1: Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working
documents of the Internet Engineering Task Force (IETF), its area, and
its working groups. Note that other groups may also distribute working
documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference material
or to cite them other than as ‘work in progress.’

To learn the current status of any Internet-Draft, please check the
‘id-abstracts.txt’ listing contained in the Internet-Drafts Shadow
Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe),
ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

This internet draft expires September 20, 1997.

2: Abstract

This paper defines a multi-master, incremental replication protocol for
the LDAP protocol [LDAPv3]. It defines the use of two types of transport
protocols for replication data, and specifies the schema which must be
supported by a server which wishes to participate in replication
activities using this protocol.

3: Introduction

LDAP is increasing in popularity as a generalized query, access, and
retrieval protocol for directory information. Data replication is key
to effectively distributing and sharing such information. Therefore,
it becomes important to create a replication protocol for use
specifically with LDAP to ensure that heterogeneous directory servers
can reliably exchange information. This document defines a multi-
master, incremental replication protocol for use with LDAP. In
addition, it defines how to use that replication protocol over two
transport mechanisms: standard email and LDAP. The new replication
protocol requires new data to be entered into the directory for use
with this protocol. Therefore, we must define new schema to hold that
information. Also, the data must be transmitted in a specific format;
we will use the proposed LDIF format [LDIF] for doing this.

2: Protocol Behavior

2.1 A glossary of replication terminology

There are 6 axes along which replication functionality can be provided.
These are:
- single-master vs. multi-master
- full vs partial
- whole vs fractional
- transactional vs loosely consistent
- complete vs. incremental
- synchronous vs. asynchronous

Each of these terms are described below.

A single-master (also known as master-slave) replication model assumes that each entry is writable on only one server. Changes flow from the master server to all of the replicas. A multi-master replication model assumes that entries can be written on multiple servers. Changes must then propagate from all masters to every replica, which requires additional work for conflict resolution.

Full replication is where every object in a database or DSA is copied to the replica. Partial replication is where some subset of the objects is copied.

Whole and fractional replication refer to the attributes transmitted during replication. If every attribute of the replicated objects are copied, this is referred to as whole replication. If only a subset of the attributes are copied, this is referred to as fractional replication.

Transactional replication requires that the replica gets and commits all changes between its copy of the data and the master’s copy of the data before the client is notified that the change was successful. Note that ‘commit’ is used in the general sense to define the action of writing changes to a data store and verifying that those changes were written successfully, it does NOT imply two-phased commit as used in databases. Loosely consistent means that there are times when the written server has data that the replicas do not, from the client’s point of view. Note also that a general replication topology may well have a mix of links that are transactional and loosely consistent.

Complete replication requires the replicating server to send a complete copy of itself to the replica every time it replicates. Incremental replication allows the replicating server to only send that data which has changed.

Synchronous replication updates the replica as soon as the source data is changed. Asynchronous replication updates the replica some time after the source data has been modified.

2.2 The basics of multi-master, incremental replication

This specification is aimed primarily at supporting multi-master, incremental, loosely consistent, asynchronous replication. To implement this, each server which wishes to master data must have the facilities necessary to track changes to the replicate data, the ability to transmit those changes to the other replicas, and the techniques to implement conflict detection and resolution. The replication protocol enables servers to transmit changes over several transport protocols. This document also provides algorithms for detecting and resolving conflicts.

2.3 The Naming Context (NC)

The Directory Information Base (DIB) is the collection of information about objects stored in the directory and their relationships. The DIB may be organized as a hierarchy (or tree), where objects higher in the
hierarchy provide naming resolution for their subordinate objects. This
tree, called the Directory Information Tree (DIT), provides the basis
for using names to query, access, and retrieve information. The DIT can
in turn be comprised of a set of subtrees.

The basic unit of replication is the NC. A Naming Context consists of a
non-leaf node (called the root of the naming context) and some subset of
its descendants subject to the following restriction: A descendant
cannot be part of a naming context unless all of its ancestors which are
descendants of the naming context root are in the naming context (e.g.
an NC is a complete subtree and cannot have any holes).

Each DSA will have one or more naming contexts. These naming contexts
will be defined and available in the Configuration container pointed to
by the root DSE of the server. The requisite schema are defined in
section 3.

To replicate a given naming context, the only requirement is that the
two servers agree on the contents of every schema entry needed to
define all the objects in the naming context. The reconciliation of
these entries is beyond the scope of this protocol.

2.3.1 Tracking changes to an NC

Borrowing from the ChangeLog draft [change], each change to a
replicated NC is logged in its own entry in the changeLog container.
This entry has object class ‘changeLogEntry’ and holds the trace of the
change, in LDIF format. For more details on the format, see [change].
However, the current ChangeLog draft is designed to provide single
master replication. To provide multi-master, incremental replication,
much more information needs to be kept.

In addition to the information required by the ChangeLog draft, servers
MUST also keep track of the following information and write it to the
changeLog entry:
- a version number for each property of every entry
- a timestamp for the time each property is changed,
- the attributes that were changed in this particular entry
- the object classes of this particular entry
- the naming context in which a given entry resides
- a unique identifier for each entry, which is NOT the DN or RDN of the
  entry

In addition, servers MUST also keep track of the following information
and conditionally write it to the changeLog entry:
- a unique identifier for each entry’s parent, which is NOT the DN or
  RDN of the parent, when the operation performed on this entry is a
  modifyDN.

2.3.2 Discussion of the required new changeLog information

The version number and timestamp are required for conflict resolution
in multi-master replication.

The attribute and object class tracking are useful for directory
synchronization with special-purpose directories. The actual changes
themselves are stored in a single binary blob in the changeLog entry.
This allows special-purpose directories (such as mail server
directories) to extract only the changes they need.

The NC is required for conflict resolution in multi-master replication.
The NC in which a given entry resides allows efficient replication of a given naming context. While this may in principle be derivable from the DN of the changed entry, adding this information allows much easier retrieval of the appropriate entries.

The unique identifier is required to handle modifyDN conflicts correctly.

In addition, the server MUST write the entry’s parentUniqueID to the changeLog entry during tracking of a modifyDN operation. This is required by the reconciliation algorithms defined below.

The new attributes are defined in section 3.

2.4 Defining the replication topology

Each server replicating a given set of naming contexts needs to have information about that naming context, including information on how to replicate it. However, this information is orthogonal to the replication protocol and as such is beyond the scope of this document.

2.5 Conflict resolution

In a multi-master environment, conflict resolution between incompatible updates is crucial. Since each change listed in the ChangeLog includes the version number of the attribute, every attribute received in a replication update is reconciled with the local version of the attribute in the following way:

A. If the version numbers are different, the higher version is favored
B. If the version numbers are the same, the version with the more recent time stamp is favored
C. If both the version and time-stamp match, the values themselves are compared and the one with the lowest value is favored. This guarantees that the system will quiesce consistently.
D. If all three of these match, the values are identical.

If an object is deleted, a server implementing this replication protocol MUST keep a ‘tombstone’ of the deleted object. This is essentially a copy of the deleted object that can be used to restore it; this document does not specify the length of time that such tombstones must be kept. When an object is deleted and there are replication changes that affect that object, there are some special rules that must be applied.

E: Deletions are allowed only on objects which have no children. If a deletion is received for an object that has a child, the reconciliation is to simply ignore the deletion.

F: If an incoming replication change is to create a new object under an already deleted object, then we reanimate the tombstones of all the ancestors and insert the new object in the correct place. This reanimation must minimally restore the RDN and object class attributes of the ancestor.

A modifyDN operation is not considered, for purposes of replication, to be a combination of a delete and an add operation unless such an operation would move the object to a new naming context.

In the case where the operation does not cross NC boundaries, it is a single operation which essentially simply modifies an entry’s
parentUniqueID. Since this attribute is treated as an attribute of the entry itself, the standard reconciliation logic applies.

In the case where the operation does cross the NC boundaries, it must be treated as a delete and add combination.

In addition, a modifyDN or modifyRDN operation may cause two objects to have the same DN. In that case, the replication system MUST algorithmically change the RDN of one or both of the objects. The algorithmically generated RDN is propagated so that the system will still reach a consistent state. The easiest way to guarantee a non-conflicting RDN is to use the object’s UID as the new RDN.

3: Schema

This section defines new attributes used in this protocol. Object classes and attributes which are not defined in this document can be found in [LSPA] or in [change].

3.1 Changes to the ChangeLog document

As noted above, multi-master replication requires a substantial amount of changes to the changeLog document. Here are the new object class and attributes.

Note that commonName, namingContexts, and description are all defined in other documents.

3.1.1 Changes to changeLogEntry

( 2.16.840.1.113730.3.2.1
   NAME 'changeLogEntry'
   SUP 'top'
   STRUCTURAL
   MUST (
       changeNumber $ targetDN $ changeType $ changes $
       changedAttribute $ entryObjectClass $ namingContext $
       uniqueIdentifier
   )
   MAY (  
       ParentUniqueIdentifier $ NewRDN $ deleteOldRDN $ newSuperior
   )
)

3.1.2 Changed attributes

( 2.16.840.1.113730.3.1.5
   NAME 'changeNumber'
   DESC 'a 64 bit number which uniquely identifies a change made to a Directory entry'
   SYNTAX 'Integer'
)

3.1.3 New attributes

( 1.2.840.113556.1.4.475
   NAME changedAttribute
   DESC 'OID of changed attribute'
   SYNTAX 'DirectoryString'
)
3.4 Changes to the LDIF document

To allow incremental efficient multi-master replication, we require two pieces of information for each attribute to be transmitted that must appear on a per-attribute basis; version number and timestamp. This should be transmitted in the LDIF format as qualifiers on the appropriate attribute: i.e. ‘commonName;2,19970308133106Z: Fred Foobar’. The version number is always the second to last qualifier, the timestamp is always the last qualifier. Note that this information is formatted this way for transmission purposes only.

4: LDAP transport

One of the two methods used to transport replication data is by using the LDAP protocol itself. The target server sets up an ordinary LDAP session with the source server, binding to the source DSA as the target server and issues a search with the new ‘replicate’ extended control. The target server will specify the changeLog container as the base of the search, and will use a filter that states that all records with changeNumber greater than the current high update number, that reside in one of the replicated naming contexts, will be given back. The source server MUST then order the results in such a way so that when they are applied to the replica in that order, the replica will be synced with the source server at the time that the replication snapshot was taken. This ordering of the changes is imperative. One possible way to provide such an ordering would be to sort the results on changeNumber. There will be a number of LDAP implementations which may not wish to provide a general sort facility for search results, however, a conformant implementation of the replicate control MUST order the results into a correct order.

Once the target starts receiving entries, it then applies each of the changeLogEntries to its own database, in the same order in which the entries were sorted, incrementing its highUpdateNumber attribute for that server appropriately. If the source server has indicated that it has more entries, the target server can then reissue the search with the new highUpdateNumber. In an environment with a rapidly changing directory, the source directory may at its discretion return a maximum highUpdateNumber indicating the highest number used by the server at the start of the session. The target server should then use that number as an additional term on the filter on subsequent search requests to allow a ‘snapshot’ of the data to be replicated. Otherwise, the target server might never close the connection to the source server, which would impact source server performance and available bandwidth.

The replicate control is included in the searchRequest and searchResultDone messages as part of the controls field of the LDAPMessage, as defined in Section 4.1.12 of [LDAPv3]. The structure
of this control is as follows:

```plaintext
replicateControl ::= SEQUENCE {
  controlType         1.2.840.113556.1.4.F
  criticality         BOOLEAN DEFAULT TRUE
  controlValue                INTEGER (1..2^64-1)
}
```

The `replicateControl` controlValue is used by the source server to return a maximum highUpdateNumber if it wishes to allow the target server to take a snapshot of the replication data.

5: Mail transport

The other method of transporting replication data is by using an email protocol. In this case, the target server mails the search command with the `replicate extended control` to the source server, and then the source server mails the results of the replication command back to the target server, in LDIF format as modified above [LDIF]. When the target server receives the changes, it processes them as appropriate. The actual mail transport protocol used is not covered in this document; it needs to be established as a bilateral agreement between the two servers. The security on this transaction is enabled by the security of the underlying mail protocol chosen.

6: Security Considerations

Replication requires secure connections and the ability to secure the change information stored in the directory. Securing the change information is covered in [change]. Standard LDAP security should be applied to the LDAP transmission of data. Standard mail security should be applied to the mail transmission of data. The information necessary to secure these connections will be stored as part of the URLs defining the connection points.

7: References


8: Author’s addresses

Chris Weider  
Cweider@microsoft.com  
1 Microsoft Way  
Redmond, WA 98052  
+1-206-703-2947

John Strassner  
Johns@cisco.com  
170 West Tasman Drive
San Jose, CA 95134
+1-408-527-1069