iOS Application State & the Snapshot Carousel

Control-F ***(Feb 2021)***

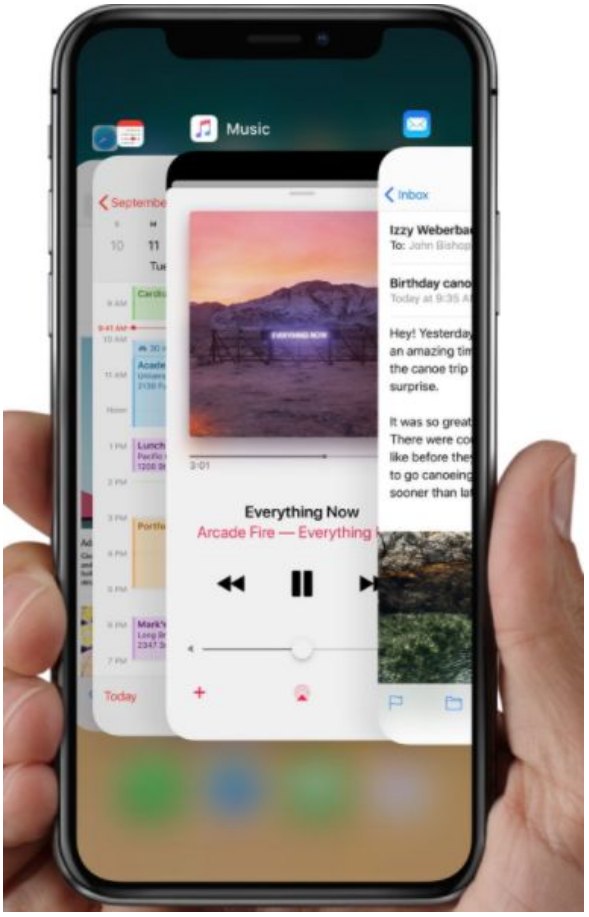
Michael Bangham

```
var applicationState: UIApplication.State { get }
```

This line of code determines an app's current state, or that of its most active state.

The behavior of this property depends on whether your app is scene-based. Apple Developer Support states the following:

> *"In a scene-based app, this property takes the value of the most active scene, which it determines from each scene's activationState property. A scene-based app launches in the background state, and transitions between its states as scenes connect, change their states, and disconnect. For scene-based apps, use UISceneDelegate to respond to changes in an individual scene's life cycle."*



Most third party apps that we as forensic investigators are interested in are scene based apps with a front end GUI that users interact with and add data to. Users can easily alter the state of a scene based app by executing it and then forcing it into the background by either minimising the application or closing it. On iOS, a background application can be reinstated by selecting it from the `Carousel` (image right) or swiping the object upwards in order to disconnect/close it.

As a forensic investigator it is important to us to know both, what an application's state was at a particular time and what may have been present on the app visually when it was pushed to the background.

Fortunately the artefacts we require are available to us, they just require a bit of digging.

The iOS Carousel in the screenshot above presents a series of snapshot images for open applications. On older versions of iOS, these snapshots were saved as PNG files that would parse with vendor tools, hassle free. Since iOS 10, Apple have transitioned to storing the Carousel snapshots using Ericsson Texture Compression. This is a texture compression format that was made standard by Khronos for OpenGL ES 2.0. Many vendor tools are still not parsing these snapshots and as such the relevance of their visual content is often missed in forensic investigation.

System .KTX snapshot files on an iOS device can be found:

/private/var/mobile/Library/Caches/Snapshots/

While snapshots for pre-installed Apple applications can be found;

/private/var/mobile/Containers/Data/Applications/<UUID>/Library/Caches/Snapshots/

`Yogesh Khatri` has created a Python script to convert/decompress the KTX files found in the above locations into PNG format. His work can be found in the below GitHub link.

https://github.com/ydkhatri/MacForensics/tree/master/IOS_KTX_TO_PNG

So now we can visualise the snapshots as images, it would be nice to give them provenance. Fortunately we can do this using an iOS database called `applicationState.db`. This a database responsible for storing information about an application's state and any objects associated with the application during runtime. The database can be found:

/root/private/var/mobile/Library/FrontBoard/applicationState.db

The following screenshots from applicationState.db show three of the tables present:

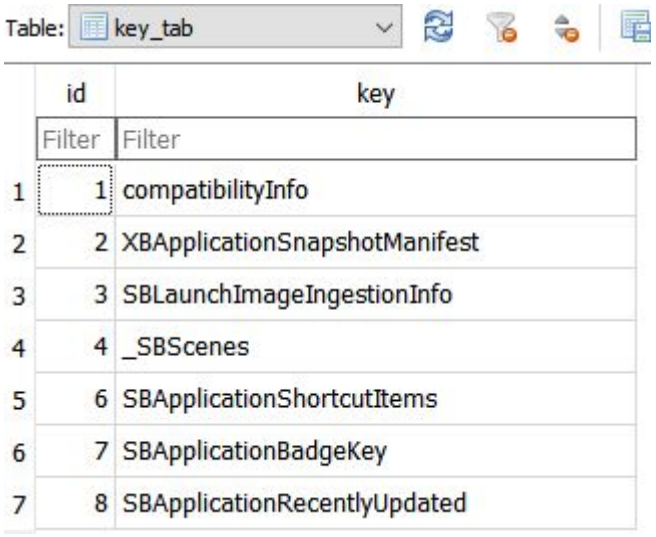


Table: key_tab

	id	key
	Filter	Filter
1	1	compatibilityInfo
2	2	XBApplicationSnapshotManifest
3	3	SBLaunchImageIngestionInfo
4	4	_SBScenes
5	6	SBApplicationShortcutItems
6	7	SBApplicationBadgeKey
7	8	SBApplicationRecentlyUpdated

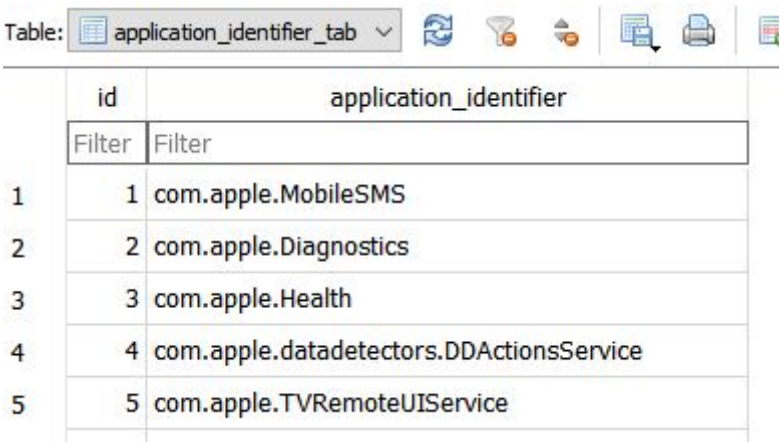


Table: application_identifier_tab

	id	application_identifier
	Filter	Filter
1	1	com.apple.MobileSMS
2	2	com.apple.Diagnostics
3	3	com.apple.Health
4	4	com.apple.datadetectors.DDActionsService
5	5	com.apple.TVRemoteUIService

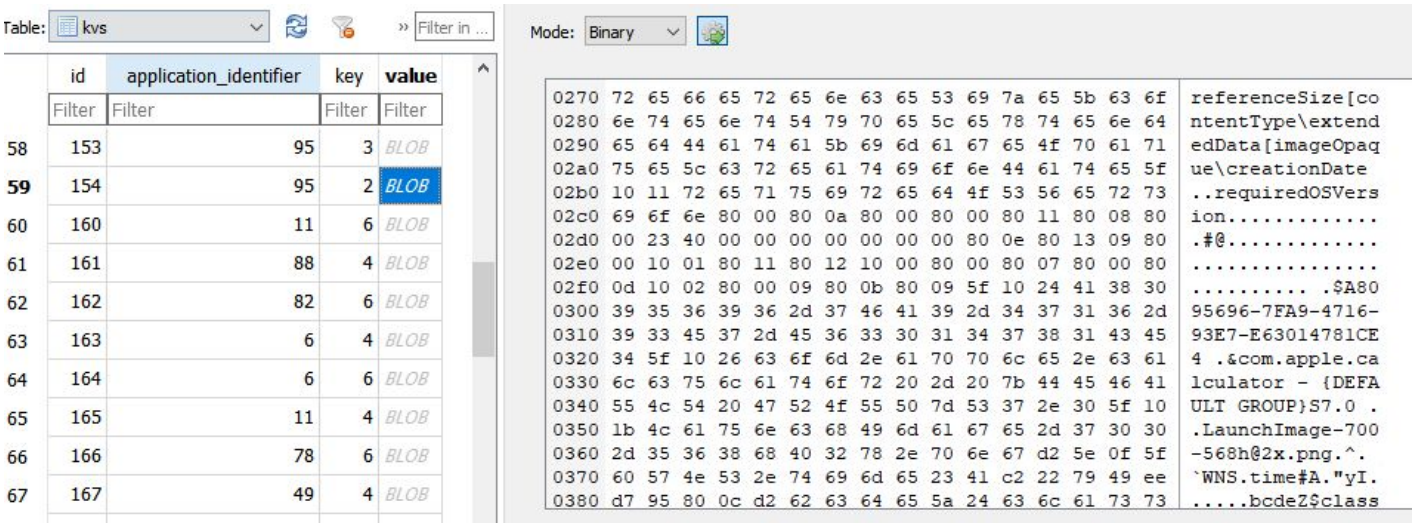


Table: kvs

	id	application_identifier	key	value
	Filter	Filter	Filter	Filter
58	153	95	3	BLOB
59	154	95	2	BLOB
60	160	11	6	BLOB
61	161	88	4	BLOB
62	162	82	6	BLOB
63	163	6	4	BLOB
64	164	6	6	BLOB
65	165	11	4	BLOB
66	166	78	6	BLOB
67	167	49	4	BLOB

Mode: Binary

```
0270 72 65 66 65 72 65 6e 63 65 53 69 7a 65 5b 63 6f  referenceSize[co
0280 6e 74 65 6e 74 54 79 70 65 5c 65 78 74 65 6e 64  ntentType\extend
0290 65 64 44 61 74 61 5b 69 6d 61 67 65 4f 70 61 71  edData[imageOpaq
02a0 75 65 5c 63 72 65 61 74 69 6f 6e 44 61 74 65 5f  ue\creationDate
02b0 10 11 72 65 71 75 69 72 65 64 4f 53 56 65 72 73  ..requiredOSVers
02c0 69 6f 6e 80 00 80 0a 80 00 80 00 80 11 80 08 80  ion.............
02d0 00 23 40 00 00 00 00 00 00 00 80 0e 80 13 09 80  .#@.............
02e0 00 10 01 80 11 80 12 10 00 80 00 80 07 80 00 80  ................
02f0 0d 10 02 80 00 09 80 0b 80 09 5f 10 24 41 38 30  .......... .$A80
0300 39 35 36 39 36 2d 37 46 41 39 2d 34 37 31 36 2d  95696-7FA9-4716-
0310 39 33 45 37 2d 45 36 33 30 31 34 37 38 31 43 45  93E7-E63014781CE
0320 34 5f 10 26 63 6f 6d 2e 61 70 70 6c 65 2e 63 61  4 .&com.apple.ca
0330 6c 63 75 6c 61 74 6f 72 20 2d 20 7b 44 45 46 41  lculator - {DEFA
0340 55 4c 54 20 47 52 4f 55 50 7d 53 37 2e 30 5f 10  ULT GROUP}S7.0 .
0350 1b 4c 61 75 6e 63 68 49 6d 61 67 65 2d 37 30 30  .LaunchImage-700
0360 2d 35 36 38 68 40 32 78 2e 70 6e 67 d2 5e 0f 5f  -568h@2x.png.^._
0370 60 57 4e 53 2e 74 69 6d 65 23 41 c2 22 79 49 ee  `WNS.time#A."yI.
0380 d7 95 80 0c d2 62 63 64 65 5a 24 63 6c 61 73 73  .....bcdeZ$class
```

Examining these three tables we can spot a link between an application, it's scene state and a large amount of metadata for that scene in the form of a bplist (BLOB). Held inside the bplist are names of snapshot KTX files and metadata such as a creation date.

An important feature of this database, unlike many other databases on iOS, is that its PRAGMA environment variable of **'auto_vaccum'** is set to **NONE**. This means this database does not clean freespace with new insertions and deletions. Artefacts relating to the state of applications that have been deleted can therefore be found here.

Using an SQL Query we can join the data so as to access the BLOB data for a snapshot:

```
SELECT

kvs.value AS bplist,

kvs_debug.value AS debug_bplist,

Application_identifier_tab.id,

application_identifier_tab.application_identifier

FROM kvs

LEFT JOIN

application_identifier_tab ON application_identifier_tab.id =
kvs.application_identifier

LEFT JOIN

key_tab ON kvs.key = key_tab.id

LEFT JOIN

kvs_debug ON application_identifier_tab.application_identifier =
kvs_debug.application_identifier

WHERE
```

```
key_tab.key = 'XBApplicationSnapshotManifest' AND key_tab.key = kvs_debug.key

ORDER BY

application_identifier_tab.id
```

Now that we have the bplist data for all the `XBApplicationSnapshotManifest` items we can use CCL's binary Plist script to convert the NSKeyArchiver structure to a native Python dictionary format.

Now we have a dictionary containing the snapshots name and the metadata attributed to it. We can now attribute our visual snapshots to its metadata. Below is a screenshot of the HTML report created using ***Control-F's*** - **`parse_ios_app_carousel_snapshots.py`** script. The script can be executed on an iOS full file system zip/tar archive (typical format from Cellebrite or GrayKey) without the need to extract any files. It will:

1. Locate and extract all KTX files by exporting them out of the archive.
2. Use Yogesh Khatri's script to decompress the KTX files to PNG
3. Execute the above SQL query on the applicationState.db
4. Parse the bplist and attribute the metadata to the snapshots.

Snapshot Properties

Snapshot Filename: 4A56B32D-F55D-400D-AAFB-D56DA8BE2722@2x.ktx.png

Metadata
relativePath: 4A56B32D-F55D-400D-AAFB-D56DA8BE2722@2x.ktx
groupID: ch.protonmail.protonmail - {DEFAULT GROUP}
imageScale: 2.0
variantID: $null
identifier: 4A56B32D-F55D-400D-AAFB-D56DA8BE2722
lastUsedDate: 2020-04-16 13:08:13
referenceSize: {320, 568}
imageOpaque: True
creationDate: 2020-04-13 20:31:05