Database Replication with the Microsoft Jet Database Engine: A Technical Overview

The Microsoft Jet database engine version 3.5 is a 32-bit engine that provides database processing and replication functionality to a variety of applications. This document is intended for experienced Microsoft Access users who want to understand database replication as implemented in Microsoft Jet and use it more effectively in their applications.

What Is Database Replication?

Database replication is a technique you can use to support multiple users of an application. Replication is the process of creating multiple copies of an application and its data, to be used at locations that are not always connected to each other. Collectively, the copies are called a *replica set*. One member of the replica set must be designated as the *Design Master;* any other copy is a *replica*. Each replica contains a common set of *replicated objects*. Any single replica can also contain *local objects* that exist only in that replica.

A replica set can contain only one Design Master, but can contain as many replicas as needed. The Design Master is the only copy where changes to the database design, or schema, are allowed. You can designate any replica as the Design Master, but be sure that only one replica is marked as the Design Master at any time. To create a Design Master, you convert an existing database to replicable format and set certain properties to identify it as the Design Master. Some of the replication tools available do this for you automatically.

A database in replicable format includes a number of fields (columns) and tables that Microsoft Jet uses to manage the replicated application. For example, one field in each record (row) in a replicated table has a globally unique identifier (GUID) that distinguishes the record from every other record in the replica set. Another field records the *lineage* (replica ID/version pair) of each record. When a record is updated, the version number for that replica's record in the lineage is incremented.

Any copy—the Design Master or a replica—can update the data. This is called a *multi-master data update* design; it permits a fully distributed system where data updates are not centralized. This is a significant benefit of using Microsoft Jet, because most other relational database engines support *single-master data updates* where data can only be updated at a single replica. A record is the most basic unit of information recognized in replication. This means that if any cell in a record is modified then the whole record is marked as changed. When changes are transmitted during synchronization, the whole record is updated. OLE objects and memo fields are exceptions; these items—due to their potential large size—are not transmitted unless they have been changed.

Synchronization, an important part of the replication process, reconciles all these changes and updates the data set in each replica. The *file tracking system* in Microsoft Jet tracks and records all changes at all replicas, in preparation for updating data during synchronization. Only records marked as changed are updated when you synchronize replicas. If two replicas simultaneously update the same record at different replicas, Microsoft Jet reconciles the updates. Synchronization can be performed on a regular schedule or as often as necessary to ensure all users have current data. This means that all information will reach all replicas, but there is no guarantee it will reach all replicas within any specified amount of time. When using Microsoft Jet, application designers must allow for this in their designs.

Microsoft Jet uses incremental replication. This means that, during a single synchronization between two replicas, the only updates made are those resulting from changes made since the last synchronization. This provides significant benefits over methods of data distribution that transmit the whole database whenever new data or objects require distribution. Each record in a replicable database has a generation counter; Microsoft Jet uses this field to control incremental exchanges. Microsoft Jet also records the maximum generation in the local database.

Not all applications that support Microsoft Jet use its features in the same way. For example, although Microsoft Excel cannot replicate a database, it can update a database replicated by another product. Microsoft Jet monitors the changes made by Microsoft Excel, Microsoft Visual Basic®, or Microsoft Access to a replica, and updates these changes when you synchronize the replicas.

Where Is Microsoft Jet supported?

Windows® 95 and Windows NT® Intel® platforms support the Microsoft Jet database engine. These products support Microsoft Jet either directly or through Microsoft Data Access Objects (DAO):

- Microsoft Access 97
- Microsoft Visual Basic
- Microsoft Excel 97
- Microsoft Visual C++®
- Microsoft Project 97
- Microsoft Office 97, Developer Edition

In addition, Windows NT, Novell NetWare, and Windows 95 peer-to-peer networks support Microsoft Jet. Banyan Vines and LANtastic do not support the Microsoft Jet Database. Microsoft Jet includes replication code that monitors whether a replica has been copied or moved to a new network location, and determines whether a replica requires a new value for the **ReplicaID** property. Microsoft Jet presumes that files are named in accordance with Microsoft and Novell network file-naming conventions, which are different from the file-naming conventions used by Banyan and LANtastic.

Replication is a feature of the Jet database engine, not of the specific applications that include the Jet database engine. Microsoft Jet performs all database processing for Microsoft Access and Visual Basic, and can provide data to Open Database Connectivity (ODBC) client applications using the Microsoft Access 2.0 ODBC driver.

When Should I Use Replication?

Replication is well suited for distributed systems that focus primarily on adding new records rather than on updating existing records. Sales representatives who visit customer offices, parcel delivery drivers, and inspectors who visit a variety of construction sites are all examples of users who might benefit from replication. There are many tools and techniques for implementing replication. Some factors to consider when choosing a replication technique are:

- How quickly you want the application to respond.
- How quickly you need data synchronized across all sites.
- Budget for hardware, software, and communication services.
- Overall system reliability requirements.

The best candidates for replication are applications that can tolerate some latency in data updates in exchange for a robust configuration that can allow updates from any replica and that supports users who are only occasionally connected. This flexibility means the system can work more effectively, potentially improving business performance. Using flexible, low-cost, off-peak asynchronous communication links and asynchronous data duplication provides "real-time-enough" updates without the expense and vulnerability of full-time connection between all nodes. When the application's users are connected, it might be via a direct connection on a local area network (LAN) or wide area network (WAN), or via the Internet. Data can be exchanged on a LAN, a WAN, or the Internet.

Microsoft Jet replication is a good solution if you want to:

- Share data among users at multiple remote locations.

- Automate the distribution of new features and updates to multiple users.
- Use different machines for system queries and transaction processing (this can improve transaction-processing performance).
- Automatically back up data without disabling the system (each replica serves as a backup, so a separate backup procedure is not needed).

If your multiuser application requires very frequent data updates, if it will update a large number of records at one or more sites, or if it is critical for data changes to be very quickly obvious to other users, replication may not be the best solution for you to use. Applications in these categories are better served by two-phase commit solutions. A "two-phase commit" is where replicas are connected all the time, and an update at any one site will only be accepted if agreement is immediately given from all other sites. It's called a two-phase commit because the initial phase is notification of a proposed update sent to all replicas, and the second phase is the actual update only when all sites have agreed (that is, committed) to the update.

Tools That Implement Replication

There are several tools you can use to implement Microsoft Jet replication. These tools allow you to convert a database to replicable format, identify a replicable database as the Design Master or a replica, initiate synchronization of the replica set, and a variety of other management tasks. You can use the following tools to implement Microsoft Jet replication:

- Microsoft Access 95 or Microsoft Access 97, running under Windows 95 or Windows NT
- Briefcase replication
- Microsoft Replication Manager, available through Microsoft Office 97, Developer Edition
- DAO programming, available in Windows 95 or Windows NT version 3.51 or later

The first three of these tools provide an easy-to-use visual interface, while the last enables programmers to build replication directly into their applications.

Microsoft Access Replication Commands

The **Replication** submenu on the **Tools** menu in Microsoft Access provides several commands to help you create a replica, synchronize a replica with another member of the replica set, resolve synchronization conflicts, and recover a replica set's Design Master. For more information about these commands, refer to the Help system provided with Microsoft Access or to *Building Applications with Microsoft Access 97*.

Briefcase Replication

The Briefcase is an accessory available in Windows 95 or Windows NT version 4.0. When Microsoft Access is installed on your computer, you can use the Briefcase as a replication tool by simply dragging an .mdb file from the Windows Explorer onto the Briefcase icon on the Windows desktop. Like magic, your database is converted into replicable format and becomes a member of your replica set. The Briefcase menus include commands to synchronize the replicas.

Behind the scenes, of course, Windows is busy making the magic work. When you install Microsoft Access, the Setup program adds Class ID (CLSID) entries for .mdb files and for the Briefcase reconciler to the Windows registry. (The Briefcase reconciler is installed only by Microsoft Access; it is not included with Visual Basic or Microsoft Excel.) The reconciler includes the code required to support replication and synchronization. When you drag an .mdb file onto the My Briefcase icon, Windows recognizes the class ID and responds by calling the reconciler. The reconciler converts the database into a replicable form, leaves the Design Master at the source, and places a replica in the Briefcase. The reconciler gives you the option of putting the Design Master in your Briefcase and leaving a replica on the desktop. When you synchronize the replicas, the Briefcase calls the reconciler to merge the replicas. With Briefcase replication, synchronization cannot be scheduled; it occurs only when the **Update** command is clicked and only between the current member and the specified member.

Note Before converting the database, Microsoft Jet asks if you want to make a backup. If you anticipate that any users will need to use a nonreplicable version of the database, it's a good idea to make this backup.

You can use the Briefcase with files other than .mdb files, and with applications other than Microsoft Access. However, doing so will not call the Microsoft Jet replication code; it will call the default Briefcase code instead. If you use the Briefcase when Microsoft Access is not installed or with a non-Microsoft Jet database, dragging a file into the Briefcase is equivalent to simply copying the file into the Briefcase—there is no conversion to replicable format. Therefore, when you update files on your main computer with files from the Briefcase, the Briefcase simply copies over the original file—changes to data and objects are not merged, they are overwritten.

Microsoft Replication Manager

Microsoft Replication Manager is included with Microsoft Office 97, Developer Edition. It is another tool that lets you use replication to administer a distributed application. It offers more functionality and more features than the Briefcase. Key features include:

- Graphical user interface and tools for system development and maintenance.
- Visual representation of replica set topologies, which greatly assist system management.
- Activity reports to assist with troubleshooting, and synchronization reports to help monitor the activity among replicas.
- Property dialog boxes that provide valuable information about components.
- Administration commands controlling the conversion, creation, location, and management of replicas.
- Scheduled exchanges between replicas administered via a graphical user interface, plus immediate synchronization with remote replicas via point-and-click commands.
- Direct and indirect exchanges provide additional support for "rarely connected" users. Laptop users may specify a networked file location where exchange information may be deposited for later processing. Synchronization is optimized over a LAN or WAN for indirect exchanges, and optimized for direct exchanges when a direct connection can be established between local replicas.
- Synchronization over the Internet.
- Exchanges can be configured to only send data, receive data, or send and receive.

Synchronizer

The Synchronizer is an agent you can use with the Replication Manager to provide scheduled background exchanges between replicas. These exchanges can be made while two replicas are directly connected, or through a file-system transport that does not require a direct connection for the exchange. In either type of transfer, the Synchronizer collects the changes at one replica and transmits them to other replicas. If the file-transfer system is used, one replica must deposit changes in a temporary file. The Synchronizer, initiated from the target replica, collects the updates at a later time and applies them to the target replica. This is a great benefit for rarely connected users; they can post changes whenever convenient rather than depending on an available connection.

The file-system transport provides an interface to the messaging service, with provision for additional messaging services in later releases.

Data Access Objects (DAO)

In Microsoft Access and Microsoft Visual Basic, DAO provides a programmatic interface to replication functions. Microsoft Jet includes a variety of extensions to the DAO programming interface. These extensions allow developers to convert a database to replicable format, make additional replicas, synchronize replicas, and manage certain properties of a replicated database. These properties include the description of a single replica or of a replica set, the ID of a particular

replica or of the replica set's Design Master, the default replica to be used in an exchange, and the local/global property of each object in the database.

Structure of a Replicable Database

Before you can use replication, you must convert the original database to replicable format. Any of the tools listed earlier can help you do this. A database in replicable format includes a number of tables and fields that are not typically present in a nonreplicable database. When you use the tools listed earlier, Microsoft Jet automatically adds the fields and tables it needs to manage your replicated application. A database in replicable format includes:

- *User tables*—The tables an application developer constructs (for example, Products, Prices, Customers, and so on). When you convert the database to replicable format, Microsoft Jet retains your table definitions and adds certain fields it needs to manage the application.
- *System tables*—The tables Microsoft Jet requires to manage the application. For example, the MSysIndex table records all indexes in an application. These tables are generally hidden from users. When a Microsoft Jet database is converted into a replicable database, Microsoft Jet adds new system tables that record the history of exchanges between replicas, the location of other replicas in the replica set, and other information required by the system.
- *GUID*—A field that provides a 16-byte globally unique identifier (GUID) for each record in each table. A GUID is guaranteed to be unique, even if two users simultaneously construct a new record at different locations. If Microsoft Jet adds this field to your tables, the default name for the field is s_GUID. If your table already includes a numeric field whose data type is Replicable ID, Microsoft Jet will use the existing field instead of adding a new one.
- *Generation*—A field used to ensure that only changed data gets exchanged when two replicas are synchronized. Microsoft Jet adds this field to each table in your application. The default name for the field is s_Generation.
- *Lineage*—A field that tracks the version and replica number of each record in each table. Microsoft Jet uses this field to keep track of changes that have already been processed and to ensure that changes to a record are not forever sent in a circle between replicas. Microsoft Jet adds this field to each table in your application. The default name for this field is s_Lineage.

Additional fields are inserted in the table where there are OLE links to embedded graphics or memo fields. This is an exception to the rule that data changes are tracked on a record level. Because OLE objects may be of a significant size, (a bit image may easily be 1 MB or more) and therefore expensive to send over a communications line, Microsoft Jet replication only sends the OLE object if it has been modified.

System Tables

System tables support code within Microsoft Jet. You should not rely upon the format remaining the same between releases. Treat the descriptions of these tables as "for information only," which may assist you in debugging certain applications.

- *MSysErrors* identifies where and why errors occurred during data synchronization. This table is replicated to all members of the replica set.
- *MSysExchangeLog* is a local table that appears in each member of the replica set, and stores information about synchronizations that have taken place between this member and other members of the replica set.
- *MSysGenHistory* stores a history of generations. It contains a record for each generation that a replica knows about. It is used to avoid sending common generations during synchronizations and to resynchronize replicas that are restored from backups. This table appears in all members of the replica set, but it is merged by a process slightly different from that used with normal replicated tables.
- *MSysOthersHistory* stores a record of generations received from other replicas. It contains one generation from every message seen from other replicas.

- *MSysRepInfo* stores information relevant to the entire replica set, including the identity of the Design Master. It contains a single record. This table is replicated to all members of the replica set.
- *MSysReplicas* stores information about all replicas in the replica set. This table is replicated to all members of the replica set.
- *MSysSchChange* stores design changes that have occurred in the Design Master so that they can be dispersed to any member of the replica set. The records in this table are deleted periodically to minimize the size of the table.
- *MSysSchemaProb* identifies errors that occurred while synchronizing the design of the replica. This table exists only if a design conflict has occurred between the user's replica and another member of the replica set.
- *MSysSchedule* stores information for scheduled synchronization. The Synchronizer for a local replica set member uses this table to determine when the next synchronization with another Synchronizer should take place, and how to synchronize data and design changes with the other Synchronizer.
- *MSysSidetable* identifies the tables that experienced a conflict and the name of the table that contains the conflicting records. This table is visible only if a conflict has occurred between the user's replica and another in the replica set. This is a local table and is only created at the losing replica in a conflict.
- *MSysTableGuids* relates table names to GUIDs. Table GUIDs are used in tables such as MSysTombstone as a reference to a table name stored in this table. This allows efficient renaming of tables. In addition, this table includes the level number used for ordering tables so that updates can be processed efficiently. This is a local table that is updated by the tracking layer at the Design Master and, as part of the processing of design changes, at all other members of the replica set.
- *MSysTombstone* stores information on deleted records, and allows deletes to be dispersed to other replicas. This table appears in all members of the replica set.
- *MSysTranspAddress* stores addressing information for Synchronizers and defines the set of Synchronizers known to this replica set. This table appears in all members of the replica set.
- *MSysTranspCoords* stores the display layout of the Synchronizers and replicas used by Replication Manager.

GUIDs

Just as a fingerprints distinguish one person from all other people, every object in a set of data must have a unique identifier to distinguish it from all other objects. This identifier is called a *GUID* (globally unique identifier) or *UUID* (universally unique identifier). Within a database, the primary key serves a similar purpose; it ensures every record in a table has a unique identifier.

When you replicate a database, Microsoft Jet adds several fields to each table in the database. One of those fields, s_GUID, contains a GUID that uniquely identifies a single record. A record, or row, in a database is the smallest unit of information that replication tools will manage. If you change information in a replicated database, the entire record is marked as changed, and the entire record will be updated when you synchronize the data in the table.

You can use the s_GUID field as the primary key in the database. The advantage of doing so is that it virtually eliminates the possibility of duplicate keys; the potential disadvantage is that the GUID field does not convey any meaning to the user.

If you choose AutoNumber or Number as the data type for a field, you can select **Replication ID** as the setting for the **FieldSize** property. The result is that a GUID is assigned for the record and stored in the field. If a table already includes a field with a GUID, the s_GUID field is not added when you replicate the table. Microsoft Jet uses the existing GUID instead.

GUID Generation

The question "How do I know a GUID is really unique?" is a common one. The process of generating a GUID includes numerous checks to ensure its uniqueness. GUIDs are created from:

- The network node ID
- A time value
- A clock sequence value
- A version value

Here is a sample GUID:

2fac1234-31f8-11b4-a222-08002b34c003

The hyphens make it easier to read and are used only when the GUID is displayed; they are not part of the GUID. A GUID is derived from the following components:

- *Time* is a 60-bit timestamp representing the number of 100ns ticks since Oct. 15, 1582 AD. This means time values are valid until approximately AD 3400.
- *Version* identifies the version of the algorithm used to generate the GUID.
- *Clock sequence* accounts for loss of continuity of the clock, for example when a clock is reset.

A GUID includes the following fields

<*time_low*>-<*time_mid*>-<*time_hi_and_version*>-<*clock_seq_hi_and_reserved*>-<*clock_seq_low*>-<*node*>

where:

- The *time_low* field is set to the least significant 32 bits of the timestamp.
- The *time_mid* field is set to bits 32 through 47 of the timestamp.
- The 12 least significant bits of the *time_hi_and_version* field are set to bits 48 through 59 of the timestamp. The four most significant bits are set to the 4-bit version number of the GUID algorithm being used.
- The six least significant bits of the *clock_seq_hi_and_reserved* field are set to the six most significant bits of the clock sequence. The most significant two bits are set to "0" and "'1", respectively.
- The *clock_seq_low* field is set to the eight least significant bits of the clock sequence.
- The *node* field stores the node ID. The construction of the node ID depends on whether a network card is present. If a network card is present, the node ID is retrieved from NetBIOS. The first six bytes are extracted from the synchronous adapter status NCB. This is the IEEE 802 48-bit node address.

If a network card is not installed, the node ID is set to a 48-bit number (a 47-bit random number plus 1 bit for local usage). This number is not guaranteed to be unique, even on the generating machine, but is unlikely to be duplicated on another machine. However, because GUIDs are *time + sequence* this is a reasonable approximation for a local GUID. The node ID returned is explicitly made into a multicast IEEE 802 address so that it will not conflict with a "real" IEEE 802 based node address. The LocalOnly bit contains 1 if the address was cooked up, 0 if it's a real IEEE 802 address. The 48-bit number will be composed of:

- The computer's name.
- The value of the performance counter.
- The system memory status.
- The total bytes and free bytes on drive C.
- The stack pointer (value).
- A LUID (locally unique ID).
- Whatever random data was in the node ID buffer at create time.

GUID Usage

There can be more than one field with coltypeGUID in a table, but there can be only one autogenerated field of coltypeGUID in a table, regardless of whether the table is replicable.

Autogenerated GUID fields are identified by the coltypeGUID and

JET_bitColumnAutogenerate

-or-

JET_bitPreventDelete (system field)

If an autogenerated GUID field is added to a table, GUID values are generated for all records in the table. The value of an autogenerated GUID cannot be changed or deleted.

Generations

When you convert a table to replicable format, Microsoft Jet adds a new field, s_Generation, to every replicable table in the replicated database.

The s_Generation field controls which records are sent during an exchange. When a record is modified, its generation is set to zero (0). In general, all records with generation 0 are sent during an exchange, and then the generation for the record is incremented to one more than the last generation, which now becomes the new highest generation.

When an exchange occurs, the sending replica knows the last generation sent to that specific receiving replica. Only records with generations higher than the previous generations or generation 0 are sent.

The receiving replica will not apply generations received out of sequence.

In some cases Microsoft Jet replication may determine that there are too many records to be sent in a single exchange message. In these situations, the first set of records for the exchange will be of one generation and the following sets of records will be of higher generations. Therefore, it is possible that a single exchange may contain records with different generations.

Generally there is one generation field per record. To optimize exchanges for databases that contain Memo or OLE Object fields (sometimes referred to as a BLOB, or binary large object) an extra generation field is associated with each BLOB. This generation value is set to 0, ensuring it is sent during the next exchange, only if the BLOB is modified. If other fields in a record are modified, but not the BLOB field, then the BLOB generation is not set to 0 and the BLOB is not sent with the exchange.

Lineage

The s_Lineage field is added to every table in the replicated database.

The lineage is used to determine which replicas have already received an update and also to determine the winner when conflicts occur. The lineage consists of a series of entries representing each replica that has changed this record. Each lineage entry consists of a shortened form (2 bytes) of the replica ID and a version number (2 bytes). The shortened version of the replica ID is the replica's *nickname*. The version number starts at 1 and is incremented each time the record is modified.

Design Considerations

Microsoft Jet enforces a 2048-byte and 255-field limit to any record. These limits include fields and data you add to tables yourself as well as any fields and data Microsoft Jet adds in the course of replication. When you design tables in an application you plan to replicate, make sure you allow for the fields Microsoft Jet is likely to add.

At a minimum, each record will receive GUID, s_Lineage and s_Generation fields, which collectively require 24 bytes per record. This means you should design tables with no more than 252 fields (255 maximum, minus 3) and 2,024 bytes (2,048 maximum, minus 24). This is not generally a hindrance to good design, however; very few well-designed applications use either the maximum allowable fields or bytes in a single record.

If your application uses long binary fields, replication will add an additional 4-byte field per long binary field. This is the result of an exchange optimization, whereby only long binary fields that have been modified are sent in an exchange.

Updates

Data Updates

The Microsoft Jet database engine tracks all data updates in the database. This tracking occurs only in replicated databases, and does not impact the performance of the Microsoft Jet engine for nonreplicated databases.

In general, updates are marked on a per-record basis. That is, when a record in a table is updated, the whole record is marked as changed. At the same time, the s_Generation field is set to zero and the version number in the s_Lineage field is incremented.

OLE Object and Memo fields are an exception to this rule. OLE Object and Memo fields are marked separately from the rest of the record, and are sent with the rest of the record only if the OLE Object or Memo field has changed. This is because OLE Object and Memo fields can be large, and therefore expensive and time-consuming to send over a communication link. Therefore, Microsoft Jet doesn't send them unless it's necessary.

Design Updates

Design changes must be made at the Design Master and then distributed to all other replicas.

In the rare event the Design Master is lost, you can designate any other replica as the new Design Master. This action should only be taken when you are certain that the original Design Master will never return, because a replica set with two Design Masters typically corrupts the whole database because conflicting design updates leave the database in an undefined condition.

If you need to make design changes in a replica that is not currently the Design Master, you can transfer the Design Master designation from the current Design Master to that replica.

There are situations where it is desirable to have private objects in the database that are not known to all users in the replica set. For example, one user might have a particular query or report of no interest to other users, or the application designer might want to work on a new feature privately until it is ready to distribute to other users. *Local objects* can be created at any replica for cases like these. Microsoft Jet ignores local objects during synchronization. If you decide that a local object should actually be part of the replicated application, add it to the Design Master and designate it as a *global object* by opening its **Properties** box and selecting the **Replicable** attribute. At the next exchange, the object will be distributed amongst the other replicas in the replica set.

Note If a local object of the same name already exists in a replica, Microsoft Jet will change its name to O*riginalName*_Local.

Synchronization

Synchronization is the process of exchanging information between two replicas about data and design changes in the replicated application. Updating an entire replica set is a series of synchronizations between pairs of replicas. Microsoft Jet replication implements *incremental synchronization*, meaning that only the modified data is exchanged between the replicas.

A replica can behave in one of three ways during an exchange:

- Only send changes (sometimes known as a push exchange).
- Only receive changes (also known as a pull exchange).
- Send and receive changes (a bidirectional exchange).

In a push (send-only) indirect exchange, the local replica collects any design changes it has not previously sent to the particular remote replica, and attempts to bundle these changes into a "message" before sending them. (The size of the message may be restricted by physical factors). If there are any generation-zero records, the generation counter at the local replica is incremented. The changes sent are the combination of all generations larger than the last generation sent, plus any new generations, which are recognized as having a generation of zero. Records of generation zero are updated with the new, incremented generation number.

In some cases, a request to only send or receive changes may be overridden. For example, a replica attempts to only send data changes to the Design Master, but the Design Master has design changes to be sent to the replica. The exchange of data changes will be delayed until both databases have the same design. Therefore, the Design Master will send the design changes to the replica, and then the data changes will be sent to the Design Master.

If an exchange gets lost between the remote and local replicas, Microsoft Jet's error-correcting protocol rejects the new message and requests a resend of the lost generations.

When running the Synchronizer to schedule exchanges between replicas, you may find it advantageous to disable the Windows 95 System Agents (such as Start, Programs, Accessories, and System Tools). Disk compression and defragmenting can make heavy demands upon your PC that prevent scheduled replication exchanges from completing in a timely fashion. To halt these System Agent tasks, right-click the System Agent icon in the Windows 95 taskbar and click **Suspend System Agent**.

Indirect and Direct Synchronization

Microsoft Jet replication supports two styles of synchronization: direct and indirect.

Direct synchronization is when the synchronization process has a “direct" connection to both replicas and is able to open both replicas simultaneously. Updates are immediately read and written to both replicas.

Direct synchronization is the default method used by Microsoft Jet replication and is ideal when the replicas are on a LAN. The synchronization is fast and reliable. In addition, direct replication does not require any additional configuration or processes.

Direct synchronization is usually a poor choice when replicas are physically dispersed and synchronization is over a WAN or dial-up connection. The Jet database engine is a file server (and not a client/server) and opening a database remotely results in substantial network traffic. In addition, if the remote communications connection is unreliable, the remote database can sometimes be left in an uncertain state. In other words, direct synchronization over a dial-up connection is usually slow and may result in the remote replica reporting that it is corrupted.

Indirect synchronization is when there is no direct connection to the remote replica. Instead the local replica collects its updates into a ”message” file, which is written to a predetermined location, referred to as a “dropbox folder”. At some later time, the remote replica looks at its dropbox folder, reads the message file and applies the updates. Indirect synchronization can either send the message file over the file system or Internet/intranet. You must have Microsoft Replication Manager in order to use indirect synchronization; the run-time license permits you install multiple copies of Replication Manager with a single purchase of the Microsoft Office 97 Developer Edition (ODE).

Indirect synchronization is almost always the better choice for synchronizing dispersed replicas over a WAN or dial-up connection. With indirect synchronization, you never open the remote replica over a potentially unreliable communications link. Instead synchronization occurs in two phases, with each phase being controlled by a process running locally to each replica. Since you are never opening a replica remotely, indirect synchronization is both fast and robust.

Indirect Synchronization over the File System

This synchronization method uses two independent synchronization processes, each local to its own replica. These processes are controlled by the Microsoft Jet Synchronizer, which is installed and configured by using Replication Manager.

Indirect synchronization over the file system requires both the local and remote sites to run a Synchronizer. This may not be an optimal choice for sites where there is no experienced personnel, or the performance impact is too high on, say, a laptop computer. However, you can initiate indirect synchronization from either site. An alternative is to use indirect synchronization over the Internet or an intranet, but this limits users to being able to initiate synchronization only from the remote client.

To use this synchronization method, first install Replication Manager at both the local and remote sites. If possible, create a shared folder local to each site. (Before creating shared folders for laptop users, see "Laptop Users" below.) Configure Replication Manager, making this shared folder the dropbox folder for each Synchronizer.

Important The synchronization process will bypass indirect synchronization if either of the Synchronizers can create a direct connection between the two replicas. The synchronization process always attempts to make a direct connection first, and only if the connection fails does it create a message file for a dropbox folder. To prevent direct synchronization from occurring, make

sure the replica is not stored in a shared folder; otherwise, the remote Synchronizer process will be able to read the replica directly.

The actual synchronization is an asynchronous process, with each element running independently of the other. A synchronization event is between one Synchronizer to another single, specific Synchronizer that's managing a replica in the same replica set. Synchronization is either initiated manually or via the schedule you set with Replication Manager. The initiating Synchronizer queries its replica for updates to be sent to the remote replica. These updates are written in a message file and deposited in the designated dropbox folder for the remote Synchronizer.

The remote Synchronizer will look in its dropbox folder every 10 seconds for any new message files and apply their contents to its managed replicas. The remote Synchronizer must be running in order to look in its dropbox folder for updates. When you install Replication Manager, it defaults to placing the Synchronizer shortcut in your Windows StartUp folder.

At this point, the synchronization process has only sent updates in one direction. For updates to be returned, the remote Synchronizer must also initiate synchronization and place its updates in the dropbox folder for the other Synchronizer.

Synchronization occurs in a Microsoft Jet transaction; therefore, an incomplete synchronization at a local replica will be rolled back to the state it was in before the process began. If ever the communications connection is lost while the remote message file is open, resulting in the message file's corruption, Microsoft Jet replication automatically requests a re-send the next time synchronization occurs.

When you have multiple remote sites, each with its own copy of Replication Manager and the Synchronizer, you must create a separate dropbox folder for each Synchronizer. If you attempt to share a dropbox folder among multiple Synchronizers, message files may get deleted, causing a re-send. This behavior occurs because a Synchronizer will open all the message files in "its" dropbox folder, and if the message file is not for one of its replicas, it will delete the message file.

Laptop Users Ideally the dropbox folder for a remote replica will be at the remote site. However, suppose your replica set topology is a replica at the head office and multiple replicas on remote, usually disconnected, laptops. It is unlikely that the head office Synchronizer will be able to write the message files to a dropbox folder on the laptops. Instead you must create a separate dropbox for each laptop user on the LAN at the central office. Then each laptop user connects to the office LAN by RAS and runs the synchronization process. This will open the message file over RAS, which may be slower than would be desired, but it will be faster and certainly be more robust than attempting a direct synchronization with the replica at the head office.

Indirect Synchronization over the Internet or an Intranet

Synchronization over the Internet or an intranet is a new feature in Microsoft Jet 3.5/Microsoft Access 97. When a replica and Replication Manager are both on an Internet server, users can synchronize by using a standard HTTP connection. Note, however, that you cannot schedule synchronization over the Internet with Replication Manager, because Replication Manager cannot control the communication link to your Internet service provider. As an alternative, you can write Visual Basic for Applications code to create the link to your Internet service provider, execute a synchronization, and then drop the communications link.

Many customers use synchronization over an intranet, rather than the public Internet to ensure that unauthorized Internet users will never be able to access their data. To set up an intranet for Microsoft Jet replication, you must have:

- Remote Access Service (RAS) running on a LAN to which your authorized users can connect. (Many corporations already have this facility for remote users to connect to their corporate network over a dial-up connection.)
- An Internet server, such as Microsoft Windows NT Server version 4.0 running Microsoft Internet Information Server (IIS).

All Internet and intranet synchronization message files are encrypted using Microsoft Jet's standard RSA 32-bit encryption, which prevents unauthorized reading.

The following material applies equally to synchronizing over the Internet or an intranet.

How Internet Synchronization Works

You need the following three things to use synchronization over the Internet: a client PC with a replica (R1), an Internet server that is also configured as a Microsoft Jet replication Synchronizer, and a second replica (R2) on the same PC as the Internet server.

The client PC, where R1 resides, starts the synchronization process with the following actions:

- Identifying the changes to be sent to R2 and bundling them in a message.
- Establishing a connection to the Internet server and getting the address of the server's FTP folder.
- Sending the bundled changes to the FTP address.

Once the changes are stored on the server, the client PC drops out of the process. The server continues the synchronization by notifying the Synchronizer that there is data in the FTP folder. The Synchronizer then takes the following actions:

- Picks up the data in the FTP folder and applies it to replica R2 on the Internet server.
- Queries replica R2 to see if there is any data to send back to R1.

If R2 has no changes to send to R1, the synchronization is complete at this point. If R2 needs to send data back to R1, the Synchronizer, Internet server, and client PC complete the process as follows:

- The Synchronizer places the data in the FTP folder and notifies the Internet server there is data for the client PC.
- The server sends a message to the client PC that there is data to collect.
- The client PC collects the data and applies it to the client replica, R1.

Internet synchronization works only between replicas on different machines. If you attempt to synchronize two replicas—one managed with an Internet Synchronizer and one unmanaged—on the same machine, you will get a message saying the exchange was unsuccessful because the Internet is slow or because there is a problem with the Internet server. However, if you have the same two replicas on two different computers, the Internet exchange succeeds.

Preparing for Internet Synchronization

Before you can synchronize over the Internet, you must property configure your Internet server and Replication Manager. For full details, see "Setting Up an Internet or Intranet Server for Replication" in the Help file provided with Replication Manager. If you're new to the Internet, here is the easiest way to configure everything:

1. Set up your Internet server with an operating system and server software. For example, you can use Microsoft Windows NT 4.0 with Microsoft Internet Information Server (IIS).
2. Install Microsoft Access 97 on the same computer. Install Replication Manager on the same computer. Note that Replication Manager runs on Intel server platforms only, and supports the Microsoft Internet Explorer and Netscape server platforms. Internet proxy servers are not supported.
3. Create a replica on the Internet server computer and manage it using Replication Manager. This will "stamp" the replica with information about the Internet server's HTTP address and other internal system information.
4. Create a copy of the replica (you can use the Windows Explorer to do this) and distribute this copy to your users. When they open the replica in Microsoft Access and synchronize, they will be able to synchronize back to the replica on the Internet server by using the automatically configured HTTP address.

Note Some Internet providers may require you to stop and restart Replication Manager and/or the communications connection if synchronization is canceled before it is complete.

The procedure above describes the easiest way to confirm you have set up your Internet server and your Microsoft Access application for replication. Once you have the system working, you may decide you do not need to install the full version of Microsoft Access on the Internet server, but only the run-time version of Microsoft Access that is included with Office 97, Developer Edition (ODE). If you decide to use only the run-time version, you can install it on the Internet server by using the ODE Setup Wizard to create a custom Setup program for the Microsoft Access application you want to replicate. When you're creating the Setup program, make sure you select the Microsoft Replication Manager and Microsoft Access Run-Time Version redistributable components. When you run the custom Setup program on the Internet server, it will extract only the files you need and create the correct registry settings.

Note The ODE Setup Wizard is the recommended tool for installing your application on an Internet server because it automatically updates the system for you. For example, the Setup programs created by the Setup Wizard update registry settings for you. Updating these entries manually is complex and prone to error, particularly with regard to Internet settings and versions of system DLLs that interact with other applications. If you manually copy your application and associated files to the server, you must also update system settings such as registry entries. For this reason, we strongly recommend you use the ODE Setup Wizard to create a Setup program for the application.

Special Considerations When Using CompuServe

Connections through CompuServe may require extra configuration steps. You may receive the error message "Dial-up Networking could not negotiate a compatible set of protocols you specified in server type settings. Check your network configuration in Control Panel then try the connection again." Follow these steps to correct the problem:

1. Start WinCim. Make note of your CompuServe password and CompuServe dial-up phone number.
2. Click the **Winsock** box and select **Configure**. Note the host name and, if available, the Host IP address.
3. Close WinCim.
4. If the Host IP address was blank go to an MS-DOS prompt and type **ping *Host Name***. Note the IP address returned by the system. The address is a number; for example, 149.174.214.47.
5. Double-click the My Computer icon on the Windows 95 desktop.
6. Double-click **Dial-Up Networking** in the My Computer window. Make sure CompuServe is among the icons in the Dial-Up Networking window.

   If CompuServe is not already an option, click **Make New Connection** in the Dial-Up Networking window. Type **CompuServe** as the name of the computer you are dialing. Enter the CompuServe phone number as the number to dial.

   If Dial-Up Networking is not on your computer, open Control Panel, double-click **Add/Remove Programs**, click the **Windows Setup** tab, click **Communications**, click the **Details** button, and then select the **Dial-Up Networking** check box.

The next step is to configure your Internet connection properties, as follows:

1. Right-click the Internet Explorer icon on your desktop and then click **Properties**.
2. Click the **Connection** tab. Select the **Connect to the internet as needed** check box in the **Dialing** section, and then select **CompuServe** in the list of Dial-Up Networking connections.
3. Click the **Properties** button. You should see the telephone number for CompuServe.
4. Under **Connect Using**, click **Configure**. You probably do not need to change the settings on the **General** tab.

5. Click the **Connection** tab. Under **Call preferences**, select the **Wait for dial tone before dialing** check box.
6. Click the **Options** tab. Under **Connection control**, make sure that neither the **Bring up terminal window before dialing** nor the **Bring up terminal window after dialing** check box is selected.
7. Click **OK** to return to the **General** dialog box.
8. Click the **Server Type** button. In the **Type of Dial-Up Server** box, make sure **PPP:Windows 95, Windows NT 3.51, Internet** is selected. Under **Advanced options**, make sure **Log on to network**, **Enable software compression**, and **Required encrypted password** are not selected. Under **Allowed network protocols**, make sure **NetBEUI** and **IPX/SPX Compatible** are not selected, and that **TCP/IP** is selected.
9. Click the **TCP/IP Settings** button. Make sure **Server assigned IP address** is selected. Click the **Specify name server addresses** option button and enter the IP address (for example, 149.174.214.47) that you obtained earlier as the setting for **Primary DNS**. Make sure the **Use IP header compression** and **Use default gateway on remote networks** check boxes are both selected.
10. Click **OK** four times to return to the desktop.
11. Click the **Start** button, point to **Programs**, point to **Accessories**, and then click **Dial-Up Scripting Tool**. With this tool, you can create a script to create a dial-up connection to CompuServe.

    If the Scripting Tool is not installed, open the Control Panel and double-click **Add/Remove Program**. Click the **Windows Setup** tab, and then click the **Have Disk** button. A dialog box appears with a prompt for the location of the manufacturer's files. Set this to the \Admin\Apptools\Dscript folder on your Windows 95 CD and click **OK**. In the next dialog box, select the **SLIP and Scripting for Dial-Up Networking** check box and then click the **Install** button. Click **OK** twice to install the tool and close Add/Remove Programs.
12. In the Dial-Up Scripting Tool, select the CompuServe connection you just created. Click **Browse**, then click **Cis.scp**, and then click **Open**. Make sure **Step through script** is not selected, and that **Start terminal screen minimized** is selected. Click **Apply**, and then click **Close**.

You should now be ready to go! Now, whenever you attempt a Microsoft Jet replication synchronization across the Internet, you will connect via the CompuServe Internet server. To do this, double-click the CompuServe icon in the Dial-Up Networking window. Enter your CompuServe ID (for example, 12345,1234) in the **User name** box, and then enter your password in the **Password** box. You can choose to save the password; if you do, you will not be prompted to enter it each time you connect to CompuServe.

Internet Synchronization with DAO

DAO includes a new constant, **dbRepSyncInternet**, that supports Internet synchronization. You cannot use **dbRepSyncInternet** independently; you must fully specify which type of synchronization you want to use. The following three statements illustrate how you can use this constant:

```
dbs.Synchronize "DatabasePathName", _
    dbRepImportChanges + dbRepSyncInternet

dbs.Synchronize "DatabasePathName", _
    dbRepExportChanges + dbRepSyncInternet

dbs.Synchronize "DatabasePathName", _
    dbRepImpExpChanges + dbRepSyncInternet
```

Note DAO provides **dbRepImpExpChanges** as a default constant only if the user does not specify any constants at all.

Conflicts and Errors

When two users simultaneously update data, there are three possible outcomes:

- The synchronization results in transparent updates with no errors or conflicts.
- A *conflict* occurs because the two users updated the same record.
- An error occurs because the changes made by the user cannot be resolved in synchronization.

The first case is the most common; in well-designed replicable database applications, users can simultaneously update data without any adverse side effects. However, if two or more users modify the same record at the same time, conflicts or errors can occur.

A *conflict* occurs when two users update the same record. When this happens, Microsoft Jet chooses one replica to win the conflict and one to lose. The record from the winning replica is placed in the table and replicated to the rest of the replica set. The losing record is returned to the loser, and reported in a special table called *TableName*_Conflict, where *TableName* identifies the table where the conflict occurred. The user of the losing replica is notified that conflicts exist and is prompted to either reapply the data or delete it. Conflicts do not cause the data in the replica set to diverge, and are usually a minor problem that is easily fixed.

Errors are simultaneous changes that can cause divergence in the data stored in the conflicting replicas. For example, an error occurs if users at two or more replicas simultaneously insert a new record and both records use an index value that was previously defined as unique. When the replicas attempt to synchronize, a duplicate key error will be returned.

Several conditions can cause an error:

- A duplicate primary key.
- A locked table during an exchange.
- Violation of a table-level validation (TLV) rule.
- Violation of a referential integrity constraint.

For example, a TLV rule may prevent one replica from accepting an update that was successfully inserted in another replica. Consider a replica set with only two members: the Design Master and one other replica. The owner of the Design Master creates a TLV rule on a field (such as Salary > $10,000). Simultaneously a user modifying the replica enters a new record with a value in the Salary field of $9,000, which is accepted since the TLV rule has not been sent to the replica. When the two replica is synchronized with the Design Master, Microsoft Jet attempts to insert the salary record from the replica into the Design Master. The Design Master rejects it and records an error. (Of course, you can avoid this situation by synchronizing the Design Master with all replicas before a new TLV rule is introduced.)

Errors are recorded in the MSysErrors table and replicated to all replicas. Errors must be corrected as soon as possible because they indicate that data in different replicas may be diverging.

Sometimes errors are self-correcting. For example, Microsoft Jet rejects updates on records that are locked when synchronization is attempted. Microsoft Jet records this as an error and attempts the update at a later time. Microsoft Jet removes the error if it later updates the record successfully.

Conflict Resolution

If two users simultaneously update the same record in different replicas, a conflict occurs when Microsoft Jet next attempts to synchronize these two replicas. When a conflict occurs, Microsoft Jet uses the following conflict-resolution algorithm to determine a winner and a loser:

- The replica where the record has been changed most often wins. (Each change to the record increments the record's version number, so the record with the highest version number is the one that has been changed most often.)
- If the version numbers are equal, the replica with the lowest replica ID wins.

This algorithm is guaranteed to be deterministic. There is no provision to modify this algorithm.

The winning record is placed in the table in both replicas. The losing record is placed in the loser's replica only in a "conflict table". The conflict-table is named *TableName*_Conflict (where *TableName* is the name of the table where the conflict occurred). Microsoft Access automatically invokes a wizard to assist the user in resolving entries in conflict tables.

Microsoft Jet uses a merge reconciler that merges individual records from multiple replicas into a single database; it does not use the default Windows 95 Briefcase reconciler. The default reconciler simply determines which version of the file has the later time stamp and copies that file over the version with the earlier time stamp. It has no provision for record-level conflict resolution; any updates outstanding in the losing file are lost in the process. By merging changes from multiple replicas, the Microsoft Jet reconciler retains changes from the losing replica rather than simply overwriting them with data from the winning replica.

You can enhance the Microsoft Jet algorithm with your own Visual Basic for Applications function. Microsoft Jet will still initially resolve conflicts using its own algorithm but you can use your code to manipulate the results. You must register your function as a database property. To do this, click **Database Properties** on the **File** menu, and then click the **Custom** tab. In the **Name** box, type **ReplicationConflictFunction**; in the **Type** box, select **Text**; and in the **Value** box, type the name of your function; for example, **Resolve()**.

Here is a simple function to display the name of any table with a conflict in the Debug window:

```
Public Function Resolve()     'Find tables with conflicts.
    Dim dbs As Database
    Dim tdf As TableDef

    Set dbs = CurrentDb
    For Each tdf In dbs.TableDefs
        If (tdf.ConflictTable <> "") Then
            Debug.Print tdf.Name & " had a conflict"
        End If
    Next tdf
End Function
```

To view the Debug window, press CTRL+G.

A more complete example resolves conflicts based upon the value in a specific field, named Date. In this example the latest date wins:

```
Public Function Resolve_Conflicts()     ' Resolve conflicts for DB.
    Dim dbs As Database
    Dim tdf As TableDef
    Dim rstWin As Recordset       ' Recordset for conflict winner.
    Dim rstConflict As Recordset  ' Recordset for conflict loser.
    Dim fld As Field

    ' Open the database & tables.
    Set dbs = CurrentDb
    For Each tdf In dbs.TableDefs
        ' Does an associated conflict table exist?
        If (tdf.ConflictTable <> "") Then
            ' Get the records from the conflict table.
            Set rstConflict = dbs.OpenRecordset("SELECT * FROM "_
                & tdf.ConflictTable & " ORDER BY s_GUID")
            ' Get the records from the base table.
            Set rstWin = dbs.OpenRecordset("SELECT * FROM " & _
                tdf.Name & " ORDER BY s_GUID")
            ' Process each record, starting at the first.
            rstWin.MoveFirst
            rstConflict.MoveFirst
            While Not rstConflict.EOF
            ' If date in the conflict record is > date in
            ' winning record, resubmit the conflict record.
                If rstConflict("s_GUID") = rstWin("s_GUID") Then
                    If rstConflict("Date") > rstWin("Date") Then
```

```
                    rstWin.Edit
                    For Each fld In rstConflict.Fields
                        rstWin(fld.Name) = rstConflict(fld.Name)
                    Next fld
                    rstWin.Update
                End If
                ' Remove conflicting record & clean up.
                rstConflict.Delete
                rstConflict.MoveNext
            End If
            rstWin.MoveNext
        Wend
        rstWin.Close
        rstConflict.Close
    End If
  Next
End Function
```

You can customize this function to resolve conflicts according to your own business rules. For example, if your application includes a special date/time field that is always updated when a record is inserted or edited, your Visual Basic for Applications function could use this field to select the record with the most recent update as the winner. However, your code will replace the Conflict Resolver provided by Microsoft; therefore your function must resolve all errors and conflict situations. It is not sufficient to resolve conflicts and ignore errors; the net result will be inconsistent replicas with unresolved errors.

Partial Replicas

A *partial replica* is one that contains a subset of data. Support for partial replicas is a new feature in Microsoft Jet 3.5/Microsoft Access 97. You can use either the Partial Replica Wizard or Visual Basic for Applications code to create a partial replica. The Partial Replica Wizard is available on the Internet; visit the Microsoft Office Developer Forum Web site at http://microsoft.com/officedev/ for more details.

A WHERE clause on a table defines a partial replica. For example, to create a partial replica with customers from California, you would specify "Region = 'CA'". You cannot use user-defined or aggregate functions, nor can you prompt the user at run-time for parameter values. By using Visual Basic for Applications, you can apply restrictions to any number of tables. The Partial Replica wizard limits you to placing restrictions on a single table.

You must enclose date variables with the number sign (#). For example, to select orders placed after March 31, 1995, and before December 31, 1996, enter the following:

```
[Order Date] > #3/13/95# AND [Order Date] < #12/31/96#
```

You must surround the contents of Text and Memo fields with quotation marks. For example, to specify Jane Doe's name, enter the following:

```
[FirstName] = "Jane" AND [LastName] = "Doe"
```

To enter numeric values, use the field's name and the value. For example, to enter 1 as the category ID, enter the following:

```
[CategoryID] = 1
```

Note Multiple filters are OR'ed together. For example, if you set a filter of "Region = 'CA'" for the Customers table and a filter of "Value > $1,000" for the Orders table, the result will include all records for customers from California PLUS all orders with a value greater than $1,000. You would *not* get orders of over $1,000 only for customers from California.

The **PartialReplica** property lets you specify which relationships to follow when creating a partial replica. For example, to select only the orders for the customers in California, set the **PartialReplica** property for the CustomersOrders relationship to **True**.

With Visual Basic for Applications, you can create more sophisticated partial replicas by applying restrictions to any number of tables. The following example shows how you can create a partial

replica, set a filter on a table, set the **PartialReplica** filter property, and populate the partial replica with data:

1. Create a partial replica. This will be an empty database with no filters, and no data records.

```
Public Function CreatePartialReplica()
    Dim dbsFull As Database     ' The source replica

    Set dbsFull = OpenDatabase("C:\NWIND.MDB")
    ' Create partial replica from full replica, description,
    ' set the dbRepMakePartial option.
    dbsFull.MakeReplica "C:\PARTIAL.MDB", _
       "Replica of NWIND.MDB", dbRepMakePartial
    dbsFull.Close
    ' You have a partial replica, but no data records.
End Function
```

2. Set a filter to select customers from Washington state.

```
Public Function CreatePartialFilter()
    Dim dbsPartial As Database
    Dim tdfCustomers As TableDef

    ' Open the partial replica in exclusive mode.
    Set dbsPartial =OpenDatabase("C:\PARTIAL.MDB", True)
    ' Open the Customers table.
    Set tdfCustomers = dbsPartial.TableDefs("Customers")
    ' Set a filter on the Customers table.
    tdfCustomers.ReplicaFilter = "Region = 'WA' "
    dbsPartial.Close
End Function
```

3. Set the **PartialReplica** property on the CustomersOrders relationship to only get the orders for records in the filtered Customers table.

```
Public Function CreatePartialRelationships()
    Dim dbsPartial As Database
    Dim Rel As Relation

    ' Open the partial replica in exclusive mode.
    Set dbsPartial = DBEngine(0).OpenDatabase("C:\Partial.MDB", True)
    For Each Rel in dbsPartial.Relations
        If (Rel.Table = "Customers") And _
            (Rel.ForeignTable = "Orders") Then
            ' Set CustOrd relationship PartialReplica property to
            ' True.
            Rel.PartialReplica = True
        Exit For
        End If
    Next Rel
    dbsPartial.Close
End Function
```

4. Populate the partial replica from a full replica.

```
Public Function PopulatePartial()
    Dim dbsPartial As Database
    Dim strFull As String

    ' Open the partial replica in exclusive mode.
    Set dbsPartial = OpenDatabase("C:\PARTIAL.MDB", True)
    ' Point to the full replica.
    strFull = "C:\NWIND.MDB"
    ' Populate the partial replica from the full replica.
    dbsPartial.PopulatePartial strFull
    dbsPartial.Close
End Function
```

You should call **PopulatePartial** under any of the following circumstances:

- Before synchronizing a full and a partial replica for the first time. This ensures the partial replica has all the system information required. If you simply create an empty partial replica and do not call **PopulatePartial** before synchronizing, you will see the message that "the filters are not synchronized" even if no filters are set on the partial replica. **PopulatePartial** is required to initialize the replica with internal system information before the first synchronization.

  Note that **PopulatePartial** works only with direct connections; it does not support indirect synchronization such as dropbox (file transfer) synchronization, Internet synchronization, or FTP folder synchronizations.

- Whenever you modify the filter for a partial replica. **PopulatePartial** removes any records from the partial replica that do not comply with the new filter. Simply using the **Synchronize** method after a change of filters will not remove these "orphaned" records.

Synchronizing between full and partial replicas may require careful attention. Say you create a replica set with three members: Full_1, Partial_1, and Full_2. Now assume Full_1 synchronizes with Partial_1, and because Partial_1 has a filter, it only gets a subset of the changes made at Full_1. Now Partial_1 synchronizes with Full_2. Obviously only the subset of the changes stored in Partial_1 can be sent to Full_2. It would be a mistake to assume that Full_2, having successfully completed the exchange, has all the updates from Full_1. The only way to guarantee that Full_2 has all the changes is to synchronize with another full replica. When synchronizing full and partial replicas, the best process is to treat partial replicas as 'leaf' nodes. That is, designate them as the end of a synchronization chain. This also increases the efficiency of synchronizations, because the protocol that ensures correct propagation of updates between partial and full replicas may result in redundant data exchange.

Partial replicas introduce a number of subtleties for deleting or updating records when referential integrity and cascading updates or deletes are used in the same application. Consider what might occur in a simple database with two tables: Customers and Orders. Referential integrity is enforced between these tables, and cascading updates or deletes are not enabled. Consider Customer A who has Orders records in the full replica. Now assume there is a partial replica with only the Customers table. If Microsoft Jet allowed the records pertaining to Customer A to be deleted at the partial replica, then this delete would fail when it was sent to the full replica, because there would be existing records for Customer A.

To prevent this, Microsoft Jet traps attempts to update or delete primary keys in the parent table in partial replicas, and permits them only if

- The parent table has no children.

Or if, for each child table:

- Any **PartialReplica** property of any enforced relationship between the parent and child is set to **True**.

-or-

- The child table's filter is set to **True** (which indicates all records in child table are present).

For partial replicas, Microsoft Jet looks only at the immediate child table to decide whether to permit an update or delete. However, it's important to remember the effect that cascading updates and deletes can have. Consider an example of three related tables (Customers, Orders, and OrderDetails). If cascading updates and deletes are enabled, an attempt to update a Customer record in a partial replica will also attempt to update any Order records, which in turn will attempt to update any OrderDetails records. If cascading updates and deletes are not enabled, an update to a Customer record would check the Order records and ignore the OrderDetails records.

Compact and Repair Utilities

Microsoft Jet provides two utilities for database maintenance: compact and repair. These utilities have special implications for replicas.

You are advised to compact your replicas before synchronizing them. In fact, there is usually a benefit in compacting twice before synchronizing. The first time you compact a replica, the compact utility reclaims unused pages on disk, making the database smaller. In addition, for replicated databases, the utility looks at the list of design changes that would be sent to a remote site and deletes those changes that are no longer relevant. These irrelevant changes usually occur where, say, you have made numerous modifications to a form, report, or code module object. Microsoft Jet replication keeps a copy of each changed version of an object, whereas only the latest version is actually used. When you run the compact utility the first time, these old versions of an object are marked as "no longer needed." When you run the compact utility the second time, these pages are reclaimed from the disk.

You should not run the repair utility on a replicated database since it will not only repair your database but also mark it as no longer replicable. There is no way to make the database replicable again. If you are notified that a replica is corrupt and that you should run the repair utility, it is best to create a new replica from another clean replica and discard the corrupted replica. The repair utility makes the database no longer replicable because if any of the replication system data was corrupted and then repaired in an incomplete manner, you wouldn't want to propagate improperly repaired data to the other replicas in the replica set.

Clean Up of Old Replicas

If you have created temporary replicas during development, you will need to take an extra step to remove them from the replica set before distributing it to your users. If you merely delete the temporary replicas, the replica set will still contain references to them; for example, the deleted replicas may still appear in the Synchronize Now dialog box in Microsoft Access.

To remove all references to these replicas, delete them and then attempt to synchronize with them even though they no longer exist. For example, in Microsoft Access, point to Replication on the Tools menu, and then click Synchronize Now. In the Synchronize Now dialog box, select the name of a replica that you deleted. When Microsoft Access tells you that the replica can't be found, it will ask if you want to remove the replica from the list. Click Yes. The replica will be marked internally as "Removed" and will no longer appear in the dialog box.

Security

Replicated databases use the same security model as nonreplicated database. The permissions assigned to a user's logon ID control the actions that user can take on the database.

The application designer must ensure that the same security information is available in each replica. There are two ways to do this:

- Make the *exact* same security file, System.mdw, available to each replica. The security file can not be replicated, but it can be physically copied to each location.
- Recreate the entries for users and groups at each location in the *local* security file. To do this, copy the user and group names and associated PIDs from System.mdw into the local file. Make sure to copy the entries exactly.

Microsoft Jet 3.5 includes a new security permission, Administrator, for a database object. By default, this permission is granted to the Users and Admins groups, as well as to the user logged on when a new database is created or converted from an earlier version of Microsoft Access. It is up to the application developer to restrict this permission to selected users if security is to be enforced.

A user with Administrator permission can do the following in the Design Master:

- Convert a nonreplicable database into a replicable database.
- Execute the **Move Replica** command for the Design Master.
- Make a local object replicable, or make a replicable object local.
- Modify the retention period.

In addition, a user with Administrator permission can execute the **Recover Design Master** command from any replica.

Note Make sure there is always at least one user with Administrator permission on the database. If the Design Master and the associated System.mdw file are destroyed (for example, through a hard disk failure), it is possible to designate another replica as the new Design Master—but only a user with Administrator permission can do this.

Registry Entries

Parameters for Microsoft Jet 3.5/Microsoft Access 97 replication components are stored in the system registry. In addition to the entries listed here, the registry includes many minor entries for the Synchronizer, such as the log file location, the last viewed replica, the security database, and so on, under the following key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\3.5\Transporter

Briefcase Reconciler

HKEY_CLASSES_ROOT\CLSID\{ClassID of Access.Database.8} \Roles\Reconciler = {DB5B8C90-7B62-11CF-A9E4-00AA00B676FC}

HKEY_CLASSES_ROOT\CLSID\{ClassID of Access.Database.8} \Roles\NotifyReplica = {DB5B8C90-7B62-11CF-A9E4-00AA00B676FC}

HKEY_CLASSES_ROOT\CLSID\{DB5B8C90-7B62-11CF-A9E4-00AA00B676FC}
\InProcServer32 = "full path of msRCLR35.dll"
\ResourceDll = "full path of msRECR35.dll"
\SystemDb = "full path of system.mdw"
\ThreadingModel = "Apartment"

HKEY_CLASSES_ROOT\CLSID\{ DB5B8C90-7B62-11CF-A9E4-00AA00B676FC}
\SingleChangeHook

Replication Manager Registry Entry

HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\3.5\Replication Manager
"Path" *path to Replman.exe*

Synchronizer Registry Entry

HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\3.5\Replication Manager\3.5
"SynchronizerPath" path to mstran35.exe

For More Information

Here is a list of publications you can refer to if you want more information about database replication, the Microsoft Jet database engine, or using replication in Microsoft products.

- *Microsoft Jet Database Engine Programmer's Guide,* Microsoft Press. 1995 ISBN 1-55615-877-7. A detailed analysis of the Microsoft Jet database engine, including extensive information on Microsoft Jet replication.
- *Building Applications with Microsoft Access 97*, a thorough guide to creating and distributing Microsoft Access applications.

Bidirectional exchange—An exchange between two replicas, in which both replicas send and receive updates.

Cascade update/cascade delete—Referential integrity options which specify that changes or deletions to the primary key in one table will be propagated to any other tables that reference that primary key value.

Conflict—An attempt by two replicas to simultaneously update the same record in a table. One replica "wins" the conflict and its value is propagated to the rest of the replica set. The losing replica is notified that it lost and is offered the chance to resubmit its update.

Data Access Objects (DAO)—The programming language-independent object-based data access language used to manipulate Microsoft Jet engine data.

Database engine—A program, or part of a program, that serves as the link between a database management system (DBMS) or application and the data. Specifically, the part of a DBMS program that reads and writes data records.

Design Master—The single member of a replica set in which design changes may be made for propagation around the whole replica set.

Error—In some situations an update received from one replica causes an error in the receiving replica (for example, it violates a unique key rule, or a TLV rule). In these cases, an error is registered and the user must manually correct the error.

Exchange—The process of sending design and data updates between replicas.

Foreign key—A reference to the primary key in another table.

Generation—A counter, specific to each replica, which is incremented each time an exchange occurs with any other replica.

Global object—An object that is replicated around the replica set. Global objects can only be created in the Design Master.

GUID—Globally unique identifier. A primary key in a database that is always guaranteed to be unique. Also known as a UUID (universal unique identifier).

Lineage—A record of each nickname/generation pair. Used to optimize exchanges between replicas and resolve conflicts.

Local object—An object that is not replicated around the replica set. A local object can be created in any replica.

Method—An action that can be applied to an object. For example, to move to the first record of a recordset, you apply the **MoveFirst** method to the **Recordset** object.

Multi-master—The ability to modify any data at any replica.

Nickname—A shortened name for of the **ReplicaID** property's GUID value, used in the lineage.

OLE—A standard method of linking and embedding objects created by one program to another program. Also, a type of field in Microsoft Jet used to store complex objects created by other programs.

Objects—A package of things created and manipulated by programs. In Microsoft Jet, Tables, Users and Query Definitions are all examples of objects.

Open database connectivity (ODBC)—A standard way to connect to and read and write records from other databases, usually server databases.

Primary key—A field (or multiple fields) in a table that ensures that a record can be uniquely identified.

Property—An attribute of an object that can be retrieved and (sometimes) set. For example, the **Index** property of a table can be set to the name of one or more fields in the table.

Pull exchange—An exchange between two replicas where one replica only receives, or pulls, update from the other replica.

Push exchange—An exchange between two replicas where one replica only sends, or pushes, its update to the other replica.

Read-only replica—A replica that is not permitted to update either data or the design.

Referential integrity—A relational database rule that requires that all foreign key values reference valid primary key values.

Replica—A special copy of a database that is created in such a way to allow changes made in the replica to be exchanged at a later time with other replicas in a replica set, eventually bringing all the replicas in the replica set into a consistent state.

Replication—The process of creating special copies of a database, where the copies have a special relationship to each other.

Replica ID—A GUID that uniquely identifies a replica.

Replica set—Replicas that share a common heritage and are able to synchronize their data and design. Replicas can synchronize only with other replicas in the same set.

Retention period—The amount of time, measured in days, that a replica set retains details of deleted records, design changes, and other system-specific information. This value can be modified only by opening the Design Master in Microsoft Replication Manager and changing the value.

Synchronization—The process of bringing two replicas into a consistent state.

Transaction—A sequence of actions that must occur as a single unit.

Transaction processing—A mode of database processing that supports the creation, and saving (**CommitTrans**) or undoing (**Rollback**) of transactions.

Two-phase commit protocol—A system used in some distributed databases systems whereby each database either agrees to or rejects a proposed change, and only if every database in the system agrees is the change actually made.

Validation rule—An expression that can be linked to a change of data so that it is always evaluated when a certain type of data modification is made.

Part No. 098-54938