Studio 3T

Knowledge Base > MongoDB Tutorials > MongoDB Best Practices for UUID Data

MongoDB Best Practices for UUID Data

In this post, we'll take a look at how MongoDB handles UUID data.

We'll also go through a few complex cases, from which we can glean best practices when working with UUID data on our MongoDB GUI, **Studio 3T**.

Download our MongoDB GUI now, in case you haven't yet.

UUID Data in MongoDB

MongoDB and **MongoDB drivers** come with built-in support for the UUID data type.

It's very easy and convenient to start using UUID immediately, without having to review the BSON specification. In most cases you'll most likely be fine. There are some scenarios however, where you might run into unexpected issues that are very difficult to debug.

As soon as you go multi-language with your deployments you might discover that your applications no longer work as expected, e.g. you might no longer be able to match orders to customers.

It's also very likely that your data gets corrupted over time and it will be very difficult to recover from such a disaster without having to restore your backups (you do backups on a regular basis, don't you?).

We'll take a closer look at scenarios where working with UUID data in MongoDB might become more complex. We'll make you aware of those configurations and provide you with a set of best practices to follow.

What is a UUID?

A **Universally Unique IDentifier** (**UUID**) is a unique number used as an identifier in computer software.

Sometimes the term **Globally Unique IDentifier** (**GUID**) is used to describe it.

A GUID is one of many implementations of the UUID standard.

UUIDs are 128-bit values and are usually displayed as 32 hexadecimal digits separated by hyphens, for example:

```
176BF052-4ECB-4E1D-B48F-F3523F7F277F
```

(The example above is a random Version 4 UUID generated by Studio 3T)

MongoDB and UUID support

MongoDB has built-in support for the UUID data type and most of the MongoDB drivers support UUID natively.

MongoDB itself stores UUIDs as `Binary` fields and when such `Binary` fields are accessed from software,

MongoDB drivers usually convert their value as found in the database to language-specific UUID or GUID objects.

For example, when you read a UUID from a MongoDB database using the Java driver, an object of type `java.util.UUID` will be returned. The C# driver on the other hand will return a struct of type `System.GUID` .

This makes it very convenient and easy to work with the UUID data type from your application code.

Complex UUID scenarios

You might run into problems when working with UUIDs in multi-platform/multi-language environments because accessing your database from different programming languages using different MongoDB drivers might not always be safe, as we will demonstrate below.

Furthermore, you should be careful when working with UUID data types in third-party MongoDB management software because only a few MongoDB tools take due care in handling UUID data types.

MongoDB drivers usually convert UUIDs between their `Binary` database representation and the language specific UUID data type. Initially the encoding method was platform specific and wasn't consistently implemented across different MongoDB drivers.

Example

An example UUID created and displayed in a Java application would be shown as follows:

```
00010203-0405-4007-8009-0A0B0C0D0E0F
```

The MongoDB Java driver (version 2.9.1 in this example) will convert it to a `Binary` database representation and store it with the following byte order:

```
07 40 05 04 03 02 01 00 0F 0E 0D 0C 0B 0A 09 80
```

However, the C# driver used to access the same field will return and display the following:

```
04054007-0203-0001-0F0E-0D0C0B0A0980
```

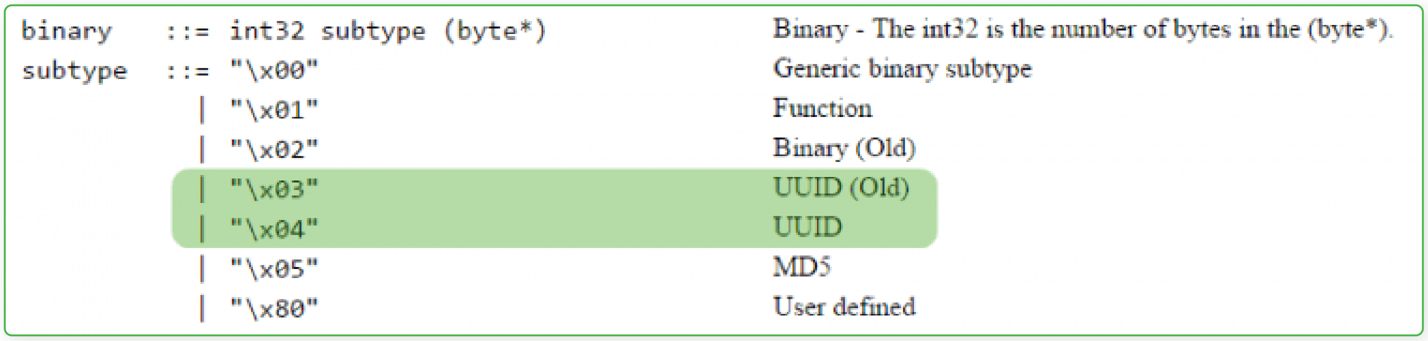
You'll notice the representation of the same UUID differs depending on the platform used to access it.

As long as you're not editing those UUIDs on both ends, you shouldn't experience any serious problems (besides confusion when analysing your data). However, if one day you do edit a

UUID, you’ll very likely run into issues that can be very difficult to locate and debug because you’ll essentially be dealing with scrambled data.

Binary subtypes `0x03` and `0x04`

While reviewing the BSON specification you’ll notice two `Binary` subtypes assigned to the UUID data type:



```
binary    ::= int32 subtype (byte*)        Binary - The int32 is the number of bytes in the (byte*).
subtype   ::= "\x00"                       Generic binary subtype
            | "\x01"                       Function
            | "\x02"                       Binary (Old)
            | "\x03"                       UUID (Old)
            | "\x04"                       UUID
            | "\x05"                       MD5
            | "\x80"                       User defined
```

Originally, only the **`0x03`** subtype was used to annotate UUIDs. To address the portability issues in multi-platform environments that arose from the language specific UUID implementations, the MongoDB designers later introduced the new **`0x04`** subtype.

The byte order used to store UUIDs as `Binary` fields with the **`0x04`** subtype is now consistently implemented across all MongoDB drivers and the legacy **`0x03`** subtype is being supported for legacy data only.

Using the `Binary` **`0x04`** subtype when working with UUIDs guarantees that you’ll be able to access your data from any platform and any programming language without these compatibility issues.

Best practices

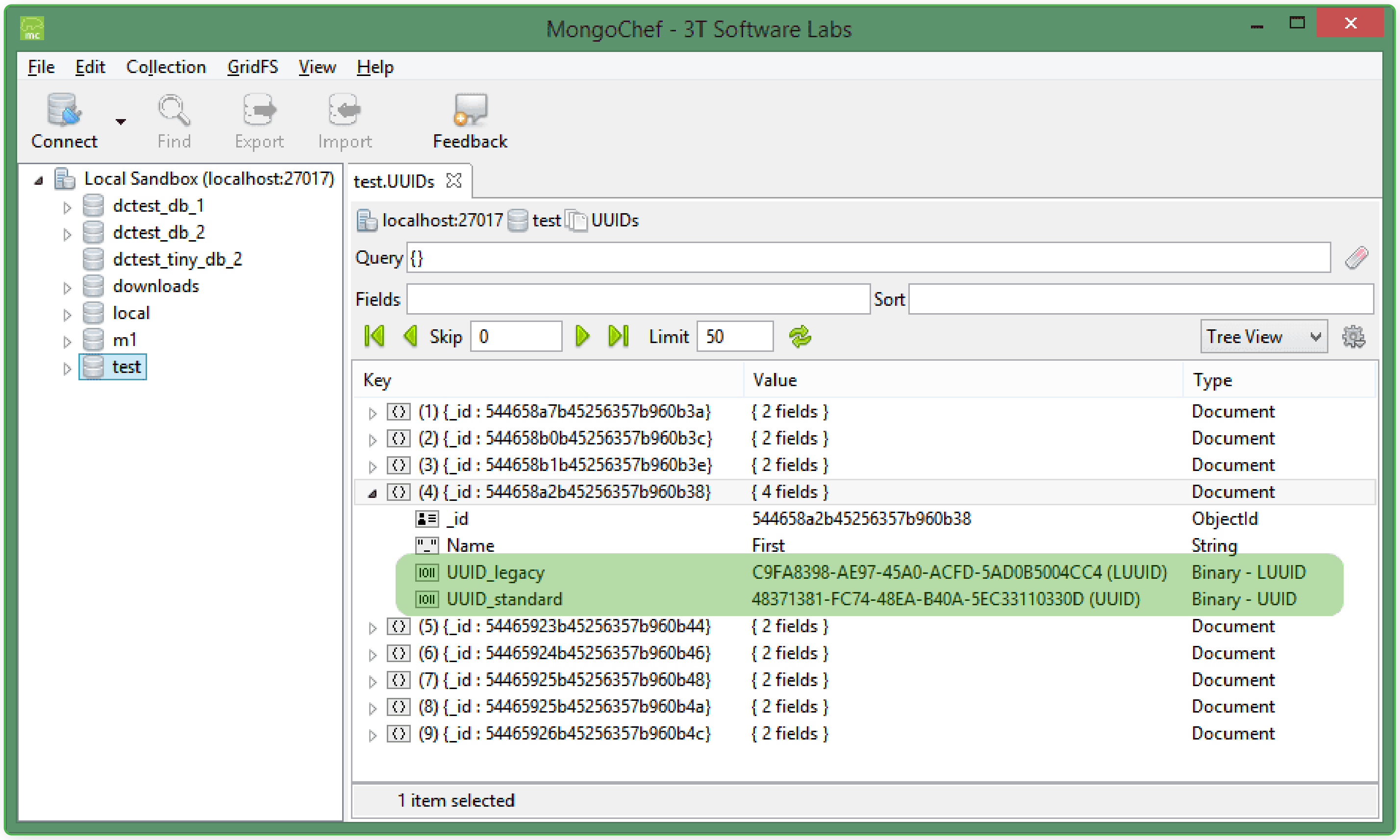
1. Make sure you always know what the UUID subtype is that your applications are using

You can find out the subtype in use by looking at the JSON representation of your existing data (e.g. in the mongo shell):

> **`BinData(3,"B0AFBAMCAQAPDg0MCwoJgA==")`**
> the **3** stands for a UUID stored as `Binary` field with the legacy 0x03 subtype
> **`BinData(4,”SDcTgfx0SOq0Cl7DMRAzDQ==”)`**
> the **4** stands for a UUID stored as a `Binary` field with the 0x04 subtype

You can also use our MongoDB GUI which has support for legacy UUIDs to access this information.

Studio 3T annotates regular UUIDs as `Binary - UUID` and for legacy UUIDs it lets you know which encoding it currently uses (e.g. Java, C# or none):


MongoChef - 3T Software Labs
File Edit Collection GridFS View Help
Connect Find Export Import Feedback
Local Sandbox (localhost:27017)
dctest_db_1
dctest_db_2
dctest_tiny_db_2
downloads
local
m1
test
test.UUIDs
localhost:27017 test UUIDs
Query {}
Fields
Sort
Skip 0
Limit 50
Tree View
1 item selected

Key	Value	Type
(1) {_id : 544658a7b45256357b960b3a}	{ 2 fields }	Document
(2) {_id : 544658b0b45256357b960b3c}	{ 2 fields }	Document
(3) {_id : 544658b1b45256357b960b3e}	{ 2 fields }	Document
(4) {_id : 544658a2b45256357b960b38}	{ 4 fields }	Document
_id	544658a2b45256357b960b38	ObjectId
Name	First	String
UUID_legacy	C9FA8398-AE97-45A0-ACFD-5AD0B5004CC4 (LUUID)	Binary - LUUID
UUID_standard	48371381-FC74-48EA-B40A-5EC33110330D (UUID)	Binary - UUID
(5) {_id : 54465923b45256357b960b44}	{ 2 fields }	Document
(6) {_id : 54465924b45256357b960b46}	{ 2 fields }	Document
(7) {_id : 54465925b45256357b960b48}	{ 2 fields }	Document
(8) {_id : 54465925b45256357b960b4a}	{ 2 fields }	Document
(9) {_id : 54465926b45256357b960b4c}	{ 2 fields }	Document

2. Configure your drivers to work with `0x04` subtype for new deployments

MongoDB drivers usually store UUIDs as `Binary` fields with the legacy **`0x03`** subtype assigned by default. This configuration can be changed:

C# / .NET

You can override the driver’s default settings and configure it to use the `Binary` **`0x04`** subtype by modifying the value of `BsonDefaults.GuidRepresentation` :

```
MongoDefaults.GuidRepresentation = MongoDB.Bson.GuidRepresentation.Standard;
```

You can also modify `GuidRepresentation` at the server, database and collection level.

Python

You can configure the behavior of your Python driver. Read more about the `uuid_subtype` attribute here.

Java

Support for UUID configuration will be added in the 3.0 release of the MongoDB driver (you might want to monitor this ticket JAVA-403). With current (pre 3.0) drivers you'll have to perform the conversion yourself:

```
/**
 * Convert a UUID object to a Binary with a subtype 0x04
 */
public static Binary toStandardBinaryUUID(java.util.UUID uuid) {
    long msb = uuid.getMostSignificantBits();
    long lsb = uuid.getLeastSignificantBits();

    byte[] uuidBytes = new byte[16];

    for (int i = 15; i >= 8; i--) {
        uuidBytes[i] = (byte) (lsb & 0xFFL);
        lsb >>= 8;
    }

    for (int i = 7; i >= 0; i--) {
        uuidBytes[i] = (byte) (msb & 0xFFL);
        msb >>= 8;
    }

    return new Binary((byte) 0x04, uuidBytes);
}

/**
```

```
     * Convert a Binary with a subtype 0x04 to a UUID object
     * Please note: the subtype is not being checked.
     */
    public static UUID fromStandardBinaryUUID(Binary binary) {
        long msb = 0;
        long lsb = 0;
        byte[] uuidBytes = binary.getData();

        for (int i = 8; i < 16; i++) {
            lsb <<= 8;
            lsb |= uuidBytes[i] & 0xFFL;
        }

        for (int i = 0; i < 8; i++) {
            msb <<= 8;
            msb |= uuidBytes[i] & 0xFFL;
        }

        return new UUID(msb, lsb);
    }
```

3. Configure your MongoDB GUI to handle legacy UUIDs with `0x03` subtype

Studio 3T has built-in support for both, the `0x03` and the `0x04` subtype. You can configure the behavior in the **Properties** dialog and select from the following options:

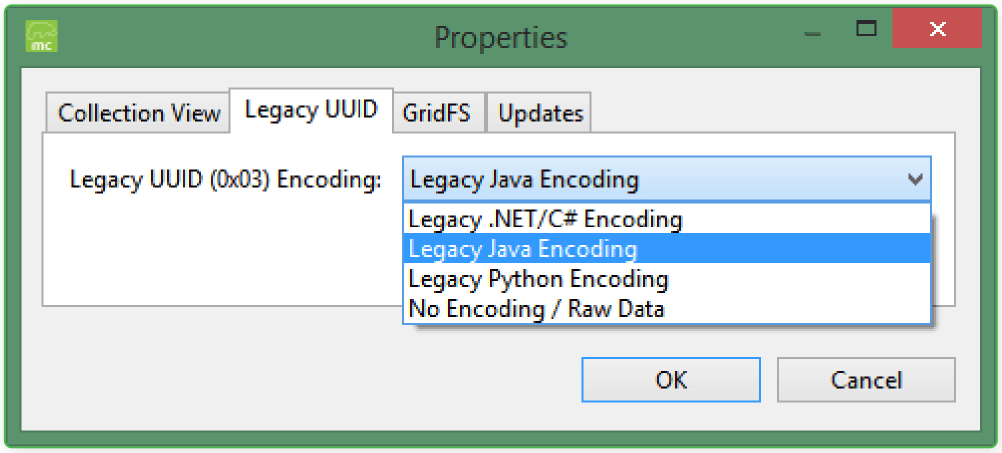
- **Legacy .NET / C# Encoding**
  - The default encoding used by the .NET/C# driver. Studio 3T will read, store and display the UUID the same way a .NET application would do

- **Legacy Java Encoding**
  - The default encoding used by the Java driver. Studio 3T will read, store and display the UUID the same way a Java application would do

- **Legacy Python Encoding**
  - The default encoding used by the Python driver. Studio 3T will read, store and display the UUID the same way a Python application would do

- **No Encoding / Raw Data**

- No conversion is being performed. The data is read, stored and displayed in the existing byte order



Legacy UUID (0x03) Encoding: Legacy Java Encoding

- Legacy .NET/C# Encoding
- Legacy Java Encoding
- Legacy Python Encoding
- No Encoding / Raw Data

Summary

Make sure your applications don't suffer from unnecessary portability issues when working with the UUID data type and avoid the legacy `Binary 0x03` subtype whenever possible. If you can't migrate your existing deployment to the new `Binary 0x04` subtype, remember to configure your MongoDB tools to the correct encoding for the legacy `Binary 0x03` subtype.

If you enjoyed this article, read more about workarounds to corner cases when querying and updating MongoDB arrays or how to query MongoDB secondary replica set members, among other tips and tricks.

Updated on July 31, 2018

Tagged: MongoDB Best Practices UUID Data

About The Author

Tomasz Naumowicz

Tomasz's first code was drawing circles on his C128D, from there he started to explore this exciting field of computing. He had fun working with devices having less than 1024 bytes of memory, has worked in software development and research, at large enterprises and small start-ups. Building rock solid software – this is what he really enjoys and has been doing for years now. Tomasz holds a

Master’s degree in Computer Science from the Freie Universtiät Berlin. He lives in Berlin and loves to spend time outdoors windsurfing and sailing.