# Active Directory Updates

## Originating Update: Initiating Changes

A Lightweight Directory Access Protocol (LDAP) directory server supports the following four types of update requests:

* Add an object to the directory.
* Modify (add, delete, or replace) attribute values of an object in the directory.
* Move an object by changing the name or parent of the object.
* Delete an object from the directory.

An LDAP directory server processes each write request as an atomic (indivisible and irreducible) transaction. It is recommended that updates to the same object be limited to a maximum of 5,000 values added, modified, or deleted at the same time.

A write request either commits and all its effects are durable, or it fails before completion and has no effect. In this case, if one attribute update fails, they all fail and the object is not updated. A write request that commits is called an originating update.

The update on the nonoriginating domain controller is called a replicated update and is distinguishable from an originating update.

An originating update enforces schema restrictions (allowable parent object types for an object, mandatory and optional attributes for an object, syntax for an attribute) according to the schema that exists on the domain controller at the moment of the update.

## Tracking Updates

Some directory services use timestamps to determine what changes need to be propagated. In these systems, it is important to keep the clocks on all directory servers synchronized.

It relies instead on the use of update sequence numbers (USNs) that are assigned by a counter that is local to each domain controller. Because these USN counters are local, it is easy to ensure that they are reliable and never “run backward” (that is, decrease in value). The trade-off is that it is meaningless to compare a USN assigned on one domain controller to a USN assigned on a different domain controller.

Active Directory replication does not use timestamps as the primary mechanism to determine which updates “win” (are preserved) in a conflict resolution. Instead, Active Directory uses volatility (number of changes) as the first element of the per-attribute “stamps” that are compared during conflict resolution. The second element is a timestamp.

\*AD do need synchronization for authentication purposes.

# Resolving Conflicts

Active Directory must ensure that when replication has occurred, all replicas agree on the value of the updated attribute.

Active Directory ensures agreement by attaching the unique stamp to each replicated attribute value (or multivalue) during an originating update. This stamp travels with the value as the value replicates. If the stamp of the value that was replicated is larger than the stamp of the current value, the current value (including the stamp) is replaced; otherwise, the current value (including the stamp) is left alone.

The stamp has the following three components:

* The version is a number that is incremented for each originating write. If the attribute was never written before, the version that was assigned to its first originating write is 1.
* The originating time is the time of the originating write, to a one-second resolution, according to the system clock of the domain controller that performed the write.
* The originating DC is a GUID that identifies the domain controller that performed the originating write.

It is possible to see all three components of the stamp in output using repadmin /showmeta in Windows 2000 and repadmin /showobjmeta in Windows Server 2003. The column labelled “Ver” contains the version, the column labelled “Org. Time/Date” contains the originating time, and the column labelled “Originating DC” contains the originating domain controller (expressed as “site name \server name” rather than GUID).

Ver > Time > GUID

So if two stamps have the same version, the originating time almost always breaks the tie. In the extremely rare event that the same attribute is updated on two different domain controllers during the same second, the originating domain controller breaks the tie in using the highest GUID.

# Update Sequence Numbers (USNs)

The *current USN* is a 64‑bit counter that is maintained by each Active Directory domain controller as the *highestCommittedUsn* attribute on the rootDSE object.

The rootDSE (DSA-specific Entry) represents the top of the logical namespace for one domain controller.

The current USN value is stored on an updated object as follows.

* **Local USN**:The USN for the update is stored in the metadata of each attribute that is changed by the update as the local USN of that attribute (originating and replicated writes).   
  View the Local USN.   
  repadmin /showobjmeta DCLIST <object\_DN> labelled “Loc. USN”
* **uSNChanged**: The maximum local USN among all of an object's attributes is stored as the object's uSNChanged attribute (originating and replicated writes). The uSNChanged attribute is indexed, which allows objects to be enumerated efficiently in the order of their most recent attribute write. This value can be examined using LDP or ADSIEDIT.
* **Originating USN**: For an originating write only, the update's USN value is stored with each updated attribute as the originating USN of that attribute. Unlike the local USN and uSNChanged, the originating USN travels with the attribute's value as it replicates.   
  View the Originating USN  
  repadmin /showobjmeta DCLIST <*object\_DN*> labelled “Org.USN”

# Replication Fundamentals

## Originating ADD

An Add request makes a new object with a unique objectGUID. The version number value is set to 1 for all attributes that are populated by the Add request.

ADD request fails if the parent object does not exist or if the originating DC does not contain a writeable replica of the new object’s partition.

## Originating MODIFY

All Modify requests can be reduced to requests to replace the current value of an attribute with a new value. A modify request can specify one of the following:

* That an attribute be deleted from the object. Attribute deletion is best thought of as replacing the attribute value with NULL. The NULL value occupies no storage of its own but does carry a stamp, as does any value that is stored as a directory attribute.
* That a value be added to the current value of an attribute, as when modifying an attribute that can have multiple values. The effect is to replace the current values with the current values plus the added value.

For each attribute in the request, a Modify request compares the new value in the request with the existing value in the object. If the values are the same, the request to modify that attribute is ignored. If the resulting Modify request does not change any attributes, the entire request is ignored.

Otherwise, a Modify request computes a stamp in the metadata for each new replicated attribute value by reading the version from the existing value (version=0 for an attribute that has never been written) and then adding 1 to this value. The Modify request replaces the old stamp values with new stamp values.

## Originating Move

A Move request is essentially a special Modify request for a single attribute, the name attribute. The operation proceeds as described for the Modify request.

## Originating Delete

A Delete request is essentially a special Modify request that does the following:

1. Sets the isDeleted attribute to TRUE.
2. Marks the object as a *tombstone*, which is an object that has been deleted but not fully removed from the directory.
3. Changes the relative distinguished name to a value that is otherwise impossible (cannot be set by an LDAP application).
4. Strips all attributes that are not needed by Active Directory. A few key attributes, including objectGuid, objectSid, distinguishedName, nTSecurityDescriptor, and usnChanged, are preserved on the tombstone.
5. Moves the tombstone to the Deleted Objects container, which is a hidden container within the directory partition.

### Viewing the tombstone

Object deletions are replicated by replicating tombstones. A tombstone is invisible to normal LDAP searches. (A tombstone is visible to searches that use the special LDAP control 1.2.840.113556.1.4.417.) Object references that formerly referred to the deleted object now refer to the tombstone. Therefore, reading such a reference returns the distinguished name of the tombstone, not the distinguished name of the object prior to the object’s deletion.

LDAP\_SERVER\_SHOW\_DELETED\_OID 1.2.840.113556.1.4.417

### Garbage Collection: Permanent Removal of Expired Tombstones

A process that runs on every domain controller to permanently remove expired tombstones from the directory database. Tombstones take up space in every directory partition replica.

Two attributes of the Directory Service object (nTDSService) in the configuration container (CN=Directory Service, CN=Windows NT, CN=Services, CN=Configuration, DC=forestRootDomain) control how garbage collection runs and what it removes, as follows:

* *Tombstone lifetime* determines the number of hours that a deleted object lives as a tombstone in the directory before being collected as garbage, and it is set in the tombstoneLifetime attribute. The default setting is 60 days, and the minimum setting is 2 days.
* *Garbage collection interval* determines how often a domain controller examines its database for expired tombstones that can be collected, and it is set in the garbageCollPeriod attribute. The default setting is 12 hours, and the minimum setting is one hour.

The maximum garbage collection interval is one-third of the tombstone lifetime (in hours). For example, if the tombstoneLifetime is set to 30 days and garbageCollPeriod to 300 hours, your actual garbage collection period is only 10 days, or 240 hours. ADSIEdit can be used to view or change the default settings for these attributes.

### Tombstone Lifetime and Active Directory Backup and Restore

Active Directory does not allow restoring from a directory backup that is older than the tombstone lifetime. A restore from backup creates a directory partition replica that has not performed replication since the time of backup (or earlier). If the backup were taken more than a tombstone lifetime before the restore, objects deleted in the meantime would have no tombstones and therefore a new directory partition replica that was created by the restore operation would never receive these deletions. For this reason, a restore procedure will not restore a backup that was taken more than one tombstone lifetime before the time of the restore. It is therefore a recommended best practice to back up Active Directory at least twice during a tombstone lifetime.

按默认，60天内至少备份两次。

It is important that the tombstone lifetime be substantially longer than the expected replication latency. The default setting of 60 days is generous to accommodate unforeseen circumstances.   
若环境恶劣，可适当延长tombstone lifetime。

### Reanimating the tombstone

Windows Server 2003 offers a new feature to reanimate the tombstone effectively allowing for an un-deletion of the object, supports an LDAP API that reanimates the tombstone of a single object.

Only attributes retained on the tombstone are restored (objectSID, objectGUID, security descriptor); all other data must be recreated. Therefore, to restore an entire deleted container or a set of multiple objects, authoritative restore is still the best option.

In order for deleted object restoration to be enabled, at least one domain controller in the domain must be running Windows Server 2003. By default, only domain administrators can restore deleted objects, though this can be delegated to others.

The following limitations apply to restoring deleted objects:

* An object cannot be restored when the tombstone lifetime for the object has expired because when the tombstone lifetime has expired, the object is permanently deleted.
* Objects that exist at the root of the naming context, such as a domain or application partition, cannot be restored.
* Schema objects cannot be restored with this process. ~~However, schema objects could not be deleted in Windows 2000~~. In Windows Server 2003 there is a feature to mark schema objects as defunct. These defunct objects cannot be restored with reanimation.
* It is possible to restore deleted containers, but the restoration of the deleted objects that were in the container before the deletion is difficult because the tree structure under the container must be manually reconstructed.