

# Chapter 14-1

## Anatomy of a Note ID

This chapter describes note IDs in detail and explains how different Domino or Notes tasks (replicator and so on) use the components of a note ID and how API programs can use them.

The parts of a note ID are:

- **UNID (Universal Note ID)** - Identifies all the copies of a note, regardless of location or time. In other words, every replica of a note has the same UNID, and the UNID does not change when a note is modified.
- **OID (Originator ID)** - Identifies a particular revision of a note, regardless of location. In other words, every replica of a note has the same OID, but the OID changes when the note is modified.
- **GNID (Global Note ID)** - Identifies a particular note in a particular database. The GNID does not change as the note is modified. The GNIDs of replica copies of a note will probably be different, since the various copies will occupy different positions in their databases.
- **NID (Note ID)** - Identifies a particular note in a given database. The NID does not contain information about the database being referred to, and it does not change when the note is modified.
- **IID (Instance ID)** - Identifies a particular revision of a note in a given database. The IID does not contain information about the database being referred to. The IID changes when the note is modified.
- **GIID (Global Instance ID)** - Identifies a particular revision of a note in a particular database. The GIID contains information about the database being referred to. The GIID changes when the note is modified.

You can examine the ID information for a particular document from the Notes user interface. Select the document in a view or open it, then select File - Document Properties. Notes displays the "Document Properties" infobox. On the information page, Notes displays the header data associated with this document, including the dates and times the document was created and modified and the note ID information.

The note ID information is displayed by Notes as three lines consisting of key characters and hex digits. For a typical data note, it looks like this:

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

These three lines contain the Originator ID (OID), the Universal Note ID (UNID), the Global Note ID (GID), and the Note ID (NID) of the open or selected document.

## The Universal Note ID (UNID) and the Originator ID (OID)

The first two lines of the ID constitute the complete Originator ID for the note. In turn, the Originator ID consists of the Universal Note ID (the whole first line) plus the sequence number and the sequence time (the second line):

Originator ID (OID) =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

Universal Note ID (UNID) =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

Sequence Time =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092


Sequence Number =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

The first two parts of the Originator ID (and the Universal Note ID) consist of a File member and a Note member. The first line of the ID information in the Document Properties infobox consists of the letters "OF" ("Originator ID - File"), followed by sixteen hex digits, followed by a hyphen, followed by the letters "ON" ("Originator ID - Note"), followed by sixteen more hex digits. The sixteen hex digits after the "OF" and before the hyphen constitute the File member of the OID. The sixteen hex digits after the hyphen and

the "ON" constitute the Note member of the OID.

OID.File =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092 

OID.Note =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

The header file nsfdata.h contains the following definition of the ORIGINATORID data structure and the UNIVERSALNOTEID data structure:

```
typedef struct {
    DBID File;          /* Unique (random) number */
                      /* (Even though this field is called "File," */
                      /* it doesn't have anything to do with the file!) */
    /*
    TIMEDATE Note;     /* Original Note Creation time/date */
                      /* (THE ABOVE 2 FIELDS MUST BE FIRST - UNID */
                      /* COPIED FROM HERE ASSUMED AT OFFSET 0) */
    DWORD Sequence;   /* LOW ORDER: sequence number, 1 for first version */
    /*
                      /* HIGH ORDER WORD: flags, as above */
    TIMEDATE SequenceTime; /* time/date when sequence number was bumped */
} ORIGINATORID;

#define OID ORIGINATORID
```

```
typedef struct {
    DBID File;          /* Unique (random) number */
                      /* (Even though this field is called "File," */
    /*
                      /* it doesn't have anything to do with the
file!) */
    TIMEDATE Note;     /* Original Note Creation time/date */
} UNIVERSALNOTEID;

#define UNID UNIVERSALNOTEID
```

The Originator ID (OID) for a note identifies all replica copies of the same note. The OID is

composed of two parts: the Universal Note ID (UNID) and the sequence number and sequence time. The UNID universally identifies all instances of the same note. Taken together, the sequence number and the sequence time distinguish different revisions of the same note from one another.

The Universal note ID (UNID) identifies a note across all servers. However, the UNID lacks the information necessary to directly access the note in a given database. The UNID is used to reference a specific note from another note. The "\$REF" (FIELD\_LINK) field of a response note contains the UNID of its parent. Similarly, DocLinks (see the NOTELINK structure in nsfdata.h) contains the UNID of the linked-to note, the UNID of the linked-to note view, plus the ID of the database where the linked-to note can be found (ViewLinks contain the same information except the linked-to note structure is set to all zeros and DatabaseLinks zero out the linked-to note and view structures.) The important characteristic of the UNID is that it continues to identify a note even after the note is updated.

## The UNID, the OID, and the Replicator

The Universal Note ID (the first half of the Originator ID) uniquely identifies all versions and all copies of the same note. Two notes are replica copies of each other if they share the same UNID. Therefore, different versions and all replica copies of the same note have the same UNID. A corollary of this rule is that one database must not contain two notes with the same UNID. If the replicator finds two notes with the same UNID in the same database, it generates an error message in the log and does not replicate the document.

The full Originator ID, on the other hand, uniquely identifies one particular version of a note. In other words, all replica copies of the same version of a note have the same OID. However, a modified version of a replica copy of a particular note will have a different OID, because Domino and Notes increment the sequence number when a note is edited and also sets the sequence time to the timedate when the sequence number was incremented. Therefore, when one replica copy of a note remains unchanged but another copy is edited and modified, the UNIDs of the two notes remain the same but the sequence number and sequence times (and therefore the OIDs) are different.

The Domino replicator uses the UNID to match the notes in one database with their respective replica copies in other databases. For example, if database A is replicating with database B, and database A contains a note with a particular UNID but database B does not, the replicator creates a copy of that note and add it to database B.

If database A contains a note with a particular UNID and database B contains a note with the same UNID, the replicator concludes that these two notes are replica copies of one another. In this case, the replicator goes on to examine the sequence number and sequence time of the two notes. If the sequence number and sequence time are the same for both notes, then the replicator concludes that the two notes are up to date with one another, and no action is required. On the other hand, if either the sequence number or the sequence time -- or both -- differ between two notes, the replicator must decide which one is more recent and update the older note with the most recent version.

If one note has been updated but the other note has not, the sequence number of the first note will be greater than that of the second. The replicator handles this case by overwriting the second note with the first, bringing the two databases into synchronization.

## Replication Conflicts

Replication conflicts arise from concurrent edits of the same note. If a user updates a note in database A and, before the change replicates to database B, another user updates the replica copy of that note in database B, the two notes have the same sequence number but different sequence times. In this case, the replicator generates a replication conflict because this represents a concurrent edit of the same note.

The replicator handles replication conflicts by generating a replica conflict document. The document with the later sequence time becomes the conflict winner and the document with the earlier sequence time becomes the conflict loser. The conflict winner is copied to the database containing the loser, and the conflict loser is turned into a response document to the winner.

The replicator generates the conflict loser (black diamond) document in the database initially holding the document with the earlier sequence time. The replicator copies the winning document from the winning database into the losing database. This winning document keeps its OID intact. Then the replicator turns the losing document into a new document by generating a distinct OID and adding the special item "\$Conflict" (VIEW\_CONFLICT\_ITEM) to the document. This \$Conflict item causes a black diamond to appear in the left-hand margin of any view that displays the conflict loser. The replicator also turns the losing document into a response document by adding to it a \$REF item that contains the UNID of the winning document.

NOTE: The conflict loser (black diamond document) may not always appear on both servers immediately after replication completes. When the server holding the document that will be the winner initiates the replication, the conflict loser does not appear on the winning server immediately after replication, but does appear on the losing server. This happens because the replicator generates the conflict loser (black diamond document) only on the server initially holding the losing document.

Specifically, when server B initiates replication with server A, B first pulls changes from A, and then A pulls changes from B. When B pulls from A, B sees the conflicting document on A, determines that B's version is the winner, and does not bring the conflicting document over from A to B. The justification for this is that B already has the winner, so B does not need to bring over the loser. Then A pulls from B, sees the conflicting document on B, and determines that B's version is the winner. Therefore A makes its own version into a replica conflict loser (black diamond) document and brings B's version over to A as the replica conflict winner. This satisfies the requirement that when replication is complete, both copies of the database present the same version of the document as the most up-to-date (winning) version of the document. However, the conflict loser (black diamond) does not yet appear on B. The next time B replicates with A the replica conflict loser (black diamond)

document will come back to B and both copies of the database will be synchronized.

## The OID and the API

API programs use `NSFNoteGetInfo` to obtain the various components of the Originator ID, specifying the `_NOTE_OID` info flag. For example:

```

    ORIGINATORID    NoteOID;

/*
 * Get the OID from the note AFTER it has been updated
 */

NSFNoteGetInfo (hNote, _NOTE_OID, &NoteOID);

```

`NSFNoteUpdate` causes Domino and Notes to change the sequence number and sequence time of the note being updated. Therefore, API programs must pay particular attention to the order in which they call `NSFNoteUpdate` and `NSFNoteGetInfo` when obtaining the OID information for a note.

Specifically, using `NSFNoteCreate` to create a new note yields a note handle, but until this note has been stored in the database (using `NSFNoteUpdate`), `NSFNoteGetInfo` does not return a valid OID. However, since the UNID has been assigned at this stage, API programs can parse the returned value from the `NSFNoteGetInfo` to retrieve the UNID of this newly created note.

## UNID.File

The File member of the OID (and UNID) contains a number derived in different ways depending on the release of Domino or Notes. Pre- 2.1 versions of Notes set the File member to the creation timedata of the NSF file in which the note is created. Notes 2.1 sets the File member to a user-unique identifier, derived partly from information in the ID of the user creating the note and partly from the database in which the note is created. Notes 3.0 and later releases of Domino and Notes sets the File member to a random number generated when the note is created.

## UNID.Note

The Note member of the OID (and UNID) is the timedata when the note was created. (For details on the TIMEDATE structure, please see the Appendix chapter "Data Types".)

## Global Note ID (GNID) and NOTEID (NID)

The Global Note ID (GNID) is composed of two parts: the Database ID and the Note ID (NID). Since the GNID identifies both the database and the note in the database, it uniquely identifies a particular copy of a note across all databases.

The GNID appears in the bottom line of the ID information displayed by the Design - Document Properties infobox. The first part of the GNID corresponds to the File member of the GLOBALNOTEID structure. The second part corresponds to the NoteID member.

Global Note ID (GNID) =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

Database ID (GNID.File) =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

Note ID (GNID.NoteID) =

ID: OF0000039D:3836C29F-ON85255DC9:0056FB94  
SD00255DF4:0057B8FA-SN00000003  
DB85255CD9:00567287-NT0000C092

The third line of ID information in the Design - Document Properties infobox consists of the letters "DB" (which stands for "Data Base"), followed by sixteen hex digits, followed by a hyphen, followed by the letters "NT" ("NoTe"), followed by eight more hex digits. The sixteen hex digits after the "DB" and before the hyphen constitute the File member of the GNID. The eight hex digits after the "NT" constitute the Note ID member of the GNID.

## Global Note ID (GNID)

Unlike the OID, the GNID does not distinguish between different versions of the same note. Since the NoteID field is relative to the file that contains the note, the GNIDs of replica notes will most likely be different.

The Database ID (the File member of the GNID) contains the replica ID of the database. The API header file nsfdata.h defines the DBID data structure as a TIMEDATE:

```
#define DBID TIMEDATE
```

The 8-byte DBID contains the TIMEDATE on which the database was originally created. You can verify that the File member of the GNID, as displayed on the third line of the ID information in the File - Document Properties infobox, agrees with the Replica ID, displayed

on the information page of the File - Database - Properties infobox. Since all replica copies of a particular database have the same DBID, all notes in all replica copies of a particular database have the same GNID.File.

```
typedef struct {
DBID File; /* Creation Date/Time of NSF where note was created */
NOTEID NoteID; /* Note's RRV in this file */
} GLOBALNOTEID;
```

```
#define GNID GLOBALNOTEID
```

## Note ID (NID)

The note ID (NID) identifies a note in a database. The NID is the file position of the Record Relocation Vector (RRV) for the note. An RRV is a DWORD offset in a file. Confusion often arises because the note ID is casually referred to as "the Note's RRV in this file." In fact, the note ID is the offset in the file to the RRV, which in turn points to the record for the note. An RRV is a general structure, while a note ID is more specific. Internal to Domino and Notes, various other objects besides notes have an associated RRV.

```
typedef DWORD RRV;
typedef DWORD NOTEID;
```

The note ID is guaranteed never to change in one database (.NSF) file, except when the note is deleted. When the note is deleted, the RRV\_DELETED bit is set in the note ID.

```
#define RRV_DELETED 0x80000000L /* indicates a deleted note */
```

Since a note ID is specific to the database file that contains it, replica copies of the same note in other databases will most likely have a different note ID.

## INSTANCEID (IID)

The Instance ID identifies an instance of a note in a given database. Each time a note is modified, Domino or Notes stamps the instance ID with the time and date that the update occurred. The Instance ID consists of the note's modification TIMEDATE and the note ID.

```
typedef struct {
TIMEDATE Note; /* Note's MODIFICATION date/time */
NOTEID NoteID; /* Note's RRV */
} INSTANCEID;
```

```
#define IID INSTANCEID
```

The modification TIMEDATE (the Note member of the IID) is not the same thing as the Revision Time in the OID. Database views use the note modification timedate in the IID to determine when to display the unread flags. You can access the modification timedate by



specifying `_NOTE_MODIFIED` to `NSFNoteGetInfo`. The replicator uses the Revision Time in the `OID` to compare replica copies of notes in different databases. You access the Revision Time by specifying `_NOTE_OID` to `NSFNoteGetInfo`.

The `NoteID` member of the `IID` is the same value found in the `NoteID` member of the `GNID`.

`NSFNoteUpdate` sets the modification `timedate` every time the note is updated to the `.NSF` file. The modification `timedate` governs the unread flags in views.

There are several ways for an API program to obtain the `IID` for a note. Get the note ID from the `GNID`, a `NIFReadEntries` buffer, or the `ID.NoteID` member of a `SEARCH_MATCH` structure. Get the modification `timedate` from the `ID.Note` member of a `SEARCH_MATCH` structure or by calling `NSFNoteGetInfo` and specifying `_NOTE_MODIFIED`.

## GLOBALINSTANCEID (GIID)

The Global Instance ID globally identifies an instance of a note across databases. It consists of the same members as the Instance ID but adds the database ID as the `File` member. The note with the latest modification date is the "most recent" instance.

```
typedef struct {
DBID File; /* database Creation time/date */
TIMEDATE Note; /* note Modification time/date */
NOTEID NoteID; /* note ID in database */
} GLOBALINSTANCEID;
```

```
#define GIID GLOBALINSTANCEID
```

API programs can initialize a `GIID` structure for a note by calling `NSFDbIDGet` to obtain the `File` member and then calling `NSFNoteGetInfo`, specifying `_NOTE_MODIFIED` to get the members that correspond to the `IID` modification `timedate`.

# TIMEDATE

The Domino and Notes TIMEDATE structure consists of two long words that encode the time, the date, the time zone, and the Daylight Savings Time settings that were in effect when the structure was initialized. The TIMEDATE structure is designed to be accessed exclusively through the time and date subroutines defined in misc.h. This structure is subject to change; the description here is provided for debugging purposes.

The first DWORD, Innards[0], contains the number of hundredths of seconds since midnight, Greenwich mean time. If only the date is important, not the time, this field may be set to ALLDAY.

The date and the time zone and Daylight Savings Time settings are encoded in Innards[1]. The 24 low-order bits contain the Julian Day, the number of days since January 1, 4713 BC. Note that this is NOT the same as the Julian calendar! The Julian Day was originally devised as an aid to astronomers. Since only days are counted, weeks, months, and years are ignored in calculations. The Julian Day is defined to begin at noon; for simplicity, Domino and Notes assume that the day begins at midnight. The high-order byte, bits 31-24, encodes the time zone and Daylight Savings Time information. The high-order bit, bit 31 (0x80000000), is set if Daylight Savings Time is observed. Bit 30 (0x40000000) is set if the time zone is east of Greenwich mean time. Bits 27-24 contain the number of hours difference between the time zone and Greenwich mean time, and bits 29-28 contain the number of 15-minute intervals in the difference.

For example, 2:49:04 P. M., Eastern Standard Time, December 10, 1996 would be stored as:

Innards[0]: 0x006CDCC0 19 hours, 49 minutes, 4 seconds GMT  
Innards[1]: 0x852563FC DST observed, zone +5, Julian Day 2,450,428

If the time zone were set for Bombay, India, where Daylight Savings Time is not observed, 2:49:04 P. M., December 10, 1996 would be stored as:

Innards[0]: 0x0032B864 9 hours, 19 minutes, 4 seconds GMT  
Innards[1]: 0x652563FC No DST, zone 5 1/2 hours east of GMT, Julian Day 2,450,428