Under the Microscope: Domino Replication

by Bret Swedeen (with Bruce Kahn and Mike O'Brien)

[Editor's note: This article resides in "Iris Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino and Notes.]

Replication is at the heart of the distributed nature of Domino and Notes. This unique feature is often imitated, but never duplicated, and it's helped to put Notes head-and-shoulders above the groupware crowd.

When it comes to replication configuration and general usage, you can refer to the **Domino Administration Help** and **Notes Help**. If, however, you want to know more about the process of replication, you have to go beyond the product documentation. You've got to look at the components of replication, and analyze the process to really understand what's going on behind the scenes. And that is exactly what we'll do in this article as we put Domino replication *Under the Microscope*.

Synching up -- simplifying replication

Before we jump into the nitty gritty details of replication, let's make sure we understand the core functionality of this feature.

Replication is easily understood when you compare it to the traditional file copy process, which is an all-or-nothing operation. For example, imagine you have a discussion database on Server A that also needs to exist on Server B. To accomplish this task, you copy the database from Server A to Server B, and you're done. You have a problem, however, once users start adding documents to either copy of the database -- they're no longer in sync. You could re-copy the database from Server A to Server B, but that would wipe out all of the changes made to the database on Server B. So, how do we maintain a single database in multiple locations in a synchronized fashion? The answer is replication.

Apply replication to the previous example, and suddenly, database synchronization is possible. Instead of copying the database from Server A to Server B, you create a new "replica" on Server B. The next time replication occurs between these servers (automatically, using Connection documents; or manually, at the server console or using menu options found in the Notes client), the source database and its replica exchange new and modified information (this can include an entire document or certain fields in a document). This approach is different than the one-sided, all-or-nothing file copy operation. It's as if Server A said to Server B, "Give me all of your new and modified information since the last time we spoke, and I'll give you mine."

This explanation of replication is extremely simplified (especially for an *Under the Microscope* article); however, the benefits and power of replication are quickly realized when we look at it in this manner.

The components of replication

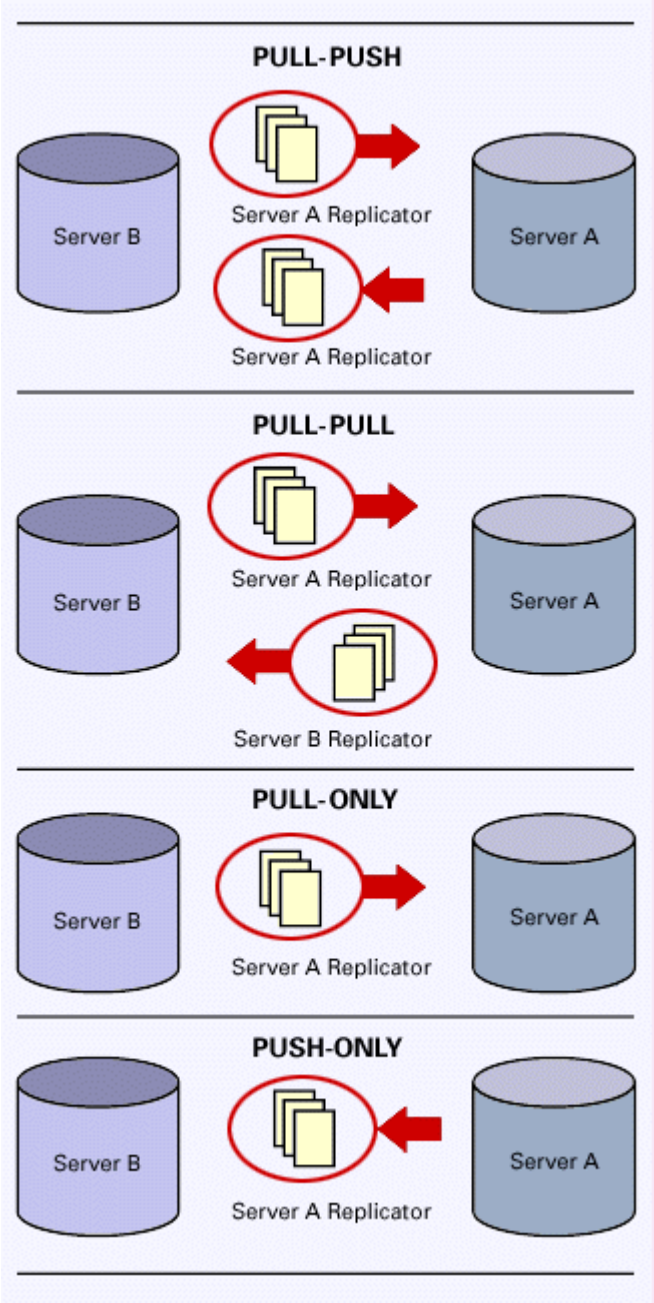
Something as powerful as replication might conjure up images of a complex series of server tasks working hard to maintain database synchronization. Surprisingly enough, replication occurs with just one server task. While you can have multiple instances of this one server task running, you really only need one. After this one server task, you're left with the pieces that help control and schedule replication. Here's a quick list:

The Replicator server task: This task appears on the server console as REPLICATOR, and does the work of synchronizing a source database with its replicas. By default, one instance of this task is loaded when the server is started; however, you can add a setting to the NOTES.INI file (REPLICATORS=*n*), or use a server console command (LOAD REPLICA) to enable multiple replicators. This feature allows a server to replicate its databases with more than one server at a time. For example, if a server has REPLICATORS=2 set in the NOTES.INI, then it can replicate its databases with two servers

simultaneously. If, however, you enable two replicators, and the server only needs to replicate with one other server, then only one replicator is used. The other Replicator task remains idle.

Connection documents: These documents are in the Public Address Book, and allow you to schedule replication between two servers (an exact time or a range of time). Along with controlling the replication schedule, a Connection document also controls which databases replicate and the type of replication they use. The following is a list of the different types of replication:

- **PULL-PUSH**: Server A contacts Server B and PULLS all new and modified information. Once Server A completes the pulling process, Server A PUSHES all of its new and modified information to Server B. The Replicator task on Server A does all of the work.
- **PULL-PULL**: Over Local Area Network (LAN) connections, Server A contacts Server B and PULLS all new and modified information. Then, Server B PULLS all new and modified information from Server A. With this type of replication, both servers "share" the work of replication. Over modem connections, however, Server A and Server B PULL simultaneously. This approach helps maximize the usage of the low bandwidth connection.
- **PUSH-ONLY**: Server A contacts Server B and PUSHES all new and modified information to Server B. The Replicator task on Server A does all the work in this one-way operation. With this replication type, nothing new or modified on Server B is pulled or pushed to Server A.
- **PULL-ONLY**: Server A contacts Server B and PULLS all new and modified information from Server B. The Replicator task on Server A does all the work in this one-way operation. With this replication type, nothing new or modified on Server A is pulled or pushed to Server B.



Server console commands: While most administrators use scheduled replication, there may be times when you need to manually initiate replication. In these situations, you can use one of three possible server console commands:

1. REPLICATE *server name*:
This command initiates a PULL-PUSH replication with the specified server. For example, let's imagine you type:

REPLICATE Server B/Acme

(where Acme is the Notes domain) at Server A's server console. This command starts the Replicator task on Server A, which begins a PULL-PUSH replication session with Server B. All of the replica databases in

common replicate during this session. If you only want to replicate a specific database, such as the Public Address Book, you can type:

REPLICATE Server B/Acme NAMES.NSF

2. PULL *server name*:
This command initiates a PULL replication with the specified server. For example, let's imagine you type:

PULL Server B/Acme

(where Acme is the Notes domain) at Server A's server console. This command starts the Replicator task on Server A, which begins a PULL replication session with Server B. All of the replica databases in common replicate in a one-way pull during this session. If you only want to pull changes from a specific database, such as the Public Address Book, you can type:

PULL Server B/Acme NAMES.NSF

3. PUSH *server name*:
This command initiates a PUSH replication with the specified server. For example, let's imagine you type:

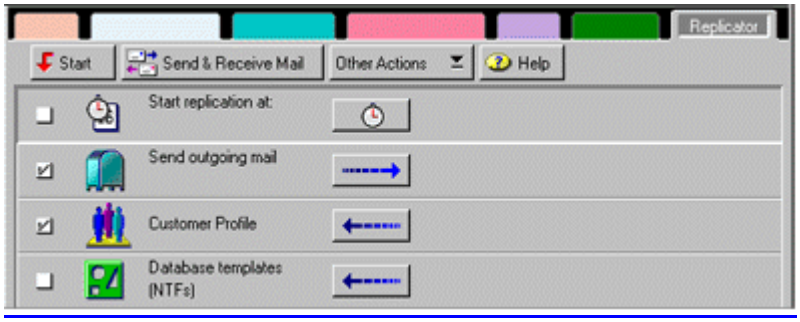
PUSH Server B/Acme

(where Acme is the Notes domain) at Server A's server console. This command starts the Replicator task on Server A, which begins a PUSH replication session with Server B. All of the replica databases in common replicate in a one-way push during this session. If you only want to push changes from a specific database such as the Public Address Book, you can type:

PUSH Server B/Acme NAMES.NSF

Note: There is no console command equivalent to the Pull-Pull replication type.

The Replicator page: This component is the last workspace page of the Notes client. Normally, you use the Replicator page during client-to-server replication sessions, such as receiving e-mail. On the Replicator page, you can schedule, customize, and launch a replication session with a server. Detailed usage instructions for the Replicator page appear in the **Notes Help.**



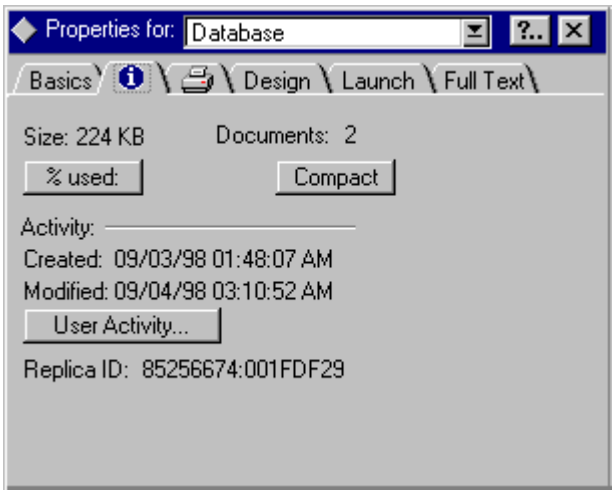
Replication step-by-step

Now that we've summed up what replication is, and listed the components of replication, we can finally explore the process of replication. This section walks through a simple replication scenario between two servers: Server A and Server B.

The replication session is triggered by a scheduled connection. The two servers authenticate, look for matching database replica IDs, and proceed to replicate. The details of what takes place during this session are listed in the following steps:

1. The Replicator task on Server A sits idle until the Connection document triggers it to start.
2. Once it's time to replicate, Server A contacts Server B.
3. To make sure these servers have access to one another, they authenticate each other in much the same way that servers authenticate users.
4. Once authentication is complete, each server, using the Access Control List (ACL) as a filter, builds a list of databases (for details about the ACL, see the *Iris Today* article **The ABC's of using the ACL**).
5. Since Server A initiated the session, it compares the lists and looks for matching replica IDs.

   Note: Databases that are replicas of one another share the same replica ID. You can view this information by accessing the Information (i) tab of the Database Properties dialog box.



6. In this scenario PROFILES.NSF is the only database that has a common replica ID on both servers. Therefore, Server A asks Server B for a list of changes since their last replication.
7. Server B responds with a list of all the new and modified Universal Note IDs (UNIDs). At this point, Server B also sends a "bookmark" for Server A to use the next time they replicate. Server A stores this "bookmark" as part of the replication information. In the next replication session, when Server A asks for a list of new and modified UNIDs, Server A also sends the "bookmark" to Server B. Server B uses the time-stamp contained in the "bookmark" as a starting point from which to create the list of new and modified UNIDs.

   Note: The UNID is a totally unique 16-byte identifier for a document. The UNID binds together the replicas of a given document. Each instance of the document has the same UNID -- regardless of how many replicas the document appears in. The UNID consists of 8 bytes of time/date-created information, and a 32-bit sequence that is a unique number indicating the document's position in the database.

8. Once Server A receives a list of changes, it builds a similar list from its replica copy based on the same time-stamp.
9. Server A moves sequentially through the lists, and compares each UNID in the list from Server B to its own list. If an UNID exists in the list from Server B but not in Server A's list, the document is PULLED

over. If, however, a UNID appears in both lists, we might have a conflict. Server A checks for and handles conflicts by executing the following steps:

a. Two documents are in conflict if they either have diverged at some point, and changes have been made past the point of divergence independently to multiple copies. The Replicator on Server A examines the two documents to see if they have diverged by looking at the Originator ID (OID) and the revisions history ($Revisions item). The OID contains the sequence number and the last modified time. If the sequence numbers are different, Server A takes the note with the latest sequence number (or if they are the same, then it picks one). Server A then looks back in its revision history for the entry that corresponds to the other document's sequence number. Server A takes this time and compares it to the other document's OID modified time. If they are different, there is a conflict.

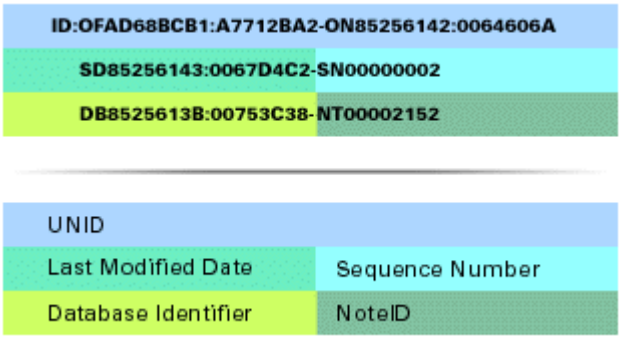
b. Next, Server A must determine a winner or loser (knowing that there is a conflict isn't enough). Server A makes this determination by locating the latest sequence number of either document. If the sequence numbers are the same, the latest modified time breaks the tie.

c. Before deeming one document the winner and the other one a loser, Server A attempts to merge the conflicts. If the form used for the document was designed with the "Merge Replication Conflicts" enabled, Server A attempts to merge the conflicts. Server A checks the individual fields for conflicts. If the modified fields are different, then Server A merges the changes between the two documents. The conflict is resolved, and Server A's Replicator changes BOTH the source and destination documents.

d. If Server A cannot resolve the conflict, it creates a conflict document. The server that contains the losing document creates the conflict document. Only one server creates the conflict document so both servers don't duplicate each other's efforts. The conflict document is a new document with an UNID algorithmically derived so that if this conflict was also generated somewhere else, it would have an identical OID. Remember, the server that has the losing document is the server that creates the conflict document. Therefore, you might not see the conflict document appear on both servers if you used a one-way replication type. For example, if Server A has the losing document when it's doing a PULL-ONLY from Server B, the conflict document only appears on Server A. Understanding this nuance of conflict handling is very important.

Note: The Originator ID, also known as the Document Version ID, describes a particular version of a document. As new versions of a document appear, this ID changes. The ID consists of four components:

--Database identifier (not the same as the replica ID)
--Time of creation of the document (last part of UNID)
--Time of last modification, a sequence number
--The 28-byte OID, which is the UNID plus a sequence number and a sequence time/date stamp



10. Once Server A PULLS all of the new and modified information from Server B, Server A sends Server B a "bookmark" for future replication sessions.
11. The roles reverse and new lists of new and modified UNIDs are created and exchanged.
12. Server A moves sequentially through the lists, and compares each UNID in its list to the list from Server B. If a UNID exists in Server A's list but not in the list from Server B, the document is PUSHED over. If, however, a UNID appears in both lists, we might have a conflict, which is handled in the same way as described in Step 9 (For information about what you can do to prevent and resolve conflicts, see the **"Preventing and Resolving Replication Conflicts"** sidebar).

Surprisingly enough, that's what takes place during replication. With something so powerful, you might have expected something much more mind boggling; however, replication is quite logical. This explanation even describes the other types of replication. Here's a quick summary:

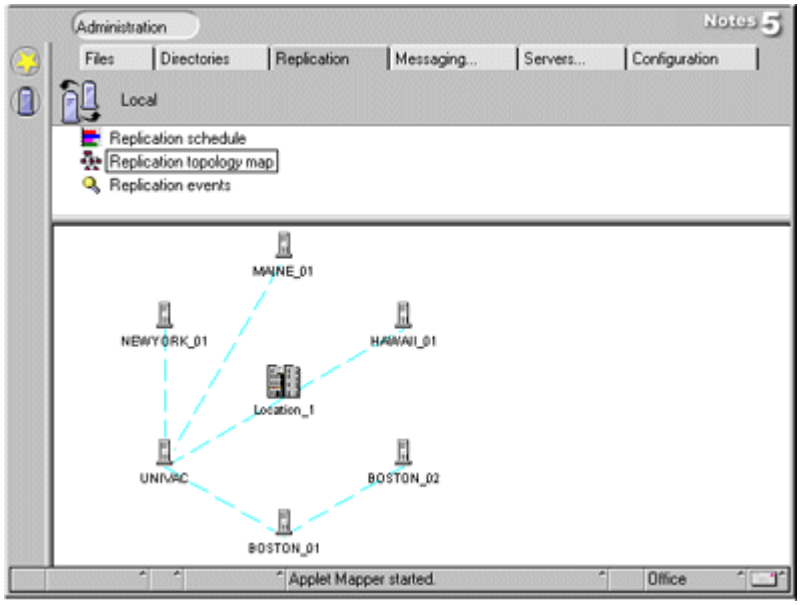
PULL-PULL: Each server takes its turn at executing Steps 1 through 9 (except over modem connections, where PULLING occurs simultaneously on both servers).

PUSH-ONLY: The initiating server executes Steps 1 through 9, except Step 9 is more like Step 12.

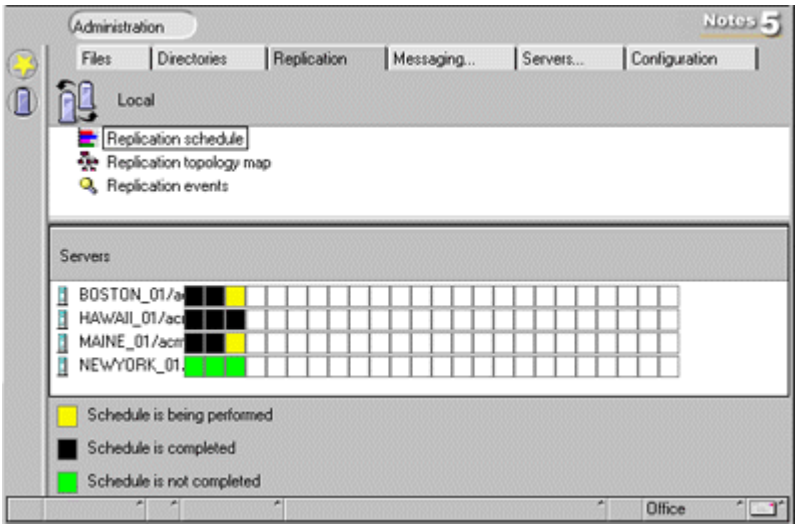
PULL-ONLY: The initiating server executes Steps 1 through 9.

What about Release 5?

What does the future hold for replication? Better administration tools such as a graphical replication topology map, which allows you to easily view your environment's replication topology. Now, instead of needing to map out your replication topology in a separate drawing application, the new Domino Administrator client does it for you. Load the server task MAPS, and presto -- you can see the replication topology defined by your Connection documents.



Along with the new replication topology map, there is a graphical replication schedule map. This new tool allows you to quickly view your replication schedule and the status of various replication events.



In addition to these administrative enhancements, replication remains a key component behind the unique power of the Domino, which continues to offer you an innovative way to distribute and disseminate information.

Preventing and resolving replication conflicts (sidebar)

Despite the sophisticated logic Domino uses to avoid replication conflicts, you can do the following things to help further prevent conflicts:

- Exploit the power of the Access Control List (ACL). Assign users Author access or lower, instead of Editor access. Limiting the number of Editors in a database reduces the possibility of someone accidentally editing another user's document.
- During form design, select the Merge replication conflicts option in the Properties InfoBox. This option allows the Replicator to automatically merge conflict documents if no fields conflict.
- Specify a versioning option when you create a form. Then, edited documents automatically become new documents.
- If you're LotusScript savvy, you can create a program that uses $ConflictActionItem to merge documents automatically if no fields conflict.
- You can also create a custom conflict handler using LotusScript. This handler allows you to handle the conflict, or let the conflict be handled automatically.

For information on designing forms, see the **Application Developer's Guide**. For information on using LotusScript, see the **Lotus Notes Designer for Domino 4.6 Programmer's Guide Part 1**, the **LotusScript Language Reference**, and the **LotusScript Programmer's Guide**.

Resolving conflicts

Despite your best efforts at preventing replication conflicts, they're bound to happen. When you're faced with such a situation, you can do one of the following:

Option 1. Delete the "loser" but save its information

1. See if the loser contains any information that should be added to the winner document. If so, copy the information and paste it into the winner document.
2. Delete the loser (the response document with a "diamond" next to it).

Option 2. Delete the "winner" and save the "loser"

1. Open the loser, switch to edit mode, and then save the document. This removes the document's [Replication or Save Conflict] status (removes the black diamond) and promotes it to a main document.
2. If the original winner had any response documents, select them and choose Edit - Cut.
3. Highlight the new winner (the document you chose in Step 1) and choose Edit - Paste. This makes the responses you cut in Step 2 into response documents of the new winner.
4. Delete the old winning document.

ABOUT BRUCE

Bruce Kahn is a senior software engineer at Iris, and has been working on the Notes calendar and scheduling (C&S) features for more than two years now. He's also spent much of that time working to help develop the new IETF C&S standards of iCalendar, iTIP and iMIP. Bruce's primary areas of responsibility are back-end C&S (the SchedMgr), evolving Internet standards for C&S and LDAP and, now, the POP3 server. He spends what free time (:^}) he has outside of Iris as the exhausted father of toddler twins, a Red Cross First Aid, CPR and Disaster Instructor and Volunteer, an amateur magician and a devoted Dilbert fan.

ABOUT MIKE

Mike O'Brien has been at Iris for almost five years. Among his recent credits are extensive work on replication and LDAP. Currently, he is deeply involved with project coordination and management of R5 work that will enhance Domino/Notes search capabilities.