORK_TYPE constant that specifies the client Net-Library to use.</td>
</tr>
</tbody>
</table>

**Remarks**

This option is useful when configuring the control to connect to the Publisher without having to use SQL Server Client Network Utility.

If the value is not DEFAULT_NETWORK (default), the **DistributorAddress** property must be specified.

**Data Type**

**NETWORK_TYPE**

**Modifiable**

Read/write
Prototype C/C++

HRESULT get_PublisherNetwork(NETWORK_TYPE *pVal);
HRESULT put_PublisherNetwork(NETWORK_TYPE newVal);

See Also

Publisher Property
PublisherAddress Property
Replication Programming
PublisherPassword Property

The PublisherPassword property sets or returns the login password used when connecting to the Publisher.

Applies To

SQLDistribution Object
SQLMerge Object
SQLSnapshot Object

Syntax

```
object.PublisherPassword [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Password used to connect to the Publisher.</td>
</tr>
</tbody>
</table>

Remarks

The default is no password.

This is a required property if PublisherSecurityMode is set to DB_AUTHENTICATION.

Data Type

String

Modifiable

Read/write

Prototype C/C++
HRESULT get_PublisherPassword(BSTR *pVal);
HRESULT put_PublisherPassword (BSTR newVal);

See Also

PublisherLogin Property
PublisherSecurityMode Property
Replication Programming
PublisherRPCLogin Property

The PublisherRPCLogin property determines the login name used by the immediate-updating Subscriber trigger RPC when connecting to the Publisher.

Applies To

SQLDistribution Object

Syntax

\textit{object.PublisherRPCLogin} [= \textit{value}]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Login to be used when connecting to the Publisher.</td>
</tr>
</tbody>
</table>

Remarks

This property is relevant only to immediate-updating subscriptions.

Data Type

String

Modifiable

Read/write

Prototype C/C++

\texttt{HRESULT get_PublisherRPCLogin(BSTR *pVal);}  
\texttt{HRESULT put_PublisherRPCLogin(BSTR newVal);}  

See Also
PublisherRPCPassword
PublisherRPCSecurityMode
Replication Programming
**PublisherRPCPassword Property**

The **PublisherRPCPassword** property determines the password used by the immediate-updating Subscriber trigger RPC when connecting to the Publisher.

**Applies To**

*SQLDistribution Object*

**Syntax**

```
object.PublisherRPCPassword [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Password string used when connecting to the Publisher.</td>
</tr>
</tbody>
</table>

**Remarks**

This property is relevant only to immediate-updating subscriptions.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**

```c
HRESULT get_PublisherRPCPassword(BSTR *pVal);
HRESULT put_PublisherRPCPassword(BSTR newVal);
```

**See Also**
Replication Programming
PublisherRPCSecurityMode Property

The **PublisherRPCSecurityMode** property determines the security mode used by the immediate-updating Subscriber trigger RPC when connecting to the Publisher.

**Applies To**

*SQLDistribution* Object

**Syntax**

```object.PublisherRPCPassword [= value]```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies the mode of security enforced during immediate-updating Subscriber RPC login at the Publisher. The value is from the REPLRPC_SECURITY_TYPE enumeration.</td>
</tr>
</tbody>
</table>

**Remarks**

This property is relevant only to immediate-updating subscriptions.

**Data Type**

Long, enumerated.

**Modifiable**

Read/write

**Prototype C/C++**

```HRESULT get_PublisherRPCPassword(REPLRPC_SECURITY_TYPE *pVal);```
HRESULT put_PublisherRPCPassword(REPLRPC_SECURITY_TYPE newVal);

**See Also**

[PublisherRPCLogin](#)

[PublisherRPCPassword](#)
Replication Programming
PublisherSecurityMode Property

The PublisherSecurityMode property sets or returns the security mode when connecting to the Publisher.

Applies To

SQLDistribution Object

SQLMerge Object

SQLSnapshot Object

Syntax

object.PublisherSecurityMode [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>SECURITY_TYPE constant that specifies the Publisher security mode.</td>
</tr>
</tbody>
</table>

Remarks

If the value is DB_AUTHENTICATION (default), PublisherLogin and PublisherPassword will be used when connecting to the Publisher.

NT_AUTHENTICATION is not supported for PublisherSecurityMode unless the Publisher runs on the Microsoft® Windows NT® 4.0 or Microsoft Windows® 2000 operating system. NT_AUTHENTICATION is not supported for any of DistributorSecurityMode, PublisherSecurityMode, and SubscriberSecurityMode unless the computer on which the replication control is running uses the Windows NT 4.0 or Windows 2000 operating system.

Data Type

SECURITY_TYPE
Modifiable
Read/write

Prototype C/C++
HRESULT get_PublisherSecurityMode(SEcurity_TYPE *pVal);
HRESULT put_PublisherSecurityMode(SEcurity_TYPE newVal);

See Also
PublisherLogin Property
PublisherPassword Property
Replication Programming
**QueryTimeout Property**

The `QueryTimeout` property returns or sets the number of seconds allowed for internal queries to complete.

**Applies To**
- `SQLDistribution Object`
- `SQLMerge Object`
- `SQLSnapshot Object`

**Syntax**

```
object.QueryTimeout [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Number of seconds allowed for internal queries to be returned.</td>
</tr>
</tbody>
</table>

**Remarks**

The default value is 30.

A value of 0 means to wait indefinitely.

**Data Type**

Integer

**Modifiable**

Read/write

**Prototype C/C++**
HRESULT get_QueryTimeout(short *pVal);
HRESULT put_QueryTimeout(short newVal);
Replication Programming
ReplicationType Property

The **ReplicationType** property returns or sets the type of replication for which the snapshot is to be used.

**Applies To**

[SQLSnapshot Object](#)

**Syntax**

`object.ReplicationType [ = value ]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <strong>SQLSnapshot</strong> object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value from the <strong>REPLICATION_TYPE</strong> enumeration.</td>
</tr>
</tbody>
</table>

**Remarks**

The types of replication are transactional replication and merge replication.

**Data Type**

**REPLICATION_TYPE**

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT ReplicationType(REPLICATION_TYPE pVal);
HRESULT ReplicationType(REPLICATION_TYPE* pVal);
```
Replication Programming
**SkipErrors Property**

The **SkipErrors** property returns or sets a list of the errors to be skipped.

**Applies To**

SQLDistribution Object

**Syntax**

\[object.SkipErrors \equiv list\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLDistribution object.</td>
</tr>
<tr>
<td>list</td>
<td>List of the errors that are to be skipped by the control.</td>
</tr>
</tbody>
</table>

**Remarks**

The SkipErrors list has this format:

\(<\text{native_error_id1}>:<\text{native_error_id2}>:<\text{native_error_id3}>\ldots\)

Native error IDs are the error numbers from the underlying database. Only the errors that are encountered when applying replication transactions at the Subscriber can be skipped.

**Data Type**

String

**Modifiable**

Read/Write

**Prototype C/C++**

HRESULT get_SkipErrors(BSTR *pList);
HRESULT put_SkipErrors(BSTR pList);

See Also

Handling Errors and Messages in Applications
Replication Programming
Source Property

The **Source** property returns or sets the name of the source where the error occurred.

**Applies To**

**SQLReplError Object**

**Syntax**

```
object.Source [= pVal]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <strong>SQLReplError</strong> object.</td>
</tr>
<tr>
<td><code>pVal</code></td>
<td>Name of the source where the error occurred.</td>
</tr>
</tbody>
</table>

**Data Type**

String

**Modifiable**

Read/write

**Prototype (C/C++)**

HRESULT Source(BSTR pVal);

HRESULT Source(BSTR* pVal);

**See Also**

- [Description Property](#)
- [ErrorNumber Property](#)
- [ErrorNumberString Property](#)
- [SourceType Property](#)
Replication Programming
**SourceType Property**

The **SourceType** property specifies the type of the source of the error information.

**Applies To**

*SQLReplError Object*

**Syntax**

`object.SourceType [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <strong>SQLReplError</strong> object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value from the <strong>ERRORSOURCE_TYPE</strong> enumeration.</td>
</tr>
</tbody>
</table>

**Data Type**

**ERRORSOURCE_TYPE**

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT SourceType(ERRORSOURCE_TYPE pVal);
HRESULT SourceType(ERRORSOURCE_TYPE* pVal);
```

**See Also**

- [Description Property](#)
- [ErrorNumber Property](#)
- [ErrorNumberString Property](#)
Source Property
Replication Programming
**Subscriber Property**

The **Subscriber** property specifies the name of the Subscriber.

**Applies To**

- **SQLDistribution Object**
- **SQLMerge Object**

**Syntax**

```
object.Subscriber [= value]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Subscriber name.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a required property.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**

```c
HRESULT get_Subscriber(BSTR *pVal);
HRESULT put_Subscriber(BSTR newVal);
```

**See Also**
SubscriberDatasourceType Property
Replication Programming
SubscriberChanges Property

The **SubscriberChanges** property specifies the total number of Subscriber changes applied at the Publisher during the last **Run** operation.

**Applies To**

*SQLMerge Object*

**Syntax**

```
[value =] object.SubscriberChanges
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <em>SQLMerge</em> object.</td>
</tr>
<tr>
<td>value</td>
<td>Total number of rows inserted, deleted, and updated.</td>
</tr>
</tbody>
</table>

**Data Type**

Long

**Modifiable**

Read-only

**Prototype C/C++**

```
HRESULT get_SubscriberChanges(long *pVal);
```

**See Also**

*PublisherChanges Property*

*SubscriberConflicts Property*
Replication Programming
SubscriberConflicts Property

The SubscriberConflicts property specifies the total number of conflicts that occurred during the upload operation from the Subscriber.

Applies To

SQLMerge Object

Syntax

object.SubscriberConflicts

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLMerge object.</td>
</tr>
</tbody>
</table>

Data Type

Long

Modifiable

Read-only

Prototype C/C++

HRESULT get_SubscriberConflicts(long *pVal);

See Also

PublisherConflicts Property

SubscriberChanges Property
Replication Programming
**SubscriberDatabase Property**

The **SubscriberDatabase** property specifies the name of the Subscriber database.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

```object.SubscriberDatabase [ = value]```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Subscriber database name.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a required property if **SubscriberDatasourceType** is SQLSERVER.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**

```HRESULT get_SubscriberDatabase(BSTR *pVal);
HRESULT put_SubscriberDatabase(BSTR newVal);```
See Also

SubscriberDatabasePath Property
Replication Programming
SubscriberDatabasePath Property

The **SubscriberDatabasePath** property specifies the path to a Microsoft® Jet 4.0 database or Microsoft SQL Server™ detached database or subscription file.

**Applies To**

- SQLDistribution Object
- SQLMerge Object

**Syntax**

`object.SubscriberDatabasePath [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Path to a Jet database or SQL Server detached database or subscription file.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a required property if **SubscriberDatasourceType** is JET4_DATABASE, or if you are using the DBADDOPTION constants ATTACH_DATABASE or ATTACH_SUBSCRIPTION when calling **AddSubscription**.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**
HRESULT get_SubscriberDatabasePath(BSTR *pVal);
HRESULT put_SubscriberDatabasePath(BSTR newVal);

See Also

AddSubscription Method
DBADDOPTION
SubscriberDatasourceType Property
Replication Programming
SubscriberDatasourceType Property

The SubscriberDatasourceType property specifies the type of Subscriber data source.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.SubscriberDatasourceType [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>DATASOURCE_TYPE constant that specifies the type of database at the Subscriber.</td>
</tr>
</tbody>
</table>

Remarks

The default is SQL_SERVER.

Data Type

DATASOURCE_TYPE

Modifiable

Read/write

Prototype C/C++

HRESULT get_SubscriberDatasourceType(
    DATASOURCE_TYPE *pVal);
HRESULT put_SubscriberDatasourceType(DATASOURCE_TYPE newVal);

**See Also**

[SubscriberDatabasePath Property](#)
Replication Programming
SubscriberLogin Property

The SubscriberLogin property specifies the login name used connecting to the Subscriber.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.SubscriberLogin [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Subscriber login name.</td>
</tr>
</tbody>
</table>

Remarks

This is a required property if SubscriberSecurityMode is set to DB_AUTHENTICATION.

Data Type

String

Modifiable

Read/write

Prototype C/C++

HRESULT get_SubscriberLogin(BSTR *pVal);
HRESULT put_SubscriberLogin(BSTR newVal);
See Also

SubscriberPassword Property
SubscriberSecurityMode Property
Replication Programming
**SubscriberPassword Property**

The **SubscriberPassword** property specifies the login password used when connecting to the Subscriber.

**Applies To**
- SQLDistribution Object
- SQLMerge Object

**Syntax**

`object.SubscriberPassword [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Subscriber password string.</td>
</tr>
</tbody>
</table>

**Remarks**

This property is used only when **SubscriberSecurityMode** is set to DB_AUTHENTICATION.

The default is no password.

**Data Type**

String

**Modifiable**

Read/write

**Prototype C/C++**

```c
HRESULT get_SubscriberPassword(BSTR *pVal);
```
HRESULT put_SubscriberPassword(BSTR newVal);

See Also

SubscriberLogin Property
SubscriberSecurityMode Property
Replication Programming
SubscriberSecurityMode Property

The **SubscriberSecurityMode** property specifies the security mode used when connecting to the Publisher.

**Applies To**

[SQLDistribution Object](#)

[SQLMerge Object](#)

**Syntax**

`object.SubscriberSecurityMode [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>SECURITY_TYPE constant that specifies the security mode enforced at the Subscriber.</td>
</tr>
</tbody>
</table>

**Remarks**

If the value is DB_AUTHENTICATION (default), **SubscriberLogin** and **SubscriberPassword** will be used when connecting to the Subscriber.

NT_AUTHENTICATION is not supported for **SubscriberSecurityMode** unless the Subscriber runs on the Microsoft® Windows NT® 4.0 or Microsoft Windows® 2000 operating systems. NT_AUTHENTICATION is not supported for any of **DistributorSecurityMode**, **PublisherSecurityMode**, and **SubscriberSecurityMode** unless the computer on which the replication control is running uses the Windows NT 4.0 or Windows 2000 operating system.

**Data Type**

[SECURITY_TYPE](#)
Modifiable
Read/write

Prototype C/C++
HRESULT get_SubscriberSecurityMode(SECURITY_TYPE *pVal);
HRESULT put_SubscriberSecurityMode(SECURITY_TYPE newVal);

See Also
SubscriberLogin Property
SubscriberPassword Property
Replication Programming
SubscriptionName Property

The SubscriptionName property specifies a display name for the subscription. This name is used in the Windows Synchronization Manager user interface.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.Subscription [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>Name of the Subscription.</td>
</tr>
</tbody>
</table>

Remarks

If SubscriptionName is not specified, a subscription name is set using Subscriber name, Subscriber database, and publication.

The SubscriptionName property is not persisted anywhere except in the operating system registry for the current user Synchronization manager settings. All subsequent merges will show the property to be formatted:

subscribername:subscriberdatabase

This is consistent with the way subscription names are shown in SQL Server Enterprise Manager.

Data Type

String
**Modifiable**
Read/write

**Prototype (C/C++)**
HRESULT get_SubscriptionName(BSTR *pVal);
HRESULT put_SubscriptionName(BSTR newVal);
Replication Programming
SubscriptionPriority Property

The **SubscriptionPriority** returns or sets the priority of the subscription.

**Applies To**

[SQLMerge Object](#)

**Syntax**

\[ object.SubscriptionPriority \[= value] \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLMerge object.</td>
</tr>
<tr>
<td>value</td>
<td>Subscription priority value from 0.0 through 100.0, inclusive.</td>
</tr>
</tbody>
</table>

**Remarks**

If the **SubscriptionPriorityType** property has the value GLOBAL_PRIORITY, use this property to set the priority.

**Data Type**

Single/Float

**Modifiable**

Read/write

**Prototype C/C++**

```c
HRESULT get_SubscriptionPriority(float *pVal);
HRESULT put_SubscriptionPriority(float newVal);
```

**See Also**
Subscriber Types and Conflicts

SUBSCRIPTION_PRIORITY_TYPE

SubscriptionPriorityType Property
Replication Programming
SubscriptionPriorityType Property

The SubscriptionPriorityType property specifies how the subscription priority is determined.

Applies To

SQLMerge Object

Syntax

object.SubscriptionType [ = value ]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLMerge object.</td>
</tr>
<tr>
<td>value</td>
<td>SUBSCRIPTION_PRIORITY_TYPE constant that specifies how the subscription priority is determined.</td>
</tr>
</tbody>
</table>

Remarks

The subscription priority can be assigned or assume the priority value of the Publisher. If the value of the property is GLOBAL_PRIORITY, use the SubscriptionPriority property to set the priority.

Data Type

SUBSCRIPTION_PRIORITY_TYPE

Modifiable

Read/write

Prototype C/C++

HRESULT get_SubscriptionPriorityType(SUBSCRIPTION_PRIORITY_TYPE_TYPE *pVal);
HRESULT put_SubscriptionPriorityType(SUBSCRIPTION_PRIORITY_TYPE newVal);

**See Also**

[Subscriber Types and Conflicts](#)

[SubscriptionPriority Property](#)
Replication Programming
SubscriptionType Property

The SubscriptionType property specifies whether the subscription is push, pull, or anonymous.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.SubscriptionType [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>SUBSCRIPTION_TYPE constant that specifies the type of subscription.</td>
</tr>
</tbody>
</table>

Remarks

The publication must be configured to support the specified subscription type. The default is ANONYMOUS.

Data Type

SUBSCRIPTION_TYPE

Modifiable

Read/write

Prototype C/C++

HRESULT get_SubscriptionType(SUBSCRIPTION_TYPE *pVal);
HRESULT put_SubscriptionType(SUBSCRIPTION_TYPE newVal);
Replication Programming
SynchronizationType Property

The SynchronizationType property specifies whether the subscription needs to be initially synchronized.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.SubscriptionType [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>value</td>
<td>SYNCHRONIZATION_TYPE constant that specifies whether initial synchronization will occur.</td>
</tr>
</tbody>
</table>

Remarks

The default is AUTOMATIC.

Data Type

SYNCHRONIZATION_TYPE

Modifiable

Read/write

Prototype C/C++

HRESULT get_SynchronizationType(SYNCHRONIZATION_TYPE *pVal);
HRESULT put_SynchronizationType(SYNCHRONIZATION_TYPE newVal);
Replication Programming
SyncToAlternate Property

The **SyncToAlternate** property returns or sets whether the synchronization is to an alternate synchronization partner.

**Applies To**

*SQLMerge Object*

**Syntax**

`object.SyncToAlternate [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <em>SQLMerge</em> object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>If set to <strong>True</strong>, an alternate synchronization partner is used.</td>
</tr>
</tbody>
</table>

**Remarks**

To select an alternate synchronization partner, the **Publisher**, **PublisherDatabase**, **Publication**, and **Distributor** properties of the *SQLMerge* object should be set to the corresponding property values of an **AlternateSyncPartner** object before the **Run** method of the *SQLMerge* object is called.

**Data Type**

Boolean

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT SyncToAlternate(VARIANT_BOOL pVal);
HRESULT SyncToAlternate(VARIANT_BOOL* pVal);
```

**See Also**
AlternateSyncPartner Object
AlternateSyncPartners Collection
Distributor Property
Publication Property
Publisher Property
PublisherDatabase Property
Replication Programming
**UndeliveredCommands Property**

The **UndeliveredCommands** property specifies the number of commands currently available to download to the Subscriber.

**Applies To**

**SQLDistribution Object**

**Syntax**

```
object.UndeliveredCommands
```

**Part** | **Description**
--- | ---
`object` | Expression that evaluates to a **SQLDistribution** object.

**Data Type**

Long

**Modifiable**

Read-only

**Prototype C/C++**

```
HRESULT get_UndeliveredCommands(long *pVal);
```

**See Also**

- [MaxDeliveredTransactions Property](#)
- [UndeliveredTransactions Property](#)
Replication Programming
**UndeliveredTransactions Property**

The **UndeliveredTransactions** property returns the number of transactions currently available to download to the Subscriber.

**Applies To**

*SQLDistribution Object*

**Syntax**

*object*.UndeliveredTransactions

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a <em>SQLDistribution</em> object.</td>
</tr>
</tbody>
</table>

**Data Type**

Long

**Modifiable**

Read-only

**Prototype C/C++**

HRESULT get_UndeliveredTransactions(long *pVal);

**See Also**

*MaxDeliveredTransactions Property*

*UndeliveredCommands Property*
Replication Programming
**UseInteractiveResolver Property**

The `UseInteractiveResolver` property returns or sets whether the interactive resolver is used during reconciliation.

**Applies To**

`SQLMerge Object`

**Syntax**

`object.UseInteractiveResolver [= value]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <code>SQLMerge</code> object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>If set to <strong>True</strong>, the interactive resolver is used.</td>
</tr>
</tbody>
</table>

**Remarks**

The interactive resolver is displayed for each row in which a conflict is detected. If the property is changed from **True** to **False** during a merge replication, all subsequent conflicts will be handled by the default resolver or the resolver specified when the article was created.

**Data Type**

Boolean

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT UseInteractiveResolver(VARIANT_BOOL pVal);
HRESULT UseInteractiveResolver(VARIANT_BOOL* pVal);
```
Replication Programming
Validate Property

The Validate property specifies the type of data validation to perform on the Subscriber data at the end of the Run.

Applies To

SQLMerge Object

Syntax

object.Validate [= value]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLMerge object.</td>
</tr>
<tr>
<td>value</td>
<td>VALIDATE_TYPE constant that specifies the type of data validation to perform.</td>
</tr>
</tbody>
</table>

Data Type

VALIDATE_TYPE

Modifiable

Read/write

Prototype C/C++

HRESULT get_Validate(VALIDATE_TYPE newVal);
HRESULT put_Validate(VALIDATE_TYPE* pVal);
Replication Programming
WorkingDirectory Property

The **WorkingDirectory** property returns or sets the working directory to which snapshot files are transferred using FTP when that option is specified.

**Applies To**

- SQLDistribution Object
- SQLMerge Object

**Syntax**

```
object.WorkingDirectory [= path]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>path</td>
<td>Fully qualified path to the working directory for copied snapshots.</td>
</tr>
</tbody>
</table>

**Data Type**

String

**Modifiable**

Read/write

**Prototype (C/C++)**

```c
HRESULT WorkingDirectory(BSTR pVal);
HRESULT WorkingDirectory(BSTR* pVal);
```
Replication Programming
Replication ActiveX Control Methods

This section defines the methods of the replication Microsoft® ActiveX® control objects and collections.

**Methods**

Add Method

AddReplError Method

AddSubscription Method

CopySubscription Method

DropSubscription Method

Initialize Method

ReinitializeSubscription Method

Run Method

SetFailoverMode Method

Terminate Method
Replication Programming
Add Method

The Add method adds a SQLReplError object to a SQLReplErrors collection.

Applies To

SQLReplErrors Collection

Syntax

collection.Add object

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>Expression that evaluates to a SQLReplErrors collection.</td>
</tr>
<tr>
<td>object</td>
<td>Expression that evaluates to a SQLReplError object.</td>
</tr>
</tbody>
</table>

Remarks

The AddReplError method can be used to add an error to a SQLReplErrors collection without creating a SQLReplError object first.

Prototype (C/C++)

HRESULT Add(
    ISQLReplError* pISQLReplError);

See Also

AddReplError Method

SQLReplError Object
Replication Programming
AddReplError Method

The AddReplError method adds a new error to a SQLReplErrors collection.

Applies To

SQLReplErrors Collection

Syntax

\[ \text{collection}.\text{AddReplError \ Description, Source, ErrorNumber, ErrorSourceType, ErrorNumberString} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{collection}</td>
<td>Expression that evaluates to a \textbf{SQLReplErrors} collection.</td>
</tr>
<tr>
<td>\textit{Description}</td>
<td>String that describes the error.</td>
</tr>
<tr>
<td>\textit{Source}</td>
<td>String that describes the component that generated the error.</td>
</tr>
<tr>
<td>\textit{ErrorNumber}</td>
<td>Long integer code for the error.</td>
</tr>
<tr>
<td>\textit{ErrorSourceType}</td>
<td>Value from the \textbf{ERRORSOURCE_TYPE} enumerating the type of error source.</td>
</tr>
<tr>
<td>\textit{ErrorNumberString}</td>
<td>String representation of the error number.</td>
</tr>
</tbody>
</table>

Remarks

The AddReplError method adds an error to a SQLReplErrors collection without the caller explicitly creating a SQLReplError object. The Add method can be used to add a SQLReplError object to a SQLReplErrors collection.

Prototype (C/C++)

\[
\text{HRESULT AddReplError(}
    \text{BSTR bstrDescription,}
\text{)}
\]
See Also

Add Method

ERRORSOURCE_TYPE

SQLReplError Object
Replication Programming
AddSubscription Method

The AddSubscription method adds a new or existing subscription based on the specified control properties.

Applies To

SQLDistribution Object

SQLMerge Object

Syntax

object.AddSubscription DBAddOption, SubscriptionHost

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>DBAddOption</td>
<td>Option to add; use a value from DBADDOPTION.</td>
</tr>
<tr>
<td>SubscriptionHost</td>
<td>Subscription host; use a value from SUBSCRIPTION_HOST.</td>
</tr>
</tbody>
</table>

Remarks

Push subscriptions are not currently supported by this method.

If DBAddOption is set to ATTACH_DATABASE and SubscriberDatasourceType is set to SQL_SERVER, AddSubscription can work only with single-file databases. Use the SubscriberDatabasePath property to specify the name and path of the .mdf file to attach.

If DBAddOption is ATTACH_SUBSCRIPTION, use SubscriberDatabasePath to specify the name and path of the Microsoft Subscription File (.msf). This file can be created with the CopySubscription method.

AddSubscription also supports creating a new Subscriber database, creating a new subscription for an existing database, and registering an existing subscription with Windows Synchronization Manager.
In Microsoft® Visual Basic®, **AddSubscription** is a **Sub** method and does not return a value.

**Prototype C/C++**

```c
HRESULT AddSubscription(
    DBADDOPTION DBAddOption,
    SUBSCRIPTION_HOST SubscriptionHost);
```

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>Method succeeded.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>General failure occurred. Check error records for detailed information.</td>
</tr>
<tr>
<td>REPLX_E_DBEXISTS</td>
<td>Specified database already exists.</td>
</tr>
<tr>
<td>REPLX_E_DBNOTFOUND</td>
<td>Specified database does not exist.</td>
</tr>
<tr>
<td>REPLX_E_SUBEXISTS</td>
<td>Specified subscription already exists.</td>
</tr>
<tr>
<td>REPLX_E_RETRYFAILURE</td>
<td>Failure occurred that might require retrying the last operation.</td>
</tr>
</tbody>
</table>

**See Also**

- [CopySubscription Method](#)
- [DropSubscription Method](#)
- [SubscriberDatasourceType Property](#)
- [SubscriberDatabasePath Property](#)
Replication Programming
CopySubscription Method

The **CopySubscription** method copies the entire subscription database to the file location specified by the parameter.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

`object.CopySubscription filespec`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>filespec</code></td>
<td>File name and path to which subscription database is copied.</td>
</tr>
</tbody>
</table>

**Remarks**

**CopySubscription** copies a specially prepared database subscription file (typically with an .msf extension) to a Subscriber, attaches it, and receives an immediately synchronized subscription at the original Publisher.

**CopySubscription** creates the .msf file. Use the **AddSubscription** method with the ATTACH_SUBSCRIPTION option to create the new subscription from the .msf file.

You can use the **CopySubscription** method to copy a subscription database that contains more than one subscription.

**Prototype (C/C++)**

```c
HRESULT CopySubscription(BSTR bstrSubscriptionFileName);
```
See Also

AddSubscription Method
DropSubscription Method
Replication Programming
DropSubscription Method

The **DropSubscription** method drops the subscription having the specified control properties.

**Applies To**
- **SQLDistribution Object**
- **SQLMerge Object**

**Syntax**

```
object.DropSubscription DBDropOption
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>DBDropOption</td>
<td>Option to drop; use a value from <strong>DBDROPOPTION</strong>.</td>
</tr>
</tbody>
</table>

**Remarks**

Push subscriptions are not currently supported by this method.

If DROP_DATABASE is specified, the database is dropped even when the subscription specified by the **SubscriptionName** property does not exist. Error notification is not provided.

**DropSubscription** also supports dropping the subscription without dropping the database and unregistering the subscription from Windows Synchronization Manager.

In Microsoft® Visual Basic®, **DropSubscription** is a **Sub** method and does not return a value.

**Prototype C/C++**

```
HRESULT DropSubscription();
```
DBDROPOPTION DBDropOption);

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>Method succeeded.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>General failure occurred. Check error records for detailed information.</td>
</tr>
<tr>
<td>REPLX_E_SUBNOTFOUND</td>
<td>Specified subscription does not exist.</td>
</tr>
<tr>
<td>REPLX_E_RETRYFAILURE</td>
<td>Failure occurred that might require retrying the last operation.</td>
</tr>
</tbody>
</table>

See Also

- AddSubscription Method
- CopySubscription Method
- DBDROPOPTION
- SubscriptionName Property
Replication Programming
Initialize Method

The Initialize method validates the control properties and establishes all database connections.

Applies To

SQLDistribution Object

SQLMerge Object

SQLSnapshot Object

Syntax

object.Initialize

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
</tbody>
</table>

Remarks

Control properties that define the Publisher, publication, Distributor, and Subscriber (for the Distribution and Merge controls) must be set before calling Initialize.

In Microsoft® Visual Basic®, Initialize is a Sub method and does not return a value.

Initialize should not be called prior to using the AddSubscription method to add a new subscription, or prior to using the DropSubscription method to drop an existing subscription.

Prototype (C/C++)

HRESULT Initialize( );

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>S_OK</td>
<td>Method succeeded.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>General failure occurred. Check error records for detailed information.</td>
</tr>
<tr>
<td>REPLX_E_DBNOTFOUND</td>
<td>Specified database does not exist.</td>
</tr>
<tr>
<td>REPLX_E_SUBNOTFOUND</td>
<td>Specified subscription does not exist (SQLDistribution and SQLMerge objects only).</td>
</tr>
<tr>
<td>REPLX_E_SUBEXPIRED</td>
<td>Subscription has expired (SQLDistribution and SQLMerge objects only).</td>
</tr>
<tr>
<td>REPLX_E_PROFILENOTFOUND</td>
<td>Specified profile does not exist.</td>
</tr>
<tr>
<td>REPLX_E_RETRYFAILURE</td>
<td>Failure occurred that might require retrying the last operation.</td>
</tr>
</tbody>
</table>

**See Also**

[Developing Replication Applications Using ActiveX Controls](#)
Replication Programming
ReinitializeSubscription Method

The `ReinitializeSubscription` method configures a subscription to reapply the initial snapshot and subsequent changes during the next `Run` operation.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

**Syntax**

`object.ReinitializeSubscription [bUploadBeforeReinit]`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <code>SQLMerge</code> object.</td>
</tr>
<tr>
<td><code>bUploadBeforeReinit</code></td>
<td>If <code>True</code>, the changes in the subscription database are uploaded to the Publisher before the snapshot is applied at the Subscriber. The default is <code>False</code>.</td>
</tr>
</tbody>
</table>

**Remarks**

`ReinitializeSubscription` is a method of both the `SQLDistribution` and `SQLMerge` objects. `ReinitializeSubscription` for the `SQLMerge` object has the `bUploadBeforeReinit` parameter. For the `SQLDistribution` object, `ReinitializeSubscription` has no parameters.

In Microsoft® Visual Basic®, `ReinitializeSubscription` is a `Sub` method and does not return a value.

**Prototype (C/C++)**

`HRESULT ReinitializeSubscription(VARIANT_BOOL bUploadBeforeReinit);`

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>Method succeeded.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>General failure occurred. Check error records for detailed information.</td>
</tr>
<tr>
<td>REPLX_E_DEADLOCK</td>
<td>Deadlock occurred.</td>
</tr>
<tr>
<td>REPLX_E_RETRYFAILURE</td>
<td>Failure occurred that might require retrying the last operation.</td>
</tr>
</tbody>
</table>
Replication Programming
Run Method

The **Run** method executes the replication process using the control properties.

Applies To

- **SQLDistribution Object**
- **SQLMerge Object**
- **SQLSnapshot Object**

Syntax

```
object.Run
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
</tbody>
</table>

Remarks

The control must call **Initialize** before calling the **Run** method.

In Microsoft® Visual Basic®, **Run** is a **Sub** method and does not return a value.

Prototype (C/C++)

```
HRESULT Run( );
```

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>Method succeeded.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>General failure occurred. Check error records for detailed information.</td>
</tr>
<tr>
<td>REPLX_E_DEADLOCK</td>
<td>Deadlock occurred.</td>
</tr>
<tr>
<td>REPLX_E_RETRYFAILURE</td>
<td>Failure occurred that might require retrying the last operation.</td>
</tr>
</tbody>
</table>
See Also

Developing Replication Applications Using ActiveX Controls
Replication Programming
SetFailoverMode Method

The `SetFailoverMode` method sets the Subscriber update mode when adding subscriptions to use immediate (DTC) updates or queued updates, or support immediate updating with a queued updating failover.

**Applies To**

`SQLDistribution Object`

**Syntax**

`object.SetFailoverMode FailoverMode`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to a <code>SQLDistribution</code> object.</td>
</tr>
<tr>
<td><code>FailoverMode</code></td>
<td>Value from the REPL_FAILOVER_MODE enumeration.</td>
</tr>
</tbody>
</table>

**Prototype (C/C++)**

`HRESULT SetFailoverMode(REPL_FAILOVER_MODE FailoverMode);`

**See Also**

`REPL_FAILOVER_MODE`
Replication Programming
Terminate Method

The **Terminate** method terminates the replication process and closes all database connections.

**Applies To**

*SQLDistribution Object*

*SQLMerge Object*

*SQLSnapshot Object*

**Syntax**

`object.Terminate`

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
</tbody>
</table>

**Remarks**

After using **Terminate**, the control host must call **Initialize** before again calling **Run**.

In Microsoft® Visual Basic®, **Terminate** is a **Sub** method and does not return a value.

**Prototype (C/C++)**

`HRESULT Terminate();`

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>Method succeeded.</td>
</tr>
<tr>
<td>E_FAIL</td>
<td>General failure occurred. Check error records for detailed information.</td>
</tr>
<tr>
<td>REPLX_E_RETRYFAILURE</td>
<td>Failure occurred that might require retrying</td>
</tr>
</tbody>
</table>
the last operation.

See Also

[Developing Replication Applications Using ActiveX Controls]
Replication Programming
Replication ActiveX Control Events

This section defines the events of the replication Microsoft® ActiveX® control objects and collections.

Events

Notify Event
Status Event
Replication Programming
Notify Event

The Notify event is raised when an error is added to the SQLReplErrors collection by a replication Microsoft® ActiveX® control.

Applies To

SQLReplErrors Collection

Prototype (Visual Basic)

Sub Objectvar_Notify( _
    ByVal Status As REPLERRXLib.AGENT_STATUS _
    ByVal Message As String)

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectvar</td>
<td>Expression that evaluates to a SQLReplErrors collection.</td>
</tr>
<tr>
<td>Status</td>
<td>Value that indicates the type of error from the AGENT_STATUS enumeration.</td>
</tr>
<tr>
<td>Message</td>
<td>Description of the error.</td>
</tr>
</tbody>
</table>

Remarks

To receive the Notify event, the program must declare Objectvar WithEvents of type REPLERRXLib.SQLReplErrors.

This event is not available through the ISQLReplErrors interface of the replication ActiveX controls.

Prototype (C/C++)

HRESULT Notify( 
    AGENT_STATUS Status, 
    BSTR Message);
See Also

AGENT_STATUS
Replication Programming
Status Event

The Status event returns information about significant occurrences in the operation of a replication Microsoft® ActiveX® control.

Applies To

SQLDistribution Object

SQLMerge Object

SQLSnapshot Object

Prototype (Visual Basic)

Function Objectvar_Status(_
    ByVal Message As String, _
    ByVal Percent As Long) _
    As STATUS_RETURN_CODE

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectvar</td>
<td>Expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>Message</td>
<td>Description of the significant occurrence.</td>
</tr>
<tr>
<td>Percent</td>
<td>Percentage of the operation completed successfully, from 0 through 100.</td>
</tr>
</tbody>
</table>

Remarks

To receive the Status event, a Microsoft Visual Basic® program must declare Objectvar WithEvents of the appropriate type.

STATUS_RETURN_CODE is an enumeration that specifies the values that can be returned from the Status callback function (event).

Prototype (C/C++)
HRESULT Status(
    BSTR Message,
    Long Percent,
    STATUS_RETURN_CODE *pReturnCode);

See Also

Developing Replication Applications Using ActiveX Controls

STATUS_RETURN_CODE
Replication Programming
Replication ActiveX Control Constants

This section defines the enumerated data types that are used as parameters and return values in Microsoft® ActiveX® replication control properties, methods, and events.
Replication Programming
The **AGENT_STATUS** constants provide agent status codes that are returned by the Notify event of the **SQLReplErrors** collection.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLAGENT_FAIL</td>
<td>4</td>
<td>Agent operation failed.</td>
</tr>
<tr>
<td>REPLAGENT_IN_PROGRESS</td>
<td>1</td>
<td>Agent operation in progress.</td>
</tr>
<tr>
<td>REPLAGENT_RETRY</td>
<td>3</td>
<td>Agent operation failed.</td>
</tr>
<tr>
<td>REPLAGENT_SUCCEED</td>
<td>2</td>
<td>Agent operation completed successfully.</td>
</tr>
</tbody>
</table>

### Remarks

REPLAGENT_RETRY indicates the agent operation failed with errors that may not recur if the operation is retried at a later time. The control does not retry the operation unless the calling program directs it to do so.

### See Also

- Notify Event
- SQLReplErrors Collection
Replication Programming
DATASOURCE_TYPE

The **DATASOURCE_TYPE** constants provide data source type values for the 
**SubscriberDatasourceType** property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2_UNIVERSAL</td>
<td>6</td>
<td>DB2 Universal database Subscriber</td>
</tr>
<tr>
<td>EXCHANGE</td>
<td>4</td>
<td>Microsoft® Exchange Subscriber</td>
</tr>
<tr>
<td>JET4_DATABASE</td>
<td>2</td>
<td>Microsoft Jet 4.0 database Subscriber</td>
</tr>
<tr>
<td>ODBC_DSN</td>
<td>1</td>
<td>ODBC data source Subscriber</td>
</tr>
<tr>
<td>OLEDB_DATASOURCE</td>
<td>3</td>
<td>OLE DB data source Subscriber</td>
</tr>
<tr>
<td>ORACLE</td>
<td>5</td>
<td>Oracle database Subscriber</td>
</tr>
<tr>
<td>SQL_SERVER</td>
<td>0</td>
<td>Microsoft SQL Server™ Subscriber</td>
</tr>
</tbody>
</table>

See Also

[SubscriberDatasourceType Property](#)
Replication Programming
DBADDOPTION

The **DBADDOPTION** constants specify if the Subscriber database exists or if it must be created or attached, or if the subscription must be attached, when calling the **AddSubscription** method of the **SQLDistribution** and **SQLMerge** objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH_DATABASE</td>
<td>2</td>
<td>Attaches a Subscriber database file, typically an .mdf (Microsoft® SQL Server™ Subscribers only).</td>
</tr>
<tr>
<td>ATTACH_SUBSCRIPTION</td>
<td>3</td>
<td>Attaches a subscription file, typically an .msf (Microsoft Subscription File).</td>
</tr>
<tr>
<td>CREATE_DATABASE</td>
<td>1</td>
<td>Creates the Subscriber database (SQL Server Subscribers only).</td>
</tr>
<tr>
<td>EXISTING_DATABASE</td>
<td>0</td>
<td>Uses an existing Subscriber database.</td>
</tr>
<tr>
<td>REGISTER_SUBSCRIPTION</td>
<td>4</td>
<td>Registers the existing subscription with Windows Synchronization Manager.</td>
</tr>
</tbody>
</table>

**See Also**

[AddSubscription Method](#)
Replication Programming
DBDROPOPTION

The **DBDROPOPTION** constants specify whether the Subscriber database should be dropped when calling the **DropSubscription** method of the **SQLDistribution** and **SQLMerge** objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP_DATABASE</td>
<td>1</td>
<td>Drops the Subscriber database and the subscription, if specified.</td>
</tr>
<tr>
<td>LEAVE_DATABASE</td>
<td>0</td>
<td>Drops the subscription without dropping the Subscriber database.</td>
</tr>
<tr>
<td>UNREGISTER_SUBSCRIPTION</td>
<td>2</td>
<td>Unregisters, but does not drop, the subscription.</td>
</tr>
</tbody>
</table>

**See Also**

[DropSubscription Method](#)
Replication Programming
The `ERRORSOURCE_TYPE` constants provide values for the `SourceType` property of the `SQLReplError` object.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALID_SOURCE_TYPE</td>
<td>0</td>
<td>Error source type is invalid.</td>
</tr>
<tr>
<td>MERGE_PROCESS</td>
<td>9</td>
<td>Merge process error.</td>
</tr>
<tr>
<td>MERGE_PROVIDER</td>
<td>8</td>
<td>Merge replication provider error.</td>
</tr>
<tr>
<td>NET_LIBRARY</td>
<td>6</td>
<td>Net-Library error.</td>
</tr>
<tr>
<td>ODBC_API</td>
<td>4</td>
<td>ODBC API error.</td>
</tr>
<tr>
<td>OPERATING_SYSTEM</td>
<td>3</td>
<td>Operating system error.</td>
</tr>
<tr>
<td>REPL_CONTROL</td>
<td>2</td>
<td>Replication ActiveX® control error.</td>
</tr>
<tr>
<td>SQL_COMMAND</td>
<td>1</td>
<td>SQL command error.</td>
</tr>
<tr>
<td>SQLDMO</td>
<td>7</td>
<td>SQL DMO error.</td>
</tr>
<tr>
<td>SQLSERVER_ENGINE</td>
<td>5</td>
<td>Microsoft® SQL Server™ error.</td>
</tr>
</tbody>
</table>

See Also

`SourceType Property`
Replication Programming
EXCHANGE_TYPE

The EXCHANGE_TYPE constants are used with the ExchangeType property of the SQLMerge object to specify whether merge replication changes should be uploaded to the Publisher, downloaded to the Subscriber, or both (uploaded and then downloaded).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPLOAD</td>
<td>1</td>
<td>Only merge Subscriber changes with the Publisher.</td>
</tr>
<tr>
<td>DOWNLOAD</td>
<td>2</td>
<td>Only merge Publisher changes with the Subscriber.</td>
</tr>
<tr>
<td>BIDIRECTIONAL</td>
<td>3</td>
<td>Merge all changes between the Publisher and Subscriber (default).</td>
</tr>
</tbody>
</table>

Remarks

You should use a BIDIRECTIONAL synchronization unless you have a specific reason for separating the UPLOAD and DOWNLOAD phases.

See Also

ExchangeType Property
Replication Programming
FILE_TRANSFER_TYPE

The FILE_TRANSFER_TYPE constants specify the type of transfer for snapshot files.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILETRANSFERFTP</td>
<td>1</td>
<td>Download snapshot files using FTP.</td>
</tr>
<tr>
<td>FILETRANSFERUNC</td>
<td>0</td>
<td>Apply snapshot from a UNC network share.</td>
</tr>
</tbody>
</table>

See Also

FileTransferType Property
Replication Programming
**NETWORK_TYPE**

The **NETWORK_TYPE** constants provide network protocol type values for the **DistributorNetwork** and **PublisherNetwork** properties.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>0</td>
<td>Use the current configured client Net-Library (default).</td>
</tr>
<tr>
<td>MULTI_PROTOCOL</td>
<td>2</td>
<td>Multiprotocol Net-Library.</td>
</tr>
<tr>
<td>TCPIP_SOCKETS</td>
<td>1</td>
<td>TCP/IP Sockets Net-Library.</td>
</tr>
</tbody>
</table>

**See Also**

[DistributorNetwork Property](#)  
[PublisherNetwork Property](#)
Replication Programming
REPL_FAILOVER_MODE

The REPL_FAILOVER_MODE enumeration specifies the Subscriber update mode when adding subscriptions to use immediate updating or queued updating options with transactional replication.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAILOVER_IMMEDIATE</td>
<td>1</td>
<td>Immediate updating.</td>
</tr>
<tr>
<td>FAILOVER_NONE</td>
<td>0</td>
<td>No updating from Subscriber.</td>
</tr>
<tr>
<td>FAILOVER_QUEUED</td>
<td>2</td>
<td>Queued updating.</td>
</tr>
</tbody>
</table>

See Also

SetFailoverMode Method
Replication Programming
REPLICATION_TYPE

The REPLICATION_TYPE enumeration specifies the type of replication for which the snapshot is to be used.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERGE</td>
<td>2</td>
<td>Specifies merge replication.</td>
</tr>
<tr>
<td>TRANSACTIONAL</td>
<td>1</td>
<td>Specifies transactional replication or snapshot replication.</td>
</tr>
</tbody>
</table>

See Also

ReplicationType Property
Replication Programming
REPLRPC_SECURITY_TYPE

The REPLRPC_SECURITY_TYPE constants provide data source type values for the PublisherRPCSecurityMode property.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC_STANDARD_MODE</td>
<td>0</td>
<td>Dynamic RPC connection is used.</td>
</tr>
<tr>
<td>RPC_STATIC_MODE</td>
<td>2</td>
<td>Static sysservers entry is used for RPC.</td>
</tr>
</tbody>
</table>

See Also

PublisherRPCSecurityMode
Replication Programming
The **SECURITY_TYPE** specifies security type values for the **DistributorSecurityMode**, **PublisherSecurityMode**, and **SubscriberSecurityMode** properties.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_AUTHENTICATION</td>
<td>0</td>
<td>Specifies SQL Server Authentication for the connection.</td>
</tr>
<tr>
<td>NT_AUTHENTICATION</td>
<td>1</td>
<td>Specifies Windows Authentication for the connection (supported by Microsoft SQL Server™ on Microsoft Windows NT® 4.0 and Microsoft Windows® 2000 operating systems only).</td>
</tr>
</tbody>
</table>

**Remarks**

If the computer on which the Microsoft ActiveX® replication control is hosted is not running the Windows NT 4.0 or Windows 2000 operating system, **NT_AUTHENTICATION** cannot be used on the Publisher, Distributor, or Subscriber.

**See Also**

- DistributorSecurityMode Property
- SubscriberSecurityMode Property
- PublisherSecurityMode Property
Replication Programming
**STATUS_RETURN_CODE**

The **STATUS_RETURN** specifies the return code values that can be returned from the status callback functions.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Operation is successful.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>1</td>
<td>Operation is canceled.</td>
</tr>
</tbody>
</table>

**See Also**

[Status Event](#)
Replication Programming
SUBSCRIPTION_HOST

The SUBSCRIPTION_HOST specifies subscription host codes for the SubscriptionHost parameter of the AddSubscription method of the SQLMerge and SQLDistribution objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>0</td>
<td>Subscription is not registered under other hosts.</td>
</tr>
<tr>
<td>SYNC_MANAGER</td>
<td>1</td>
<td>Subscription is to be registered in Windows Synchronization Manager.</td>
</tr>
</tbody>
</table>

See Also

AddSubscription Method
Replication Programming
SUBSCRIPTION_PRIORITY_TYPE

The SUBSCRIPTION_PRIORITY_TYPE constants specify subscription priority type values for the SubscriptionPriorityType property of the SQLMerge object.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_PRIORITY</td>
<td>1</td>
<td>Subscription has an assigned priority value.</td>
</tr>
<tr>
<td>LOCAL_PRIORITY</td>
<td>2</td>
<td>Subscription uses the priority value of the Publisher.</td>
</tr>
</tbody>
</table>

See Also

SubscriptionPriorityType Property
Replication Programming
SUBSCRIPTION_TYPE

The SUBSCRIPTION_TYPE specifies subscription type values for the SubscriptionType property of the SQLMerge and SQLDistribution objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANONYMOUS</td>
<td>2</td>
<td>Anonymous subscription.</td>
</tr>
<tr>
<td>PULL</td>
<td>1</td>
<td>Pull subscription.</td>
</tr>
<tr>
<td>PUSH</td>
<td>0</td>
<td>Push subscription.</td>
</tr>
</tbody>
</table>

See Also

SubscriptionType Property
Replication Programming
SYNCHRONIZATION_TYPE

The SYNCHRONIZATION_TYPE specifies subscription synchronization type codes for the SynchronizationType property of the SQLMerge and SQLDistribution objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOMATIC</td>
<td>1</td>
<td>Initial synchronization is to be provided to the Subscriber.</td>
</tr>
<tr>
<td>NOSYNC</td>
<td>2</td>
<td>No initial synchronization is needed by the Subscriber.</td>
</tr>
</tbody>
</table>

See Also

SynchronizationType Property
Replication Programming
**VALIDATE_TYPE**

The **VALIDATE_TYPE** constants specify the type of data validation to perform on the Subscriber data at the end of the run. **VALIDATE_TYPE** is used with the **Validate** property of the **SQLMerge** object.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAST_ROWCOUNT_AND_CHECKSUM</td>
<td>4</td>
<td>Perform a fast row count and checksum validation of the Subscriber data.</td>
</tr>
<tr>
<td>FAST_ROWCOUNT_AND_BINARYCHECKSUM</td>
<td>6</td>
<td>Perform a fast row count and binary checksum validation of the Subscriber data. BINARYCHECKSUM is not supported by Microsoft® SQL Server™ 2000 Subscribers.</td>
</tr>
<tr>
<td>FAST_ROWCOUNT_ONLY</td>
<td>3</td>
<td>Perform only a fast row count validation of the Subscriber data.</td>
</tr>
<tr>
<td>NO_VALIDATION</td>
<td>0</td>
<td>Do not validate the Subscriber data (default).</td>
</tr>
<tr>
<td>ROWCOUNT_AND_CHECKSUM</td>
<td>2</td>
<td>Perform a full row count and checksum validation of the Subscriber data.</td>
</tr>
<tr>
<td>ROWCOUNT_AND_BINARYCHECKSUM</td>
<td>5</td>
<td>Perform a full row count and binary checksum validation of the Subscriber data. BINARYCHECKSUM</td>
</tr>
</tbody>
</table>
is not supported by SQL Server 2000 Subscribers.

| ROWCOUNT_ONLY | 1 | Perform only a full row count validation of the Subscriber data |

See Also

Help with Replication

Validate Property
Replication Programming
Developing Replication Merge Conflict Resolvers Through a Custom Resolver

Microsoft® SQL Server™ 2000 supports two types of user-implemented conflict resolvers:

- Custom conflict resolvers built as COM components and compiled into dynamic-link libraries (.dll) through products such as Microsoft Visual Basic® and Microsoft Visual C++®. The COM custom conflict resolver is a DLL that implements the ICustomResolver interface, methods, and properties. There are interfaces and type definitions designed specially for conflict resolution. For information about the required header file, see COM Conflict Resolver Header File.

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 computer, and for a pull subscription, it is the Subscriber computer. When using Visual C++ or Visual Basic, the name of the project becomes the name of the .DLL. The DLL name must be a unique resolver name. Run the sp_enumcustomresolvers stored procedure to ensure uniqueness. sp_enumcustomresolvers displays all the resolvers currently registered on the system.

- User-built Transact-SQL stored procedures can be used instead of a COM component. The stored procedures must implement a specific set of parameters required for a conflict resolver. For more information about the use of each field defined, see Custom Stored Procedure Conflict Resolver.

The Microsoft SQL Server 2000 CD-ROM ships with some resolver samples. For more information, see Replication Resolver Samples.

See Also

Custom Stored Procedure Conflict Resolver
Merge Replication Conflict Detection and Resolution
Other Microsoft Resolvers

Replication Architecture
Replication Programming
COM Conflict Resolver Header File

A file named sqlres.h is located in C:\Program Files\Microsoft SQL Server\80\Dev Tools\include, if the replication sample programs were installed to the default folder. The file, sqlres.h, contains several important definitions with which you should be familiar. Do no modify this file.

The custom resolver must implement the interface ICustomResolver, which is defined in this file.

```
#define INTERFACE ICustomResolver
DECLARE_INTERFACE_(ICustomResolver, Iunknown)
{
    //** IUnknown methods
    STDMETHODCALLTYPE QueryInterface (THIS_ IUnknown *ppvObject, REFIID riid, void **ppvObject) PURE;
    STDMETHODCALLTYPE AddRef (THIS) PURE;
    STDMETHODCALLTYPE Release (THIS) PURE;

    //** ICustomResolver methods
    STDMETHODCALLTYPE Reconcile (THIS_ IReplRowChange *pRowChange, DWORD dwFlags, PVOID pvReserved) PURE;
    STDMETHODCALLTYPE GetHandledStates (THIS_ DWORD *pResolverBm) PURE;
}
```

ICustomResolver inherits from IUnknown, similar to all COM classes. The IUnknown methods usually do not need to be modified from the supplied resolver, but the ICustomResolver methods Reconcile and GetHandledStates must be implemented. Reconcile is the method called for each table row that contains a conflict. GetHandledStates defines the conflict conditions that the resolver will handle.
The important parameter for **Reconcile** is a reference to an **IReplRowChange** object; **IReplRowChange** is defined in this include file. Through the method of **IReplRowChange**, the resolver determines the columns in conflict, examines the conflicting data, and then copies the appropriate data to the result row.

Other definitions in the file include **IConnectionInfo**, which is used when a resolver needs to access a stored procedure, and **ITranDataChange**, which is used in a transactional resolver. Only the **Get<xxx>** methods in **IConnectionInfo** are accessible to user-implemented resolvers.

Several enumerations are defined in **sqlres.h**. Symbols from these enumerations should be used wherever possible instead of using hard-coded constants.

<table>
<thead>
<tr>
<th>Enumeration Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPOLE_CHANGE_TYPE</td>
<td>Codes for the database operation (insert, update, delete), whether there is a conflict, and whether column tracking is active. Many symbolic definitions of aggregates of these change types are also available.</td>
</tr>
<tr>
<td>REPOLE_CONFLICT_TYPE</td>
<td>Codes for the database operation and whether the failure occurred at upload or download.</td>
</tr>
<tr>
<td>REPOLE_COLSTATUS_TYPE</td>
<td>Codes for the conflict status of an individual column.</td>
</tr>
<tr>
<td>REPOLE_PRIORITY_TYPE</td>
<td>Codes for what have higher priority (source, destination, neither source nor destination).</td>
</tr>
</tbody>
</table>
Replication Programming
Programming Replication from Heterogeneous Data Sources

Microsoft® SQL Server™ enables third-party products to become Publishers within the SQL Server replication framework. The Replication Distributor Interface allows replication from heterogeneous databases that provide 32-bit OLE DB drivers to Subscribers running SQL Server 2000. Heterogeneous data sources include:

- Oracle databases
- DB2 databases
- Microsoft Access databases
- Other databases that comply with SQL Server ODBC or OLE DB Subscriber requirements.

When integrated, the Replication Distributor Interface exposes the SQL Distributor, and allows heterogeneous data sources to store meta data and replicated transactions in the SQL Server Distributor database. The Replication Distributor Interface is an OLE DB service provider that allows users to store replicated SQL statements, scripts, and .bcp files in the Distributor store-and-forward database. The Replication Distributor Interface is based on the OLE DB connection model and supports a subset of the DataSource, Session, and Error objects. An additional Distribution object is added to the Session object and is used to store transactions marked for replication in a SQL Server distribution database.

**Note** The Replication Distributor Interface is a special purpose OLE DB service that is used only to distribute replicated SQL Server transactions. It does not support the minimal set of interfaces necessary to be considered a standard OLE DB data provider.

The Replication Distributor Interface cannot be used with replication types that
need updates to be made at the Subscriber. The only types of replication that can be used with the Replication Distributor Interface are snapshot replication and transactional replication that has read-only Subscribers. Merge replication, and transactional replication with immediate updating, queued updating, or immediate updating with queued updating as failover are not available from heterogeneous Publishers to SQL Server Subscribers.

Here are the steps to using the Replication Distributor Interface.

1. This step uses Microsoft Visual Basic® or Microsoft Visual C++® and the replication SQL-DMO objects, makes calls for setting up the publication, articles, and subscription information. These calls differ from the typical SQL-DMO setup calls because they are made on the distribution server and not at the Publisher. A third-party replication tool can also be used to implement this first step.

2. This step takes the place of the Snapshot Agent and Log Reader Agent. The Replication Distributor Interface is used to store the replication transactions on a server that is currently acting as the Distributor, which will then be distributed by the Distribution Agent. The following C++ code is a sample of an object that is used to place commands into the distribution database.

   // Instantiate a data source object for the SQL Server Publisher // provider.
   hr = CoCreateInstance(CLSID_SQLDistribution, NULL, CLSCTX_INPROC_SERVER, IID_IDBInitialize, (void**)&pIDBInit);

Using the Replication Distributor Interface leaves the responsibility of data modification detection to the developers because the Log Reader Agent is not available. The use of the monitoring and troubleshooting tools, alerts, and notifications are still available in SQL Server Enterprise Manager. The Log Reader Agent can be activated by using Microsoft Host Integration Server 2000 and its Distribution Store Interface (iDistributionStore). For more information, see the Host Integration Server 2000 documentation.

The Microsoft SQL Server CD-ROM ships with some Replication Distributor Interface samples. For more information, see Replication Distributor Interface.
Samples.

When deploying an application using the Replication Distributor Interface independently of SQL Server 2000, you must include additional files in the installation kit you use to distribute your application. If you will be deploying your application to a computer where SQL Server 2000 will also be installed, these files will already be present.
Replication Programming
SQL-DMO Replication Objects

SQL-DMO provides a set programming interface for administering and monitoring Microsoft® SQL Server™ replication, including the ability to administer replication from heterogeneous computers. Here are the SQL-DMO objects used in replication:

- **Distributor** object
- **DistributionDatabase** object
- **DistributionPublisher** object
- **DistributionPublication** object
- **DistributionSubscription** object
- **DistributionArticle** object
- **RegisteredSubscriber** object
- **Replication** object

The SQL-DMO replication objects are a subset of the SQL-DMO COM object model. For information about how these object fit into the larger replication object model, see [Developing SQL-DMO Applications](#).

**See Also**

- [Distributor Object](#)
- [DistributionArticle Object](#)
DistributionDatabase Object
DistributionPublication Object
DistributionPublisher Object
DistributionSubscription Object
RegisteredSubscriber Object
Replication Object
Replication Programming
Replication Distributor Interface Reference

The Replication Distributor Interface can be used to enable Microsoft® SQL Server™ 2000 replication services on heterogeneous databases. The Replication Distributor Interface is based on the OLE DB connection model. The objects are available using Microsoft Visual C++® and the OLE library.

The Replication Distributor Interface exposes a Distribution object that is generated from a Session object. The Distribution object is used to store replicated transactions in the distribution database on the SQL Server Distributor, and is used to log history and error information.

The Replication Distributor Interface is implemented using these files:

- Repldist.dll

- Repldist.h

The Replication Distributor Interface objects, methods, properties, events, and structures can be found in the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Distributor Interface Objects</td>
<td>Description of Replication Distributor Interface Distribution objects.</td>
</tr>
<tr>
<td>Replication Distributor Interface Properties</td>
<td>Description of Replication Distributor Interface properties.</td>
</tr>
<tr>
<td>Replication Distributor Interface Methods</td>
<td>Description of Replication Distributor Interface methods.</td>
</tr>
<tr>
<td>Replication Distributor Interface Structures</td>
<td>Description of Replication Distributor Interface structures.</td>
</tr>
</tbody>
</table>

See Also

Replication Distributor Interface Samples
Replication Programming
## Replication Distributor Interface Objects

The Replication Distributor Interface exposes these objects.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistributionLog Object</td>
<td>Stores history and error information about the Replication Distributor Interface.</td>
</tr>
<tr>
<td>DistributionStore Object</td>
<td>Stores transactions in a Distributor.</td>
</tr>
</tbody>
</table>
Replication Programming
DistributionLog Object

The DistributionLog object stores history and error information about the Replication Distributor Interface. This information is used to monitor replication.

Methods

AddLog Method
Replication Programming
DistributionStore Object

The DistributionStore object stores transactions in a Distributor. This is a custom interface supported only by the Distributor.

Methods

Abort Method
AddTransactionCommands Method
Commit Method
GetLastTransaction Method
StartTransaction Method
Replication Programming
Replication Distributor Interface Properties

This section defines the properties of the Replication Distributor Interface.
Replication Programming
DBPROP_APPLICATION_NAME Property

The DBPROP_APPLICATION_NAME property specifies the name of the application.

**Applies To**

[DistributionStore Object](#)

**Syntax**

```
object.DBPROP_APPLICATION_NAME
```

**Data Type**

VT_BSTR

**Modifiable**

Read/write
Replication Programming
**DBPROP_APPLICATION_TYPE Property**

The **DBPROP_APPLICATION_TYPE** property specifies the Publisher application type. The application type can be either snapshot or transactional (incremental updates).

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_APPLICATION_TYPE`

**Data Type**

VT_I1

**Modifiable**

Read/write
Replication Programming
DBPROP_AUTH_PASSWORD Property

The **DBPROP_AUTH_PASSWORD** property specifies the password used when connecting to the data source.

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_AUTH_PASSWORD`

**Data Type**

`VT_BSTR`

**Modifiable**

Read/write
Replication Programming
**DBPROP_AUTH_USERID Property**

The `DBPROP_AUTH_USERID` property specifies the user ID used when connecting to the data source.

**Applies To**

*DistributionStore Object*

**Syntax**

`object.DBPROP_AUTH_USERID`

**Data Type**

`VT_BSTR`

**Modifiable**

Read/write
Replication Programming
**DBPROP_DBMSNAME Property**

The **DBPROP_DBMSNAME** property specifies the name of the product accessed by the provider.

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_DBMSNAME`

**Data Type**

VT_BSTR

**Modifiable**

Read-only
Replication Programming
DBPROP_DBMSVER Property

The DBPROP_DBMSVER property specifies the version of the product accessed by the provider.

Applies To

DistributionStore Object

Syntax

object.DBPROP_DBMSVER

Data Type

VT_BSTR

Modifiable

Read-only
Replication Programming
DBPROP_INIT_DATASOURCE Property

The DBPROP_INIT_DATASOURCE property specifies the name of the distribution database to which to connect.

Applies To

DistributionStore Object

Syntax

object.DBPROP_INIT_DATASOURCE

Data Type

VT_BSTR

Modifiable

Read/write
Replication Programming
**DBPROP_INIT_LOCATION Property**

The **DBPROP_INIT_LOCATION** property specifies the location of the Distributor to which to connect (typically, the server name).

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_INIT_LOCATION`

**Data Type**

`VT_BSTR`

**Modifiable**

Read/write
Replication Programming
**DBPROP_INIT_PUBLISHERDATASOURCE**

**Property**

The **DBPROP_INIT_PUBLISHERDATASOURCE** property specifies the name of the Publisher database on whose behalf the connection is made.

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_INIT_PUBLISHERDATASOURCE`

**Data Type**

`VT_BSTR`

**Modifiable**

Read/write
Replication Programming
DBPROP_INIT_PUBLISHER_NAME Property

The DBPROP_INIT_PUBLISHER_NAME property specifies the name of the Publisher on whose behalf the connection is made.

Applies To

DistributionStore Object

Syntax

object.DBPROP_INIT_PUBLISHERNAME

Data Type

VT_BSTR

Modifiable

Read/write
Replication Programming
**DBPROP_INIT_TIMEOUT Property**

The `DBPROP_INIT_TIMEOUT` property specifies the connection time-out.

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_INIT_TIMEOUT`

**Data Type**

VT_I4

**Modifiable**

Read/write
Replication Programming
DBPROP_INIT_XACT_SEQNO_SIZE Property

The `DBPROP_INIT_XACT_SEQNO_SIZE` property specifies the size of the transaction sequence number.

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_INIT_XACT_SEQNO_SIZE`

**Data Type**

VT_I1

**Modifiable**

Read/write
Replication Programming
**DBPROP_INIT_XACTID_SIZE Property**

The **DBPROP_INIT_XACTID_SIZE** property specifies the size of the transaction ID.

**Applies To**

[DistributionStore Object](#)

**Syntax**

`object.DBPROP_INIT_XACTID_SIZE`

**Data Type**

VT_I1

**Modifiable**

Read/write
Replication Programming
DBPROP_PUBLICATION_NAME Property

The **DBPROP_PUBLICATION_NAME** property specifies the name of the publication serviced by the application.

**Applies To**

[DistributionStore Object](#)

**Syntax**

```
object.DBPROP_PUBLICATION_NAME
```

**Data Type**

VT_BSTR

**Modifiable**

Read/write
Replication Programming


**Replication Distributor Interface Methods**

This section defines the methods of the Replication Distributor Interface.
Replication Programming
Abort Method

The **Abort** method rolls back the storage of transaction commands.

**Applies To**

DistributionStore Object

**Syntax**

HRESULT Abort( );

**Remarks**

The method returns S_OK if it succeeds and DB_E_ERROROCCURRED if it fails.
Replication Programming
AddLog Method

The **AddLog** method adds history and error information to a distribution store.

**Applies To**

[DistributionLog Object](#)

**Syntax**

```
HRESULT AddLog(
    DWORD dwStatusID,
    LPSTR szComment,
    BYTE* pXactSeqno,
    ULONG ulTransactions,
    ULONG ulCommands,
    ULONG cErrorDescs,
    const DISTERRORDESC rgErrorDescs[]);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dwStatusID</code></td>
<td>Status of the log message:</td>
</tr>
<tr>
<td></td>
<td>• 1 = STARTUP</td>
</tr>
<tr>
<td></td>
<td>• 2 = SUCCESS</td>
</tr>
<tr>
<td></td>
<td>• 3 = INPROGRESS</td>
</tr>
<tr>
<td></td>
<td>• 4 = RETRY</td>
</tr>
<tr>
<td></td>
<td>• 5 = FAILURE</td>
</tr>
<tr>
<td><code>szComment</code></td>
<td>Log message text</td>
</tr>
<tr>
<td><code>XactSeqno</code></td>
<td>Transaction sequence number</td>
</tr>
</tbody>
</table>
### Remarks

The method returns S_OK if it succeeds and DB_E_ERROROCCURRED if it fails.

### See Also

DISTERRORDESC Structure
Replication Programming
AddTransactionCommands Method

The AddTransactionCommands method stores a group of replicated transaction commands. If the StartTransaction method is not called before this method, AddTransactionsCommands uses an implicit transaction.

Applies To

DistributionStore Object

Syntax

HRESULT AddTransactionCommands(
 ULONG cCommandDescs,
 const DISTCOMMANDDESC rgCommandDescs []);

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cCommandDescs</td>
<td>Number of DISTCOMMANDDESC structures in the rgCommandDescs array</td>
</tr>
<tr>
<td>rgCommandDescs</td>
<td>Array of DISTCOMMANDDESC structures that describe the replicated command</td>
</tr>
</tbody>
</table>

Remarks

The method returns S_OK if it succeeds and DB_E_ERROROCCURRED if it fails.

See Also

DISTCOMMANDDESC Structure
Replication Programming
Commit Method

The **Commit** method commits the storage of the transaction commands.

**Applies To**

[DistributionStore Object](#)

**Syntax**

HRESULT Commit( );

**Remarks**

The method returns S_OK if it succeeds and DB_E_ERROROCCURRED if it fails.
Replication Programming
GetLastTransaction Method

The **GetLastTransaction** method retrieves information about the last stored transaction.

**Applies To**

[DistributionStore Object](#)

**Syntax**

```c
HRESULT GetLastTransaction(
    BYTE* pXactID,
    BYTE* pXactSeqno );
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XactID</td>
<td>Transaction identifier that uniquely identifies the transaction. It can be up to 255 bytes. The default is NULL.</td>
</tr>
<tr>
<td>XactSeqno</td>
<td>Transaction sequence number that identifies the sequence in which transactions are committed (in big-endian format). Transactions with lower sequence numbers are committed first.</td>
</tr>
</tbody>
</table>

**Remarks**

The method returns S_OK if it succeeds and DB_E_ERROROCCURRED if it fails.
Replication Programming
StartTransaction Method

The StartTransaction method begins a transaction.

Applies To

DistributionStore Object

Syntax

HRESULT StartTransaction();

Remarks

The method returns S_OK if it succeeds and DB_E_ERROROCCURRED if it fails.
Replication Programming
Replication Distributor Interface Structures

This section defines the structures exposed by the Replication Distributor Interface.
Replication Programming
DISTCOMMANDDESC Structure

This is the definition of the DISTCOMMANDDESC structure.

Syntax

typedef struct tagDISTCOMMANDDESC{
    INT PublicationID;
    INT ArticleID;
    INT CommandID;
    DISTCMDTYPE CommandType;
    BOOL fPartialCommand;
    LPSTR szCommand;
    BYTE* pXactID;
    BYTE* pXactSeqno;
    LPSTR szOriginator;
    LPSTR szOriginatorDB;
} DISTCOMMANDDESC;

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PublicationID</td>
<td>Publication ID.</td>
</tr>
<tr>
<td>ArticleID</td>
<td>Article ID.</td>
</tr>
<tr>
<td>CommandID</td>
<td>Uniquely identifies commands within a transaction. Each command added to a transaction should have a unique, monotonically increasing command ID.</td>
</tr>
<tr>
<td>CommandType</td>
<td>Identifies the type of the command. The Microsoft® SQL Server™ Distribution Agent can handle the following command types:</td>
</tr>
<tr>
<td></td>
<td>• SQL_CMDTYPE_SQL = Transact-SQL command.</td>
</tr>
<tr>
<td></td>
<td>• SQL_CMDTYPE_SCRIPT = File path to a Transact-SQL script file.</td>
</tr>
</tbody>
</table>
- **SQL.CmdType_Native_BCP** = File path to a `.bcp` file in native format.

- **SQL.CmdType_Char_BCP** = File path to a `.bcp` file in character format.

- **SQL.CmdType_WorkingDir** = File path to the directory in which snapshot files are stored. Used by the Distribution Cleanup Agent.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fPartialCommand</td>
<td>Determines whether the command wraps more than one row.</td>
</tr>
<tr>
<td>szCommand</td>
<td>Command text.</td>
</tr>
<tr>
<td>pXactID</td>
<td>Pointer to the transaction ID.</td>
</tr>
<tr>
<td>pXactSeqno</td>
<td>Pointer to the transaction sequence number.</td>
</tr>
<tr>
<td>szOriginator</td>
<td>Name of the originating server.</td>
</tr>
<tr>
<td>szOriginatorDB</td>
<td>Name of the originating database.</td>
</tr>
</tbody>
</table>
Replication Programming

**CommandType Text Formats**

The *CommandType* member of the DISTCOMMANDDESC structure requires specific text formatting when specifying these items

- Transact-SQL data types
- Working directories
- Schema files
- *bcp* files

**Transact-SQL Data Types**

When using DISTCMDTYPE_SQL, data types in Transact-SQL statements have these formats.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Format</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datetime</td>
<td><code>{ts 'yyyy’mm’dd hh:mm:ss.[mmm]'}</code></td>
<td>Milliseconds are optional.</td>
</tr>
<tr>
<td>Datetime</td>
<td><code>{ts 'yyyy’mm’dd hh:mm:ss.[mmm]'}</code></td>
<td>Milliseconds are optional.</td>
</tr>
<tr>
<td>SmallDATETIME</td>
<td><code>{ts 'yyyy’mm’dd hh:mm:ss.[mmm]'}</code></td>
<td>Milliseconds are optional.</td>
</tr>
<tr>
<td>Binary</td>
<td><code>{b 'data'}</code></td>
<td>Where data is one or more characters within the range: [0-9a-f]. It should not contain a leading 0x.</td>
</tr>
<tr>
<td>Binary</td>
<td><code>{b 'data'}</code></td>
<td>Where data is one or more characters within the range: [0-9a-f]. It should not contain a leading 0x.</td>
</tr>
<tr>
<td>Long Binary</td>
<td><code>{lb 'data'}</code></td>
<td>Where data is one or more characters within the range: [0-9a-f]. It should not include a leading 0x.</td>
</tr>
<tr>
<td>Character</td>
<td>'data'</td>
<td>Where data is any sequence of characters. Single quotation marks within the data portion must be expanded to two adjacent single quotation marks.</td>
</tr>
</tbody>
</table>
Long Character text | {lc 'data'} | Where data is any sequence of characters. Single quotation marks within the data portion must be expanded to two adjacent single quotation marks.

**Working Directories**

When using DISTCMDTYPE_WORKINGDIR, include escape characters (\) in the file paths.

```c
const char szDir[] = "\Program Files\Microsoft SQL Server\mssql\r
DISTCOMMANDDESC aCommand[60];
INT NumCommands = 0;
DistByteArray XactId;
DistByteArray XactSeqno;
CHAR szWrkDir[_MAX_PATH];

// Get the current working directory.
sprintf(szWrkDir, "%c", (char)(_getdrive() + 'A' - 1));
strcat(szWrkDir, szDir);

// Set working directory.
NumCommands++;
aCommand[i].PublicationId = 1;
aCommand[i].ArticleId = 1;
aCommand[i].CommandId = NumCommands;
aCommand[i].CommandType = DISTCMDTYPE_SQL_WORKINGDIR;
aCommand[i].fPartialCommand = FALSE;
aCommand[i].pXactId = (BYTE *)&XactId;
aCommand[i].pXactSeqno = (BYTE *)&XactSeqno;
aCommand[i].szOriginator = NULL;
aCommand[i].szOriginatorDB = NULL;
```
aCommand[i].szCommand = (LPSTR)szWrkDir;
i++;}

**Schema Files**

When using DISTCMDTYPE_SCRIPT, include escape characters (\) in file paths.

const char szDir[] = "\Program Files\Microsoft SQL Server\mssql\r
const char szScriptCmd[] = "%ssamptab.sch";
DISTCOMMANDDESC aCommand[60];
char pszCmdBuf[60][255];
INT NumCommands = 0;
DistByteArray XactId;
DistByteArray XactSeqno;

// Get the current working directory.
sprintf(szWrkDir, "%c", (char)(_getdrive() + 'A' - 1));
strcat(szWrkDir, szDir);

// Execute script - table schema.
NumCommands++;
aCommand[i].PublicationId = 1;
aCommand[i].ArticleId = 1;
aCommand[i].CommandId = NumCommands;
aCommand[i].CommandType = DISTCMDTYPE_SCRIPT;
aCommand[i].fPartialCommand = FALSE;
aCommand[i].pXactId = (BYTE *)&XactId;
aCommand[i].pXactSeqno = (BYTE *)&XactSeqno;
aCommand[i].szOriginator = NULL;
aCommand[i].szOriginatorDB = NULL;

sprintf(pszCmdBuf[i], szScriptCmd, szWrkDir);
aCommand[i].szCommand = pszCmdBuf[i];
i++;

Use this format in .sch files:

```
SET QUOTED IDENTIFIER ON
GO
SET ANSI_PADDING OFF
GO
CREATE TABLE [Samptbl1] (C1 INT, C2 VARCHAR(20))
GO
```

**bcp Files**

When using DISTCMDTYPE_CHAR_BCP or DISTCMDTYPE_NATIVE_BCP, you can use these switches with the `sync` command.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t</td>
<td>Destination table.</td>
</tr>
<tr>
<td>-o</td>
<td>Destination owner.</td>
</tr>
<tr>
<td>-d</td>
<td>Data file.</td>
</tr>
<tr>
<td>-f</td>
<td>Field delimiter. Default field delimiter: <code>\n&lt;x$3&gt;\n</code></td>
</tr>
<tr>
<td>-r</td>
<td>Row delimiter. Default row delimiter: <code>\n&lt;,@g&gt;\n</code></td>
</tr>
<tr>
<td>-u</td>
<td>Unicode. This switch applies only when using DISTCMDTYPE_NATIVE_BCP type bcp files.</td>
</tr>
<tr>
<td>-m</td>
<td>Denotes that the file is a character bcp file with a Unicode marker at the beginning of the file.</td>
</tr>
</tbody>
</table>

Include escape characters (`\`) in file paths. Begin switch arguments with quotation marks (`"`) and end the arguments with a backslash and quotation marks (`"`).

```c
const char szDir[] = "\:\Program Files\Microsoft SQL Server\mssql\r
const char szBCPCmd[] = "sync -t"SampTable1" -d"%ssamptab.bcp'`
```

DISTCOMMANDDESC aCommand[60];
char pszCmdBuf[60][255];
INT NumCommands = 0;
DistByteArray XactId;
DistByteArray XactSeqno;

// Get the current working directory.
sprintf(szWrkDir, "%c", (char)(_getdrive() + 'A' - 1));
strcat(szWrkDir, szDir);

// Import data - char bcp.
NumCommands++;
aCommand[i].PublicationId = 1;
aCommand[i].ArticleId = 1;
aCommand[i].CommandId = NumCommands;
aCommand[i].CommandType = DISTCMDTYPE_CHAR_BCP;
aCommand[i].fPartialCommand = FALSE;
aCommand[i].pXactId = (BYTE *)&XactId;
aCommand[i].pXactSeqno = (BYTE *)&XactSeqno;
aCommand[i].szOriginator = NULL;
aCommand[i].szOriginatorDB = NULL;

sprintf(pszCmdBuf[i], szBCPCmd, szWrkDir);
aCommand[i].szCommand = pszCmdBuf[i];
i++;
Replication Programming
**DISTERRORDESC Structure**

This is the definition of the DISTERRORDESC structure.

**Syntax**

typedef struct tagDISTERRORDESC{
  INT   SourceType;
  CHAR szSourceName[31];
  CHAR szErrorCode[31];
  BSTR  bstrErrorText;
} DISTERRORDESC;

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SourceType</strong></td>
<td>Identifies the type of the command. Microsoft® SQL Server™ 2000 supports the following command types:</td>
</tr>
<tr>
<td></td>
<td>• APPLICATION</td>
</tr>
<tr>
<td></td>
<td>• DISTOLE</td>
</tr>
<tr>
<td></td>
<td>• OS</td>
</tr>
<tr>
<td><strong>szSourceName</strong></td>
<td>Name of the component responsible for the error (for example, ODBC).</td>
</tr>
<tr>
<td><strong>szErrorCode</strong></td>
<td>Error code string of the source.</td>
</tr>
<tr>
<td><strong>bstrErrorText</strong></td>
<td>Error text.</td>
</tr>
</tbody>
</table>
Replication Programming
Replication Programming Samples

The following samples illustrate Microsoft® SQL Server™ 2000 replication application development in various environments and languages, including Transact-SQL, Microsoft Visual C++®, and Microsoft Visual Basic®. The samples and associated headers and libraries are required for successful execution of several sample applications.

To install the samples and related headers and libraries using the SQL Server Installation Wizard

1. On the Setup Type page, select Custom.

2. On the Select Components page, under Components, select Code Samples.

3. In the Select Sub-Components dialog box, select Replication.


5. In the Select Sub-Components dialog box, select Headers and Libraries.

After installation is complete, if you have accepted the default installation location, the path C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\sqlrepl will be on your computer. The sqlrepl directory contains an executable file, unzip_sqlrepl.exe, which expands the samples into useable files. Double-click unzip_sqlrepl.exe, and you will be prompted to enter that path to the folder where you want the samples stored. The samples assume that the default is selected, and samples are not installed to an alternate location. The expansion adds several subdirectories to sqlrepl.

The Visual C++ samples were tested with Visual C++ version 6.0, Service Pack 3. The Visual Basic samples were tested with Visual Basic version 6.0, Service
Pack 3. The samples have been run on Microsoft Windows NT® version 4.0, Service Pack 6, and on Microsoft Windows® 2000 operating systems. They have not been compiled or tested on any other hardware platform supported by any other compiler.

For Visual C++ samples to compile, header and libraries file paths must be set properly to obtain the required replication files. After installation, set your project options for **include** files to C:\Microsoft SQL Server\80\Tools\Devtools\Include, and set your options for the **library** files to C:\Microsoft SQL Server\80\Tools\Devtools\Lib.
Replication Programming
Replication Syntax Conventions

Replication programming samples use the following conventions to distinguish elements of text.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Constants and enumerated data types.</td>
</tr>
<tr>
<td>courier new</td>
<td>Sample command lines and program code.</td>
</tr>
<tr>
<td>italic</td>
<td>Information that the user or the application must provide.</td>
</tr>
<tr>
<td>bold</td>
<td>Replication component objects; object events, methods or properties; data types; and other syntax that must be typed exactly as shown.</td>
</tr>
</tbody>
</table>
Replication Programming
## Replication ActiveX Control Samples

Microsoft® SQL Server™ 2000 comes with the following sample applications to help you use Microsoft ActiveX® replication controls in your application.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Using SQL Merge and SQL Distribution Controls in a Custom Visual Basic Application</strong></td>
<td>Microsoft Visual Basic® sample that demonstrates how to include the SQL Distribution and SQL Merge controls in a custom application.</td>
</tr>
<tr>
<td><strong>Using SQL Merge and SQL Distribution Controls in a Custom Visual C++ Application</strong></td>
<td>Microsoft Visual C++® sample that demonstrates how to include the SQL Distribution and SQL Merge controls in a custom application.</td>
</tr>
<tr>
<td><strong>Using SQL Merge and SQL Distribution Controls in a Custom Web Application</strong></td>
<td>HTML sample that demonstrates how to include the SQL Distribution and SQL Merge controls in a custom application.</td>
</tr>
<tr>
<td><strong>Creating a Transformable Subscription Using Visual Basic</strong></td>
<td>Visual Basic sample that creates and saves a Data Transformation Services (DTS) package to DTS LocalPackages. The package contains Data Driven Query code to synchronize a subscription, and ActiveX code to transform the data before it is entered into the subscription database.</td>
</tr>
</tbody>
</table>
Replication Programming
Using SQL Merge and SQL Distribution Controls in a Custom Visual Basic Application

The programs in the replctrl folder are samples of how to include the SQL Merge and SQL Distribution controls in a custom application. This sample is located in C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\sqlrepl\replctrl.

To run the sample programs

1. On the computer that will be a Publisher with a local Distributor, verify that the SQL Server Agent is running. If it is not running, start it.

2. Open SQL Query Analyzer, open \Samples\sqlrepl\replctrl\instsamp.sql, and then run instsamp.sql. This SQL script configures the computer for publishing and distribution, enables the computer as a Subscriber, adds a distribution database with the name distributor, creates a transactional publication named SampleTransactionalPublication, and creates a merge publication named SampleMergePublication.

3. The instsamp.sql script creates pull and push subscriptions for each publication and creates and configures a database called Northwind_replica as the subscription database. Any warnings from the script regarding tables that have been created with a maximum row size that exceeds the maximum number of bytes per row can be ignored. The computer is now configured as a Publisher with a local Distributor and enabled as a Subscriber.

   The Northwind publication database will have two publications: SampleTransactionalPublication and SampleMergePublication. The instance of SQL Server contains a new subscription database, Northwind_replica.

4. After the instsamp.sql script has completed successfully, in SQL
Server Enterprise Manager, expand Replication Monitor, expand the Agents folder, and then click the Snapshot Agents folder. In the right pane, for each agent listed, right-click, and then click Start Agent. This starts the Snapshot Agents for each publication and creates a snapshot for each publication.

5. Start Visual Basic, and then open \Samples\sqlrepl\replctrl\VB\replsamp.vbp.

6. On the Project menu, click References, and then in the Available References list, verify that the following controls are selected:
   - Microsoft SQL Distribution Control 8.0
   - Microsoft SQL Merge Control 8.0
   - Microsoft SQL Snapshot Control 8.0
   - Microsoft SQL Replication Errors 8.0

7. On the File menu, click Make ReplSamp.exe, and then save the executable to a directory.

8. Run replsamp.exe. The following options will be displayed:

   **Generate Snapshot for Transactional Publication**
   Creates a snapshot for the Northwind transactional publication. The snapshot activity can be monitored interactively in SQL Server Enterprise Manager using Replication Monitor and the Agents folder.

   **Generate Snapshot for Merge Publication**
   Creates a snapshot for the Northwind merge publication. The snapshot activity can be monitored interactively in SQL Server Enterprise Manager using Replication Monitor and the Agents folder.
Update Transactional Subscription Tables

Runs the Distribution Agent and applies the snapshot and schema at the subscription database, Northwind_replica. After the SQL Replication Sample dialog box shows that the task has completed, the Northwind_replica subscription database shows the new tables. The distribution activity can be monitored interactively in SQL Server Enterprise Manager using Replication Monitor and the Agents folder.

Update Merge Subscription Tables

Runs the Merge Agent and applies the snapshot data and schema at the merge subscription database, Northwind_replica. After the SQL Replication Sample dialog box shows that the task has completed, the Northwind_replica subscription database shows the new tables. The merge activity can be monitored interactively in SQL Server Enterprise Manager using Replication Monitor and the Agents folder.
Replication Programming
Using SQL Merge and SQL Distribution Controls in a Custom Visual C++ Application

The programs in the replctrl folder are samples of how to include the SQL Merge control and the SQL Distribution control in a custom application. This sample is located in C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\sqlrepl\replctrl.

To run the sample programs

1. On the computer that will be a Publisher with a local Distributor, verify that the SQL Server Agent is running. If it is not running, start it.

2. Open SQL Query Analyzer, open \Samples\sqlrepl\replctrl\instsamp.sql, and then run instsamp.sql. This SQL script configures the computer for publishing and distribution, enables the computer as a Subscriber, adds a distribution database with the name distributor, creates a transactional publication named SampleTransactionalPublication, and creates a merge publication named SampleMergePublication.

3. The instsamp.sql script creates pull and push subscriptions for each publication and creates and configures a database called Northwind_replica as the subscription database. Any warnings from the script regarding tables that have been created with a maximum row size that exceeds the maximum number of bytes per row can be ignored. The computer is now configured as a Publisher with a local Distributor and is enabled as a Subscriber. The Northwind database will have two publications: SampleTransactionalPublication and SampleMergePublication. The instance of SQL Server contains a new subscription database, Northwind_replica.

4. After the instsamp.sql script has completed successfully, in SQL
Server Enterprise Manager, expand **Replication Monitor**, expand the **Agents** folder, and then click the **Snapshot Agents** folder. In the right pane, for each agent listed, right-click, and then click **Start Agent**. This starts the Snapshot Agent and creates a snapshot for each publication.

To use the Microsoft Visual C++® sample code that demonstrates how to include the SQL Distribution control and the SQL Merge control in a custom application, the two programs must first be built into executables.

**To build the distribution sample executable using Visual C++**

1. Open Visual C++. On the main menu, click **File**, click **Open Workspace**, navigate to the C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp directory, and then open distsamp.dsw.

2. On the **Tools** menu, click **Options**, and then on the **Directories** tab, in the **Show directories for** drop-down list, select **Include files**. Add the path C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Include. This path assumes the samples were installed to the default directory. If this path does not exist, navigate to the path where the include files were installed.

3. On the **Directories** tab, in the **Show directories for** drop-down list, click **Library files**, and then add the path C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Lib.

   This path assumes the samples were installed to the default directory. If this path does not exist, navigate to the path where the Lib files were installed.

4. On the **Build** menu, click **Build distsamp.exe**. The default location of the resulting distsamp.exe will be in C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp\Debug or C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp\Release, depending on the build configuration.
To build the distribution sample executable using a batch command

1. Open a command prompt window.

2. Navigate to C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp.

3. Run the batch file **buliddst.cmd** with the following parameters:

   buliddst.cmd [x86] [debug|retail] [clean]

   • Enter buliddst.cmd /? for help.

To run the sample, run the executable from its location using a command prompt window.

The **distsamp.exe** sample activates the Distribution Agent and moves the data from the snapshot into the tables defined as articles for the transactional subscription in the database **Northwind_replica**.

To build the merge sample executable using Microsoft Visual C++.

1. Open Visual C++. On the main menu, click **File**, click **Open Workspace**, and then navigate to the C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp directory and open mergsamp.dsw.

2. On the **Tools** menu, click **Options**, and then on the **Directories** tab, in the **Show directories for** drop-down list, click **Include files**. Add the path C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Include.

   This path assumes the samples were installed to the default directory. If this path does not exist, navigate to the path where the include files were installed.

3. On the **Directories** tab, in the **Show directories for** drop-down list, click **Library files**, and then add the path C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Lib.
This path assumes the samples were installed to the default directory. If this path does not exist, navigate to the path where the Lib files were installed.

4. On the **Build** menu, click **Build mergsamp.exe**. The default location of the resulting mergsamp.exe will be in C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp\Debug or C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp\Release, depending on the build configuration.

**To build the merge sample executable using a batch command**

1. Open a command prompt window.

2. Navigate to C:\Program Files Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\replctrl\cpp.

3. Run the batch file **buildmrg.cmd** with the following parameters:
   
   buildmrg.cmd [x86] [debug|retail] [clean]

   Enter buildmrg.cmd /? for help.

To run the sample, run the executable from its location using a command prompt window.

The **mergsamp.exe** sample activates the Merge Agent and moves data from the snapshot into the tables defined as articles for the merge subscription in the **Northwind_replica_html** database.
Replication Programming
Using SQL Merge and SQL Distribution Controls in a Web Application

The programs in the replctrl folder are samples of how to include the SQL Merge control and the SQL Distribution control in a custom application. This sample is located in C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\sqlrepl\replctrl.

To run the sample programs

1. On the computer that will be a Publisher with a local Distributor, verify that the SQL Server Agent is running. If it is not running, start it.

2. Open SQL Query Analyzer, open \Samples\sqlrepl\replctrl\instsamp.sql, and then run instsamp.sql. This SQL script configures the computer for publishing and distribution, enables the computer as a Subscriber, adds a distribution database with the name distributor, creates a transactional publication named SampleTransactionalPublication and a merge publication named SampleMergePublication. instsamp.sql creates a Snapshot Agent for both publications.

3. The instsamp.sql creates pull and push subscriptions for each publication and creates and configures a database called Northwind_replica as the subscription database. Any warnings from the script regarding tables that have been created with a maximum row size that exceeds the maximum number of bytes per row can be ignored.

The Northwind database will have two publications: SampleTransactionalPublication and SampleMergePublication. The SQL Server contains a new subscription database, Northwind_replica.

4. After the instsamp.sql script has completed successfully, in SQL
Server Enterprise Manager, expand **Replication Monitor**, expand the **Agents** folder, and then click the **Snapshot Agents** folder. In the right pane, for each agent listed, right-click, and then click **Start Agent**. This starts the Snapshot Agent and creates a snapshot for each publication.

**To modify the HTML sample program**

1. Open the `\Samples\sqlrep\replctrl\html\replsamp.htm` file in Notepad or other HTML editor. There are comments in the file noting that the local computer name needs to be specified in the line of code following the comment.

2. Save and the file.

**To execute the HTML sample program**

- Using Microsoft Internet Explorer or another Internet browser, open the `replsamp.htm` file. These options are available:

**Synchronize Transactional Subscription**

Runs the Distribution Agent and applies the snapshot data and schema at the subscription database, **Northwind_replica**. After the HTML page has completed, the snapshot will be applied, and the **Northwind_replica** database will show the new tables, with each table corresponding to an article from the publication. The distribution activity can be monitored interactively in SQL Server Enterprise Manager using **Replication Monitor** and the **Agents** folder.

**Synchronize Merge Subscription**

Runs the Merge Agent and applies the snapshot data and schema at the subscription database, **Northwind_replica**. After the HTML Page has completed, the snapshot will be replicated, and the **Northwind_replica** database will show the new tables, with each table corresponding to an article from the publication. The merge activity can be monitored interactively in SQL Server Enterprise Manager using **Replication Monitor**
and theAgents folder.
Replication Programming
Creating a Transformable Subscription Using Visual Basic

Advanced users familiar with the Data Transformation Services (DTS) object model can build their own transformable subscription DTS packages in Microsoft® Visual Basic™. For information about programming to the DTS object model, see DTS Programming Reference.

A transformable subscription is a subscription in which the data is modified as it flows from Publisher to Subscriber. In replication programming, a Visual Basic program for a transformable subscription closely resembles that of a DTS Visual Basic program used outside of replication. This sample is located in C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\sqlrep\repldts.

How to Run the Transformable Subscription Sample

The program in the repldts folder is a sample of how to write a transformable subscription using Visual Basic.

To run the sample program

1. On the computer that will be a Publisher with a local Distributor, verify that the SQL Server Agent is running. If it is not, start it.

2. Open Visual Basic 6.0. Open ReplDTS.vbp, and then open the code window for ModReplDTS (ReplDTS.bas).

3. Select the following Project / References:
   - Microsoft DTSPackage Object Library (required)
   - Microsoft DTSDataPump Scripting Object Library (required to use a Microsoft ActiveX® script or custom transformation)
- **Microsoft DTS Custom Tasks Object Library** (required to use one of the DTS custom tasks)

4. Edit the following line of code to include your own connection information:
   ```csharp
goPackage.SavetoSQLServer "MyServerName", "sa", ""
```

5. In Visual Basic, start the ReplDTS program.

6. When the program is finished executing, you should receive a message indicating the Employees package was saved successfully. When you receive this message, save the project, and then close Visual Basic.

7. In SQL Server Enterprise Manager, ensure that your server is configured for replication.

8. In SQL Server Enterprise Manager, click **Data Transformation Services**, click **Local Packages**, and then on the **Action** menu, click **Refresh** to refresh the view. The package Employees should appear in the right pane.

9. Open SQL Query Analyzer, open the `repldts.sql` script supplied with sample, and then edit the `@subscriber` parameter of `sp_addsubscription` so that it contains your server name. The `sp_addsubscription` stored procedure is the last SQL statement in the file.

10. Run the `repldts.sql` script. You can run the entire script at once, or you can run the script a block at a time and check each message.

11. After the `repldts.sql` script has completed successfully, in SQL Server Enterprise Manager, expand **Replication Monitor**, expand the **Agents** folder, and then click the **Snapshot Agent** folder.
12. In the right pane, right-click the Snapshot Agent for the Employees publication, and then click **Start Agent**.

13. When the Snapshot Agent has completed, view or query the data in **MyEmployees** table of the subscription database **ReplDTS_SubDB**, and then compare it to the data in the Employees table of the publication database ReplDTS_PubDB.

14. The **Address** column of the **MyEmployees** table in the subscription database, **ReplDTS_subDB**, contains data concatenated from several columns of the Employees table of the publication database.

## Examining the Sample Code

This section describes key parts of the sample Visual Basic program.

- The Publisher connection is always set to the Microsoft SQL Server Replication OLE DB Provider for DTS. This is specified in the section of code where the connections are created. In addition, a required property specified for this provider is the column list for **ConnectionProperties**, which provides DTS Designer with the number and names of the source columns in the package. The Subscriber connection (not shown here) is set to the Microsoft OLE DB Provider for SQL Server.

```
Dim oConnection As DTS.Connection
Set Connection = goPackage.Connections.New("SQLReplication.OLEDB")
oConnection.Name = "Publisher article 'Employees'",
oConnection.ID = 1
oConnection.ConnectImmediate = False
oConnection.ConnectionProperties("Column List") = _
  "[EmployeeID],[LastName],[FirstName],[Title],[BirthDate],[HireDate],[Address],[City],[Region],[PostalCode],[Country],[HomePhone],[Extension],[ReportsTo]"
goPackage.Connections.Add oConnection
```
Set oConnection = Nothing

- If the destination table schema is not defined, use an Execute SQL custom task containing an SQL statement to generate the schema for the destination (Subscriber) table.

Use additional Execute SQL custom tasks to define SQL scripts to be applied after the data has been copied to the destination (for example, a script containing index generation statements (not shown)).

Use the following conventions when naming tasks in a replication DTS program (required by the replication agents):

- For a Data Driven Query task, the task name is the same as the article name.

- For custom tasks, such as an Execute SQL task, the name is a concatenated string consisting of the article name, the prefix "pre" (if the task executes before the snapshot data is copied) or "post" (if the task executes after the snapshot data is copied), and an optional part ("ignore_error") if an instruction is given to continue program execution when a query script error is encountered. To determine what the name should be, run sp_helparticledts in SQL Query Analyzer. For more information, see sp_helparticledts.

In the following code sample, the Execute SQL task name "Employees_pre_ignore_error" (line 5) means that the article name is Employees, the task occurs before the snapshot data is copied, and that program execution should continue if a script error is encountered.

Dim oTask As DTS.Task
Dim oCustomTask0 As DTS.ExecuteSQLTask
Set oTask = goPackage.Tasks.New("DTSExecuteSQLTask")
Set oCustomTask0 = oTask.CustomTask
oCustomTask0.Name = "Employees_pre_ignore_error"
oCustomTask0.Description = "Pre script for article employees"
oCustomTask0.SQLStatement = _
"If object_id('MyEmployees') is NOT NULL _
BEGIN Drop Table MyEmployees END _
Create Table MyEmployees _
([EmployeeID] [int] NOT NULL,
[LastName] [nvarchar] (20) NOT NULL,
[FirstName] [nvarchar] (10) NOT NULL,
[Title] [nvarchar] (30) NULL,
[Birthdate] [datetime] NULL,
[HireDate] [datetime] NULL,
[Address] [nvarchar] (255) NULL,
[HomePhone] [nvarchar] (24) NULL,
[Extension] [nvarchar] (4) NULL,
[ReportsTo] [int] NULL,
CONSTRAINT [PK_Employees] PRIMARY KEY _
   CLUSTERED([EmployeeID]))"
CustomTask0.ConnectionID = 2
goPackage.Tasks.Add oTask
Set CustomTask0 = Nothing
Set oTask = Nothing

- With transformable subscriptions, data movement is always done with a Data Driven Query task, never with a Transform Data task, which is commonly used in DTS packages that do not use replication. Therefore, you must define a Data Driven Query custom task:
Dim oTransformation As DTS.Transformation
Dim oTransProps As DTS.Properties
Dim oColumn As DTS.Column
Dim oCustomTask1 As DTS.DataDrivenQueryTask
Set oTask = goPackage.Tasks.New("DTSDataDrivenQueryTask")
Set oCustomTask1 = oTask.CustomTask
oCustomTask1.Name = "Employees"
oCustomTask1.Description = "Transformations for article Employees"
oCustomTask1.SourceConnectionID = 1
oCustomTask1.SourceObjectName = "nothing" 'Experiment
oCustomTask1.DestinationConnectionID = 2
oCustomTask1.DestinationObjectName = "MyEmployees"

• Among the properties you need to define for the Data Driven Query
  task are the parameterized queries associated with each type of
  incremental update (INSERT, UPDATE, and DELETE). The
  parameterized query works by selecting for data movement source rows
  that satisfy the conditions in the query statement.
  oCustomTask1.InsertQuery = "INSERT INTO _
   MyEmployees values (?,?,?,?,?,?,?,?,?,?)"
  oCustomTask1.UpdateQuery = "UPDATE _MyEmployees _
   SET LastName=?, FirstName=?, Title=?, Birthdate=?, _
   HireDate=?, Address=?, HomePhone=?, Extension=?, _
   ReportsTo=? where EmployeeID=?"
  oCustomTask1.DeleteQuery = "DELETE MyEmployees _
   WHERE EmployeeID = ?"

• Another essential set of operations is setting the source and destination
  column collections. Because one of the transform operations performed
  by the DTS package is a concatenation of several of the source
  columns, the number of destination columns added to the destination
  collection is fewer than the number of source columns added to the
  source collection. An example of one source column and one
  destination column added to their respective collections is shown here.
  The column "Address" for the destination is actually the concatenation
  of the source columns "Address", "City", "Region", and "Postal Code".
  Set oColumn = oTransformation.DestinationColumns.New("Hi
  oColumn.Name = "HireDate"
  oColumn.Ordinal = 6
  oTransformation.DestinationColumns.Add oColumn
  Set oColumn = Nothing
  ...

  Set oColumn = oTransformation.DestinationColumns.New("Ad
oColumn.Name = "Address"
oColumn.Ordinal = 7
oTransformation.DestinationColumns.Add oColumn
Set oColumn = Nothing
...

- To complete the Data Driven Query task transformation, add column definitions (in sequential order) to each of the incremental update operations described earlier. For example, because an INSERT operation was defined with parameters for the 10 destination columns of the article, INSERT column definition code for each of the 10 columns is required (only the first two are shown here):

  Set oColumn = oCustomTask1.InsertQueryColumns.New("EmployeeID")
oColumn.Name = "EmployeeID"
oColumn.Ordinal = 1
oCustomTask1.InsertQueryColumns.Add oColumn
Set oColumn = Nothing

  Set oColumn = oCustomTask1.InsertQueryColumns.New("LastName")
oColumn.Name = "LastName"
oColumn.Ordinal = 2
oCustomTask1.InsertQueryColumns.Add oColumn
Set oColumn = Nothing
...

Similar code is used for the UPDATE and DELETE queries. When defining the columns for those queries, remember that the order of the columns must follow the order of the parameters specified by the question mark (?) characters in the InsertQuery and DeleteQuery definitions. For example, in this sample, the DELETE query uses only the EmployeeId column; therefore, only code for that column is used.

  Set oColumn = oCustomTask1.DeleteQueryColumns.New("EmployeeID")
oColumn.Name = "EmployeeID"
oColumn.Ordinal = 1
In the example, a Microsoft ActiveX® script performs the actual transformation of data, which is the concatenation of several columns. The entire ActiveX script code should be viewed in the sample. This example shows how to set the transform server property to handle scripts, and how to set the line of script code used to perform the column concatenation.

Set oTransProps = oTransformation.TransformServerProperties
...

```
oTransProps("Text") = oTransProps("Text") &
    "DTSDestination("Address") = DTSSource("Address")
    & "," & DTSSource("City") & "," & DTSSource("Region")
    & "," & DTSSource("PostalCode") & vbCrLf
```

Transformable subscription DTS packages are typically saved to an instance of SQL Server. They can also be saved as a .dts structured storage file, but cannot be saved to the repository. In the following line of code, the package is saved to an instance of SQL Server:

```
GoPackage.SaveToSQLServer "myServerName", "sa", ""
```

DTS includes several options for handling data conversions. These options are handled by a set of transformation flags whose values can be viewed in the Visual Basic Object Browser under the DTSPump component, DTSTransformFlags enumeration. When building a replication DTS package in Visual Basic, if an ActiveX script transformation is used, as in this sample, the transformation flags do not need to be explicitly set. If you build a replication DTS package without ActiveX scripts (using only Copy Column mappings), the TransformFlags property must be explicitly set to DTSTransformFlag_Default (a value of 63).

```
oTransformation.TransformFlags = 63
```
Replication Programming
## Merge Replication Samples

Microsoft® SQL Server™ 2000 comes with the following samples to help you implement merge replication in your application.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subscriber-Based Resolver using C++ and a Stored Procedure</strong></td>
<td>Microsoft Visual C++ language sample that builds a stored procedure custom resolver to use in merge replication.</td>
</tr>
<tr>
<td><strong>Transact-SQL Custom Stored Procedure Resolver</strong></td>
<td>Transact-SQL stored procedure that is the custom resolver used in a merge replication.</td>
</tr>
<tr>
<td><strong>Generating Merge Dynamic Snapshot Jobs</strong></td>
<td>The procedures in this sample show how to enumerate a list of users from within a Microsoft Windows® group and generate dynamic snapshot jobs for each user.</td>
</tr>
</tbody>
</table>
Replication Programming
Subscriber-Based Resolver Using C++ and a Stored Procedure

This sample application builds a custom stored procedure resolver that executes the stored procedure at the Subscriber. It contains a sample script to set up a publication on Northwind, and install the sample resolver that calls sp_authority_resolver. sp_authority_resolver retrieves the 'authority' value from the table at the Publisher and Subscriber, and the row with the highest authority will win and be returned to the merge process to be applied to both servers. If both rows have the same authority, the source table will win. The Microsoft® Visual C++ sample code is located in the \Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\resolver\subspres directory.

To run the sample program

1. On the computer that will be the Publisher, verify that the SQL Server Agent is running. If it is not running, start it.

2. Verify that the computer is configured for publishing and distribution. If it is not configured, configure it using the Configure Publishing and Distribution Wizard. The defaults given in the wizard are all acceptable for this sample.
Replication Programming
Transact-SQL Custom Stored Procedure Resolver

The program in the C:\Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\deflt_sp directory is a sample application, which builds a custom stored procedure resolver that executes at the Publisher. The resolver uses the Northwind sample database, a distributed query to obtain information from the Subscriber, and then computes the average price if the values between the Publisher and Subscriber are different. The resolver then makes the average price the resolved value, and logs the conflict at the Publisher so it can be viewed, and if necessary, changed. Finally, the application sends an e-mail stating that the price was changed to an average due to a conflict. If any columns other than price have changed, the resolver uses the values from the Publisher.

The resolver is designed to run on the Products table in Northwind sample database. It is assumed that the sample code was saved to the default directory offered during installation, and that the files can be found in C:\Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\deflt_sp.

Note To run this sample, two computers are required. For the distributed query to work, the two computers must both be running the Microsoft Windows NT 4.0 or Windows 2000 operating system, or the Publisher must be running on Windows 2000 with the linked server running on Windows NT 4.0.

To run the sample program

1. On the computer that will be the Publisher, verify that the SQL Server Agent is running. If it is not running, start it.

2. Using the Create Publication Wizard, create a merge publication based on the Northwind database, and then select the Products table as an article in the publication.

How to create publications and define articles
Replication Programming
Generating Merge Dynamic Snapshot Jobs

This sample generates multiple merge dynamic snapshot jobs for a set of Windows Group users. After you create a merge publication with a dynamic filter and generate a standard snapshot, you can run the extended stored procedure and stored procedures included with this sample to generate dynamic snapshot jobs.

The sample extended stored procedure enumerates a Windows group and identifies the members for which dynamic snapshot jobs will be created. After the members are identified, you can execute the sample stored procedure that will create the actual dynamic snapshot jobs. After the dynamic snapshot jobs are created, you will need to start them manually or ensure they will start according to a schedule.

If you have a dynamically filtered merge publication with multiple subscriptions, this sample shows how you can create multiple dynamic snapshot jobs for the users of those subscriptions. 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. For more information, see Dynamic Snapshots.

The sample code is located in the \Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\dynsnapjob directory.

To run the sample programs

1. In the \Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\dynsnapjob\xp directory, copy the compiled xp_enumntusers.dll into your Windows system32 folder.

2. Register the name of the extended stored procedure to Microsoft SQL Server™ by running the following Transact-SQL command in SQL Query Analyzer:
   use master
go
exec sp_addextendedproc 'xp_enumntusers', 'xp_enumntusers'.

go

3. To create the stored procedure, open SQL Query Analyzer, and then run spdynsnapsample.sql, which is located in the \Samples\sqlrepl\dynsnapjob\sp directory.

4. Create a merge publication with dynamic filtering enabled (use a system function such as SUSER_SNAME() as the criteria for the filter), and then generate a standard snapshot for the publication.

5. Allow a Windows Local (or Global) group access to the SQL Server instance that contains the merge publication in Step 4. This group needs to have public access only to the database on which you created the publication. This step can be performed using SQL Server Enterprise Manager.

6. Add the Windows Group to the Publication Access List (PAL), which is found in the merge publication properties.

You can now run the extended stored procedure and stored procedure to generate dynamic snapshot jobs for the Windows Group users. For more information about how to execute each procedure, see Using xp_enumntusers and Using sp_addntgroupmergedynsnapshotjob.

Using xp_enumntusers

This extended stored procedure enumerates through Microsoft® Windows® Local and Global group users.

xp_enumntusers [ @servername = ] 'domain_server_name'
, [ @groupname = ] 'group_name'

Arguments
[@servername = ] 'domain_server_name'

The domain server to retrieve the group from. Use '.' or '' for local server.

[@groupname = ] 'group_name'

The name of the group from which to enumerate users.

Both @servername and @groupname must be server strings.

Using sp_addntgroupmergedynsnapshotjob

This stored procedure generates a dynamic snapshot job for each user found when xp_enumntusers was run. The stored procedure accepts parameters that allow you to specify the following:

- Generate a single dynamic snapshot job for all users or generate a separate dynamic snapshot job for each user.

- Filter the enumerated user using the LIKE operand.

- Specify schedule information.

sp_addntgroupmergedynsnapshotjob [ @publication = ] 'publication'
 , [ @ntserver_name = ] 'ntserver_name'
 , [ @group_name = ] 'group_name'
 , [ @destination = ] 'destination'
 , [ @job_name = ] 'job_name'
 , [ @like_string = ] 'like_string'
 , [ @as_one_job = ] as_one_job
 , [ @frequency_type = ] frequency_type
 , [ @frequency_interval = ] frequency_interval
 , [ @frequency_subday = ] frequency_subday
 , [ @frequency_subday_interval = ] frequency_subday_interval
 , [ @frequency_relative_interval = ] frequency_relative_interval
 , [ @frequency_recurrence_factor = ] frequency_recurrence_factor
 , [ @active_start_date = ] active_start_date
 , [ @active_end_date = ] active_end_date
 , [ @active_start_time_of_day = ] active_start_time_of_day
Arguments

[@publication = ] 'publication'

The name of the publication. publication is sysname, with no default.

[@ntserver_name = ] 'ntserver_name'

The Windows server or domain name on which to enumerate users. Use '.' for local server. ntserver_name is nvarchar(100) with no default.

[@group_name = ] 'group_name'

The name of the group on the Windows server or domain. group_name is nvarchar(256) with no default.

[@destination = ] 'destination'

The destination directory path. destination is nvarchar(3500) with no default.

[@job_name = ] 'job_name'

The job name. job_name is sysname with a default of NULL. If NULL, a default job name is used. This parameter is ignored when @as_one_job=0

[@like_string = ] 'like_string'

The string to be used in WHERE clause used when creating jobs for user names. For example, the WHERE clause is constructed of "WHERE user name LIKE @like_string". like_string can contain any of the valid wildcard characters such as the percent sign ('\%'). like_string is nvarchar(1000) with a default of NULL.

[@as_one_job = ] as_one_job

Specifies how many dynamic snapshot jobs to create. as_one_job is a bit with a default of 1. 1 specifies that one dynamic snapshot job will be created for all users. 0 specifies that an individual dynamic snapshot job will be created for each user.

[@frequency_type = ] frequency_type

@frequency_type int = 4,
Is a value indicating when the job is to be executed. \textit{freq\_type} is \textbf{int} with a default of 4, and can be one of these values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Once</td>
</tr>
<tr>
<td>4</td>
<td>Daily</td>
</tr>
<tr>
<td>8</td>
<td>Weekly</td>
</tr>
<tr>
<td>16</td>
<td>Monthly</td>
</tr>
<tr>
<td>32</td>
<td>Monthly, relative to \textit{frequency_interval}</td>
</tr>
<tr>
<td>64</td>
<td>Run when SQL Server Agent service starts</td>
</tr>
<tr>
<td>128</td>
<td>Run when computer is idle</td>
</tr>
</tbody>
</table>

[@\textit{frequency\_interval} = ] \textit{frequency\_interval}

Is the days that the job is executed. \textit{freq\_interval} is \textbf{int} with a default of 1, and the value used is dependent on the value of \textit{freq\_type}.

<table>
<thead>
<tr>
<th>Value of \textit{frequency_type}</th>
<th>Effect on \textit{frequency_interval}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Once)</td>
<td>\textit{frequency_interval} is ignored.</td>
</tr>
<tr>
<td>4 (Daily)</td>
<td>Every \textit{frequency_interval} days.</td>
</tr>
</tbody>
</table>
| 8 (Weekly)                       | \textit{frequency\_interval} is one or more of the following (combined with an OR logical operator):
|                                 | 1 = Sunday
|                                 | 2 = Monday
|                                 | 4 = Tuesday
|                                 | 8 = Wednesday
|                                 | 16 = Thursday
|                                 | 32 = Friday
|                                 | 64 = Saturday |
| 16 (Monthly)                     | On the \textit{frequency\_interval} day of the month. |
32 (Monthly relative)  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description of Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1</td>
<td>At the specified time.</td>
</tr>
<tr>
<td>0x4</td>
<td>Minutes.</td>
</tr>
<tr>
<td>0x8</td>
<td>Hours.</td>
</tr>
</tbody>
</table>

64 (When SQL Server Agent service starts)  

 frequency_interval is unused.

128 (When computer is idle)  

 frequency_interval is unused.

[@frequency_subday = ] frequency_subday

Specifies the units for frequency_subday_interval. frequency_subday is int with a default of 8, and can be one of these values.

[@frequency_subday_interval = ] frequency_subday_interval

Is the number of frequency_subday_type periods to occur between each execution of the job. frequency_subday_interval is int, with a default of 6.

[@frequency_relative_interval = ] frequency_relative_interval

Is the scheduled job’s occurrence of frequency_interval in each month, if frequency_interval is 32 (monthly relative). frequency_relative_interval is
**int** with a default of 1, and can be one of these values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description of Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First</td>
</tr>
<tr>
<td>2</td>
<td>Second</td>
</tr>
<tr>
<td>4</td>
<td>Third</td>
</tr>
<tr>
<td>8</td>
<td>Fourth</td>
</tr>
<tr>
<td>16</td>
<td>Last</td>
</tr>
</tbody>
</table>

[@frequency_recurrence_factor = ] *frequency_recurrence_factor*

Is the number of weeks or months between the scheduled execution of the job. *frequency_recurrence_factor* is used only if *frequency_type* is 8, 16, or 32. *frequency_recurrence_factor* is **int** with a default of 0.

[@active_start_date = ] *active_start_date*

Is the date on which execution of the job can begin. *active_start_date* is **int** with a default of 0, which indicates today’s date. The date is formatted as YYYYMMDD. If *active_start_date* is not NULL, the date must be greater than or equal to 19900101.

[@active_end_date = ] *active_end_date*

Is the date on which execution of the job can stop. *active_end_date* is **int** with a default of 99991231, which indicates December 31, 9999. Formatted as YYYYMMDD.

[@active_start_time_of_day = ] *active_start_time_of_day*

Is the time on any day between *active_start_date* and *active_end_date* to begin execution of the job. *active_start_time* is **int**, with a default of 0, which indicates 12:00:00 A.M. on a 24-hour clock. The value for this parameter must be entered using the form HHMMSS.

[@active_end_time_of_day = ] *active_end_time_of_day*

Is the time on any day between *active_start_date* and *active_end_date* to end execution of the job. *active_end_time* is **int** with a default of 235959, which indicates 11:59:59 P.M. on a 24-hour clock. The value for this parameter
must be entered using the form HHMMSS.
Replication Programming
# Replication Distributor Interface Samples

Microsoft® SQL Server™ 2000 comes with the following sample applications to help you implement the Replication Distributor Interface.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="#">Programming Snapshot or Transactional Replication from Heterogeneous Data Sources</a></td>
<td>Microsoft Visual Basic® SQL-DMO application to configure the Publisher, publication, and a subscription for use in a heterogeneous publishing environment. Also includes a Microsoft Visual C++® application to deliver transactions to the Distributor.</td>
</tr>
</tbody>
</table>

[Programming Snapshot or Transactional Replication from Heterogeneous Data Sources](#): Microsoft Visual Basic® SQL-DMO application to configure the Publisher, publication, and a subscription for use in a heterogeneous publishing environment. Also includes a Microsoft Visual C++® application to deliver transactions to the Distributor.
Replication Programming
Programming Snapshot or Transactional Replication from Heterogeneous Data Sources

SQL-DMO and the Replication Distributor Interface sample explains how to support a third-party publication. Because this sample uses transactional replication, this sample will work only with Microsoft® SQL Server™ 2000 Standard Edition and SQL Server 2000 Enterprise Edition.

SQL-DMO configures a third-party publication and Distributor using SQL Server. The Replication Distributor Interface logs any transaction that it marks for replication to this Distributor. The third-party publication uses the Distribution Agent to distribute the transactions as if they were coming from an instance of SQL Server.

If the samples were installed to the default location, the files used in this sample will be located in the C:\Microsoft SQL Server\80\Tools\Devtools\Samples\sqlrepl\samppub directory.

To run the sample programs

1. On the computer that will be the Publisher, verify that SQL Server Agent is running. If it is not running, start it.

2. In SQL Query Analyzer, open \Samples\sqlrepl\samppub\samppub.sql, and then execute it. This script creates two databases, SampleSubscriberDB1 and SampleSubscriberDB2, and sets up a subscription to one of the databases. The script also configures the computer for publishing and distribution, adds a distribution database named distribution, and enables the computer as a Subscriber.

3. Execute the BAT file iniwkdir.bat, in the in \Microsoft SQL Server\80\Tools\Devtools\Samples\Sqlrepl\Samppub directory, by either double-clicking it in Microsoft Windows Explorer or running it at a command line. The result of the execution creates the working directory at C:\mssql8\repldata\UNC\samppub that contains the files Samptab.sch and Samptab.bcp. These schema and .bcp data files
will be used by the Visual C++ application.

4. On the Publisher, open Visual Basic, open the SQL-DMO application workspace, *samppub.vbp*, in the `samppub\sqlmdmo` directory. Build the application by selecting Make `samppub.exe` on the File menu. The resulting executable will be named `samppub.exe`, (the same name as the Visual C++ executable); therefore, it is recommended that this one be kept in the SQLDMO directory or in a directory of your choosing where the two executables can be distinguished from each other.

5. Run the Visual Basic version of the `samppub.exe` from inside Visual Basic on the Run menu by clicking Start, or by starting the executable from where it was saved. The application has one form with four buttons.

6. Click the Create Sample Publication button to create a publication on a third-party vendor. You can view the results in SQL Server Enterprise Manager. Expand Replication, and there will be a new folder called Heterogeneous Publications. There will be a folder called Sample Vendor, with a publication called SamplePublication. You can right-click SamplePublication:SampleDatabase and select Properties to view information about the publication. You will see that there is no subscription to this publication. If the Replication folder was already expanded, click Refresh to show the new objects or close and open SQL Server Enterprise Manager again.

7. Click the Add Subscription button. This creates a push subscription, with the data going to the SampleSubscriberDB1 database. If you have not clicked the Create Sample Publication button, clicking this button will give an error because there will be no publication on which to add a subscription. After clicking this button, you can view the results in SQL Server Enterprise Manager. Expand Replication, expand Heterogeneous Publications, expand Sample Vendor Right-click SamplePublication:SampleDatabase and select Properties. The Subscriptions tab shows the new subscription.
8. Open Visual C++, and then open the workspace \samppub.dsw in the \samppub\repldist directory. On the Tools menu, point to Options, and then click the Directories tab. In the Show directories for box, select Include files and point to the \Microsoft SQL Server\80\Tools\Devtools\Include directory. For the Library setting, navigate to \Microsoft SQL Server\80\Tools\Devtools\Lib. On the Build menu, select Build samppub.exe. The application will be compiled and saved to the \samppub\repldist directory.

9. Run the Visual C++ version of samppub.exe. On the Build menu, click Execute samppub.exe. It can also be executed directly running it from the directory where it was saved. This will deliver transactions to the Distributor. The application will perform Snapshot Agent and Log Reader Agent operations for the publication. When run, messages will appear in the Command window, with the messages:

Sample Publisher Agent Startup
Added 1 Transaction(s) consisting of 3 Command(s)
Sample Publisher Snapshot Agent Succeeded.
Added 1 Transaction(s) consisting of 30 Command(s)
Sample Publisher Logreader Agent Succeeded.

After a successful execution, you should see a table SampleTable1 in the SampleSubscriberDB1 database. Opening the table shows 20 rows of data.

10. Optionally, create a push subscription to the SampleSubscriberDB2 from the publication database using SQL Server Enterprise Manager. When creating this subscription, do not create an initial snapshot. Running the Visual C++ program results in transactions being propagated to both subscriptions.

11. Execute the Visual Basic samppub.exe file from where it was saved, and then click Drop Subscription to drop the push subscription. After clicking this button, you can view the results in SQL Server Enterprise
Manager. Expand the SQL Server Group, the **Replication** folder, **Heterogeneous Publications** folder, and **Sample Vendor**. On **SamplePublication:SampleDatabase**, right-click and select **Properties**. On the **Subscriptions** tab is a **Properties** button that shows the **Subscription Properties** dialog box. There are no subscriptions showing. If you have not clicked the **Create Sample Publication** button, clicking this button will give an error, as there will be no subscription to drop because one has not been created. If you have not clicked the **Add Subscription** button, clicking this button will give an error, as there is no subscription to drop.

12. Click the **Drop Sample Publication** button to removes the publication. In SQL Server Enterprise Manager, in the **Replication** folder, the **Heterogeneous Publications** folder is removed if this was the only heterogeneous publication on the computer. If you have not clicked the **Create Sample Publication** button, clicking this button will give an error, as there will be no Publication to drop. The subscription and the publication have been removed.

Data and schema files were installed on the local computer to a directory created by the **iniwkdir.bat**. The directory is C:\mssql8\repldata\uncsamppub. They were used by the Visual C++ application as the data copied into the **SampleTable1**.