

Revisiting Apple Notes (2): Easy Embedded Objects

13 Jan 2020 · 28 mins read

TL;DR: Embedded objects are really easy to do wrong when parsing Apple Notes, each type is like a snowflake, unique and special in how it is represented in the database. This post covers how to find embedded objects and how to piece five of them back together, along with their thumbnails.

Background

Embedding inlined pictures, improved rich text, and iCloud support for Apple Notes was introduced with iOS 9, necessitating serious changes in how Notes were stored. Gone were the days of plaintext in notes.sqlite, here are the days of large relations contained in NoteStore.sqlite. The ability to embed objects, such as pictures from your camera roll, doodles on the screen, or to do lists (actually not an embedded object, technically it is just formatting) required Apple to change from storing the text as is to finding another format. This post isn't specifically looking at the database structure or the individual Note objects, how the embedded objects in the Note are represented.

As another caveat, you won't find any one-query-to-rule-them-all in here for doing this. Each step is granular to put the emphasis on learning and understanding what is happening. Once understood, the knowledge can be applied to your specific situation easily, as opposed to learning you need to take all the output and hunt for your answer.

Where and How to Find Embedded Objects

From Note to Object

For the sake of this post, assume I have already explained the format of the Notes databas and that you know that within NoteStore.sqlite the `ZICNOTEDATA.ZDATA` field contains a GZip'd protobuf that conforms to the protobuf format Apple is using for Notes. You also know that the

protobuf has two important sections, the note text and a lot of "Attribute Runs" that explain things about that text, each of which has a length to know how much of the text it applies to.

As you parse the text in a given Note's protobuf (i.e. you have gunzipped the blob in ZICNOTEDATA.ZDATA, parsed the protobuf, and are looking at the note text field), you will come across the Unicode object replacement character "" (0xffff, integer 65532). These single-width characters represent where Apple Notes is going to replace the object, which is somewhat obvious based on the name and the intended purpose of the character. Each of these characters will have a corresponding entry in the Note protobuf's Attribute Run attachments, giving an object type, and UUID for what goes in its place. For an example, take this excerpt of a Note's protobuf which consists entirely of a Drawing, no text (output from [protobuf-inspector](#)):

```
root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = ""
      5 Attribute Run = attribute_run:
        1 Length = 1
        12 Attachment Info = attachment_info:
          1 Attachment Identifier = "2344980E-9D5D-493E-96C1-461AADB87F67"
          2 Type UTI = "com.apple.drawing.2"
```

It is hard to see, but the Note text field is the single character . There is only one attribute run, which has a length of 1 (since there's obviously only one length to the character) and it gives the UUID 2344980E-9D5D-493E-96C1-461AADB87F67 and Type UTI "com.apple.drawing.2". I say Type UTI because these types will be found both in the Note protobuf and in the ZICLOUDSYNCINGOBJECT 's ZTYPEUTI column.

Notes with a lot of text and more objects will have a lot more text and objects, but identifying each object's UUID and type is as simple as walking through the Note text, attribute run by attribute run (each will have a length, you start at 0 and go til the end), and every time you have the character, you should have attachment information for it.

```
root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = "Lots of embedded things\n\n\nFirst pic\n\n\nScanned docum\n\n\nTables above\n\n\nOrdered list\n\n\nOrdered list 2\n\n\nChecked\n\n\nchecked\n\n\nCheckbox unchecked\n\n\nCheckboxes above\n\n\n"
      5 Attribute Run = attribute_run (1 Length = 24)
      5 Attribute Run = attribute_run (1 Length = 1)
      5 Attribute Run = attribute_run:
        1 Length = 1
        12 Attachment Info = attachment_info:
          1 Attachment Identifier = "9FBFAEBC-DF04-4A6F-AE17-3DF6773FD72A"
          2 Type UTI = "public.jpeg"
```

```

5 Attribute Run = attribute_run (1 Length = 11)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run:
    1 Length = 1
    12 Attachment Info = attachment_info:
        1 Attachment Identifier = "A6957758-3E4A-469E-91A7-9131E0659E9C"
        2 Type UTI = "com.apple.notes.gallery"
5 Attribute Run = attribute_run (1 Length = 18)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run:
    1 Length = 1
    12 Attachment Info = attachment_info:
        1 Attachment Identifier = "37A789E3-6320-4ECE-8C9B-5627971094AA"
        2 Type UTI = "com.apple.notes.table"
5 Attribute Run = attribute_run (1 Length = 14)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run (1 Length = 28)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run (1 Length = 17)
5 Attribute Run = attribute_run (1 Length = 18)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run (1 Length = 17)
5 Attribute Run = attribute_run (1 Length = 1)
5 Attribute Run = attribute_run:
    1 Length = 1
    12 Attachment Info = attachment_info:
        1 Attachment Identifier = "5953C479-111E-4E81-89FA-A3579868EC91"
        2 Type UTI = "com.apple.drawing.2"

```

This example, again with a lot of bits snipped out, shows how a lot of embedded objects can be paired with the corresponding replacement character by following the lengths in each attribute run. The first object, a `public.jpeg` image with UUID 9FBFAEBC-DF04-4A6F-AE17-3DF6773FD72A, is in the third attribute run. Attribute Runs 1 and 2 have lengths 24 and 1, respectively, for a total offset of 25 and at position 25 is the first replacement character.

Continuing to add Attribute Run lengths together, you see that embedded object 2, a `com.apple.notes.gallery` with UUID A6957758-3E4A-469E-91A7-9131E0659E9C, is at offset 38, embedded object 3, a `com.apple.notes.table` with UUID 37A789E3-6320-4ECE-8C9B-5627971094AA, is at offset 58, and embedded object 4, a `com.apple.drawing.2` with UUID 5953C479-111E-4E81-89FA-A3579868EC91, is at offset 158. With that information, you have enough information to take the text and insert the embedded objects where they go! Enough information, that is, once you understand how to put the objects back together (see below).

From Object to Note

To go the opposite direction is a lot easier. You can find a given object's corresponding Note by looking in the `ZICLOUDSYNCINGOBJECT.ZNOTE` field. To use the first example from above, this query...

```
SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZNOTE,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE ZIDENTIFIER LIKE "2344980E-9D5D-493E-96C1-461AADB87F67"
```

... identifies that Note 34 had that image in it.

Z_PK	ZNOTE	ZIDENTIFIER
35	34	2344980E-9D5D-493E-96C1-461AADB87F67

How to Parse Embedded Objects

The bad news about embeded objects is each type is handled differently. The good news is that for images and some other objects, Apple keeps multiple copies of thumbnails and the image, including for doodles the user draws. This increases the opportunity for recovering information if you know how to look. Below is a breakdown of all but two of the major types I've encountered on Apple Notes running on iOS 13, how they are represented, and how to put it all back together. The two that will wait for their own post due to complexity are

`com.apple.notes.gallery` and `com.apple.notes.table`. These are specific to Notes and get more involved as a result. If you don't want to do this by hand, feel free to check out the [Apple Cloud Notes Parser](#), which handles all of the below.

public.jpeg / public.png

The Type UTIs of `public.jpeg` and `public.png` represent normal images taken by the camera, not something specific to Apple Notes. For example, you can generate these types by taking pictures directly from within the Notes application, or by selecting images you have previously taken. For the sake of this section, assume that `public.[jpeg|png]` behave the same, just the underlying image format is different.

To put a `public.jpeg` back together you look up the `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` that is listed in the Attachment Identifier and extract the `ZICLOUDSYNCINGOBJECT.ZMEDIA` field from that row. That value will be the `ZICLOUDSYNCINGOBJECT.Z_PK` of the photo's entry in `ZICLOUDSYNCINGOBJECT`, with the image's filename listed in that row's `ZICLOUDSYNCINGOBJECT.ZFILENAME` field. That filename, that file's `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` and the Account's `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` are what are needed to find the location on disk of the image. Thumbnails for the image (see further below) are found using the UUID listed directly in the Note, not the second UUID found by following the `ZMEDIA` link.

That's a mouthful, so let's take this Note as an example:

```

root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = "Picture from camera roll directly\n\n\nIt is a pencil"
      5 Attribute Run = attribute_run (1 Length = 34)
      5 Attribute Run = attribute_run (1 Length = 1)
      5 Attribute Run = attribute_run:
        1 Length = 1
      12 Attachment Info = attachment_info:
        1 Attachment Identifier = "4411963D-3FFD-4422-9FD4-22466013822E"
        2 Type UTI = "public.jpeg"

```

The plaintext would have us believe that there is a pencil picture taken directly from the camera roll. This query which pulls the Attachment Identifier from the corresponding Attribute Run in the protobuf...

```

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZMEDIA,
       ZICLOUDSYNCINGOBJECT.ZNOTE,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE ZIDENTIFIER="4411963D-3FFD-4422-9FD4-22466013822E"

```

... shows that the image is identified in entry 74.

Z_PK	ZMEDIA	ZNOTE	ZIDENTIFIER
75	74	73	4411963D-3FFD-4422-9FD4-22466013822E

With that knowledge these queries (the last comes from the results of the second)...

```

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZFILENAME,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=74;

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZACCOUNT3
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=73;

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=2;

```

... tell us everything we need to know to build the filepath.

Z_PK	ZFILENAME	ZIDENTIFIER
74	IMG_0002.jpeg	23D4CEAD-89CB-496D-A97B-DA4940B540C1

Z_PK	ZACCOUNT3
73	2

Z_PK	ZIDENTIFIER
2	LocalAccount

You build the filepath using this relative path from the Apple Notes directory:

Accounts/[Account's ZIDENTIFIER]/Media/[UUID of ZMEDIA row]/[Filename from ZMedia row] .

In our example, that becomes Accounts/LocalAccount/Media/23D4CEAD-89CB-496D-A97B-DA4940B540C1/IMG_0002.jpeg which is present in the iTunes backup and potentially relevant to our interests, showing the author was lying about the contents of the picture. Or to misquote a famous movie, "That's not a pencil! This is a pencil!"



com.apple.drawing.2

The Type UTI of `com.apple.drawing.2` (and ostensibly `com.apple.drawing` which I've also seen reference to) represents the user's ability to doodle on the screen and have Notes store the resulting image. Apple will try to parse this as handwriting (see OCR section below), but will

put its guess as to what the words say into the
 ZICLOUDSYNCINGOBJECT.ZADDITIONALINDEXABLETEXT field.

To put a `com.apple.drawing.2` back together you can do two things. The easiest answer is to use the images that Apple generates and use the `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` that is listed in the Attachment Identifier and the Note Account's
`ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` to build a filepath to the fallback image. Thumbnails will also be generated using the Attachment Identifier. For example, if this is the note:

```
root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = "Lots of embedded things\n\n\nFirst pic\n\n\nScanned docum
        "\n\n\nTables above\n\n\nOrdered list\n\n\nOrdered list 2\n\n\nChe
        "checked\n\n\nCheckbox unchecked\n\n\nCheckboxes above\n\n\n"
      5 Attribute Run = attribute_run:
        1 Length = 1
        12 Attachment Info = attachment_info:
          1 Attachment Identifier = "5953C479-111E-4E81-89FA-A3579868EC91"
          2 Type UTI = "com.apple.drawing.2"
```

Then these queries (the second and third are both derived from from the results of the prior one)...

```
SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZNOTE,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE ZIDENTIFIER="5953C479-111E-4E81-89FA-A3579868EC91"

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZACCOUNT3
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=88;

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=2;
```

... give us what we need to build the file path.

Z_PK	ZNOTE	ZIDENTIFIER
102	88	5953C479-111E-4E81-89FA-A3579868EC91

Z_PK	ZACCOUNT3
------	-----------

Z_PK	ZACCOUNT3
88	2

Z_PK	ZIDENTIFIER
2	LocalAccount

You build the filepath using this relative path from the Apple Notes directory:

Accounts/[Account's ZIDENTIFIER]/FallbackImages/[Attachment's ZIDENTIFIER].jpg . As a fun fact, the extension says JPEG, but this file is actually a PNG format. Never, ever, trust a file extension.

In our example, that becomes Accounts/LocalAccount/FallbackImages/5953C479-111E-4E81-89FA-A3579868EC91.jpg , or this ugly mess.



The harder answer is to pull the ZICLOUDSYNCINGOBJECT.ZMERGEABLEDATA1 field from the row identified by the Attachment's ZICLOUDSYNCINGOBJECT.ZIDENTIFIER , parse the embedded protobuf, and then follow the directions therein to rebuild the lines. I do not recommend this course of action unless you only have the Notes database and cannot get access to the logical or physical backups, thumbnails, etc.

That query would be...

```
SELECT ZICLOUDSYNCINGOBJECT.ZMERGEABLEDATA1
FROM ZICLOUDSYNCINGOBJECT
WHERE ZICLOUDSYNCINGOBJECT.ZIDENTIFIER="5953C479-111E-4E81-89FA-A3579868EC91"
```


... and dealing with the results is left as an exercise for the reader.

public.url

The Type UTI of `public.url` represents shared URL objects, although these do not come from directly pasting a URL into the Note (which results in only text). These are generated by applications such as Maps, Safari, or Firefox that have a "share to" option using Notes. In the case of a web browser, this is obvious, but Maps and some other applications are more interesting in that they generating URLs to represent what you look at in the application.

To put a `public.url` back together is trivial, you look up the `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` that is listed in the Attachment Identifier and extract the `ZICLOUDSYNCINGOBJECT.ZURLSTRING` from that row. For example, if this is the note:

```
root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = "Embedded thing from safari's share menu\n\n"
      5 Attribute Run = attribute_run (1 Length = 40)
      5 Attribute Run = attribute_run:
        1 Length = 1
        12 Attachment Info = attachment_info:
          1 Attachment Identifier = "035634F8-0F88-4BDC-A183-943C0775122A"
          2 Type UTI = "public.url"
      5 Attribute Run = attribute_run (1 Length = 1)
```

Then you could use this query...

```
SELECT ZICLOUDSYNCINGOBJECT.Z_PK, ZICLOUDSYNCINGOBJECT.ZTYPEUTI,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER, ZICLOUDSYNCINGOBJECT.ZURLSTRING
FROM ZICLOUDSYNCINGOBJECT
WHERE ZICLOUDSYNCINGOBJECT.ZIDENTIFIER="035634F8-0F88-4BDC-A183-943C0775122A"
```

... to get these results:

Z_PK	ZTYPEUTI	ZIDENTIFIER	ZURLSTRING
156	public.url	035634F8-0F88-4BDC-A183-943C0775122A	https://www.ciofecaforensics.com/

public.vcard

The Type UTI of `public.vcard` represents the storage of a contact, likely shared from the Contacts application or a contact directly. These are kept in VCF format in a plaintext file.

To put a `public.vcard` back together you follow essentially the same steps as `public.jpeg`. First you look up the `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` that is listed in the Attachment Identifier and extract the `ZICLOUDSYNCINGOBJECT.ZMEDIA` field from that row. That value will be the `ZICLOUDSYNCINGOBJECT.Z_PK` of the vcard's entry in `ZICLOUDSYNCINGOBJECT`, with the vcard's filename listed in that row's `ZICLOUDSYNCINGOBJECT.ZFILENAME` field. That filename, that file's `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` and the Account's `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` are what are needed to find the location on disk of the image. For example, if this is the note:

```
root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = "Embedded contact, synced from mac's Contacts application\
      5 Attribute Run = attribute_run: (1 Length = 57)
      5 Attribute Run = attribute_run:
        1 Length = 1
        12 Attachment Info = attachment_info:
          1 Attachment Identifier = "F6434D7E-6D78-4FD6-95E0-DBF8D751C497"
          2 Type UTI = "public.vcard"
      5 Attribute Run = attribute_run: (1 Length = 1)
```

This query...

```
SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZMEDIA,
       ZICLOUDSYNCINGOBJECT.ZNOTE,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE ZIDENTIFIER = "F6434D7E-6D78-4FD6-95E0-DBF8D751C497"
```

... shows that the vcard is identified in entry 173.

Z_PK	ZMEDIA	ZNOTE	ZIDENTIFIER
169	173	168	F6434D7E-6D78-4FD6-95E0-DBF8D751C497

With that knowledge these queries (the last comes from the results of the second)...

```
SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZFILENAME,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=173;

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZACCOUNT3
```

```
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=168;

SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER
FROM ZICLOUDSYNCINGOBJECT
WHERE Z_PK=10;
```

... tell us everything we need to know to build the filepath.

Z_PK	ZFILENAME	ZIDENTIFIER
173	Apple Inc..vcf	16CA0042-928E-45D4-BFFF-4A56AC62BA32

Z_PK	ZACCOUNT3
168	10

Z_PK	ZIDENTIFIER
10	D34714F2-F2F7-4AD0-8EA5-A54E31A74D72

You build the filepath using this relative path from the Apple Notes directory:

Accounts/[Account's ZIDENTIFIER]/Media/[UUID of ZMEDIA row]/[Filename from ZMedia row] .

In our example, that becomes Accounts/D34714F2-F2F7-4AD0-8EA5-A54E31A74D72/Media/16CA0042-928E-45D4-BFFF-4A56AC62BA32/Apple Inc..vcf which contains the contact's details:

```
BEGIN:VCARD
VERSION:3.0
PRODID:-//Apple Inc.//Mac OS X 10.15.1//EN
N:;;;
FN:Apple Inc.
ORG:Apple Inc.;
TEL;type=MAIN;type=pref:1-800-MY-APPLE
item1.ADR;type=WORK;type=pref:;;1 Infinite Loop;Cupertino;CA;95014;United States
item1.X-ABADR:us
item2.URL;type=pref:http://www.apple.com
item2.X-ABLabel:_$!<HomePage>!$_
X-ABShowAs:COMPANY
END:VCARD
```

Other Notes (haha) about Embedded Objects

Thumbnails

It is worth mentioning the value in how thumbnails are identified and named as Apple encodes some valuable information in them and makes a *lot* of them. Regardless of the type of object that is thumbnailed, the `ZICLOUDSYNCINGOBJECT.ZIDENTIFIER` field follows a format of `[ZIDENTIFIER]-[ZSCALE]-[ZWIDTH]-[ZHEIGHT]-[ZAPPEARANCETYPE]`. All of those parts are named after columns in the `ZICLOUDSYNCINGOBJECT`'s (iOS 13) database because the values are also held in there. `ZIDENTIFIER` is the UUID for the base image or object, `ZSCALE` is the scale to display the image at, `ZWIDTH` is the image's width, `ZHEIGHT` is the image's height, `ZAPPEARANCETYPE` is whether it is light mode (0) or dark mode (1).

All thumbnails appear to have a `ZICLOUDSYNCINGOBJECT.Z_ENT` field equal to 5 and a `ZICLOUDSYNCINGOBJECT.ZATTACHMENT` field equal to the base image's `ZICLOUDSYNCINGOBJECT.Z_PK`. Finally, you likely will want to know what type of file it is, to finish the filename. It seems like for `com.apple.notes.gallery` the file format is png (and has a png extension) but for the other public sources, the file format is JPEG and uses .jpg.

Let's walk through all of this with a Maps example which generated no less than 6 thumbnail images.

```
root:
  2 <document> = document:
    3 Note = note:
      2 Note Text = "Maps shared object of Caffè Nero, from Mac synced with iCloud"
      5 Attribute Run = attribute_run: (1 Length = 63)
      5 Attribute Run = attribute_run:
        1 Length = 1
        12 Attachment Info = attachment_info:
          1 Attachment Identifier = "089EFB19-A4C6-454A-B874-70C84C6645E2"
          2 Type UTI = "public.url"
      5 Attribute Run = attribute_run: (1 Length = 1)
```

This query for all of the potential thumbnails (and the base object)...

```
SELECT ZICLOUDSYNCINGOBJECT.Z_PK,
       ZICLOUDSYNCINGOBJECT.ZIDENTIFIER,
       ZICLOUDSYNCINGOBJECT.ZATTACHMENT
FROM ZICLOUDSYNCINGOBJECT
WHERE ZIDENTIFIER LIKE "089EFB19-A4C6-454A-B874-70C84C6645E2%"
ORDER BY ZIDENTIFIER
```

... results in these values.

Z_PK	ZIDENTIFIER	ZATTACHMENT
160	089EFB19-A4C6-454A-B874-70C84C6645E2 (the base object)	
165	089EFB19-A4C6-454A-B874-70C84C6645E2-1-384x384-0	160

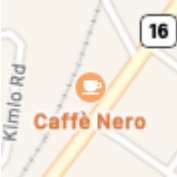
Z_PK	ZIDENTIFIER	ZATTACHMENT
162	089EFB19-A4C6-454A-B874-70C84C6645E2-1-88x88-0	160
163	089EFB19-A4C6-454A-B874-70C84C6645E2-2-384x384-0	160
166	089EFB19-A4C6-454A-B874-70C84C6645E2-2-384x384-1	160
164	089EFB19-A4C6-454A-B874-70C84C6645E2-2-88x88-0	160
167	089EFB19-A4C6-454A-B874-70C84C6645E2-2-88x88-1	160


To see how and why those images differ, this is how each thumbnail name breaks out:

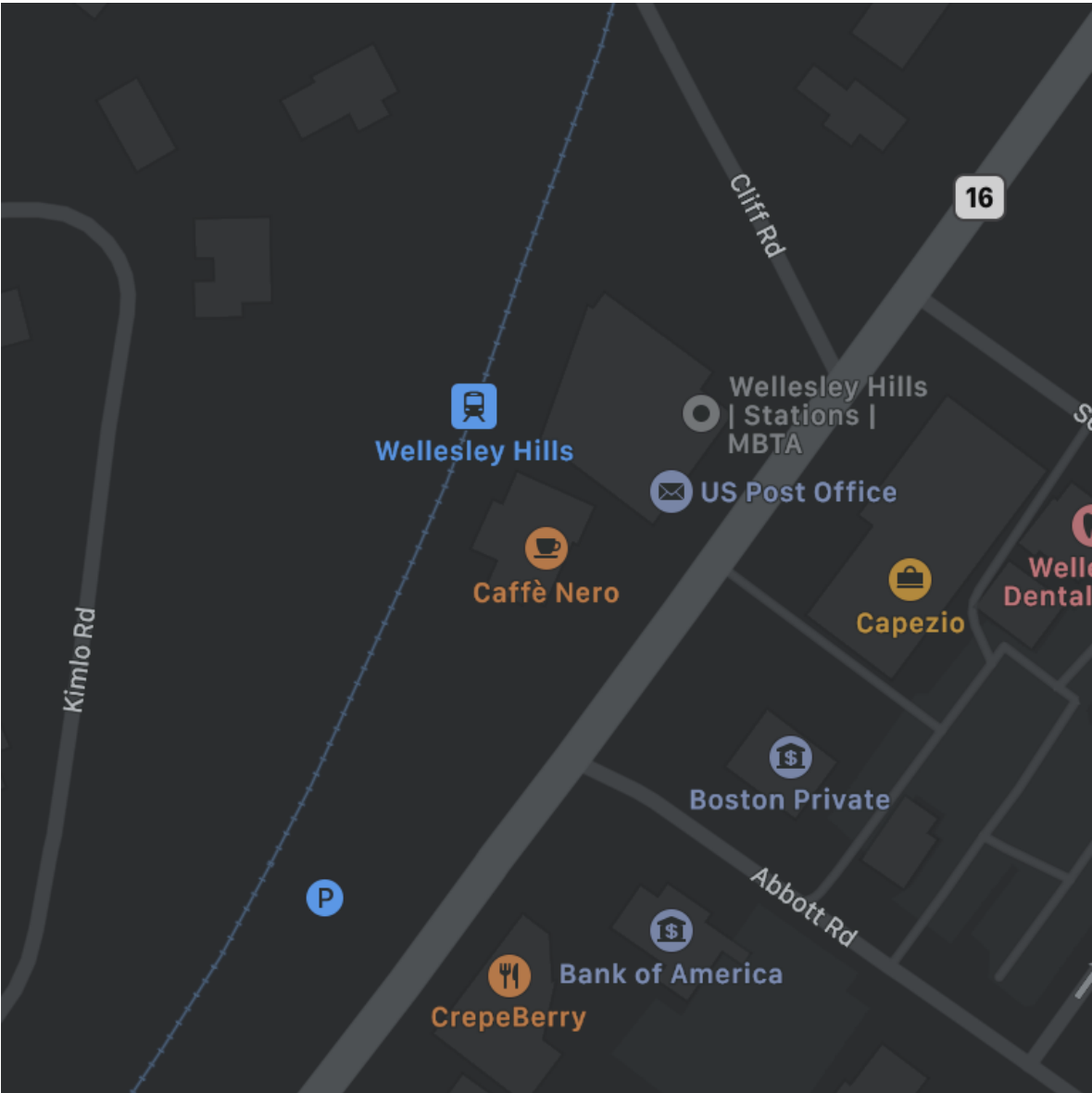

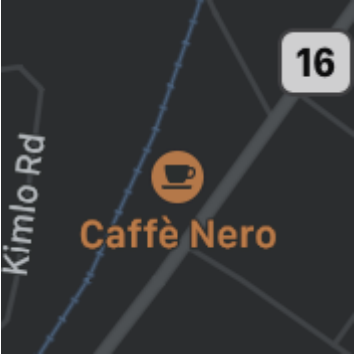
Z_PK	UUID	Scale	Width	Height	Appearance
165	089EFB19-A4C6-454A-B874-70C84C6645E2	1	384	384	Light
162	089EFB19-A4C6-454A-B874-70C84C6645E2	1	88	88	Light
163	089EFB19-A4C6-454A-B874-70C84C6645E2	2	384	384	Light
166	089EFB19-A4C6-454A-B874-70C84C6645E2	2	384	384	Dark
164	089EFB19-A4C6-454A-B874-70C84C6645E2	2	88	88	Light
167	089EFB19-A4C6-454A-B874-70C84C6645E2	2	88	88	Dark

... and these would be the images themselves, extracted from the logical backup (no matter how I tried, I couldn't figure out a "nice" way to display this):

Z_PK	Thumbnail
-------------	------------------

Z_PK	Thumbnail
165	
162	
163	

Z_PK	Thumbnail
	

Z_PK	Thumbnail
166	
164	
167	

So what does this tell you? As you go through your forensic materials, expect to find a *lot* of copies of the same image in different places. Also, expect that you can recreate some of the database values from the thumbnail name itself, which may help when reconstructing deleted rows, or database fragments to try to work back to which note something may have been removed from.

OCR

Much like Google, Apple has a better product (you) if they can tell everything you do. When you take a picture, that image isn't searchable (what you'll care about) or taggable (what they'll care about) on its own. So naturally, Apple runs OCR against it to give you the ability to search for the image and tags it. It is worth noting that this appears to have shown up in iOS 13 for Notes.

Two relevant database columns to look at are

`ZICLOUDSYNCINGOBJECT.ZIMAGECLASSIFICATIONSUMMARY` and `ZICLOUDSYNCINGOBJECT.ZOCRSUMMARY` (and as noted above, `ZICLOUDSYNCINGOBJECT.ZADDITIONALINDEXABLETEXT` is where text parsed from `com.apple.drawing.2` objects would appear). The former is a list of the image classifications generated by Apple and the latter is the text it thought it saw in the image. To remove any confusion, it needs to be noted that Apple generates a `ZICLOUDSYNCINGOBJECT.ZSUMMARY` and `ZICLOUDSYNCINGOBJECT.ZTITLE` for the images it runs OCR on, based on the OCR'd text, this is not user generated.

For an example, let's revisit the `public.jpeg` image used above. This query...

```
SELECT ZICLOUDSYNCINGOBJECT.ZIMAGECLASSIFICATIONSUMMARY,  
       ZICLOUDSYNCINGOBJECT.ZOCRSUMMARY,  
       ZICLOUDSYNCINGOBJECT.ZSUMMARY,  
       ZICLOUDSYNCINGOBJECT.ZTITLE  
FROM ZICLOUDSYNCINGOBJECT  
WHERE ZIDENTIFIER LIKE "4411963D-3FFD-4422-9FD4-22466013822E"
```

... shows that Apple does not know how to classify an obvious picture of a knife, but can recognize the make of my EDC. It also set the summary and title based on the text it ripped from the picture. My favorite part is that it took the cutouts in the handle to be zeros.

ZIMAGECLASSIFICATIONSUMMARY	ZOCRSUMMARY	ZSUMMARY	ZTITLE
	WINCHESTER WINCHESTER® WINCHESTER.00 000	WINCHESTER WINCHESTER® WINCHESTER.00 000	WINCHESTER

Trying this on other objects proved more helpful and made it clear that Apple appends all possible matches to the `ZICLOUDSYNCINGOBJECT.ZIMAGECLASSIFICATIONSUMMARY` field. For

example, a sticky note I took a picture of hit on Document, Written Document, Handwriting, Paper Documents, etc. All of those are in the field, separated by spaces.

This can be helpful if a device has a lot of pictures on it. While other tools likely will also do some form of image classification, it may be worth checking out Apple's built-in classifications to see if there are any obvious wins.

Conclusion

Apple has consistently improved Notes by adding new ways to collaborate and take Notes over the years. The ways users can embed objects are fairly important to understanding the plaintext taken from the note (was the todo list of 'steal stuff' checked, or unchecked? Was the meeting time written underlined, or strikethrough?) and this post hopes to help explain those embedded objects.

A future post will cover the Apple Notes specific object types `com.apple.notes.gallery` and `com.apple.notes.table`, but for now hopefully this is a help. If you don't want to do this by hand, seriously consider my [Apple Cloud Notes Parser](#) over on Github, but even if you use that, you'll want to know the underlying theory behind what it is doing.

Previous

< Revisiting Apple Notes (1):...

Next

Revisiting Apple Notes (3):... >



© 2020 Ciofeca Forensics. All rights reserved.