

IPL

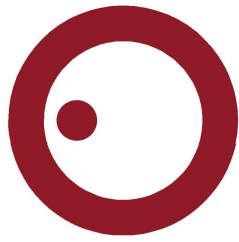
escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Eng.^a Informática – Computação Móvel

DIGITAL FORENSIC ARTIFACTS OF SQLITE-BASED
WINDOWS 10 APPLICATIONS

LUÍS MIGUEL ANTÓNIO ANDRADE

Leiria, Novembro de 2020



IPL

escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Eng.^ª Informática – Computação Móvel

**DIGITAL FORENSIC ARTIFACTS OF SQLITE-BASED
WINDOWS 10 APPLICATIONS**

LUÍS MIGUEL ANTÓNIO ANDRADE

Número: 2180234

Dissertação realizada sob orientação do Professor Patrício Rodrigues Domingues e Professor Miguel Monteiro de Sousa Frade.

Leiria, Novembro de 2020

AGRADECIMENTOS

Quero agradecer, em primeiro lugar, aos meus orientadores, Professor Patrício Rodrigues Domingues e Professor Miguel Monteiro de Sousa Frade, por todo o trabalho e ajuda que dedicaram a mim e à minha dissertação. Especialmente por nunca terem desistido ou perdido a confiança em mim, quando a minha motivação estava num ponto mais baixo.

Gostaria de agradecer também a dois amigos, Bruce Coelho e Luís Paulo, que me ajudaram quando foi necessário a testar certos cenários para a dissertação. Sem eles não teria conseguido realizar todos os testes necessários à realização do meu trabalho.

RESUMO

O Windows 10 é um dos [Operating System \(OS\)](#) mais populares e utilizado. Contém vários serviços, como o [Windows Push Notification Services \(WNS\)](#) e o *Timeline*, que usam bases de dados SQLite. O Windows 10 tem também uma plataforma, [Universal Windows Platform \(UWP\)](#), para suportar o desenvolvimento de aplicações. As aplicações desta plataforma podem guardar os seus dados em bases de dados SQLite, como o *Photos* da Microsoft e o *Messenger* do Facebook.

Esta dissertação estuda, numa perspectiva de análise digital forense, dois componentes do Windows 10, o ambiente Your Phone, e o [WNS](#). O primeiro consiste de uma aplicação Android, [Your Phone Companion \(YPC\)](#), e uma aplicação [UWP](#), Your Phone. O último é um sistema do Windows 10 que disponibiliza o serviço de notificações. No âmbito desta dissertação foram desenvolvidos *scripts* para analisar esses componentes, extraindo-se os artefactos forenses considerados mais relevantes. As soluções desenvolvidas estão integradas com o conhecido software de análise forense Autopsy.

Para ajudar a desenvolver e manter estas soluções de forense digital que analisam artefactos produzidos por aplicações [UWP](#), foi desenvolvido o [UWP scanner](#). Trata-se de um analisador de aplicações focado na deteção de alterações ao nível das bases de dados SQLite empregue por aplicações [UWP](#). Esta ferramenta ajuda a manter um histórico da evolução das bases de dados utilizadas por certas aplicações [UWP](#).

ABSTRACT

Windows 10 is one of the most popular and widely used [Operating System \(OS\)](#). It contains services, such as [Windows Push Notification Services \(WNS\)](#) and *Timeline*, that use SQLite databases. Windows 10 also has a platform, [Universal Windows Platform \(UWP\)](#), to support application development. The platform applications can store their data in SQLite databases, such as Microsoft's *Photos* and Facebook's *Messenger*.

This dissertation studies, from a digital forensic analysis perspective, two components of Windows 10, the Your Phone environment, and [WNS](#). The former consists of the Android application [Your Phone Companion \(YPC\)](#), and the desktop [UWP](#) application Your Phone. The latter is the notification system of Windows 10 which provides the notification service. In the scope of this dissertation, several scripts were developed to analyze these components, extracting the forensic artifacts considered to be relevant. The developed solutions are integrated with the known digital forensics analysis software Autopsy.

To help develop and maintain digital forensic solutions to analyze the digital forensic artifacts produced by [UWP](#) applications, the [UWP](#) scanner was developed. It is an application analyzer focused on detecting changes in SQLite databases of [UWP](#) applications. This tool allows us to keep a history of the evolution of the databases used in certain [UWP](#) applications.

TABLE OF CONTENTS

Acknowledgments	i
Resumo	iii
Abstract	v
Table of contents	vii
List of Figures	ix
List of Tables	xi
List of Acronyms	xiii
1 INTRODUCTION	1
1.1 Context	1
1.2 Motivation	2
1.3 Objectives	2
1.4 Contributions	3
1.5 Organization of the dissertation	4
2 BACKGROUND	5
2.1 Digital forensics	5
2.1.1 Autopsy	5
2.2 Databases: definition and origins	8
2.3 SQLite	9
2.3.1 In detail	9
2.3.2 In mobile devices and apps	15
2.3.3 Microsoft, Windows 10, and UWP	16
2.4 Windows 10's Your Phone and YPA	18
2.4.1 Previous work	18
3 UWP APPLICATIONS	21
3.1 Universal Windows Platform	21
3.2 Keeping up with apps' updates	22
3.2.1 Server schema and rules	25
3.2.2 Delving into the script	25
4 YOUR PHONE	31

TABLE OF CONTENTS

4.1	Applications	31
4.1.1	Your Phone Companion	32
4.1.2	Desktop application	38
4.2	Your Phone Analyzer	54
4.2.1	Ingest module	54
4.2.2	Communications Visualization	57
4.2.3	Report module	59
5	WINDOWS NOTIFICATIONS	65
5.1	WNS and Action Center	65
5.1.1	User data	68
5.1.2	NotifAnalyzer	71
5.2	Standalone script	71
5.3	Autopsy ingest module	73
5.3.1	Integration with YPA	74
5.3.2	Running in a digital forensics image	75
6	CONCLUSIONS	77
6.1	Future work	78
	BIBLIOGRAPHY	81
	Appendices	
A	APPENDIX A	89
B	APPENDIX B	93
	DECLARAÇÃO	99

LIST OF FIGURES

Figure 1	Autopsy GUI	6
Figure 2	Autopsy workflow	7
Figure 3	Autopsy Communications Visualization	8
Figure 5	PRAGMAs of an SQLite3 database.	12
Figure 4	PRAGMA command syntax.	12
Figure 6	CREATE VIRTUAL TABLE command syntax.	14
Figure 7	UWP overview.	16
Figure 8	Your Phone in the Microsoft Store	17
Figure 9	Data directory hierarchy of an UWP application.	22
Figure 10	Scanner architecture.	23
Figure 11	Example of a scanner execution.	23
Figure 12	Scanner database dashboard.	24
Figure 13	List of monitored apps in the scanner.	28
Figure 14	Scanner displaying YourPhone’s evolution.	29
Figure 15	YPC screens.	32
Figure 16	YPC ’s screen for pre 7.0 Android devices.	33
Figure 17	Android API level distribution.	34
Figure 18	Share a link with YPC	34
Figure 19	YPC notification in the smartphone.	35
Figure 20	YPC local data path tree.	37
Figure 21	Your Phone interface.	39
Figure 22	Your Phone messages and calls interface.	40
Figure 23	Your Phone settings interface, with Media Controls.	41
Figure 24	YPC ’s Media Controls notification.	41
Figure 25	Your Phone local data path tree.	42
Figure 26	Your Phone interface for disabling notifications.	53
Figure 27	YPA ’s artifacts in Autopsy’s interface.	55
Figure 28	YPA ’s ingest module GUI	56
Figure 29	Communications Visualization of a case using YPA	58
Figure 30	YPA ’s report module GUI	59
Figure 31	YPA ’s report module - <i>Conversations</i> page.	60
Figure 32	YPA ’s report contact modal.	61

LIST OF FIGURES

Figure 33	YPA’s report module - <i>Address book</i> page.	61
Figure 34	YPA’s report module - <i>Photos</i> page.	62
Figure 35	YPA’s report module - <i>Photos</i> ’ modal.	62
Figure 36	YPA’s report module - <i>Call history</i> page.	63
Figure 37	YPA’s report module - <i>Apps</i> page.	63
Figure 38	WNS’s diagrams of the data flow of a push notification . . .	66
Figure 39	Windows 10 Action Center and notification settings.	67
Figure 40	Types of Windows 10 notifications.	68
Figure 41	Different Windows 10 notification toasts. Source: Microsoft Docs (2020b).	68
Figure 42	NotifAnalyzer’s flow diagram	71
Figure 43	NotifAnalyzer standalone script execution.	72
Figure 44	NotifAnalyzer standalone script toast.	73
Figure 45	NotifAnalyzer’s artifacts in Autopsy’s interface.	73
Figure 46	NotifAnalyzer’s ingest module GUI.	74
Figure 47	YPA’s integration with notifications from NotifAnalyzer. . .	75
Figure 48	NotifAnalyzer artifacts ran on a forensic disk image.	76
Figure 49	YPC’s <code>content.db</code>	94
Figure 50	YPC’s <code>eventstore</code>	95
Figure 51	YPC’s <code>YourPhoneSettings</code>	95
Figure 52	YPC’s <code>ContentTransferDatabase</code>	96
Figure 53	YPC’s <code>com.google.android.datatransport.events</code>	96
Figure 54	YPC’s <code>com.microsoft.appcenter.persistence</code>	97

LIST OF TABLES

Table 1	Types of temporary SQLite files.	11
Table 2	Scanner dependencies.	25
Table 3	Android permissions declared by Your Phone Companion . .	36
Table 4	Databases of Your Phone - version 1.20071.95.0	43
Table 5	Fields of table <code>call_history</code> / <code>calling.db</code> database	44
Table 6	Fields of table <code>contact</code> / <code>contacts.db</code> database	45
Table 7	Fields of table <code>phonenumber</code> / <code>contacts.db</code> database	46
Table 8	Record for notification of a group of WhatsApp messages in table <code>notifications</code>	47
Table 9	Tables of database <code>phone.db</code>	48
Table 10	Record of a conversation in <code>phone.db</code> 's <code>conversation</code> table	49
Table 11	Record of an MMS in <code>phone.db</code> 's <code>mms</code> table	50
Table 12	Fields of table <code>media</code> / <code>photos.db</code> database	51
Table 13	Record for Your Phone Companion in table <code>phone_apps</code> . .	52
Table 14	YPA 's recovered data artifacts.	56
Table 15	YPA 's Your Phone artifacts.	57
Table 16	YPA 's native artifacts.	58
Table 17	WNS ' <code>wpndatabase.db</code> tables	69
Table 18	Notification handler from WNS	70
Table 19	WhatsApp notification artifact from NotifAnalyzer.	74

LIST OF TABLES

LIST OF ACRONYMS

ADB	Android Debug Bridge.
API	Application programming interface.
CLI	Command-line interface.
DBMS	Database Management System.
DLL	Dynamic-link libraries.
FTS	Full-Text Search.
GPL	GNU Public License.
GUI	Graphical User Interface.
NoSQL	Not only SQL .
OS	Operating System.
OSDFCon	Open Source Digital Forensics Conference.
SDK	Software Development Kit.
SEQUEL	Structured English Query Language.
SMIL	Synchronized Multimedia Integration Language.
SQL	Structured Query Language.
SSL	Secure Sockets Layer.
TSK	The Sleuth Kit.
URI	Uniform Resource Identifier.
UWP	Universal Windows Platform.

- WAL Write-Ahead Log.
- WNS Windows Push Notification Services.
-
- YPA Your Phone Analyzer.
- YPC Your Phone Companion.

INTRODUCTION

The SQLite database has been gaining traction within the Microsoft software ecosystem. This is particularly evident in Windows 10, where a plethora of services and applications have adopted SQLite databases as data containers. This chapter introduces the significance of these databases in Windows 10, and explores the motivation and objectives of the dissertation.

1.1 CONTEXT

Microsoft's Windows 10 is one of the most popular desktop OSs, with over 75% of the desktop market of OSs in most of 2020, according to NetMarketShare (2020a) and StatCounter GlobalStats (2020a).

SQLite is the most deployed database engine, even more than all the other database engines combined, according to SQLite (2020e). It is estimated that there are over a trillion SQLite databases in active use, heavily due to its usage in mobile devices (SQLite, 2020e)

Android is also a very popular OS, in this case for mobile devices. According to NetMarketShare (2020b) and StatCounter GlobalStats (2020b), Android is almost as popular for mobile as Windows 10 is for desktop, with over 70% market share worldwide for most of 2020.

Windows 10's UWP mimics Android's and iOS ecosystem. It contains a store, Microsoft Store, which allows the user to install or update applications, just like Google's Play Store or Apple's App Store. These three OS also have applications from their store installed by default on a user's device. In a mobile OS it is very common to use certain mechanisms, such as SQLite databases, to store user data. Windows 10's UWP applications share this similarity, as shall be seen later.

Digital tools are frequently involved in crimes, either directly - hacking, cryptomining, data stealing, etc - or indirectly as tools of communications and data sharing (photos, videos, etc). Thus, when a criminal situation needs to be investigated, often there is the requirement of analyzing devices for digital forensic artifacts. The

complexity of modern OS and applications, coupled to large volume of data makes for a high demand on digital forensic examiners.

1.2 MOTIVATION

An important motivation for this work is to study, from a digital forensic point of view, the usage of SQLite databases within the Windows OS, exploring forensic artifacts.

Two different scenarios are analyzed: Microsoft's Your Phone ecosystem and WNS, as both rely on SQLite databases. The former links an Android smartphone to Windows 10, and can allow for indirect access to certain data of the smartphone in a forensic examination, while the latter is a Windows 10 service that holds user and system notifications, which can result in interesting forensic data. Both of these Windows components have a considerable amount of users. Your Phone is installed by default in Windows 10 since version 1809¹, and its Android app, YPC, has over 100 million downloads in the Play Store. WNS notifications are also enabled by default in Windows 10.

Complementary, another driving motivation is to develop open source software tools that can assist digital forensic examiners to easily deal and process the artifacts from the studied scenarios. This is fulfilled by developing software seamlessly integrates with the Autopsy software, which is well known within the digital forensic community. Indeed, due to the current data deluge in digital forensics, solutions are needed that can assist digital forensic practitioners in their demanding tasks (Olivier, 2020; Quick and Choo, 2014).

1.3 OBJECTIVES

The focus of the dissertation is in digital forensics of SQLite3 databases used in Windows applications and services. As such, the objectives are: *i*) to survey SQLite from a digital forensics point of view, *ii*) identify Windows 10 components that use SQLite, *iii*) provide detailed analysis on these components, *iv*) and to develop tools or solutions to extract digital forensic artifacts from their databases.

A challenge that surfaced when attempting to accomplish these initial objectives was the maintenance of the developed solutions. As such, another objective is to

¹ <https://docs.microsoft.com/en-us/windows/application-management/apps-in-windows-10>

ease the development process and maintenance of these solutions, as the OS is in constant evolution. It is also important to retain a repository of information regarding the updates of certain applications.

1.4 CONTRIBUTIONS

The main contributions of this work are:

- In-depth study of the Your Phone ecosystem, that is, the Windows Your Phone application and Android application YPC.
- Development of the Autopsy module [Your Phone Analyzer \(YPA\)](#), which parses the Your Phone SQLite databases allowing to *i*) import the main relevant forensic artifacts within Autopsy, and *ii*) to produce HTML-based reports.
- In-depth study of [WNS](#), with emphasis on the main SQLite database.
- Development of the Python 3 script NotifAnalyzer to parse and extract the main relevant digital forensic artifacts, exporting them in a JSON format
- Development of the Autopsy module “Windows Notifications Analyzer”, which relies on the NotifAnalyzer script to import the [WNS](#)-related artifacts within Autopsy.
- Development of the [UWP](#) scanner solution to detect changes and updates of certain Windows applications.
- Paper published in the international scientific journal *Digital Forensics*²: Patricio Domingues et al. (2019). “Digital forensic artifacts of the Your Phone application in Windows 10”. In: *Digital Investigation* 30, pp. 32–42. URL: <https://www.sciencedirect.com/science/article/pii/S1742287619301239>.
- Submission of the paper “Keeping track of [UWP](#) application changes for digital forensic purposes” (Luis Miguel Andrade, Patricio Domingues, Miguel Frade) to ConfTele’2021³.
- All the developed code and tools are available at the following GitHub addresses:

i) <https://github.com/labcif/YPA/> (Andrade et al., 2020);

² <https://www.journals.elsevier.com/digital-investigation>

³ <http://conftele2021.ipleiria.pt/>

ii) <https://github.com/L-Andrade/UWPAppsScanner> (Andrade, 2020).

1.5 ORGANIZATION OF THE DISSERTATION

The remainder of the dissertation is organized as follows. Chapter 2 gives the necessary background of the tools used or studied. Chapter 3 details how the UWP works internally for applications, and the development of the UWP scanner solution. Chapter 4 contains the study of Your Phone Windows 10 and Android applications, and the development of YPA. Chapter 5 consists of the examination of WNS, and the development of the NotifAnalyzer script and its Autopsy module counterpart. Lastly, Chapter 6 contains the conclusions and a summary of the dissertation. This last chapter also lists possible future work.

BACKGROUND

This chapter contains the background required to understand the study, analysis, and development that is present in the next chapters of the dissertation. It defines digital forensics; databases, their origins and necessity, categorizing and differentiating implementations and architectures. After these principles, the dissertation will dive into SQLite and the mobile database ecosystem, which is then followed by the potential value of these databases, focusing on Microsoft's Windows 10, which contains several of these SQLite artifacts.

2.1 DIGITAL FORENSICS

Årnes (2017) defines *digital forensics* as applying forensic science to digital information, and a *digital investigation* as investigations in the digital domain. The book then goes more in depth about these terms.

The work presented in this dissertation is meant to aid in these so called digital investigations, first in the form of documenting how certain applications or tools work (such as the internals of the Your Phone app, and tools used by the app itself - like SQLite3), and by providing scripts to analyze according to the documentation and findings. One of the goals is to bring to light anything that might be used as evidence by a practitioner, often termed as digital forensic artifact, or simply forensic artifact.

2.1.1 *Autopsy*

The Sleuth Kit (TSK)'s Autopsy is an easy to use, open-source [Graphical User Interface \(GUI\)](#)-based program to TSK and other digital forensics tools that allows the user to analyze hard drives and smart phones (The Sleuth Kit, 2019). Autopsy can be extended through three types of modules, *i*) file ingest, *ii*) data source ingest, *iii*) and report. The first two can be classified as ingest modules. Figure 1 shows the

BACKGROUND

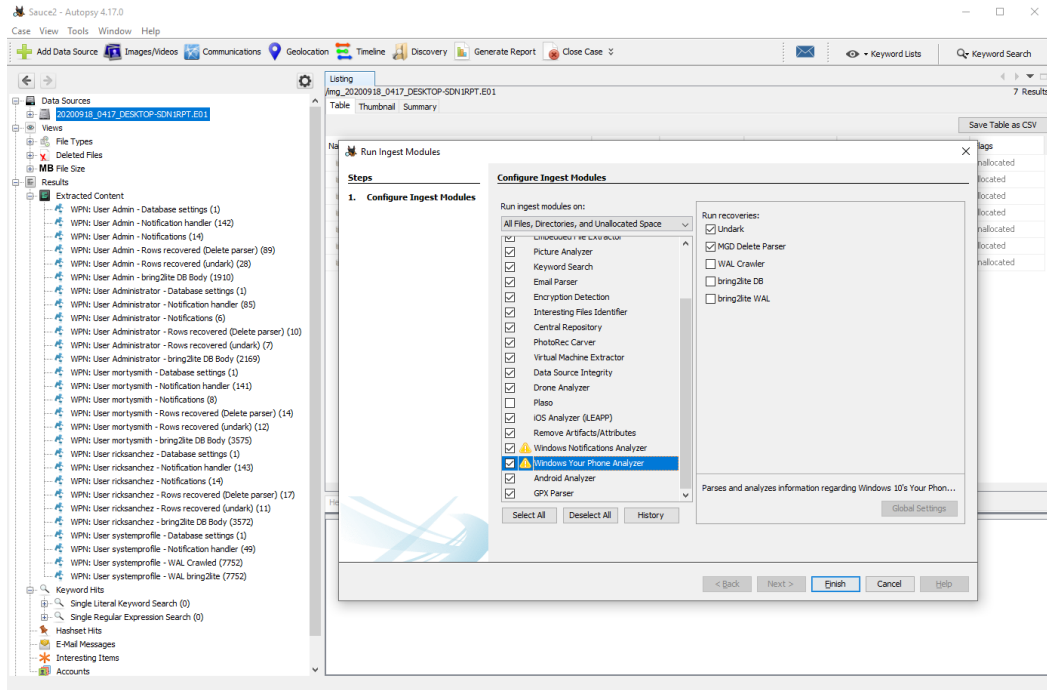


Figure 1: Autopsy GUI

GUI of Autopsy, in this case the interface where a practitioner can choose which ingest modules to be run in a data source.

Autopsy installs certain modules by default, such as Exif Parser¹, PhotoRec Carver², Recent Activity³, and others. The community can also develop their own, and the creators of Autopsy incite the community to do so. These modules are usually written in Java or Jython (Python for Java). The latter is based on Python 2.7⁴. An official repository containing third-party modules can be found at https://github.com/sleuthkit/autopsy_addon_modules, and the workflow of Autopsy can be seen in Figure 2.

Internally, a centerpiece of Autopsy is the *Blackboard*. It allows modules to communicate with each other and the UI (Basis Technology, 2017). Modules post their results as *artifacts* to the blackboard, and they are displayed in the interface, on the left-hand side, under *Results*, which can be seen in Figure 1.

1 http://sleuthkit.org/autopsy/docs/user-docs/4.17.0/_e_x_i_f_parser_page.html

2 https://sleuthkit.org/autopsy/docs/user-docs/4.17.0/photorec_carver_page.html

3 http://sleuthkit.org/autopsy/docs/user-docs/4.17.0/recent_activity_page.html

4 <https://www.jython.org/>

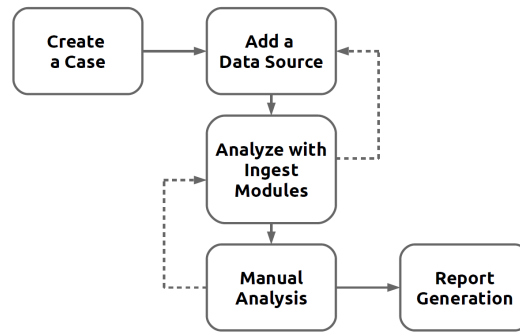


Figure 2: Autopsy workflow. Source: Basis Technology (2016)

2.1.1.1 *Ingest modules*

As mentioned before, these modules have two variants: data source and file. The difference between these variants is simple, the first runs on a data source and the second runs for each file found (Basis Technology, 2019). Looking at it in a programmer’s perspective, a file ingest runs a `for` loop (module runs once per file) while the data source receives a list of files (module runs once for the whole data source). Another difference between a file ingest and data source module is that the former includes files that are found via carving or inside ZIP files, while the latter does not (Autopsy, 2015).

Ingest modules are typically run immediately after adding a data source, and its purpose is to find *artifacts* in files and add them to the *blackboard*. These modules can take some time to finish, evidently depending on the size of the data source that is being analyzed.

2.1.1.2 *Report modules*

Generally, report modules are ran after ingest modules, and tend to take less time to finish. These modules usually take the artifacts created by the ingest modules and enhance this information, either by creating relations, or by computing statistics, or other visual aids of the information, such as graphs. Reports end with one or more files, depending on the developer. The most common report formats are HTML, Excel, and CSV. Some modules even allow the user to choose the desired format, from an array of options. These formats are widely spread and can be opened in any OS.

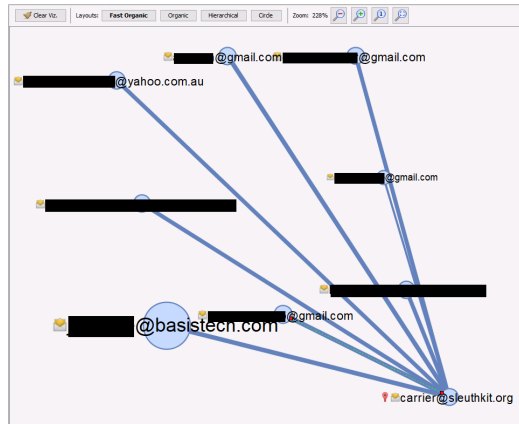


Figure 3: Autopsy Communications Visualization. Source: Basis Technology (2018)

2.1.1.3 Communications Visualization

This feature was introduced in Autopsy 4.6.0, and improved upon in the next version, 4.7.0 (Basis Technology, 2018). Figure 3 shows an example of how Communications Visualization can be seen. It will look different depending on the modules that are ran by the user, and the artifacts that are extracted from the data source. It is a way to sort through large amounts of messages, be it emails, SMS, MMS, etc, and it makes visualizing the relation between the sender and receiver easier (Carrier, 2018).

To add artifacts to this interface, a developer needs to use Autopsy’s native artifacts, and he/she must create an *account* in Autopsy’s *Communications Manager*. Each account type, which can be a phone, email, credit card, among other types⁵, has a different unique identifier. In the case of a phone account, the unique identifier is the phone number, and it can be a limitation since it must be a valid phone number. There are times that text messages are received from contacts without an explicit phone number, where the display name is *Google*, *GitHub*, or any other service that sends messages under its name.

2.2 DATABASES: DEFINITION AND ORIGINS

Starting simply: what is a database? According to Abiteboul et al. (1995), a database is “a large amount of data stored in a computer” and follows that thought stating that the software that manages this data is called a [Database Management](#)

⁵ http://sleuthkit.org/sleuthkit/docs/jni-docs/4.10.1/classorg_1_1sleuthkit_1_1datamodel_1_1_account_1_1_type.html

System (DBMS). In the scope of this dissertation, this definition is acceptable - considering modern mobile phones are practically pocket computers.

Relational databases date back to 1970, then called **Structured English Query Language (SEQUEL)** (Kline et al., 2004). Nowadays, these databases are referred to as **Structured Query Language (SQL)** or relational databases.

Oracle (2019) defines a database as “an organized collection of structured information, or data, typically stored electronically in a computer system”, which is simple and to the point. This definition has held, even since **Not only SQL (NoSQL)** has entered the field. Vaish (2013) claims that **NoSQL** is used to refer to any class of databases that do not follow the traditional relational **DBMS**. These databases have schemaless data representation, falling into the category of data, not structured information, in Oracle (2019)’s definition (Vaish, 2013).

Google’s Bigtable is considered to be one of the first types of **NoSQL** databases, which was released in February 2005 (Chang et al., 2008; Li and Manoharan, 2013; Wikipedia contributors, 2019).

2.3 SQLITE

SQLite is an open-source relational database. That does not, however, mean that it is open-contribution. SQLite is the most deployed database engine, and it is estimated to be over one trillion SQLite databases in active use (SQLite, 2020e), mostly due to its usage in Android OS, iOS, and many mobile applications. Due to the enormous quantity and potential value of these artifacts, it is important to study how the software works, its quirks, and algorithms - such as record deletion and alteration - to understand how it can then be used as evidence in digital forensics cases.

2.3.1 *In detail*

According to *About SQLite*: “SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine”, (SQLite, 2020a). Unlike other **SQL** databases, SQLite is an embedded SQL database engine, which means it does not run in a separate server, and reads/writes to ordinary disk files. A complete database can be held in a single file. With all features enabled, the SQLite library can be less than 600KiB, depending on platform and

compiler settings (SQLite, 2020a). All in all, SQLite is a very lightweight database engine, perfect for local storage in any device, easily exported to other devices due to its small and single files.

An atomic commit means that either all changes occur within a single transaction, or none of them occur. SQLite, like other transactional databases, implements this feature, and as a default it uses a rollback journal (SQLite, 2020b; SQLite, 2020i). Another option to the rollback journal is [Write-Ahead Log \(WAL\)](#), which will be explained further. The rollback journal is a disk file which contains the database name and a `-journal` suffix. Whenever a database is to be changed, it is first saved in this `journal` file (SQLite, 2019b). Although it is enabled by default, the rollback journal can be disabled. The behaviour of the rollback journal, like many other features of SQLite, can be changed using a `PRAGMA` statement⁶.

2.3.1.1 *Temporary files in SQLite3*

According to *Temporary files used by SQLite*, “One of the distinctive features of SQLite is that a database consists of a single disk file. This simplifies the use of SQLite since moving or backing up a database is as simple as copying a single file. It also makes SQLite appropriate for use as an application file format. But while a complete database is held in a single disk file, SQLite does make use of many temporary files during the course of processing a database” (SQLite, 2020g).

There are, at the time of writing, nine distinct types of temporary SQLite files. These types are listed in Table 1.

Rollback journals are used to implement atomic commit and rollback capabilities. A [WAL](#) file is an alternative to a rollback journal when the database is operating in [WAL](#) mode. *Shared-memory files* are related to a journal mode, [WAL](#), and exist only to allow multiple processes to access the same database in [WAL](#) mode.

Super-journal files are used when a single transaction affects multiple databases under a single database connection using `ATTACH`. At least two of the affected databases must not be in [WAL](#) mode, along with other requirements, in order to create this file. A *statement journal file* is used to rollback partial results of a single statement within a larger transaction.

TEMP databases are created when using the `CREATE TEMP TABLE` syntax, and are only visible to the database that issued that statement. This separate database

⁶ https://www.sqlite.org/pragma.html#pragma_journal_mode

Table 1: Types of temporary SQLite files.

TYPE	DESCRIPTION
Rollback journals	To implement atomic commit/rollback capabilities
Super-journals	Single transaction affects multiple databases
Write-Ahead Log files	Alternative to rollback journal
Shared-memory files	To allow access by multiple processes to a database in WAL mode
Statement journals	Used to rollback partial results of a single statement
TEMP databases	When using <code>CREATE TEMP DATABASE</code> commands
Materializations of views and subqueries	Result of a subquery needs to be stored
Transient indices	Used in certain SQL language features
Transient databases used by <code>VACUUM</code>	Created by a <code>VACUUM</code> command

can also have an associated rollback journal. This file is deleted when the database connection is closed.

Materializations of views and subqueries occur when the result of a subquery needs to be stored in a temporary file. When the query is concluded, the file is deleted. *Transient indices* are used in certain SQL language features. The temporary file is automatically deleted at the end of the statement that uses it. *Transient databases used by VACUUM* are created by the `VACUUM` command. It creates a temporary database, and then builds the entire database into that temporary file, which then copies the content of the temporary file into the original database and deletes the temporary file.

2.3.1.2 PRAGMA

PRAGMA statements are exclusive to SQLite, and are used to modify the operation of SQLite or query internal SQLite data. It can be used like other commands, such as `SELECT` and `INSERT`, but is different in (SQLite, 2020f):

- No guarantees of backwards compatibility, can be removed or added as necessary in new versions.
- No error messages: unknown pragmas are ignored.
- Some pragmas take effect during compilation.

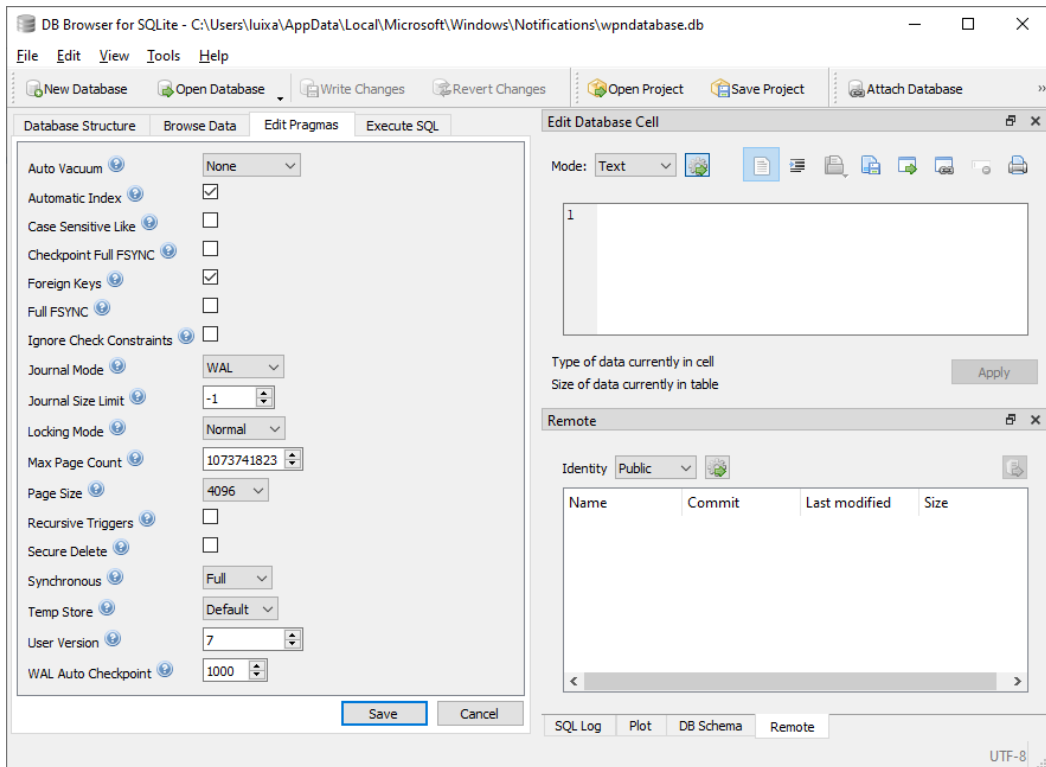


Figure 5: PRAGMAs of `wpndatabase.db`, shown in DB Browser for SQLite.

It seems the second item exists to avoid errors due to the first. This should allow SQLite’s developers to deprecate or remove statements without breaking any existing software, however, there are still some pragmas which are deprecated and not removed to support backwards compatibility, such as `temp_store_directory`⁷. The syntax for PRAGMA statements can be seen in Figure 4. A pragma can have the value of either one (true) or zero (false). Also, a pragma may have an optional `schema-name`, which is the name of an attached database, or `main` or `temp` for the main and temp databases respectively. In some pragmas, this `schema-name` is meaningless. Figure 5 shows a screenshot of DB Browser for SQLite⁸, a tool which can be used to check the PRAGMA values of a SQLite database, among many other features. In this case, it is showing the PRAGMA values of a `wpndatabase.db` database, which is the SQLite database that supports WNS (Chapter 5).

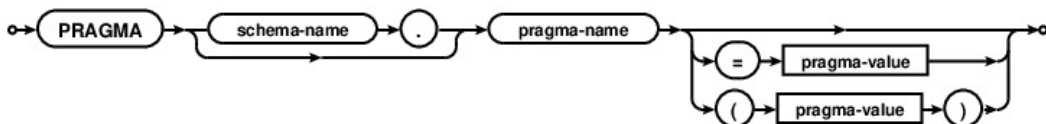


Figure 4: PRAGMA command syntax. Source: SQLite (2020f).

⁷ https://www.sqlite.org/pragma.html#pragma_temp_store_directory

⁸ <https://sqlitebrowser.org/>

At the time of writing, there are seventy pragma statements, some of which are deprecated. Some pragmas will be explored the next. Nevertheless, a list of the existing pragmas can be found at <https://www.sqlite.org/pragma.html>, in the “List of PRAGMAs” section.

The following pragmas are important in the scope of this dissertation:

i) `auto_vacuum`⁹, *ii*) `user_version`¹⁰, *iii*) `journal_mode`¹¹. Next, we detail each of these pragmas. Pragma `auto_vacuum` has three possible modes, `NONE`, `FULL`, and `INCREMENTAL`. The last two modes correspond to an enabled state, meaning the database wipes the records that are deleted, overwriting them with zeros and thus eliminating any chance of recovery. The key difference between `FULL` and `INCREMENTAL` is that the latter does not automatically perform the vacuum. Fortunately for digital practitioners, this option is disabled by default, using the `NONE` mode. The `user_version` is a rather simple pragma and is entirely up to the application to manage, and has no impact in an SQLite database itself. The `journal_mode` pragma gets or sets the journaling mode, and has five possible modes, `DELETE`, `TRUNCATE`, `PERSIST`, `MEMORY`, `WAL`, `OFF`. The `DELETE` mode is the default journaling mode. It controls how the journal files are stored and processed, which are used to roll back transactions or handle unrecoverable errors. The journal files are required for both auto-commit and explicit transactions (Kreibich, 2010).

2.3.1.3 *Write-Ahead Log (WAL)*

As specified in a previous section, `WAL` is an alternative to rollback journals. Like the rollback journal, `WAL` uses ordinary disk files. These files have the suffixes `-wal` and `-shm` (shared-memory) (SQLite, 2020i). A read-only `WAL`-mode database can be opened if these files exist, or those files can be created if the database is immutable (read-only and cannot be modified, even by a process with elevated privileges) (SQLite, 2020h).

The `-wal` file is cross-platform, exists for as long as the database is open, and is usually deleted when the last connection to the database closes or a checkpoint occurs (SQLite, 2020i). In case of apps that are started when the `OS` starts and killed only when the `OS` is killed, this file might almost always exist. If the last process to have the database open does not exit cleanly, or if the `SQLITE_FCNTL_PERSIST_WAL` is used, it might persist on the disk. If this file is separated from its database file, the database can end up with lost transactions or even corrupted. It is enabled with

⁹ https://www.sqlite.org/pragma.html#pragma_auto_vacuum

¹⁰ https://www.sqlite.org/pragma.html#pragma_user_version

¹¹ https://www.sqlite.org/pragma.html#pragma_journal_mode

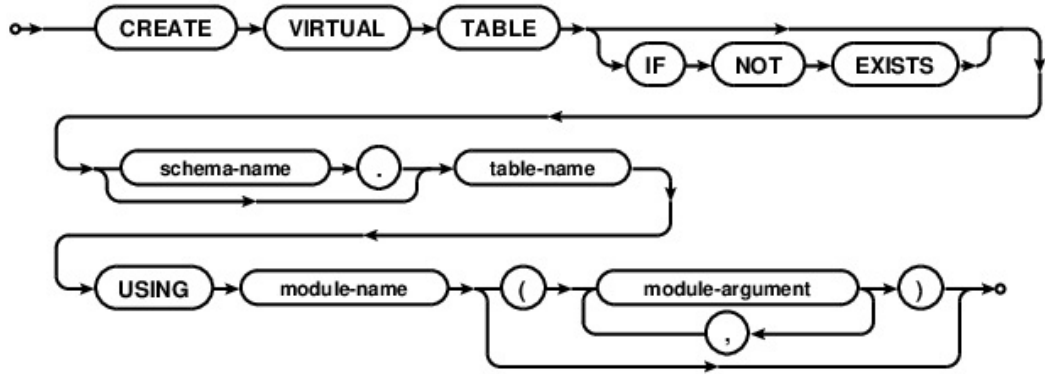


Figure 6: CREATE VIRTUAL TABLE command syntax. Source: SQLite (2020c).

the `journal_mode` PRAGMA statement, using the WAL value, and two other statements can be defined for it, *i*) `wal_checkpoint`, and *ii*) `wal_autocheckpoint`. The first causes a *checkpoint* operation, which consists of moving the WAL transactions to a database, on a certain database or on all attached databases, if `database` is omitted (SQLite, 2020f; SQLite, 2020i). The second defines when a checkpoint should be ran, after N pages. The default threshold is 1000 pages (SQLite, 2020i).

2.3.1.4 Full-Text Search (FTS)

Full-Text Search (FTS) is a common and effective way to facilitate searching for something. Users can input a term, or series of terms, which can be connected by a binary operator or group together in a phrase, and the full-text query system finds the best matches according to those terms.

For SQLite3, there are four extension modules for FTS, from FTS1 to FTS5. FTS1 and FTS2 are obsolete. The tables created by these modules are virtual tables, and are in many ways equal to normal tables, data can be added, modified, and removed using the same commands (`SELECT`, `SELECT`, `UPDATE`, `DELETE`) (SQLite, 2020d).

To use this SQLite feature, one has to create virtual tables using the `CREATE VIRTUAL TABLE` command syntax (SQLite, 2020c). The full syntax can be seen in Figure 6.

The sole purpose of this feature is to speed up and/or facilitate a user's search, meaning that there is not much value in these tables from a forensics point of view, as the data on these tables is already present elsewhere (most likely in their original tables).

2.3.2 *In mobile devices and apps*

SQLite is actually present in most mobile device OS (Android and iOS) by default, and mobile applications, and each device can have hundreds of these databases (SQLite, 2020e). This subsection aims to briefly summarize the options and recommendations of various frameworks to implement local storage in their apps.

Due to its lightweight and administration-less capabilities, SQLite is a good choice for local caching or data storing in mobile devices (SQLite, 2019a). Android has full native support for SQLite and any database created by an app is solely accessible by the own app (Android Developers, 2019a). In the present-day there are two main libraries in the block to ease the implementation of SQLite in Android, Cash App's SQLDelight, and Google's Room, which is the recommended option by Google (Android Developers, 2019b; Cash App, 2019). For iOS native, Apple recommends CoreData, which is a framework for managing an object graph, and can use SQLite at its core (Apple Developer Documentation, 2019; Jacobs, 2016). Nevertheless, there are still popular libraries to help implement SQLite, such as SQLite.swift¹² and FMDB¹³. Flutter is one of the most recent cross-platform framework for mobile development, created by Google¹⁴. In their documentation, developers searching for local storage are directed to a plugin, sqflite¹⁵ which is an SQLite plugin for Flutter (Flutter, 2019). React Native has a different approach to local storage - as its web counterpart, the most popular library is Redux, which does state management (Facebook, 2019). There is another library that augments Redux and adds the persistence functionality - redux-persist¹⁶ - which is a prominent solution. Redux-persist allows developers to choose their own storage engine, which can in turn be SQLite, although it does not seem that popular¹⁷. On the side of Xamarin, Microsoft also proposes SQLite out of the box (Microsoft Docs, 2018c). Finally, Ionic offers two options for local storage: *i*) Ionic Offline Storage, which is a NoSQL solution, and *ii*) Ionic Storage, which prioritizes SQLite (Ionic Documentation, 2019). This covers the most popular frameworks and their choices for local caching or data storing, which supports the idea that SQLite has a lot of importance in the mobile ecosystem.

¹² <https://github.com/stephencelis/SQLite.swift>

¹³ <https://github.com/ccgus/fmdb>

¹⁴ <https://flutter.dev/>

¹⁵ <https://pub.dev/packages/sqflite>

¹⁶ <https://github.com/rt2zz/redux-persist>

¹⁷ <https://github.com/prsn/redux-persist-sqlite-storage>



Figure 7: *UWP* overview. Source: Microsoft Docs (2018b).

2.3.3 *Microsoft, Windows 10, and UWP*

Microsoft and SQLite are tightly coupled. Microsoft Docs (2018a) has the following statement in their documentation: “The Windows version of SQLite is maintained by Microsoft in coordination with SQLite.org”. Moreover, Microsoft’s Entity Framework also has SQLite support (Microsoft Docs, 2016). Windows’ build 18362 introduced tools and an *Software Development Kit (SDK)* to help developers create *UWP*¹⁸ apps (Microsoft Docs, 2019a). *UWP* apps are secure apps that use a common API on all devices that run Windows 10, and are able to adapt to the user’s device. These apps are available in the Microsoft Store, and can interact with Windows Timeline (Horsman et al., 2019). Developers can choose their programming language from C#, C++, Visual Basic, to Javascript (Microsoft Docs, 2018b). An overview of *UWP* can be seen in Figure 7, and the Microsoft Store, displaying Your Phone, in Figure 8.

Microsoft understands SQLite’s advantages of an open-source, lightweight, serverless, client-side database, and *UWP* apps can use SQLite to store data (Microsoft Docs, 2018a). It seems Microsoft wants to give developers a way to create native Windows 10 apps, with interoperability between devices. Microsoft is already shipping *UWP* apps in Windows 10 from the get-go, such as Your Phone, Photos, Skype, among others. The first two apps are available from a clean Windows 10 installation since versions 1809 and 1703 respectively, and both use SQLite databases locally (Domingues et al., 2019; Microsoft Docs, 2019b). The interaction between *UWP* apps and Windows Timeline is important. Timeline has been available since Windows 10’s April 2018 Update and it essentially shows snapshots of the user’s

¹⁸ Complete *UWP* details: <https://docs.microsoft.com/en-us/windows/uwp/>.

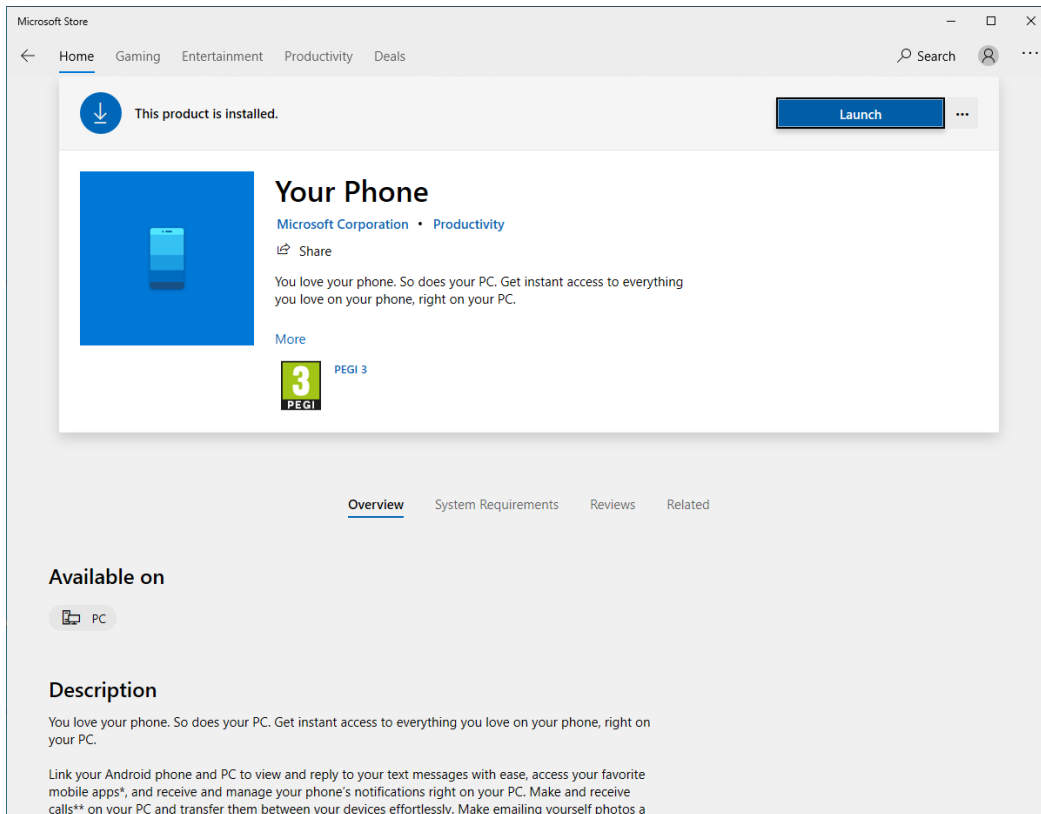


Figure 8: Your Phone in the Microsoft Store.

activity (Windows support, 2018). It is imaginable that Timeline will be the epicenter of the data of UWP apps, and it is important to note that Timeline also takes advantage of SQLite databases, which is valuable from the digital forensic point of view (Horsman et al., 2019).

2.3.3.1 In digital forensics

Data inside an SQLite database is simple to fetch. One needs only the database, and a tool, such as DB Browser for SQLite, to query it. In this manner, the user can query all the *current* data and PRAGMA values. These can be valuable of course, however one must look at the possibility of recovering deleted records, or seeing what is left behind by a temporary file, such as a WAL file (Casey et al., 2019).

By default, SQLite does not delete records, it simply marks them as deleted, which means it is likely to find records, in full or partially, in the unallocated area within a SQLite database (Meng and Baier, 2019). WAL files store the latest changes to a database, and it is a richful source to recover deleted SQLite information (Meng and Baier, 2019). “Ten years of critical review on database forensics

research” highlights the importance of database forensics. Articles Chopade and Pachghare (2019), and Meng and Baier (2019) delve into the importance, reliability, and results of certain SQLite recovery tools.

The software created with this dissertation relies heavily on open-source tools, namely Daniels (2020)’s `undark`, Meng and Baier (2019)’s `bring2lite`, Miller and Bryce (2019)’s `WAL-Crawler`, and DeGrazia (2015)’s `SQLite-Deleted-Records-Parser`.

The higher number of SQLite data recovery tools used, the higher the chance a practitioner has of recovering meaningful data. There are no tools which can handle all corners of SQLite, according to Chopade and Pachghare (2019). It is important when creating software that automates the process of analyzing SQLite databases to give the choice of using these tools, as it might bring additional value to the practitioner, and each tool might complement another. This dissertation applies this advice, resorting to four different tools for recovering SQLite records.

2.4 WINDOWS 10’S YOUR PHONE AND `YPA`

As specified in Subsection 2.3.3, Your Phone is a Windows 10 `UWP` application. This application uses SQLite databases heavily, which in turn use `WAL`. It has an Android counterpart, `YPC`¹⁹, which at the time of writing has over 100 million downloads. Your Phone requires a smartphone with Android 7.0 or newer to use. The author of this dissertation has worked beforehand with three others to study the desktop application, which at the time used a single SQLite database, and create a two-part Autopsy module, `YPA`²⁰ (Domingues et al., 2019). `YPA` is open-source, and is licensed under `GNU Public License (GPL)` version 3²¹. Your Phone leaves databases on the user’s disk. As can be expected, these databases contain valuable information regarding the user’s phone and phone usage. `YPA` extracts the data left behind by these databases and creates an easy-to-read HTML report.

2.4.1 *Previous work*

Since the work documented in “Digital forensic artifacts of the Your Phone application in Windows 10” (Domingues et al., 2019), Your Phone has been updated

¹⁹ <https://play.google.com/store/apps/details?id=com.microsoft.appmanager&hl=en>

²⁰ <https://github.com/labcif/YPA>

²¹ <https://github.com/labcif/YPA/blob/master/LICENSE>

and saw meaningful additions to not only the database schema, but the number of SQLite databases. While YPA was still functional on both old and newer versions of the software, it was fundamental to update the module to support the new features of Your Phone and fix some breaking changes from these recent updates. **At that time**, YPA had the following key features:

- Extract SMS and MMS from Your Phone's databases.
- Create an HTML report with the extracted data.
- Use `undark` and `SQLite-Deleted-Records-Parser` to recover deleted data.

These features will be expanded on in Chapter 3, along with their improvements and additions in the most recent YPA updates. It is also possible to see the state of the repository at that time²².

Further will be presented the ongoing development solely by the author, and the changes of Your Phone since the publication of the paper (Domingues et al., 2019).

²² <https://github.com/labcif/YPA/tree/c26c2388a140f2ad332c3715a95eb7b0387759dc>

UWP APPLICATIONS

This chapter contains details about the findings and development for analysing **UWP** applications. One of the objectives of this work was to analyse **UWP** apps. Most of these apps consist of the same thing - a desktop application with local SQLite databases, following the framework imposed by **UWP**, developed by Microsoft and installed by default in Windows 10.

3.1 UNIVERSAL WINDOWS PLATFORM

Contrary to regular Windows applications, **UWP** applications have to adhere to a rigid directory layout. Specifically, the executable file(s) and specific **Dynamic-link libraries (DLL)**s are kept in an appropriate directory hierarchy based at `C:\ProgramFiles\WindowsApps\`, considering that the **OS** is installed in the `C:\` partition. For example, Your Phone's executable and **DLL**s are located at `C:\Program Files\WindowsApps\Microsoft.YourPhone_1.20101.99.0_x64__8wekyb3d8bbwe` where `1.20101.99.0` is the application version, `x64` indicates that the application is for a 64-bit Windows 10 version, and `8wekyb3d8bbwe` is the producer ID of the application - in this case, Microsoft.

The data of the **UWP** application is kept in a different directory, localized by the user's application data directory. For example, data from Your Phone are kept in the subdirectory `Microsoft.YourPhone_8wekyb3d8bbwe` of the parent directory `C:\Users\%USERNAME%\AppData\Local\Packages\`, where `%USERNAME%` is the user's Windows 10 username. The directory hierarchy within an application's data directory is normalized. It is comprised of eight subdirectories, as shown in Figure 9.

Applications often need to persist data. **UWP** applications usually resort to SQLite. The connection between **UWP** and SQLite was previously established in Section 2.3.3. From the digital forensic point of view, SQLite databases are well known, and well supported by digital forensic software.

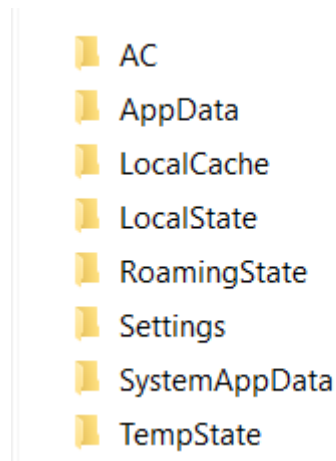


Figure 9: Data directory hierarchy of an **UWP** application.

3.2 KEEPING UP WITH APPS' UPDATES

From a forensic point of view, it is fundamental to know the potential artifacts of an application, especially those from databases. Some **UWP** applications are evolving at a fast pace, such as Your Phone going from one database in its first version to eight. It is important to know when apps are updated and the schema database changes that can occur. In this way, to detect any alterations in **UWP** applications, the UWP Apps Scanner was developed¹.

It is hard to know if apps are locally up to date, and it is also complicated to know if the app is being used in a certain Windows machine. For example, even though Your Phone is installed by default in a computer, it is possible that the databases are not created, since the app could have not been started yet or the device setup has not been yet completed. To avoid doing manual checks for every single **UWP** app, an architecture was idealized, which can be seen in Figure 10. The idea is a script that checks the local machine, and tells the user if the machine has new additions or is running an older version, compared to other users. Figure 11 shows the scanner being executed.

The steps of the scanner are as follows:

1. Get configuration and data from the server database.
2. The server returns the data. Expected data are the applications data, and the schema version of the scanner.
3. If the scanner is up-to-date, it will compute according to the returned data.

¹ <https://github.com/L-Andrade/UWPAppsScanner>

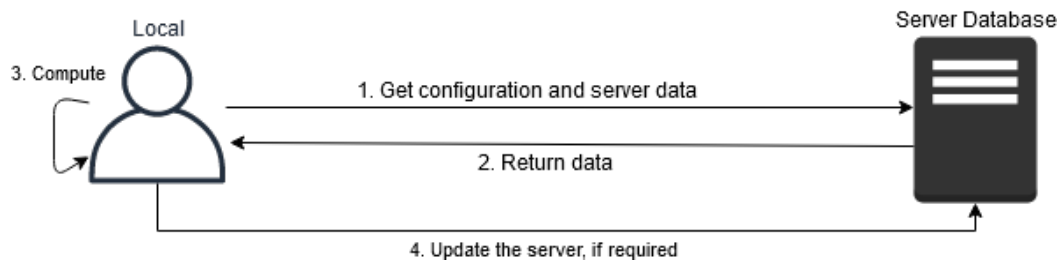


Figure 10: Scanner architecture.

```

Command Prompt
C:\Users\luixa\Desktop\esc\UWPAppsScanner>python uwp_apps_scanner.py
Your script is up-to-date, with version 6.
Found 2 DBs for FacebookMessenger, with version [75, 4, 124, 62046]
Found 0 DBs for FacebookMessengerBeta, with version [79, 2, 115, 55903]
Found 1 DBs for Photos, with version [2020, 20090, 1002, 0]
Found 4 DBs for Skype, with version [8, 66, 0, 74]
Found 8 DBs for YourPhone, with version [1, 20101, 99, 0]
Elapsed time: 2.2s

```

Figure 11: Example of a scanner execution.

4. If an application is not up-to-date in the server, the scanner will update the server with its data.

Most of the logic is in the third step. It is the step where the scanner checks its own version, and each application locally. It first checks the version of the application, and if it is the same or newer, it will check each database the application might have for the tables and the value of `user_version` PRAGMA.

The fourth step is done for each app that is updated locally, but not on the server.

Ideally, the user would run this script every once in a while, to know if the local instance is in synch with the server, and up to date. If the script tells the user there are updates to an app, it is time to investigate and update any necessary tool that might have been impacted. This scanner was entitled `UWPAppsScanner`, and henceforth it will be referred to as “scanner”. It is an open-source Python script and it is available under a [GPL 3.0](#) license.

It is intended that the scanner has most capabilities, but it is still necessary to have the information centralized, being able to supply several users and maintain a version history, including the date/time and version of an update (the date being when the scanner receives the update information, not the date of the app update itself). If the script functioned only locally, it would not be of much use to others, and all the data generated by the first users would not be accessible to new users, who might want to create backwards compatible tools that interact with a certain application.

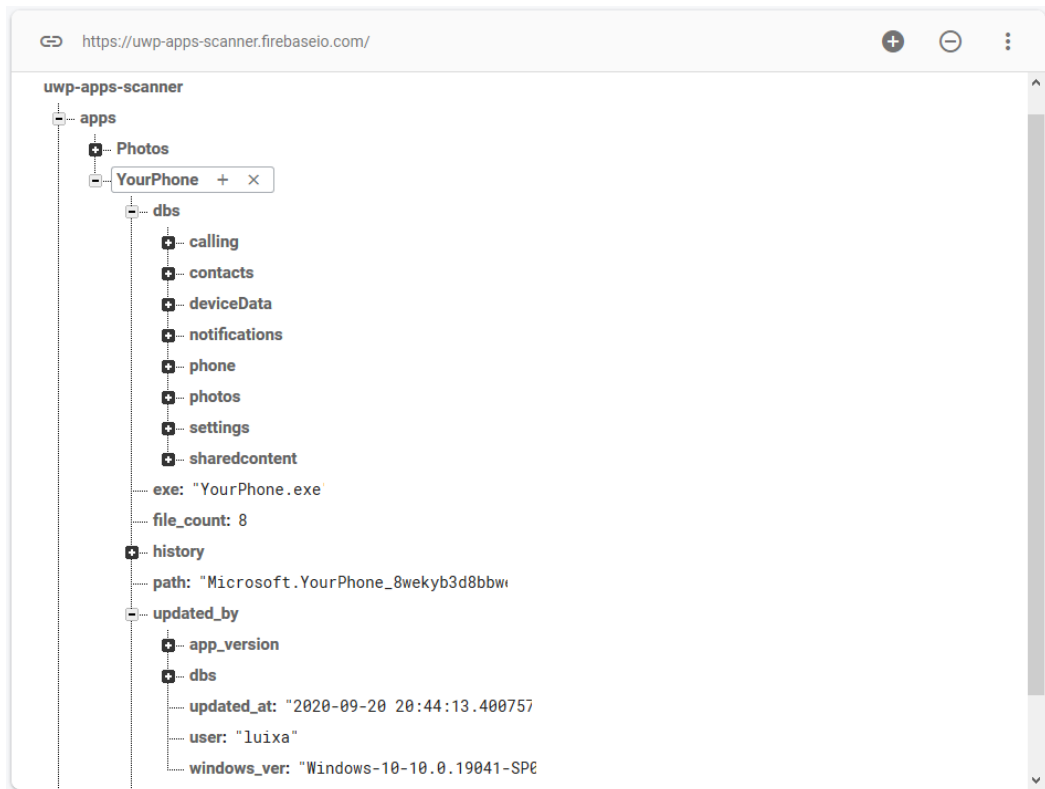


Figure 12: Firebase Console’s Realtime Database dashboard of the scanner’s database.

Since the scanner server does not require much computation and serves mostly as a source of truth and data, it is not required to implement a solution from scratch. For this reason, Firebase’s Realtime Database² was chosen. Realtime Database requires almost no effort for maintainability, and its free plan is enough to keep the solution free to run and use. It also has numerous libraries to communicate with it, easing the development of the scanner. Essentially, it is an easy-to-use JSON file in the cloud.

Figure 12 shows how the database looks in the Firebase Console dashboard. In the given example, only two UWP applications are considered: Microsoft Your Phone (seen expanded), and Microsoft Photos. The console dashboard is easier to check and navigate than the JSON that is outputted by the script. Unfortunately, the access to this dashboard can not be freely shared with everyone.

The scanner uses Python 3.7, and depends on the external libraries listed in Table 2.

² <https://firebase.google.com/docs/database>

Table 2: Scanner dependencies.

Dependency	Description
<code>pyrebase4</code> ^a	Handle Firebase Realtime Database.
<code>filetype</code> ^b	Check a file's mime type.
<code>dictdiffer</code> ^c	Difference between two Python dictionaries.
<code>pywin32</code> ^d	Windows 10 Python extensions.

^a <https://github.com/nhorvath/Pyrebase4>

^b <https://github.com/h2non/filetype.py>

^c <https://github.com/inveniosoftware/dictdiffer>

^d <https://github.com/mhammond/pywin32>

3.2.1 *Server schema and rules*

The database schema is one of the critical points of the scanner. It defines what is saved and what other users can see. The information gathered needs to be not only of the apps, but of the user reporting changes and the time of those changes. This way, it is possible to have an history of updates, and to see the difference between updates. An example of the database can be seen in JSON format in Listing 4. In this example, the object `dfs` is shortened after its first appearance, to avoid visual clutter.

As the tool is available to everyone, the [Application programming interface \(API\)](#) key is also open to everyone. There is a need for security to avoid having breaking changes to the schema which could even result in loss of data. This issue is solved by adding rules to the database³. As can be seen by the rules in Listing 1, the write access to the database is always declined. Only users with an administrator *Service account*⁴ key file (JSON format) associated to the Realtime Database can update the data. A user of the scanner can use the `-key` option to define the path to the key file. After using this argument, the path is saved in a local JSON file so that the user does not have to keep using this option.

3.2.2 *Delving into the script*

The following subsections explain the internals of the script, to understand how the objectives were accomplished.

³ <https://firebase.google.com/docs/database/security>

⁴ <https://firebase.google.com/support/guides/service-accounts>

Listing 1: Server database schema rules.

```

1  {
2    "rules": {
3      ".read": true,
4      "schema_version": {
5        ".read": true,
6        ".write": false
7      },
8      "apps": {
9        ".read": true,
10       "$app": {
11         ".write": false
12       }
13     }
14   }
15 }

```

Listing 2: Scanner execution with the `version` argument.

```

1  C:\Users\luixa\Desktop\Dissertation\UWPAppsScanner>python3
   ↪ uwp_apps_scanner.py --version
2  Your script is outdated.
3     Local is at version 3.
4     Server is at version 9.

```

3.2.2.1 *Control local and server*

The script is also updated from time to time to add features and correct bugs. Thus, the script always checks whether the local version is the most up-to-date one. Indeed running an older version might break the schema of the server data. The server has a `schema_version` attribute, with an integer value. Every time there is an update to the scanner which adds or changes other attributes, this integer should be incremented. The script also contains a `schema_version` constant, which should be updated accordingly, by downloading or pulling the scanner updates.

Running the script with the `version` argument will show the user the local version of the scanner and also say if it is up to date with the server. With this, it is safe to use the script with multiple users and/or computers. Listing 2 shows the output of this argument.

3.2.2.2 *Get or export server data*

There are three possible arguments for this feature: *i)* `info`, *ii)* `infohistory`, and *iii)* `export`. The first two options get the available information from the server and print it in the user's command-line. The key difference between the two being that

`infohistory` also prints the update history of an app. On the other hand, `export`, as the name indicates, exports all the information to a JSON file. This can be useful to save backups, or accessing the information offline.

3.2.2.3 *Check app version*

To check an app version, the `win32api` Python module was used. With this module, it is possible to fetch the version number of an executable file, based on the high and low order values of `FileVersionMS` (major) and `FileVersionLS` (minor), which form the version code using the formula `high major, low major, high minor, low minor` such as `5.0.1.0`. The scanner compares the local app version with the server version, one integer at a time, from the most impactful to the least.

Cases where the local version is equal to the server will be discussed next.

3.2.2.4 *Calculate differences*

As seen in Table 2, the scanner depends on `dictdiffer` to calculate differences between two dictionaries. This is a very important part of the algorithm behind the scanner. When the local version of an application is the same as the server's, but the content is not the same, it is of paramount importance to the user to know what is different between local and server, as the user can then update the server with the development local data, or on the contrary, see that the local data might not be up-to-date.

The script receives JSON-formatted data from the server, which is then handled internally as a dictionary. The data fetched from the local databases of an UWP app is also handled as a dictionary, so all the script has to do is calculate the difference between the two dictionaries. The dictionaries vary a lot between applications. For example, at the time of writing, Your Phone is comprised of eight databases, each with a few tables, while Photos has only one database, but with more than a hundred tables. By comparing both sources of data, the scanner computes the differences between local and server, and the user can then choose whether he/she wishes to create a new update entry at the server.

3.2.2.5 *Add to history*

Whenever there is an update in the server, an entry is also added to the history in the server. This history serves to keep track of app updates and users. Each

```

C:\Users\luixa\Desktop\esc\UWPAppsScanner>python uwp_apps_scanner.py --appsdetail
-----|-----|-----|-----|
| App | Version | Last updated | Databases |
-----|-----|-----|-----|
| FacebookMessenger | [75, 4, 124, 62046] | 2020-11-16 18:45:43.681163 | ['Cookies', 'msys_100000745123816'] |
-----|-----|-----|-----|
| Photos | [2020, 20090, 1002, 0] | 2020-11-16 18:22:11.202225 | ['MediaOb'] |
-----|-----|-----|-----|
| Skype | [8, 66, 0, 74] | 2020-11-16 18:52:53.478603 | ['Cookies', 'Databases', 'QuoteManager', 'siimcore-aria-cache'] |
-----|-----|-----|-----|
| YourPhone | [1, 20101, 99, 0] | 2020-11-16 18:22:11.830302 | ['calling', 'contacts', 'deviceData', 'notifications', 'phone', 'photos', 'settings', 'sharedcontent'] |
-----|-----|-----|-----|

```

Figure 13: List of monitored apps in the scanner.

entry contains information about the user who performed the information update (time of the update, the Windows username, and the Windows version), the app version of the app in question, and all the database information at that time of that update (database list, including tables and `user_version`). This allows the scanner to behave as a sort of version control, containing data about each app version. In the future, someone who receives an UWP app's databases to analyse can use this scanner's information to at least know when that version was first reported, what data to expect, and if there are currently new versions of the software.

3.2.2.6 Database and table data

UWP apps in the script are scanned for SQLite databases. Depending on the option chosen by the user, it might be all the apps, or a single one using the `app` option. Whenever a database of this type is found (using `filetype` and the mime type `application/x-sqlite3`), the scanner gets all the tables of that database, and the `user_version` PRAGMA value. Microsoft's UWP apps use this PRAGMA. When there are differences in the database schema, it contains an update to the `user_version` aswell. Other app publishers, such as Facebook with their *Messenger* app, do not use this PRAGMA.

3.2.2.7 Getting a list of monitored apps

By using the `apps` option, the user can check all the apps that are being monitored. The `appsdetail` option delivers the same list, but with additional details, such as when each app was last updated, and its list of databases. Figure 13 shows an example of the detailed output.

3.2.2.8 Application evolution

Since the server has a history of records per application, it is possible to create a detailed output of the evolution if the server has sufficient data of that application. The longer the app has been in the scanner, the more history records it can have.


```

Command Prompt
C:\Users\luixa\Desktop\esc\UWPAppsScanner>python uwp_apps_scanner.py --appevo YourPhone
Getting existing info from Firebase...
-----
App name: YourPhone
-----
Version [1, 19112, 113, 0]:      ['contacts', 'deviceData', 'notifications', 'phone', 'photos', 'settings']
Version [1, 19122, 89, 0]:      ['contacts', 'deviceData', 'notifications', 'phone', 'photos', 'settings']
Number of tables:      [New] 7      [Old] 4      (contacts)
Number of tables:      [New] 3      [Old] 3      (deviceData)
Number of tables:      [New] 2      [Old] 2      (notifications)
Number of tables:      [New] 14     [Old] 13     (phone)
Number of tables:      [New] 4      [Old] 4      (photos)
Number of tables:      [New] 4      [Old] 4      (settings)
Differs inside the databases:
('change', ['contacts', 'tables', 1], ('content_sequence', 'contactdate'))
('change', ['contacts', 'tables', 2], ('onenumber', 'content_sequence'))
('change', ['contacts', 'tables', 3], ('sqlite_sequence', 'emailaddress'))
('add', 'contacts.tables', [(4, 'onenumber'), (5, 'postaladdress'), (6, 'sqlite_sequence')])
('change', 'contacts.user_version', (1, 6))
('change', ['phone', 'tables', 8], ('rcs_filetransfer', 'rcs_conversation'))
('change', ['phone', 'tables', 9], ('sending_message', 'rcs_filetransfer'))
('change', ['phone', 'tables', 10], ('sqlite_sequence', 'sending_message'))
('change', ['phone', 'tables', 11], ('subscription', 'sqlite_sequence'))
('change', ['phone', 'tables', 12], ('sync', 'subscription'))
('add', 'phone.tables', [(13, 'sync')])
('change', 'phone.user_version', (28, 31))
Version [1, 19122, 138, 0]:      ['calling', 'contacts', 'deviceData', 'notifications', 'phone', 'photos', 'settings']

```

Figure 14: Scanner displaying YourPhone's evolution.

However, it can also depend on often an app is updated. If the app does not see many updates, the evolution will not have much information.

There are two options to check the evolution: `-evolution`, and `-appevo`. The former lists the evolution for all applications in the server. The latter receives an application name, and shows the evolution for that app only. If there is no application with that name in the server, it prints a list of all apps in the server. As can be seen in Figure 14, from Your Phone's version 1.19112.113.0 to 1.19122.89.0, a three tables were added to the `contacts` database: `contactdate`, `emailaddress`, and `postaladdress`. The `rcs_conversation` table was also added to the `phone` database. Also seen at the bottom of Figure 14, from version 1.19122.89.0 to 1.19122.138.0, the `calling` database was added.

YOUR PHONE

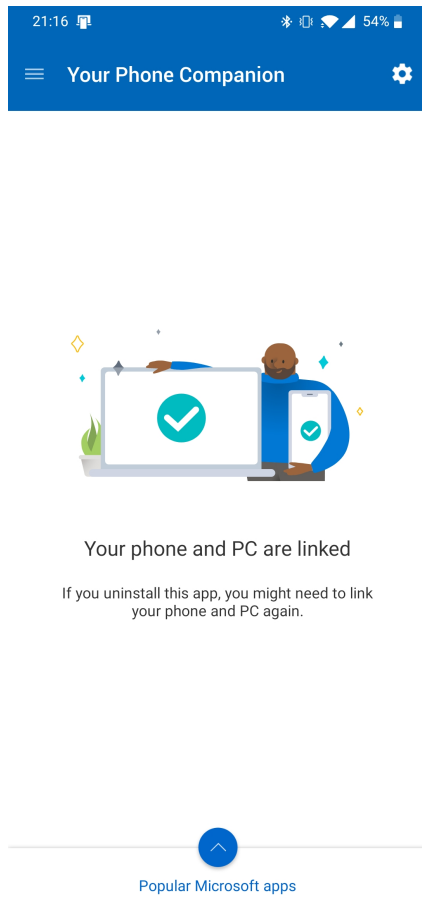
This chapter is about the findings of the Your Phone ecosystem. Your Phone is a **UWP** application. Its purpose is to create a link between the user's Android phone and Windows 10 computer, allowing the user to reply to text messages, receive and manage notifications, make and receive calls, and share resources between these two classes of devices effortlessly. How does Microsoft do it?

4.1 APPLICATIONS

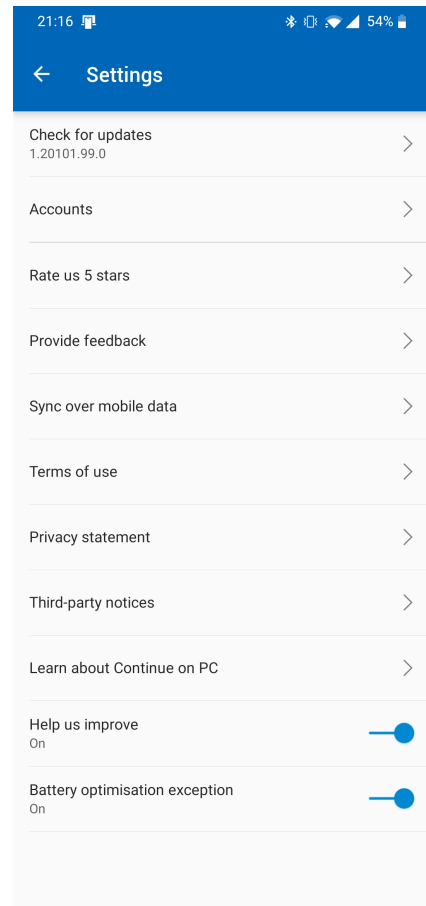
To achieve the computer/smartphone link, it is necessary to have some sort of communication from the phone to the user's computer. For this purpose, Your Phone requires that the user installs the **YPC** app from the Play Store¹. This app then connects with its desktop counterpart, Your Phone. **YPC** already accounts over 100 million installs, a remarkable number, placing **YPC** on the top apps in the Google Play Store². All this popularity increases the probability of a given Windows 10 computer to have a functional Your Phone link to one or more smartphones, therefore having some potential value in digital forensics investigations. It is possible to gather most content of a phone without accessing the phone at all, circumventing common issues of investigating mobile devices, such as the unavailability of a direct access to the internal memory of the device (Distefano et al., 2010). The devices of Your Phone environment are linked to each other through a Microsoft cloud account. The account can be either from the `hotmail.com` or `outlook.%domain%` domains, where for the latter `%domain%` can be a country domain, such as `pt` or the most common `com` domain. **YPC** needs to be configured with a cloud account, and it is accessible through the app's settings menu. This is an account solely at the application level. On the Windows 10 device, the account is configured at the **OS** level, committing the account to the whole machine. Attaching Windows 10 to a cloud account is done through the *Your account info* interface.

¹ <https://play.google.com/store/apps/details?id=com.microsoft.appmanager&hl=en>

² AppBrain's Google Play statistics for **YPC**: <https://www.appbrain.com/app/your-phone-companion-link-to-windows/com.microsoft.appmanager>



(a) Initial YPC screen.



(b) YPC settings screen.

Figure 15: YPC screens.

In the next section, the contents of YPC are explored. It will detail the app's permissions, features, and data stored in the user's device. In the following subsection, the same is done for the desktop counterpart, Your Phone. Following those two subsections, YPA is explored as a tool to extract data (and consequently add any possible valuable information) left behind by Your Phone. The initial screen of YPC can be seen in Figure 15a.

4.1.1 *Your Phone Companion*

YPC requires an Android 7.0 phone (API level 24) to function, but it can be installed on devices with Android 4.4 (API level 19) or higher. While a phone with a lower API level can install the app, it blocks those lower versions of using most features, meaning that the app can be developed as if its minimum level was higher.

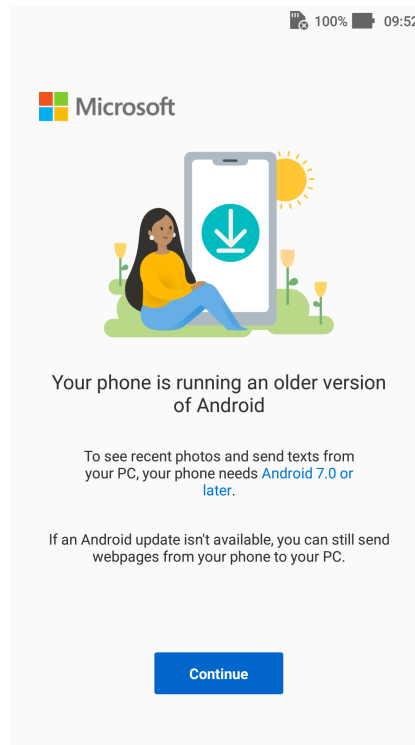


Figure 16: [Your Phone Companion \(YPC\)](#)'s screen for pre 7.0 Android devices.

A user with a pre-7.0 device sees the screen shown in Figure 16. The Android 7.0 requirement has existed since at least February of 2019 (when the work of [YPA](#) was started). The only feature that can be used with a pre-7.0 device is sharing links of webpages between the phone and computer.

Figure 17 shows the distribution of API levels from May 2019 to May 2020. In June, pre-7.0 Android devices accounted for 28.3 percent of Android devices (according to Statcounter's data), so there must be a strong reason for Microsoft to disregard this big of an amount of users - it can only be assumed that this number was bigger back in February. It might be related to their analytics and user statistics back then, but it is easier to believe it was due to platform reasons.

The [YPC](#) app does not have many features by itself, as can be seen in Figure 15a. From the phone, the most a user can do is *i*) link to a computer, *ii*) see Microsoft advertising its other Android apps, *iii*) go to Settings, which can be seen in Figure 15b (where the user can be redirected to the app in the Play Store, offer direct feedback to Microsoft, sync on-demand, and some other small options), and *iv*) share links from phone to the computer (only feature available to pre-7.0 Android devices). The latter can be triggered by using the default Android sharing interface and clicking the [YPC](#) option: *Continue on PC*. When this option is selected, the screen in Figure 18 is shown, where the user can select a connected machine to

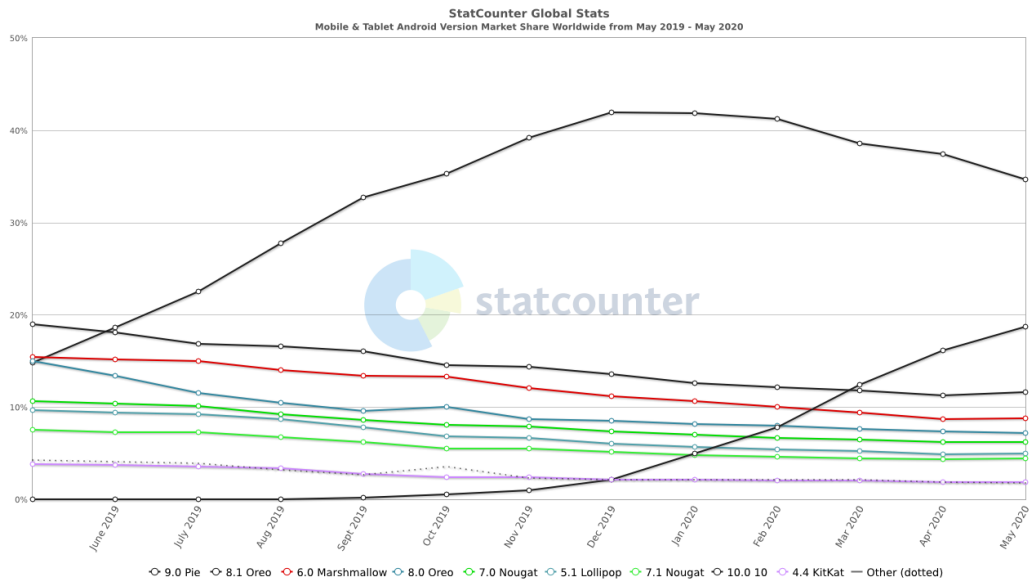


Figure 17: Android API level distribution, May 2019 - May 2020. Source: <https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide>

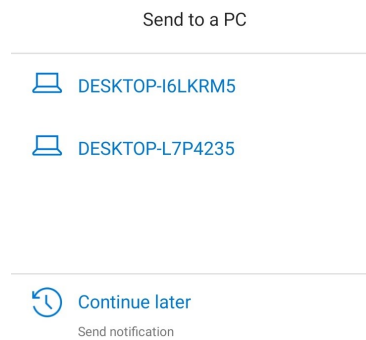


Figure 18: Share a link with YPC.

share the link. In the case shown in Figure 18, two Windows machines connected to the smartphone via Your Phone: DESKTOP-I6LKRM5, and DESKTOP-L7P4235. The link can be opened on either the user's or the OS' default browser, results may vary. In case the user has no active session at the selected machine, the browser is still launched and is accessible whenever the users logs in. If a machine is turned off, that option is greyed out and displays a message that the PC is not available. Lastly, if the user clicks on *Continue later*, it will send a notification to all the machines.

Also seen in Figure 18 is a phone using YPC linked to multiple Windows 10 computers. However, it can not synchronize the phone content with all linked computers at the same time, only one. Syncing with one of the devices, disconnects it from the other, and the notification on the phone (seen in Figure 19) updates to the connected computer. The links remain, nonetheless. The notification shown

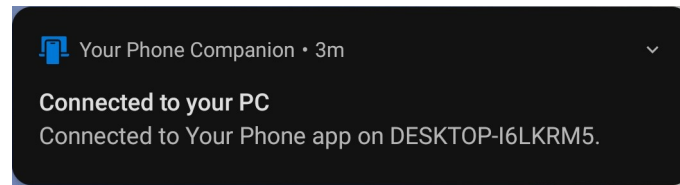


Figure 19: *YPC* notification in the smartphone.

in Figure 19 stays while the computer and phone are connected, and lasts a while longer after the desktop is turned off (probably until the phone tries to sync and fails since the desktop disconnected).

4.1.1.1 *Permissions*

The many features of Your Phone rely heavily on sensitive user data. This data is protected in an Android phone by permissions³. Google states that “the purpose of a permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data”. As such, *YPC* must request the user for permission to access certain *OS* features in the user’s phone.

Table 3 shows the permissions declared by *YPC*, and while there are a lot of permissions requested compared to other apps, it is understandable seeing as each major feature of Your Phone requires a different Android permission (reading SMS, sending an SMS, calls, bluetooth pairing...). The location permission is misleading, but it is necessary since a Bluetooth scan can be used to gather information about the location of the user (Android Developers, 2020). Some of these permissions are required to be requested to the user (e.g. contacts, messages), while others just need to be declared by the application (e.g. internet connectivity). A permission being requested can also depend on the Android version of the user’s phone.

4.1.1.2 *User app data*

It is possible to fetch the local data of an Android application, such as files and databases. It is a simple method, using *Android Debug Bridge (ADB)*’s backup functionality, and works as long as the app has not disabled it⁴. The package name for *YPC* is

³ <https://developer.android.com/guide/topics/permissions/overview>

⁴ <https://blog.shvetsov.com/2013/02/access-android-app-data-without-root.html>

Table 3: Android permissions declared by [Your Phone Companion](#)

PERMISSION	OBSERVATION
Identity	Find/add/remove accounts on the device
Location	Precise location (GPS and network-based)
Wi-Fi	View Wi-Fi connections
Storage	Read and write USB storage
SMS	Read text messages (SMS/MMS). Send SMS
Camera	Take pictures and record videos
Phone	Read call log/phone status and identity
Device/Apps History	List of running apps
Contacts	Read Contacts
Bluetooth	Access settings and pair
Other	Download files without notification
Other	Receive data from the Internet
Other	Full network access
Other	Set an alarm
Other	Pair with Bluetooth
Other	View network connections
Other	Modify system settings
Other	Disable screen lock
Other	Prevent device from sleeping
Other	Create accounts and sets passwords

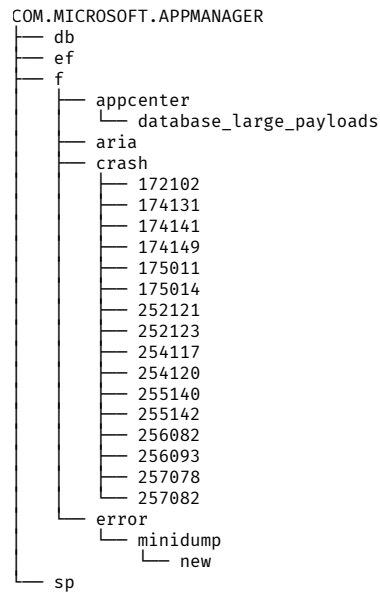


Figure 20: YPC local data path tree.

`com.microsoft.appmanager` - it can be seen in the page link for the app in the Play Store⁵.

This process was followed using a OnePlus 7 with OxygenOS 10.0 (Android 10). The folder structure of the result can be seen in Figure 20.

In Appendix B are the diagrams of the SQLite databases found in YPC’s local app data. In all these databases there is a `room_master_table`⁶. This means that YPC uses Google’s architecture component “Room”, seeing as this is the library’s master table that contains metadata information. Multiple databases in one application is not a recommended practice or usually seen, due to the complexity of having multiple connections, where each connection is quite expensive. This might be well handled by the app though, as a result of most of the app’s work being done in the background. The app does not require to be opened to communicate with the user’s personal computer, as it runs in the background.

YPC’s databases do not have much information about or of the user, meaning that they do not have much forensic value. Most of the content seem to be used to control or aid the exchange of information between the phone app and desktop app. A few of the databases also contain WAL files. Below are some of the databases found in the app data:

5 <https://play.google.com/store/apps/details?id=com.microsoft.appmanager>, notice the end of the URL

6 https://developer.android.com/reference/androidx/room/Room#MASTER_TABLE_NAME

- Database `eventstore`, Appendix B, Figure 50, table `agent_service_event`: it might hold a log message of the form *Connected to machine M*, where *M* is the Windows name of the Windows 10 device.
- Database `YourPhoneSettings`, Appendix B, Figure 51, table `settings`: it holds the smartphone's applications that can produce notifications and whether these notifications should be forwarded to Your Phone Windows 10 devices.
- Database `com.google.android.datatransport.events`, Appendix B, Figure 53, table `event_metadata`: this table holds some data regarding the device hardware and OS, such as the `model`, the `SDK-version` and the `OS-build`. This database is created by the Android Firebase SDK⁷.

Inside the `f` directory, there are some files and other directories. Among those, one stands out: `%ID%.cdp`, where `%ID%` stands for a unique identifier linked to the user's email address. The extension `.cdp` is short for *Certificate Distribution Point*. This file holds information about the phone, IP addresses, and computers that are connected to that phone. This can be valuable to identify Windows 10 computers that might not be known in the investigation, or to link a computer to a phone in an investigation. Inside the file, there is a key `StableUserId` with the value of the filename, meaning that the filename is from the value of that key. With this `cdp` file are a `cer` and `key` files, with the same naming (`%ID%.cdp`).

The rest of the files left by the app are XML files, which some are from Shared Preferences⁸. However, some of those Shared Preferences XML files have some interesting flags, such as `telemetry_consent_shown` (which is in `home_view.xml`), but it has no other content.

4.1.2 Desktop application

The analyzed version of Your Phone was 1.20101.99.0, from November 2020.

Opposite to `YPC`, the desktop application, which henceforth will be mentioned as just Your Phone, has a lot of features. Your Phone was originally installed through the Windows Store, although since Windows 10 version 19.03 it now comes with a clean installation, with updates available through the Windows Store.

⁷ <https://github.com/firebase/firebase-android-sdk/blob/master/transport/transport-runtime/src/main/java/com/google/android/datatransport/runtime/scheduling/persistence/SchemaManager.java>

⁸ <https://developer.android.com/reference/android/content/SharedPreferences.html>

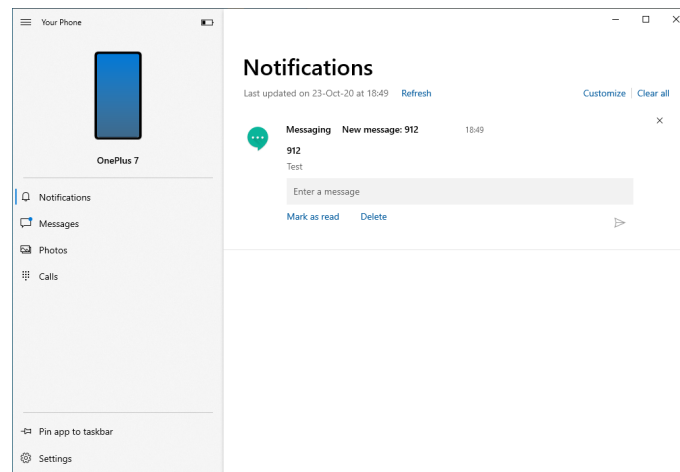


Figure 21: Your Phone interface.

Your Phone has two executables, `YourPhone.exe` and `YourPhoneServer.exe`. The former is the executable used by the scanner to track Your Phone’s version. These executables are both in `C:\ProgramFiles\WindowsApps\%YourPhoneDIR%` where `%YourPhoneDIR%` is `Microsoft.YourPhone`, the numeric version of Your Phone, and the Microsoft publisher ID, `8wekyb3d8bbwe`, all concatenated with `_` as separators. The latter is visible in other [UWP](#) apps that belong to Microsoft.

4.1.2.1 Interface

The Your Phone interface is fairly straightforward. It can be seen in [Figure 21](#). It has a side bar with six options, where each changes the main view, except the *Pin app to taskbar* option. All options show the same title and refresh option. The refresh option allows the user to manually sync the phone and computer.

The default selected side bar option is *Notifications*, which displays an image if the user has no notifications, or the user’s phone notifications. The user can take action from this screen, as is also seen in the [Figure 21](#) (*Mark as read* and *Delete*). These actions depend on the app that sent this notification. The notifications shown in this screen also show on Windows’ Action Center, using [WNS](#). Also worth mentioning that the *Customize* button redirects to the Settings screen, with an inner option selected (*Features*).

The *Messages* option shows the user’s messages – SMS and MMS from the last 30 days. The user can read, send and reply to messages without leaving the computer, as shown in the left image of [Figure 22](#). The user can also search through the phone’s contact list when sending messages (by choosing who to send the message to). However, the user cannot delete messages within [YPC](#).

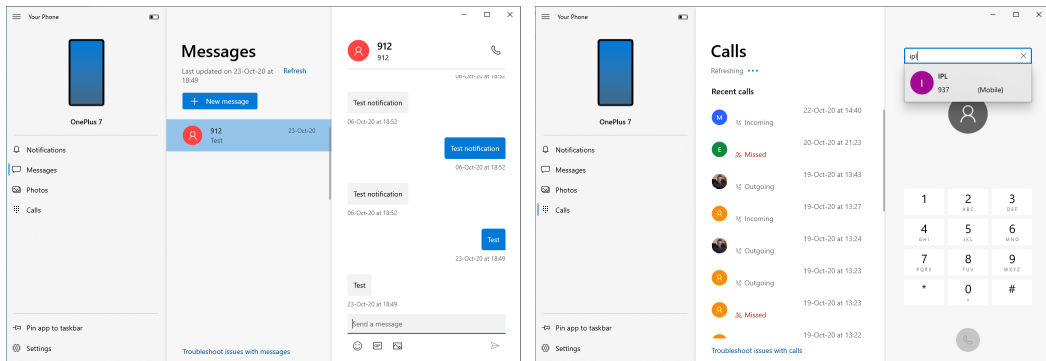


Figure 22: Your Phone messages (left) and calls interface (right).

Photos shows the phone's photos. From this screen, the user can choose a photo, open it with any desktop application, copy, save, or share. However, any changes on a photo are not synchronized back to the phone's photo.

The *Calls* screen only has content if the user's computer has Bluetooth. If it does not, it says the feature does not work with the computer. What Your Phone does not tell the user is that it still has the user's call log (if the user granted that permission). Your Phone could show the call log even if the user had no Bluetooth, due to the call log being present in the database. Anyway, in case the user has Bluetooth and connects the phone, he/she can see the call log, search contacts, and perform calls. This is shown on the right side of Figure 22. Calls will use the computer's sound and microphone devices.

Settings screen shows a lot of options for the user to customize his Your Phone experience, which can be seen in Figure 23. In this last Figure, there's also what appears like a Spotify notification, showing what is playing on the phone (bottom left corner). This is more than a simple notification, it is the Media Controls⁹. That is why it does not appear in the *Notifications* screen, even though it appears in the notifications database, as will be seen further. It also features actions for the user, in this case *Previous*, *Play/Pause*, *Next* (same order as in the Figure's).

An easy to miss detail is the phone's battery near the top left corner, seen in all of the Figures 21, 22, and 23. In all these cases, the phone is low on battery. It shows the percentage of the battery as a tooltip by hovering the icon.

Despite not having a contact view or screen, the user can search the contacts from both *Messages* and *Calls*. This is important, since it means Your Phone also stores the user's contact list. As mentioned above, Your Phone can also have the

⁹ <https://developer.android.com/guide/topics/media/media-controls>

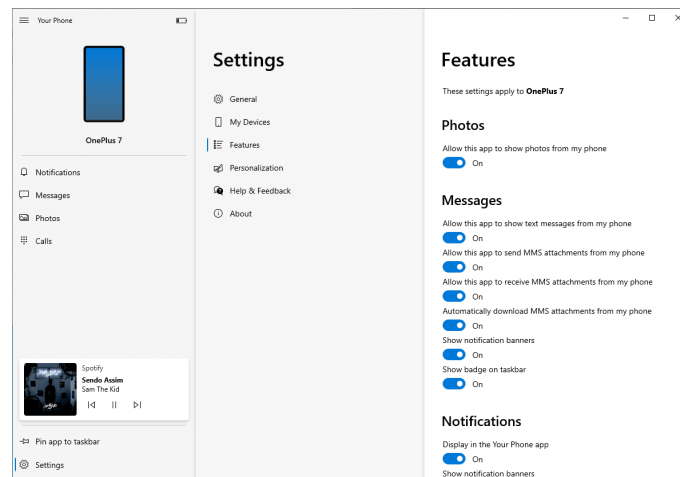


Figure 23: Your Phone settings interface and Media Controls (bottom left corner).

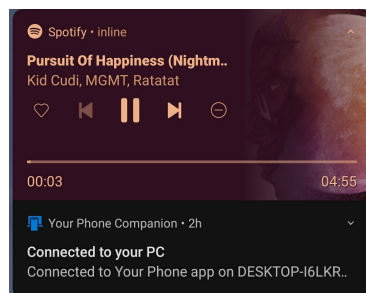


Figure 24: *Your Phone Companion*'s Media Controls notification.

call log, even if not connected by Bluetooth to the phone. These are two seemingly important assets in a digital forensics point of view.

Both *Messages* and *Calls* screens show content up to 30 days old. Even in visible conversations that have happened in the current month, it will not show older content. The two are also ordered from most recent to oldest.

The *Photos* screen shows up to 2000 photos/screenshots taken by the smartphone. It presents the photos sorted from the most recent to the oldest, like the previously mentioned screens. The newer photos have both a thumbnail and full sized local file, while the older ones have only a thumbnail. Old photos can have a full sized photo by exporting the photo or by opening the full size screen in *Your Phone*. It also shows a *Syncing from your phone...* message while fetching the photo. When *Your Phone* fetches a photo from the phone, it is downsized to no larger than 1.5 MiB. The original aspect ratio and filename are maintained, but the EXIF metadata is only partially preserved (*Camera Model Name* and *Maker* fields).

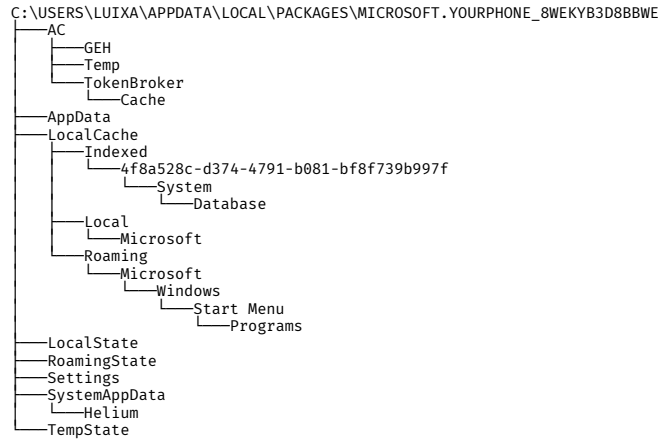


Figure 25: Your Phone local data path tree.

4.1.2.2 User local data

The user data is stored in `%LOCALAPPDATA%\Packages\Microsoft.YourPhone_8wekyb3d8bbwe`. `%LOCALAPPDATA%` is mapped to `C:\Users\%USERNAME%\AppData\Local`, where `%USERNAME%` corresponds to the Windows 10's username. This means that Your Phone can be used by multiple users in the same desktop, and needs to be accounted for when developing software to automate the extraction of data from the files in this directory. As seen before, `8wekyb3d8bbwe` is Microsoft's publisher ID. Figure 25 shows the path tree of the local data. It should be emphasized that `TempState` holds quite a few directories, each named after the package name of a mobile app that is installed in the device, following an integer (e.g `com.microsoft.appmanager_48`). Most of these directories are empty, except a few (some of the ones with the highest integer) which contain a `png` image file with the app's icon.

From the directories seen in Figure 25, only one holds meaningful data - `LocalCache`, which contains a subdirectory (inside `Indexed`) with fix blocks of hexadecimal symbols, separated by dashes, which is a GUID that corresponds to the ID assigned by Microsoft. Inside this directory lies the `System\Database` subdirectory, which contains the Your Phone databases, the “holy grail” of data of Your Phone. The number of databases in this directory has increased over time, starting from one SQLite3 database (`phone.db`) to, at the moment of writing, eight SQLite3 databases.

These databases are listed in Table 4. Each database will be explored in the next sections. All these databases are accompanied with their respective `WAL` files.

Table 4: Databases of Your Phone - version 1.20071.95.0

NAME	DESCRIPTION
<code>calling.db</code>	Call log
<code>contacts.db</code>	Contact lists
<code>deviceData.db</code>	Device-related data (e.g. wallpaper)
<code>notifications.db</code>	Pending phone notifications
<code>phone.db</code>	SMS/MMS/RCS
<code>photos.db</code>	Photos and metadata
<code>settings.db</code>	List of phone apps
<code>sharedcontent.db</code>	Not populated yet

Before detailing each database, bear in mind that the tables `content_sequence` and `sqlite_sequence` are not accounted for. All databases contain the latter, as SQLite3 creates it automatically for databases that contain an `AUTOINCREMENT` column¹⁰. For the former, `notifications.db` and `settings.db` are missing it - this table is probably used internally by Your Phone, but it has no meaningful data.

The date/time fields are in UTC, unless stated otherwise. When mentioning `filetime64`, it is short for Microsoft Filetime, which is the number of 100 ns intervals since January 1st 1601¹¹. `UNIX` is the UNIX Epoch, the number of seconds since January 1st 1970, and `UNIXms` is the same, but in milliseconds.

4.1.2.3 `calling.db`

The only table in this database is `call_history`. Table 5 shows the fields of this table with a brief description of each field.

This table keeps the user's call log of the last month. Each row is a call, with the phone number of the other party, duration of the call (in seconds), and the start date/time. The field `call_type` is an integer with four known types: 1 is *incoming*, 2 is *outgoing*, 3 is *missed*, 5 is *declined*, and 6 is *blocked*. Value 4 was not found during our research, so it is not possible to determine if it exists or not, and which state it corresponds to. It might also have been deprecated or discontinued internally. `is_read` can be either 0 (seen) or 1 (missed/not seen). Calls that have `is_read` with value 1 have `call_type` with value 3 (missed). This does not mean

¹⁰ <https://sqlite.org/autoinc.html>

¹¹ https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/2c57429b-fdd4-488f-b5fc-9e4cf020fcdf

Table 5: Fields of table `call_history` / `calling.db` database

FIELD	TYPE	DESCRIPTION
<code>call_id</code>	integer	Primary key
<code>phone_number</code>	text	Called/calling phone number
<code>duration</code>	integer	Duration of the call (seconds)
<code>call_type</code>	integer	1=incoming 2=outgoing 3=missed 5=declined
<code>start_time</code>	integer	GMT date/time start of call (filetime64)
<code>is_read</code>	integer	0 = seen, 2 = not taken/not seen
<code>last_update_time</code>	integer	GMT date/time last data update (filetime64)
<code>phone_account_id</code>	integer	Always the same with only one phone/SIM card

that `call_type` 3 always has value 1 for `is_read`. The value can be 0 if the missed call was purposely silenced in the phone, but not declined - e.g. when someone calls another person and the second person silences his phone (by clicking the lock screen button). The `last_update_time` field records when the call (table row) was last updated. Usually, the value corresponds to `start_time` plus the duration of the call and a small delay (under 60 seconds) or higher. Both `last_update_time` and `start_time` are in filetime 64-bit format. The last field, `phone_account_id`, always had the same value. It was observed as always 1 in one desktop/phone pair, and as 8935101811520418416 in another pair. It is suspected that the field is used for multi-SIM card support, since Your Phone can handle phones with more than one active SIM¹².

4.1.2.4 `contacts.db`

This database contains 38 tables, and while that might seem like a lot of tables, 32 of those are `FTS` tables, which do not have much forensic value. The information present in most of these `FTS` tables is present in other tables. As mentioned in Section 2.3.1.4, these are used to ease searching for something. If the user types either the display name or nickname of a person in the contact search (be it in the `Calls` or `Messages` screens), it will appear. Unfortunately, other fields are not updating to their actual values yet in `contacts.db`, so searching for those fields retrieves no values.

From the other six tables, only two hold data, `contact` and `phonenumber`. The other four tables are empty in the current Your Phone version, but might have some importance in the future.

¹² <https://www.windowscentral.com/your-phone-app-now-supports-dual-sim-insiders>

Table 6: Fields of table `contact` / `contacts.db` database

FIELD	TYPE	UPDATES	DESCRIPTION
<code>contact_id</code>	integer	Yes	Primary key
<code>display_name</code>	text	Yes	Contact's full display name
<code>nickname</code>	text	Yes	Contact's nickname
<code>last_updated_time</code>	integer	Yes	GMT date/time last updated (filetime)
<code>thumbnail</code>	blob	Yes	Linked photo thumbnail
<code>checksum</code>	integer	Yes	Changes when contact is updated
<code>company</code>	text	No	Empty
<code>job_title</code>	text	No	Empty
<code>notes</code>	text	No	Empty
<code>name_prefix</code>	text	No	Empty
<code>name_suffix</code>	text	No	Empty
<code>given_name</code>	text	No	Empty
<code>middle_name</code>	text	No	Empty
<code>family_name</code>	text	No	Empty

Each row in the `contact` table matches an entry in the user's phone contact list, with the `display_name`, `last_updated_time` timestamp, and the `thumbnail` if available, which corresponds to a contact's photo. Updating a contact's prefix, suffix, middle name, and/or surname in the phone does not populate the independent columns in this table (such as `name_prefix`, `name_suffix`), but it does append these values to the contact's display name as follows: `%PREFIX% %NAME% %MIDDLE NAME% %SURNAME%, %SUFFIX%`. There are more fields that are empty and not updated, such as `company` and `job_title`. These fields can be seen in Table 6, along if the field is updated or not.

Table `phonenumber` is a simpler table. It has its own ID column, along with a mandatory contact's ID, which is a foreign key and can be linked to the `contact` table. The table has the `phone_number` and `display_phone_number` fields, which for a row contain the same number with different formats: the former contains the international code, such as `+351` for Portugal and no whitespaces, while the second might also contain the international code but is formatted with certain spacings and serves as a display-friendly name. There is also a `phone_number_type` field which is an integer that maps to a type of phone number as follows: 1 is home, 2 is mobile, 3 is work, 4 is work mobile, 5 is main, and 6 is for any other label the phone might have. All these fields can be seen in Table 7.

The other four tables of this database are: *i)* `contactdate`, *ii)* `contacturl`, *iii)* `emailaddress`, and *iv)* `postaladdress`. As mentioned above, and like many other fields in the `contact` table, these are empty.

Table 7: Fields of table `phonenumbers` / `contacts.db` database

FIELD	TYPE	DESCRIPTION
<code>phone_number_id</code>	integer	Primary key
<code>contact_id</code>	integer	Called/calling phone number
<code>phone_number</code>	text	Clean phone number (no spaces)
<code>display_phone_number</code>	text	Display-friendly phone number
<code>phone_number_type</code>	integer	1=home, 2=mobile, 3=work, 4=work mobile, 5=main, 6=other
<code>label</code>	integer	Empty
<code>checksum</code>	integer	Updates when row is updated

4.1.2.5 *deviceData.db*

The only table in this database is `wallpaper`. It contains a row with the current wallpaper, which can have the blob of the wallpaper (it is nullable, and for some included phone wallpapers the blob is null, e.g. in the case of a OnePlus 7), which can then be shown in Your Phone. Forensically, this database does not have much value as of yet, unless the wallpaper is an important or incriminating photo.

4.1.2.6 *notifications.db*

This database has only one table: `notifications`. It stores the notifications that are displayed in the phone screen, such as a message, missed phone call, or any other notification. It can also have the Media Controls notification, such as a song that is playing on Spotify, or a YouTube video that is being cast to another device (e.g. smart television). The phone media controls can be seen in Figure 24, and the Your Phone variant can be seen in Figure 23. Notifications can only be sent from the phone to the desktop if the user has the setting enabled for that certain app. The user can change these settings at any time, and the interface for enabling/disabling the notification forwarding from the phone to the desktop is shown in Figure 26.

From a forensic point of view, the most important fields are `json` and `post_time`. The former holds the full JSON representation of that notification alert, including the app name and other JSON fields of the originator of that alert. The `post_time` represents the date/time (filetime64) when the notification was posted. Whenever a notification is dismissed from the phone, the row is deleted, and due to that the probability of capturing certain records is not evident, although recovery tools, such as `undark`, `bring2lite` to name a couple of open source software, can be used

Table 8: Record for notification of a group of WhatsApp messages in table `notifications`

NAME	DESCRIPTION
<code>id</code>	6
<code>notification_id</code>	0 com.whatsapp 1 3fu8LLvJtKLhn8VIWAQJ8bOKh+RBjudquZ/c4+D7JQY= 10250
<code>package_name</code>	<code>com.whatsapp</code>
<code>json</code>	Shown in Listing 5 (Appendix A)
<code>post_time</code>	132484855169050000
<code>state</code>	0
<code>anonymous_id</code>	52DC3EF6-1A85-4B7D-A12D-41B71AB60790

to attempt to recover deleted data, as shall be seen later on (Daniels, 2020; Meng and Baier, 2019).

A table row from `notifications` corresponding to a WhatsApp notification of multiple messages can be seen in Table 8, while the JSON can be seen in Listing 5.

As can be seen in Table 8 and Listing 5, `notification_id` is the concatenation of multiple fields of the JSON, such as the package name, tag, and the value of the `notification_id` field is the same as the JSON's `key` value. As a side note, the `post_time` field is set with the 64-bit filetime value of 132484855169050000, which corresponds to Thursday, October 29th, 2020 10:51:56PM.

4.1.2.7 *phone.db*

Unlike the other databases, the name of this database, `phone.db`, can be misleading and reflects the architecture of the first version of Your Phone, where this database held all the data for Your Phone (Domingues et al., 2019). Since then, the `phone.db` database has been heavily modified, as it has the highest PRAGMA value for `user_version` out of all of the Your Phone databases - 31, comparing to the other highest `user_version` values 10, of `settings.db` and `contacts.db`. These modifications have added support for multiple SIM cards and RCS, while leaving behind legacy databases such as `message_to_address`, and `mms_to_address`, which are no longer used, and are empty now. The full list of tables of this database can be seen in Table 9.

As a side note, this database shows the fast evolution of Your Phone, and the Scanner (3.2) allows us to always be up-to-date with these constant changes.

Table 9: Tables of database `phone.db`

TABLE	LEGACY	EMPTY	DESCRIPTION
<code>conversation</code>	No	No	Conversation threads table
<code>message</code>	No	No	SMS table
<code>message_to_address</code>	Yes	Yes	Legacy SMS table
<code>mms</code>	No	No	MMS table
<code>mms_address</code>	Yes	Yes	Legacy MMS table
<code>mms_part</code>	No	No	Attached MMS data
<code>rcs_chat</code>	No	Unknown	RCS table
<code>rcs_conversation</code>	No	Unknown	RCS table
<code>rcs_filetransfer</code>	No	Unknown	RCS table
<code>sending_message</code>	No	No	Messages not yet sent
<code>subscription</code>	No	No	One record per phone SIM
<code>sync</code>	No	No	Synchronization data

This database mostly contains tables holding data linked to phone-based messaging, such as SMS, MMS, and RCS. The table `message` holds the data and metadata of SMS from the last 30 days. Besides the text of the message, which is kept in the `body` field, the `message` table holds the `status` (1 is unread, 2 is read), the `from_address` that identifies the sender (empty for a sent SMS), and `type`, which is 1 for a received message, and 2 for a sent one. The `timestamp` (`filetime64`) records the date/time the SMS was sent/received. The `subscription_id` acts as a foreign key, linking to the `subscription` table. This table keeps the data about the SIM cards attached to the phone - one record per SIM card - and the associated configurations. Besides the primary key `subscription_id`, the other fields of this row have fairly straightforward names such as `country_iso`, `is_mms_enabled`, and `is_rcs_supported`. A less straightforward field name is `name`, which in certain cases holds the SIM card provider (e.g. Vodafone), while in others is simply set to SIM1. A `subscription` row holds 27 fields at the time of writing.

Another foreign field of the `message` table is `thread_id`, which links to the `conversation`'s table primary key. It identifies the thread to which a message belongs to. A thread is a set of messages, be it SMS, MMS, or RCS, that are shared among the same recipients (can be one or more). The recipient list is under the `recipient_list` field, where recipients are separated by a comma if there is more than one (e.g. `contact1,contact2`). Other fields are counters for the conversation, such as `unread_count`, which is the number of unread messages in Your Phone (`phone_unread_count` is the number of messages that are not read yet in the phone), and `msg_count`, which is the total number of messages, while `has_rcs` is 0 when there are no RCS messages in the conversation. Finally, `timestamp`

Table 10: Record of a conversation in `phone.db`'s `conversation` table

FIELD	VALUE
<code>thread_id</code>	43
<code>recipient_list</code>	938*****
<code>timestamp</code>	132485692195370000
<code>msg_count</code>	12
<code>unread_count</code>	0
<code>has_rcs</code>	0
<code>checksum</code>	40876
<code>phone_unread_count</code>	1

holds the date/time of the last exchanged message. A very interesting thing about the `conversation` table from a forensics point of view is that the records from conversations that have no messages in Your Phone are still preserved in this `conversation` table. This occurs when the conversations are still present in the phone, but no longer processed by Your Phone due to the 30-day expiration. These records can be easily identified as all fields are zero, except for `thread_id` and the `recipient_list`. Table 10 shows an example of a conversation row, with a 64-bit Filetime formatted timestamp - 132485692195370000, corresponding to Friday, October 30th, 2020 10:06:59PM

For MMS, there are three tables in `phone.db`: `mms`, `mms_address`, and `mms_part`. As previously mentioned, `mms_address` no longer contains any data and is a legacy table. Table `mms` holds a row per MMS sent/received, recording the `thread_id` (which, like SMS messages, links to `conversation`), `status` with a value of 1 (unread) or 2 (read), `type` with a value of 1 (received) or 2 (sent). Similarly to `message`, `mms` has a `subscription_id` which links to the `subscription` table. Table 11 shows a record of an MMS.

As the name indicates the `mms_part` table holds parts for MMS messages. For an MMS comprised of two parts, for example text and photo, there are three records: the two parts plus the record that keeps the [Synchronized Multimedia Integration Language \(SMIL\)](#) data. [SMIL](#) is an XML markup language for interactive multimedia presentations (Bulterman and Rutledge, 2008). Within the context of an MMS, it serves to configure the presentation of the MMS on the screen of the phone. Table 11 holds a record of an MMS from the `mms` table.

Table 11: Record of an MMS in `phone.db`'s `mms` table

NAME	DESCRIPTION
<code>message_id</code>	1
<code>thread_id</code>	16 (associated conversation thread)
<code>status</code>	2 (read)
<code>type</code>	2 (sent)
<code>subscription_id</code>	1 (associated SIM card)
<code>subject</code>	(empty text)
<code>charset</code>	0
<code>timestamp</code>	132485707550000000
<code>pc_status</code>	1
<code>from_address</code>	<code>insert-address-token</code> (default value for sent MMS)
<code>checksum</code>	1

There are three tables for RCS messages: `rsc_chat`, `rsc_conversation`, and `rsc_filetransfer`. Without an RCS compatible phone, it is not possible to address these tables and verify that they are already in-use by Your Phone.

Lastly, the table `sending_message`, as the name suggests, holds messages sent from Your Phone which have not yet been sent by the phone. This can occur when either the computer or phone have no internet access. As soon as the connection is established, the messages in this table are sent and the record is deleted from the table.

4.1.2.8 *photos.db*

There are two tables in `photos.db`: `photo` and `media`. The `photo` table has stopped seeing use in recent Your Phone versions, and is now considered legacy. In older versions, this table would hold up to 25 photos/screenshots. This table is no longer synchronized with Your Phone, so if there was any data left by an older synchronization, it is still retrievable. For installations after this table has stopped seeing updates, this table is empty. Similarly, these older photos/screenshots (up to 25) from the first versions of Your Phone might still be found in the filesystem, in the directory `\\LOCALAPPDATA\\LocalCache\\Indexed\\%GUID\\User\\%PHONENAME\\Recentphotos`, where `LOCALAPPDATA` and `GUID` have the same values as explained before, and `PHONENAME` is the phone's display name (e.g. Aquaris, OnePlus 7) (Domingues et al., 2019).

Table 12: Fields of table `media` / `photos.db` database

FIELD	TYPE	EXISTED IN PHOTOS	DESCRIPTION
<code>id</code>	integer	As <code>photo_id</code>	Primary key
<code>name</code>	text	Yes	Media file name in the phone
<code>last_updated_time</code>	integer	Yes	Date/time of last update (filetime64)
<code>taken_time</code>	integer	Yes	Date/time when media taken (filetime64)
<code>orientation</code>	text	No	Media angle (values: 0/90/180/270)
<code>last_seen_time</code>	integer	No	Date/time last media access in Your Phone (filetime64)
<code>mime_type</code>	text	No	Media file mimetype
<code>height</code>	integer	No	Media resolution height
<code>width</code>	integer	No	Media resolution width
<code>size</code>	integer	Yes	Media size (in bytes)
<code>uri</code>	text	Yes	<code>content://media/external/ images/media/ %ID%</code>
<code>thumbnail</code>	blob	Yes	Blob of media thumbnail
<code>media</code>	blob	As <code>blob</code>	Blob of full media (or null)
<code>checksum</code>	integer	No	Updates when row is updated

In more recent versions of Your Phone, photos and media are kept in the `media` table. The fields of this table are summarized in Table 12.

In the old `photos` table, `uri` was the full phone path to the image. In the new `media` table, it is `content://media/external/images/media/ %ID%`, where `ID` is the media ID. The `media`'s table new fields suggest that Your Phone has support for multiple media types, but at the time of writing it only has photos inside, with the mimetype `image/jpeg`.

Both `thumbnail` and `media` can be null. `media` is not null when the file is synchronized with the phone, normally when the photo is not a month old. Once it is a month old and is not manually loaded, it is null. `thumbnail` is usually not null, and contains a much more reduced image, but keeps the aspect ratio too (an image with the resolution 4608×3456 is reduced to 320×240 , keeping its $4/3$ aspect ratio, however it has over 200 times fewer pixels as the original). The reduced photos are primarily used to show the thumbnails in the Your Phone interface, but it might also be to preserve the bandwidth usage while transferring photos from the phone to the computer, and disk space of the Windows 10 desktop, as holding the maximum of 2000 photos in the database might require more than 2 GiB of disk space, considering the maximum storage size of 1.5 MiB per photo (Section 4.1.2.1) kept in the `media` table.

Table 13: Record for [Your Phone Companion](#) in table `phone_apps`

NAME	DESCRIPTION
<code>app_name</code>	Your Phone Companion
<code>package_name</code>	<code>com.microsoft.appmanager</code>
<code>version</code>	1.20101.99.0
<code>etag</code>	252
<code>favorite_rank</code>	NULL
<code>blob</code>	Application icon (blob)

Other fields are `name`, and `taken_time`, the latter holding the date/time of when the photo was taken. The table has two more GMT date/times, `last_updated_time` and `last_seen_time`. The former is the timestamp of when the photo was last synchronized within Android, and the latter corresponds when the photo was last displayed in Your Phone. The fields `height`, `width`, and `size` hold the dimensions of the original photo.

4.1.2.9 *settings.db*

The `settings.db` database possesses three tables: `phone_apps`, `phone_requests`, and `settings`. Of these three, `phone_requests` has no records.

The table `phone_apps` contains all the apps installed in the smartphone. Each row is an app, and contains the self explanatory fields: `app_name`, `package_name`, and `version`. Other fields are `blob`, which is the app's icon, `favorite_rank`, which has the value NULL for all records (even for apps with rating and reviews), and `etag`, which has the same value for all records: 252 (the value itself can be different for a user, but it is the same for all records anyway). An example of a record of this table can be seen in Table 13 - in this case it is the record of [YPC](#).

The `settings` table does not have many columns. It holds the settings of Your Phone relative to the applications installed on the phone. The table fields are `setting_group_id`, `setting_key`, `setting_type`, and `setting_value`. The first two form the primary key, which has some significance as will be seen ahead.

The value of `setting_group_id` is `NotificationSyncSetting` for all rows, leaving the suspicion that this is the only available setting at the moment of writing.

The table contains this setting with a value for some of the apps present in the test phone that can use push notifications. The criteria is that the app must have

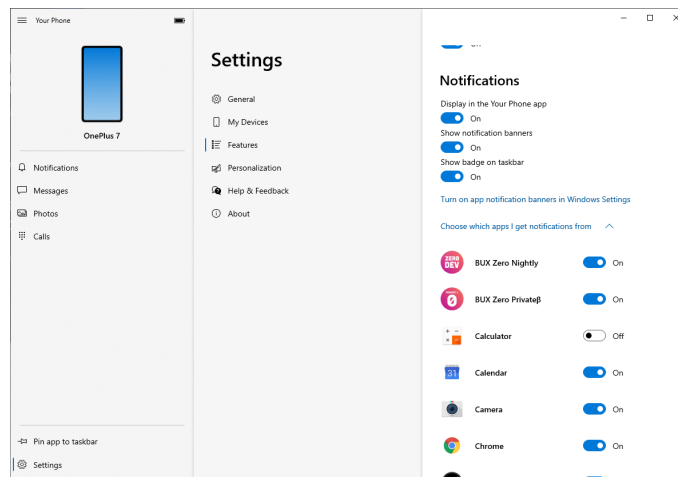


Figure 26: Your Phone interface for disabling notifications.

sent at least one notification. It serves as a way to tell the phone which notifications can be pushed from the phone to the desktop.

`setting_value` contains an integer that represents if the setting is active or not. In the case of *NotificationSyncSetting*, 0 means that the notification is not delivered to the desktop, and 1 means it is. The user can enable or disable this setting in the desktop. To do so the user needs to disable it via Your Phone - Settings - Features - Notifications (or by clicking *Customize* in the notification screen) and disable the app notification there - all apps are shown in this list, even the ones that can not push notifications. Figure 26 shows this menu. As soon as it is disabled, the table is updated with the new value. The disabled notifications in Your Phone do not appear in `notifications.db` either. By accessing *Notifications & action settings* in the Windows 10's *System settings*, the user can disable notifications from appearing in the Action Center, but they are still delivered to Your Phone. This last method does not update the value of `setting_value` either.

The field `setting_key` is the package name of the app in question, and is naturally the same value as `phone_apps`'s `package_name`. Lastly, the `setting_type` field always has the same value, 11. The purpose of this field is unclear.

Since `settings_group_id` and `setting_key` form the primary key of this table, Your Phone can have multiple settings for an app. As of now, only *NotificationSyncSetting* has been observed.

4.1.2.10 *sharedcontent.db*

This database has two tables: `sharedcontent` and `shareduri`. In the studied version of Your Phone, both tables have no data. Even `sqlite_sequence` is empty. From the database, table, and column names, it is likely that it will have content related to sharing something from the phone. Perhaps when sharing a link from the phone, using the native Android sharing interface implemented in [YPC](#), and other media sharing between the phone and computer.

This database was first spotted in June 14th 2020 according to the scanner. This is still the most recent Your Phone database, and it might see use in future releases.

4.2 YOUR PHONE ANALYZER

[YPA](#) is an open source Autopsy module, licensed under a [GPL 3.0](#) license¹³. Autopsy is a well known open source software that combines several tools under a [GUI](#) to form a versatile solution for digital practitioners. Autopsy is detailed in [Section 2.1.1](#). Autopsy can be extended by external modules, and that is exactly what [YPA](#) is, adding support for Your Phone within Autopsy.

Autopsy plugins can be developed either in Java, or Jython 2.7, which is Python code that is ran within a JVM environment. In reality, [YPA](#) is comprised of two Autopsy Jython modules: *i*) a data source ingest module and *ii*) a report module. The latter is entirely dependent on the first, and it this approach can be described as a two-part module, giving the practitioner full control of how the data is presented.

4.2.1 *Ingest module*

The ingest module is responsible for finding Your Phone databases, extracting information from their content into Autopsy's [Blackboard](#), which is the artifact repository. The data is then displayed in Autopsy's interface, as shown in [Figure 27](#), which displays data from the user [Demo](#).

The artifacts shown in [Figure 27](#) will be explained further in this section. Every artifact follows the naming `YourPhone: User %USER% - %ARTIFACT%`, where the `%USER%` is the Windows 10 user, and `%ARTIFACT%` is the actual artifact name. The app prefix is to distinguish the artifacts from the many other artifacts that a case

¹³ <https://github.com/labcif/YPA/blob/master/LICENSE>

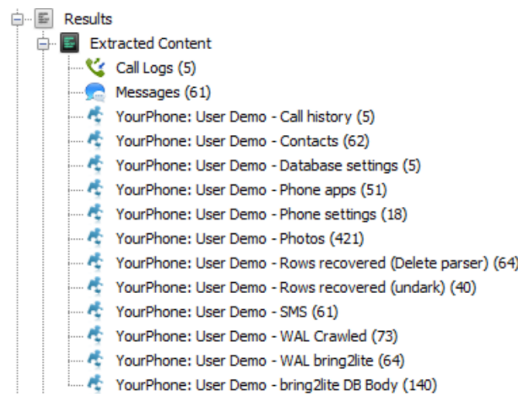


Figure 27: *YPA*'s artifacts in Autopsy's interface.

might have, while the user is to distinguish from the many users a computer might have with Your Phone - *YPA* has support for multiple users in the same machine.

After downloading *YPA* from its repository¹⁴, all that is left for the user is to run the ingest module. *YPA* requires little to no work to run, as the only thing that the practitioner needs to do is put the module in the correct Autopsy folder. The practitioner is greeted by the *GUI* shown in Figure 28.

The practitioner can then choose which recovery data tools he/she wants, from a list of *i*) *undark* (Daniels, 2020), *ii*) *SQLite-Deleted-Records-Parser* (DeGrazia, 2015), *iii*) *WAL-Crawler* (Miller and Bryce, 2019), and *iv*) *bring2lite* (Meng and Baier, 2019). The last tool, *bring2lite*, has two separate options, as the practitioner can choose whether he/she wants to run another *WAL* tool or not. *SQLite-Deleted-Records-Parser* will be henceforth referred as *MDG-Parser*.

The settings chosen by the practitioner are saved, and kept for the next time the module is ran. *undark* and *MDG-Parser* are enabled by default, as their runtime is relatively short.

As previously explained, Your Phone, for many of its databases tables, keeps records only of the last 30 days. This makes the data recovery tools a very important asset for *YPA*, as executing all these tools gives higher chances of recovering important user data, possibly from a time period older than 30 days. Furthermore, *YPA* implements these tools without requiring the practitioner to install anything else by him/herself, functioning in a plug-and-play manner.

Of course, the data produced by these tools is often not structured enough to create the same artifacts that can be extract from the plain *SQLite* databases, and as such, the artifacts produced by these tools are usually in a free text format, leav-

¹⁴ <https://github.com/labcif/YPA>

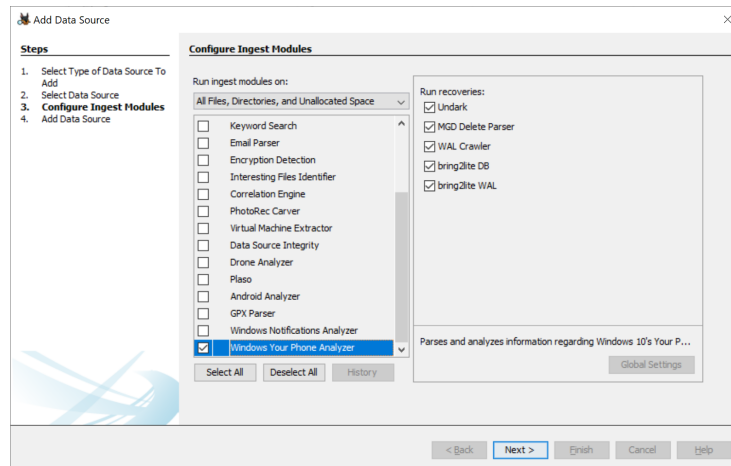


Figure 28: YPA's ingest module GUI.

Table 14: YPA's recovered data artifacts.

ARTIFACT	TOOL	ATTRIBUTES
Rows recovered	undark	Data recovered from unvacuumed row (free text)
Rows recovered	MDG-Parser	Type (unallocated/free block), Offset, Length, Data (free text)
bring2lite DB body	bring2lite	Page and row content (free text)
WAL Crawled	WAL-Crawler	Frame, Salt-1, Salt-2, Frame Offset, Cell, Cell Offset, ROWID, Data (free text)
WAL bring2lite	bring2lite	Page and row content (free text)

ing the practitioner to interpret the results. Table 14 shows the artifacts produced by these tools.

The last two artifacts from Table 14 are extracted from the database's **WAL** files, while the other artifacts are directly from the SQLite databases.

After the practitioner chooses the data recovery tools for his/her investigation, YPA attempts to find Your Phone databases (**phone.db**). Once a database is found, it will extract all available data from that database, and find other Your Phone databases that should be in the same directory as **phone.db**, and extract data from those tables too. Table 15 shows the artifacts extracted from these databases and tables.

It is worth noting that as YPA queries the databases, it also runs the requested data recovery tools, creating the aforementioned artifacts for each tool.

Table 15: YPA's Your Phone artifacts.

ARTIFACT	DATABASE	TABLE
Call history	calling.db & contacts.db	call_history, phonenumber & contact
Contacts	contacts.db	contact & phonenumber
Database settings	All databases	-
Notifications	notifications.db	notifications
Phone apps	settings.db	phone_apps
Phone settings	settings.db	settings
Photos	photos.db	media & photo
Recent photos	-	-
SMS	phone.db & contacts.db	message, phonenumber, conversation & contact
MMS	phone.db & contacts.db	mms, mms_address, phonenumber & contact

YPA copies the databases from the data source into a temporary folder, and the copies are where the queries are executed, in order to not tamper any data. For the WAL files, the same procedure is applied when using the data recovery tools.

Each of these artifacts from Table 15 are the result of the rows from a query to the databases' tables. It can also be seen that some artifacts are from a certain database and contacts.db. Some queries need this database attached in order to add more details to the artifact, such as the contact display name of whom performed a call or sent a message, instead of just having the phone number. Attaching a database with SQLite is simple - ATTACH DATABASE %DB_PATH% AS %ALIAS%, where %DB_PATH% is the path to the database to be attached, and %ALIAS% is the alias that can be later used in queries.

The *Recent photos* artifacts is related with the older photo directory of Your Phone. Even though Your Phone no longer puts photos in that directory, YPA still checks for them. These image files are not dependent on any database queries.

4.2.2 Communications Visualization

Other than the data recovery artifacts and Your Phone artifacts, YPA also indexes native Autopsy artifacts. These artifacts can be seen in Table 16. Autopsy's *Communications* module uses the native artifacts to populate its interface. YPA does

Table 16: YPA's native artifacts.

ARTIFACT	ARTIFACT ID	BASED OF
Call Logs	TSK_CALLLOG	YPA's Call history
Messages	TSK_MESSAGE	YPA's SMS

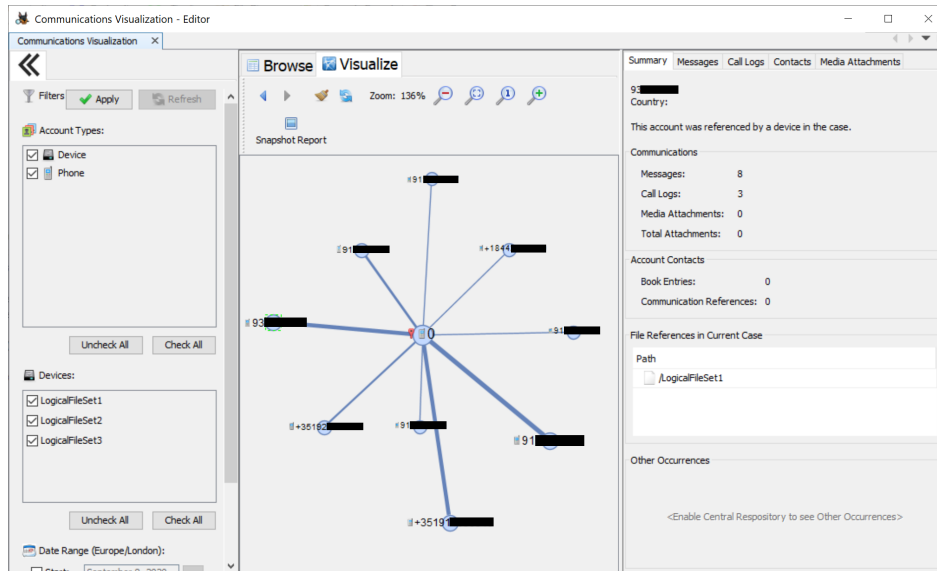


Figure 29: Communications Visualization of a case using YPA.

not add as much data to these artifacts as it does for its own artifacts, as the native artifacts should be paired with native attributes.

Autopsy 4.15.0 has introduced a condition to the communication accounts that does not allow non-valid phone numbers (such as messages from services, e.g. *Google*, *GitHub*), and with that condition the amount of YPA and native artifacts can be different. Nevertheless, this feature allows for a great visualization of the user's communications, as can be seen in Figure 29. It is worth mentioning that this condition is still present in Autopsy 4.17.0, which is the version which the module was last tested.

The interface shown in Figure 29 shows information about the selected account in the right part of the interface - this account can be seen with a green dotted square around the account's circle. From this interface, the practitioner can see the messages, call logs and other details. It also allows filtering by data sources, account types, and between dates (most of these filters are in the left side of the interface).

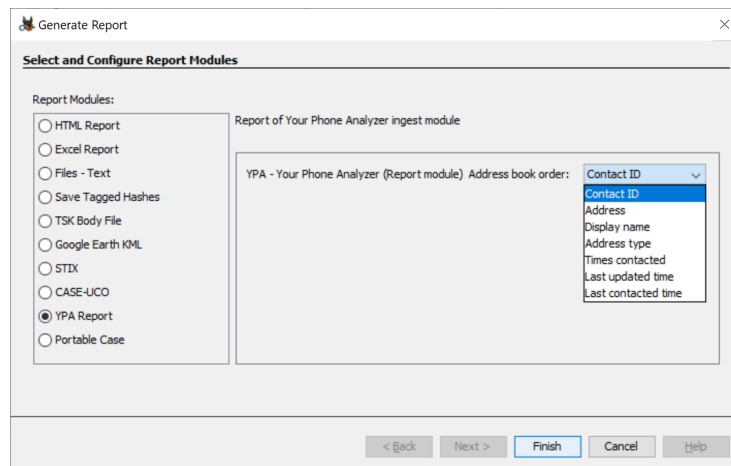


Figure 30: YPA's report module GUI.

4.2.3 Report module

Following the flow of a conversation can be a pain in Autopsy's interface, as it is a table of artifacts, which can be out of order or with other conversations inbetween. The report module of YPA's primary objective was to give the user a better interface to follow these conversations, while also allowing the user to export Your Phone data to a format that can be interpreted in most devices. The report module then evolved to present more data to the practitioner.

The report module can also handle multiple Windows 10 users, usually showing to which user a certain conversation or table row belongs to.

The dates shown in the report module are all in the UTC+0 timezone and formatted with ISO 8601, unless stated otherwise.

The GUI for YPA's report module can be seen in Figure 30. It only has an option to choose the order that the practitioner wants for the address book. A label is shown in the GUI if the case has no artifacts created by YPA, as the report module requires the ingest to be ran, as it uses the artifacts created by the ingest.

Running the report module generates five HTML files from the templates - one for each page of the report: *i) Conversations*, *ii) Address book*, *iii) Photos*, *iv) Call history*, and *v) Apps*. The initial page, *Conversations*, can be seen in Figure 31. The templates contain the skeleton of a page, however the rest of the HTML content is generated by the module, using BeautifulSoup¹⁵.

¹⁵ <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

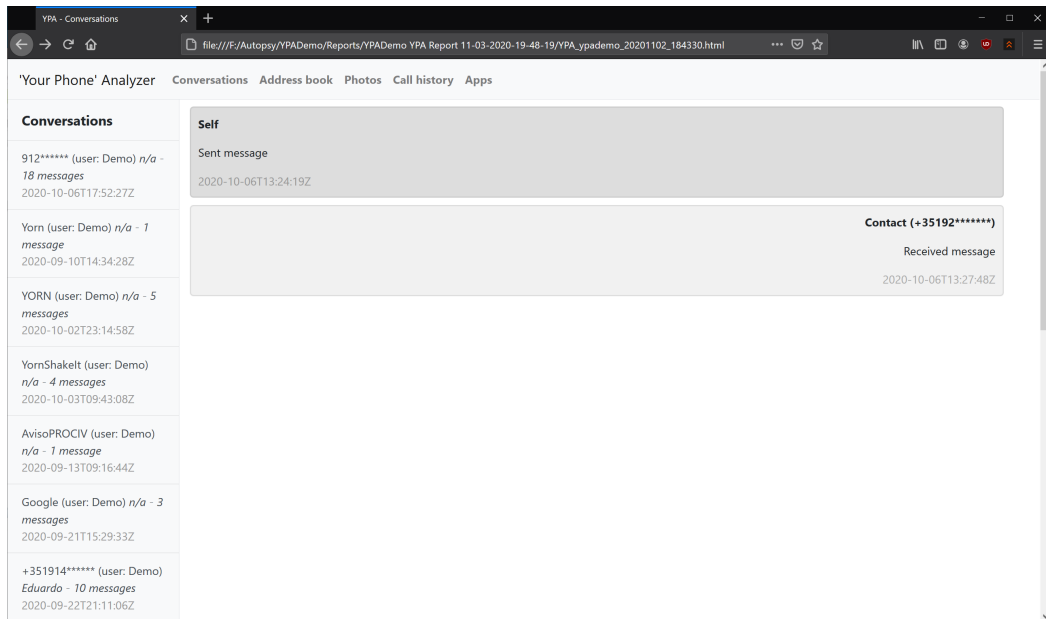


Figure 31: YPA's report module - *Conversations* page.

At the top of Figure 31 is the navigation bar, where the practitioner can choose a page to display. On the left side is the side bar with all conversations. Each conversation option has the phone number, Windows 10 user, contact display name (if the contact is saved, *n/a* if it is not saved), number of exchanged messages and the date of the last sent or received message. Messages sent have a darker gray background, while received messages have a lighter gray.

Clicking on a contact's message will display a modal with the contact's information. If the message is not from a contact, it will do nothing. The modal can be seen in Figure 32.

The *Address book* page shows the contacts found in Your Phone databases. A thing that the report module does over the ingest here is that it extracts the thumbnail blobs from the contacts database, giving the practitioner an automated way to see these images. In case the contact does not have a thumbnail, a default image is shown. Figure 33 shows a contact with a default image, and another with an image from the contact's thumbnail. The thumbnail is usually a small resolution (e.g. 320×240), and due to that there is no click for fullscreen functionality.

For the *Photos* page, YPA also extracts the media from the `photos.db` database. If the media is null, it checks if the media thumbnail is also null. If both the media and thumbnail are null, it shows the same default image as the *Address book* page. Other than the image, it shows some other information about the photo, present in the photo artifact. Figure 34 shows the *Photos* page. Clicking in a photo opens

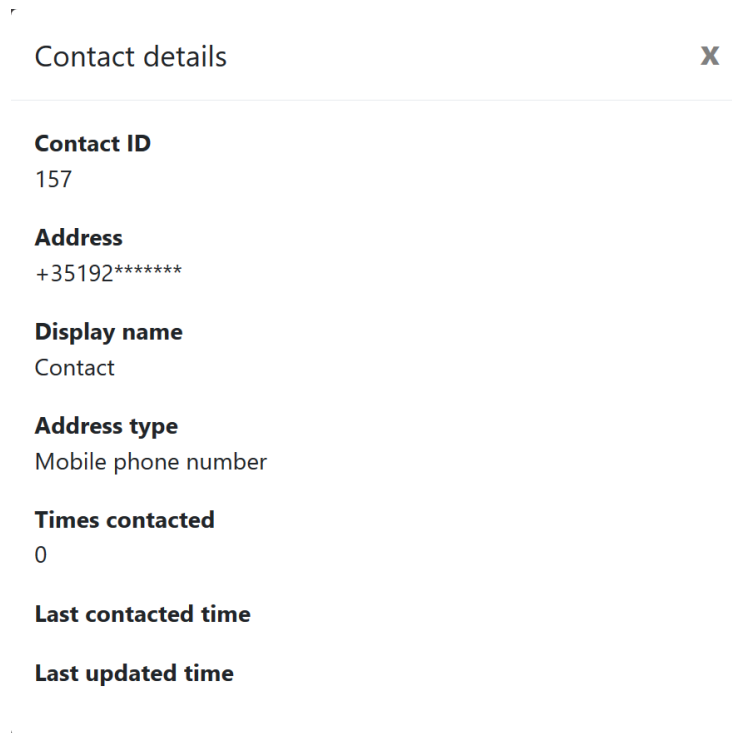


Figure 32: YPA's report contact modal.

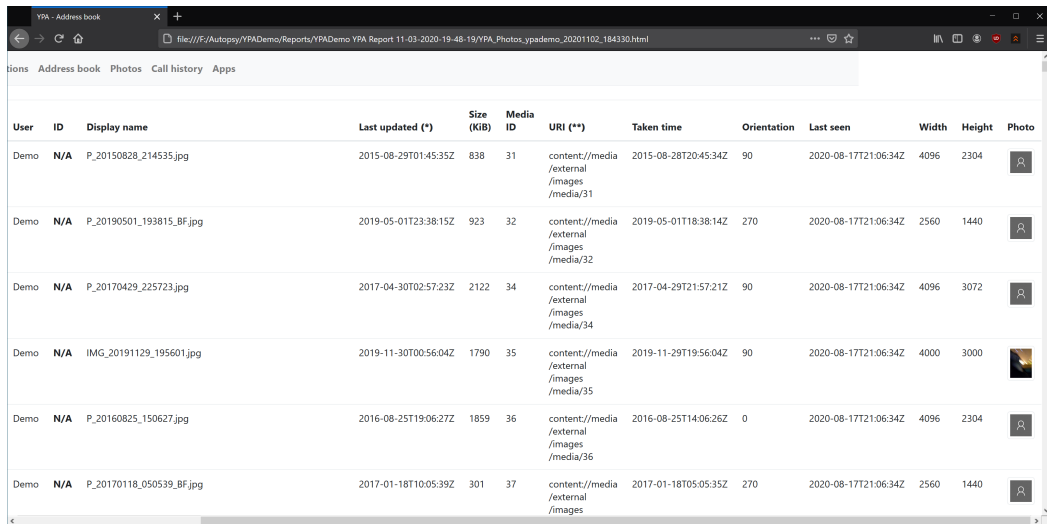
User	ID	Address	Display name	Address type	# contacted (*)	Last updated (**)	Thumbnail
Demo	1	+351244****	Contact 1	Home phone number	0	2020-06-15T18:05:34Z	
Demo	169	+351915****	Contact 2	Mobile phone number	0	2020-06-19T12:57:52Z	

(*) Might not be accurate (progresses by blocks of 10 for values larger than 10) and includes contact done via WhatsApp and other instant messaging apps. More recent versions do not have this attribute anymore (0).

(**) Affected by updates of the smartphone SMS application.

Figure 33: YPA's report module - Address book page.

YOUR PHONE









User	ID	Display name	Last updated (*)	Size (KiB)	Media ID	URI (**)	Taken time	Orientation	Last seen	Width	Height	Photo
Demo	N/A	P_20150828_214535.jpg	2015-08-29T01:45:35Z	838	31	content://media/external/images/media/31	2015-08-28T20:45:34Z	90	2020-08-17T21:06:34Z	4096	2304	
Demo	N/A	P_20190501_193815_BF.jpg	2019-05-01T23:38:15Z	923	32	content://media/external/images/media/32	2019-05-01T18:38:14Z	270	2020-08-17T21:06:34Z	2560	1440	
Demo	N/A	P_20170429_225723.jpg	2017-04-30T02:57:23Z	2122	34	content://media/external/images/media/34	2017-04-29T21:57:21Z	90	2020-08-17T21:06:34Z	4096	3072	
Demo	N/A	IMG_20191129_195601.jpg	2019-11-30T00:56:04Z	1790	35	content://media/external/images/media/35	2019-11-29T19:56:04Z	90	2020-08-17T21:06:34Z	4000	3000	
Demo	N/A	P_20160825_150627.jpg	2016-08-25T19:06:27Z	1859	36	content://media/external/images/media/36	2016-08-25T14:06:26Z	0	2020-08-17T21:06:34Z	4096	2304	
Demo	N/A	P_20170118_050539_BF.jpg	2017-01-18T10:05:39Z	301	37	content://media/external/images	2017-01-18T05:05:35Z	270	2020-08-17T21:06:34Z	2560	1440	

Figure 34: YPA's report module - *Photos* page.

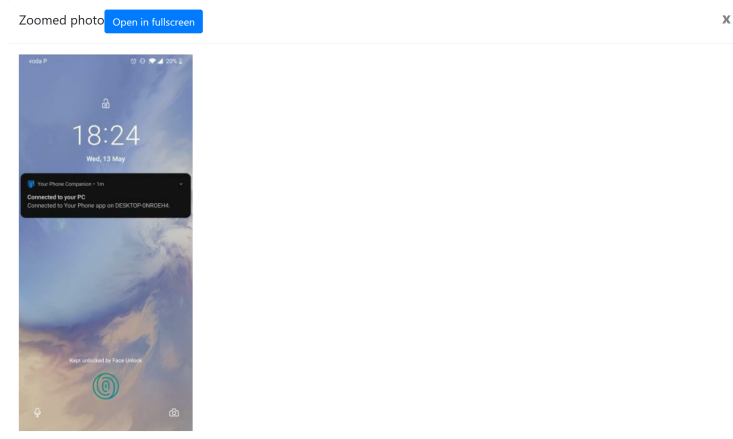


Figure 35: YPA's report module - *Photos's* modal.

it in a modal, which can be seen in Figure 35. This modal also gives the option to see the photo in fullscreen.

The remaining pages, *Call history* and *Apps* are shown in Figures 36 and 37 respectively. *Apps' Play Store Link* is created from each app's package name and the URL prefix of the Play Store.

User	ID	Display name	Address	Duration (s)	Call type	Start time (UTC)	Last updated (UTC)	Is read
Demo	43	Contact 1	914*****	39	Incoming	2020-09-17T14:02:24Z	2020-09-21T09:05:50Z	Taken
Demo	44	Contact 2	938*****	69	Incoming	2020-09-22T14:08:09Z	2020-09-27T19:01:18Z	Taken
Demo	45	Contact 3	938*****	0	Declined	2020-09-23T11:06:07Z	2020-09-27T19:01:18Z	Taken
Demo	46	Contact 4	919*****	880	Incoming	2020-10-04T18:22:47Z	2020-10-04T18:37:31Z	Taken
Demo	47	Unknown	938*****	54	Incoming	2020-10-06T18:02:32Z	2020-10-06T18:03:29Z	Taken

Figure 36: YPA's report module - *Call history* page.

User	Display name	Package name	Version	Play Store Link	Notifications
Demo	Chrome	com.android.chrome	85.0.4183.127	Link	
Demo	Phone	com.android.dialer	4.6.1	Link	
Demo	Settings	com.android.settings	10	Link	
Demo	Maps	com.google.android.apps.maps	10.51.1	Link	
Demo	Duo	com.google.android.apps.tachyon	106.0.332327649.DR106_RC00	Link	
Demo	Calendar	com.google.android.calendar	2020.38.3-334096848-release	Link	

Figure 37: YPA's report module - *Apps* page.

WINDOWS NOTIFICATIONS

Microsoft [WNS](#) were introduced in Windows 8, allowing developers to send toast, tile, badge, and raw updates from their own cloud service to a Windows application (Microsoft Docs, [2015](#)). These services will be explored in the following sections, along with the relevant forensic findings of any user data left behind. Software was developed to analyze the data, a standalone script and an Autopsy module. This software was then integrated with [YPA](#), due to Your Phone sending these notifications. In fact, as the Your Phone [UWP](#) application makes a massive use of Windows notifications, as it mirrors Android notifications, this chapter references whenever appropriate the Your Phone application.

5.1 [WNS](#) AND ACTION CENTER

[WNS](#) are comprised of two services: *i*) a system service, referred as *Windows Push Notification System Service*, and *ii*) a user service known as *Windows Push Notification User Service*. Both these services can be queried in the Windows 10 *Services* interface, where they also have a small description.

The description in Windows' *Services* for the system service is: “This service runs in session 0 and hosts the notification platform and connection provider which handles the connection between the device and [WNS](#) server”. The system service handles notifications related to the Windows [OS](#), such as Windows Defender, Windows updates, and Bluetooth device pairing.

For the user service, the description is “This service hosts Windows notification platform which provides support for local and push notifications. Supported notifications are tile, toast and raw.”.

Since Microsoft Docs ([2015](#)), [WNS](#) have joined the Windows 10 [UWP](#) ecosystem (Microsoft Docs, [2020c](#)). In fact, based on the documentation these services have not changed much - from the 2015 Windows 8 documentation to the 2020 [UWP](#) Windows 10, there are some differences in the wording of how [WNS](#) works, but

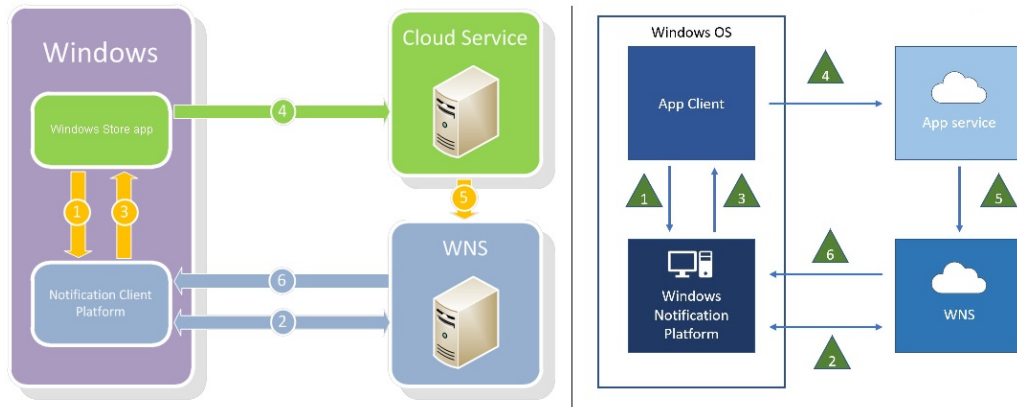


Figure 38: WNS' diagrams of the data flow of a push notification. Left Microsoft Docs (2015), right Microsoft Docs (2020c).

the overall idea is the same. Even the diagrams of both documentations are very similar, as shown in Figure 38.

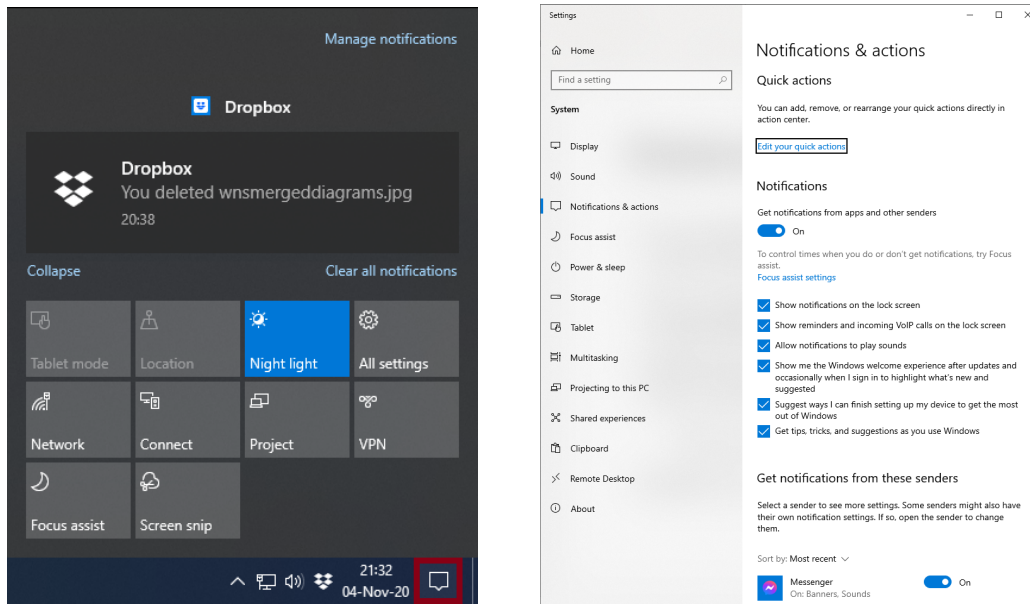
Since the steps are almost the same, only the most recent descriptions will be specified below (Microsoft Docs, 2020c):

1. App requests a push notification channel from WNS.
2. Windows asks WNS to create the channel. This channel is returned to the calling device in a form of a Uniform Resource Identifier (URI).
3. Notification channel URI is returned to the app by WNS.
4. The app sends the URI to a cloud service. The URI is stored in the service, so that it can be accessed when sending notifications. The URI serves as an interface between the app and the cloud service, and it is the developer's responsibility to implement the interface safely and with security standards.
5. When the cloud service has an update to send, it notifies WNS using the channel URI, via an HTTP POST request with the notification payload, over Secure Sockets Layer (SSL). This step requires authentication.
6. WNS receives the request and routes the notification to the appropriate device.

Notifications can also be sent locally, via a *NuGet* package for UWP apps (Microsoft Docs, 2020a), and there is even a Python package to send notifications¹.

These notifications are shown to the user in the Action Center, which was brought from Windows Phone back in 2014 to a pre-release version of Windows 10 (Fingas, 2014). Figure 39a shows the Action Center. To open it, a user has to click the icon

¹ <https://github.com/jithurjacob/Windows-10-Toast-Notifications>



(a) Windows 10's Action Center.

(b) Windows 10 notification settings.

Figure 39: Windows 10 Action Center and notification settings.

(outlined with a red square in the Figure). From there, the user can see his/her current notifications, and has access to some quick actions, such as *Night light*, or *VPN*. The user can also click, on the top right, *Manage notifications* to be redirected to a settings panel where the user can choose which apps are allowed to send notifications, edit the quick actions, and other notification settings. The settings screen can be seen in Figure 39b.

There are three types of notifications: *i*) badges (Figure 40a), *ii*) tiles (Figure 40b, and *iii*) toasts (Figure 40c). Badges can be used with tiles and toasts, the latter can be seen in Figure 40c. Badges can have numeric values between 1 and 99. Above 99 is represented by 99+.

Toasts are notifications that pop up from the right bottom corner of the screen. The name stems from the similarity of toast popping out of a toaster. There is a wide variety of toasts, some interactive, such as *Quick reply text box* (seen in Your Phone message notifications), *Progress bar*, and *Context menu actions*, among others (Microsoft Docs, 2020b). Figure 41a shows a toast with a reply box, Figure 41b shows a toast with a progress bar, and Figure 41c shows a toast with button icons.

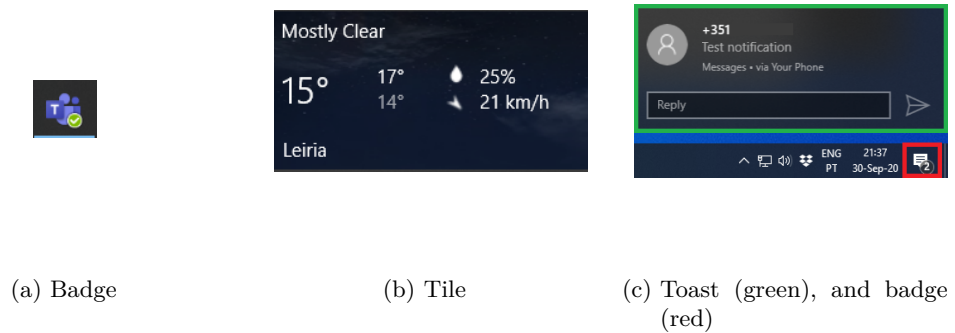


Figure 40: Types of Windows 10 notifications.

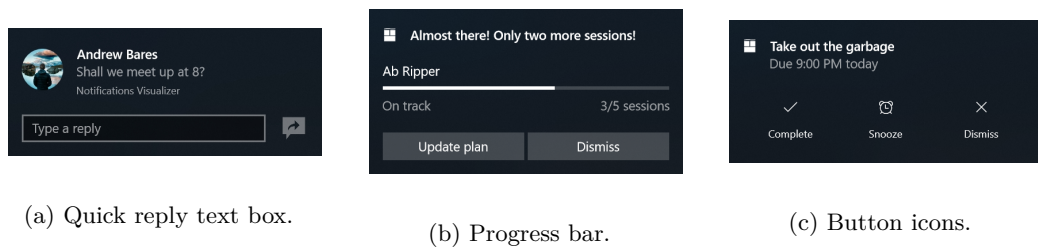


Figure 41: Different Windows 10 notification toasts. Source: Microsoft Docs (2020b).

5.1.1 User data

Windows 10 resorts to SQLite3 databases to handle the notifications and configurations of the Action Center - `wpnatabase.db`. For each service there is one database. These databases are located in the `%LOCALAPPDATA%\Microsoft\Windows\Notifications` directory for each Windows 10 user, and `C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Windows\Notifications` for the system service. Just like Your Phone’s databases, these databases have WAL enabled, and the WAL file can be found in the same directory of each database. The databases are identical, and have eight tables, listed in Table 17.

The tables from `wpnatabase.db` revolve around notification handlers. The table for handlers is `NotificationHandler`. Some of the other tables have a handler’s attributes, and are linked to a handler by the `HandlerId`.

`HandlerAssets` contains the handler’s assets, such as *Display name*, in a key-value format (`AssetKey` and `AssetValue` columns). These assets are not mandatory. In a Your Phone enabled system, there is a handler per app in the user’s phone, and each of these handlers has three `AssetKeys`: *i*) `IconUri`, *ii*) `DisplayName`, and *iii*) `LaunchArgs`.

Table 17: WNS' wpndatabase.db tables

NAME	DESCRIPTION
HandlerAssets	Notification handler's assets
HandlerSettings	Notification handler's settings
Metadata	Database metadata and WNS settings
Notification	Handler's notifications
NotificationData	Extra Notification data
NotificationHandler	Notification handlers
TransientTable	Unknown
WNSPushChannel	Applications WNS URIs

All `IconUri` have the same value: `ms-appx:///Assets/AppTiles/AppIcon.png` - this is probably the Your Phone icon, which shows in all Windows 10's Your Phone notifications in the Action Center. The `DisplayName` is the app's name in the phone, and finally `LaunchArgs` seems to be a sort of deeplink to open the Your Phone app in a certain page. Some examples are `ms-phone://phonenotifications/`, `ms-phone://calling/`, and `ms-phone://messages/`. When a user clicks a notification with this deeplink, he is redirected to Your Phone with a specific page selected - the first should open Your Phone in the *Notifications* page, the second in *Calls*, and the third in *Messages*. `IconUri` and `LaunchArgs` do not seem to have much forensic value as they are more for display and usability, but `DisplayName` allows to easily link a notification to a phone app.

`HandlerSettings` contains the handlers' settings, also in a key-value format (`SettingKey` and `Value`) - the value seems to have only two possible values, 0 or 1. These do not seem to have much value.

`Notification` consists of the handlers' notifications. A notification has a `Payload`, which might have several types since it also has a `PayloadType`, but all observed notifications have this set as `Xml`, which results in a `Payload` with an XML string. Other notable fields of this table are `ArrivalTime`, and `ExpiryTime`, which are both 64-bit filetimes. The former is when the notification arrives, and the latter is when the notification expires, which is up to three days of the arrival, but can change according to the app developer. `Type` is the notification type, from `tile`, `toast`, `badge`, or `toastCondensed`. `NotificationData` can contain extra notification information, in a key-value format (`Key` and `Value` fields), however this table never seemed to be populated in the observations.

Table 18: Notification handler from [WNS](#)

COLUMN	TYPE	VALUE
<code>RecordId</code>	integer	159
<code>PrimaryId</code>	text	Microsoft.YourPhone_8wekyb3d8bbwe! YourPhoneNotifications_com.whatsapp
<code>WNSId</code>	text	(empty)
<code>HandlerType</code>	text	app:immersive
<code>WNFEventName</code>	int64	NULL
<code>SystemDataPropertySet</code>	blob	NULL
<code>CreatedTime</code>	datetime	2020-08-18 20:57:40
<code>ModifiedTime</code>	datetime	2020-08-18 20:57:40
<code>ParentId</code>	text	Microsoft.YourPhone_8wekyb3d8bbwe! App
<code>ContainerSid</code>	text	NULL

When a notification is removed from the Action Center, it is removed from the `Notification` table. Consequently, clearing all of the Action Center’s notification clears the whole table. Just like with Your Phone, these records might be recoverable using data recovery tools.

An example of a notification handler can be seen in Table 18. In this case, it is a Your Phone handler, from the app WhatsApp. As can be seen by the Table, the `PrimaryId` is generated from Your Phone’s prefix (`Microsoft.YourPhone_8wekyb3d8bbwe!`) and the app’s package name (`com.whatsapp`, separated by a `_`). The field `WNSId` is always empty for Your Phone handlers, as it does not use a cloud service for notifications. The `ParentId` seems to follow similar logic to `PrimaryId`, Your Phone’s prefix plus `App`, indicating that the parent of this handler is the Your Phone app.

The table `TransientTable` seems to be a caching mechanism for notifications that are sent by a cloud service, but there were no records observed from this table.

Lastly, table `WNSPushChannel` contains information about which apps can send notifications from their cloud service using a [URI](#), as previously explored.

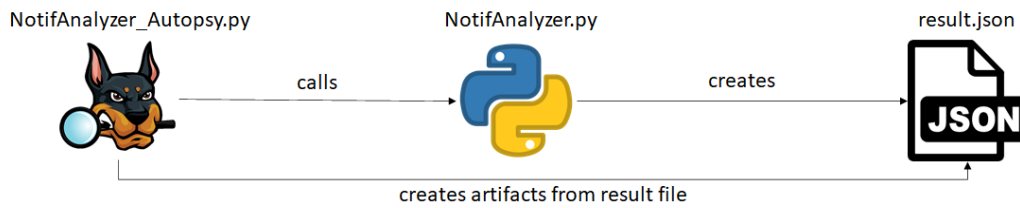


Figure 42: NotifAnalyzer’s flow diagram

5.1.2 *NotifAnalyzer*

NotifAnalyzer is a Python 3 script and Autopsy module. It can be used as a standalone script, or in Autopsy as an ingest module. It is licensed under a [GPL 3.0](#) license, and hosted in the same repository as [YPA](#)².

Figure 42 shows a diagram of how NotifAnalyzer works when executed from Autopsy. When ran, the standalone script processes the `wpndatabase.dbs` producing as output a JSON file.

By having a standalone script and an Autopsy module that executes the standalone, it is possible to process the [WNS](#) databases without depending on Autopsy, and thus practitioners can use NotifAnalyzer without using Autopsy. This does however have other drawbacks, as the practitioner must have Python installed in the machine, and the script dependencies must be installed by the user manually, when compared to [YPA](#) which requires no setup other than the installation of Autopsy. However, Python is nowadays a must have interpreter in a forensic practitioner’s machine, and installing modules is straightforward with Python’s *pip* utility.

5.2 STANDALONE SCRIPT

The standalone Python 3 script receives the path to a `wpndatabase.db` and extracts the data from this SQLite database to a JSON file. Figure 43 shows the execution of this script. The JSON output file is formatted in a certain way, for easy readability and processing. It contains the `user_version` of the database, which like Your Phone databases is important since Microsoft uses it as version control, and `assets`. The `assets` are a list of the notification handlers with their most important fields, the handler’s assets (in a list), handler’s notifications (also in a list), and the `AppName` of the handler, which is only populated if the handler has an asset with the key

² <https://github.com/labcif/YPA/>

```

Command Prompt
C:\Users\luixa\AppData\Roaming\autopsy\python_modules\YPA>python NotifAnalyzer.py -p "C:\Users\luixa\Desktop\Databases\Demo\AppData\Local\Microsoft\Windows\Notifications\wpndatabase.db" -j "C:\Users\luixa\Desktop\output.json"
Elapsed time: 0.01s
C:\Users\luixa\AppData\Roaming\autopsy\python_modules\YPA>

```

Figure 43: NotifAnalyzer standalone script execution.

Listing 3: Output JSON example of NotifAnalyzer.

```

1 {
2   "user_version": 7,
3   "assets": {
4     "159": {
5       "HandlerId": 159,
6       "HandlerPrimaryId": "Microsoft.YourPhone_8wekyb3d8bbwe!YourPhoneNotifi
↵ fications_com.whatsapp",
7       "ParentId": "Microsoft.YourPhone_8wekyb3d8bbwe!App",
8       "WNSId": "",
9       "HandlerType": "app:immersive",
10      "WNFEventName": null,
11      "SystemDataPropertySet": null,
12      "CreatedTime": "2020-08-18 20:57:40",
13      "ModifiedTime": "2020-08-18 20:57:40",
14      "OtherAssets": [],
15      "Notifications": [],
16      "AppName": "WhatsApp"
17    }
18  }
19 }

```

DisplayName. Listing 3 shows an example of an output of NotifAnalyzer. In order to display all that data about a handler, the script queries `NotificationHandler`, `HandlerAssets`, and `Notification`.

The script also attempts to format the notification payload, if the type is `Xml`, in a fail-safe manner. It first tries to use `lxml`³ to format it. If this fails, it will fallback to adding a newline after every `>`, and in case that fails, it will finally fallback to the original value of the payload. This is done in order to aid whoever is parsing the output to easily display the notification payload.

As a side note, if the script is ran and Python's `win10toast` is installed, it will display a Windows 10 notification to the practitioner with the elapsed time - this is optional, and the package is not required for the script to function. The toast notification can be seen in Figure 44.

³ <https://lxml.de/>

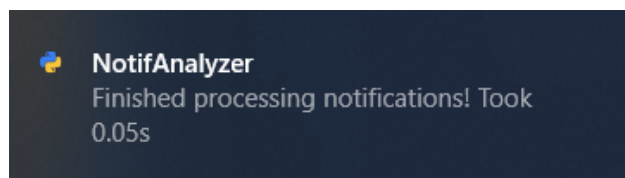


Figure 44: NotifAnalyzer standalone script toast.

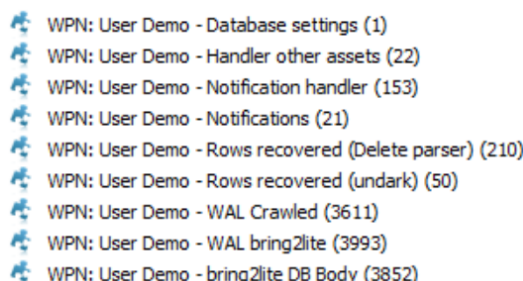


Figure 45: NotifAnalyzer's artifacts in Autopsy's interface.

5.3 AUTOPSY INGEST MODULE

The Autopsy module adds to the standalone script. As can be spotted in Figure 42, it runs the script and creates the artifacts from the output JSON file. The artifacts can be seen in Figure 45.

Since it is needed to run the script from outside of the Autopsy environment, it is required that the practitioner has Python installed. Figure 46 shows the GUI for NotifAnalyzer's ingest module. It has a text field to input the path to Python, and the same options as YPA for the data recovery tools. The artifacts produced from the result of the data recovery tools are the same as for YPA, but they have the WPN prefix, as is shown in Figure 45. The database version artifact contains the database `user_version` PRAGMA value.

The *Handler other assets* artifacts are the assets of a handler, except `DisplayName` which is already in the *Notification handler* artifact under *App name (Your Phone)*. The *Notification handler* artifact contains all the fields shown in the JSON in Listing 3. The *Notification* artifact itself is the most important one, as it contains the notification payload and the app that displayed it. It can also contain the phone app that displayed it, if the notification was from Your Phone. Table 19 shows an example of a notification artifact, in this case a message from WhatsApp, sent by Your Phone. The *Payload* was cut since it is quite long.

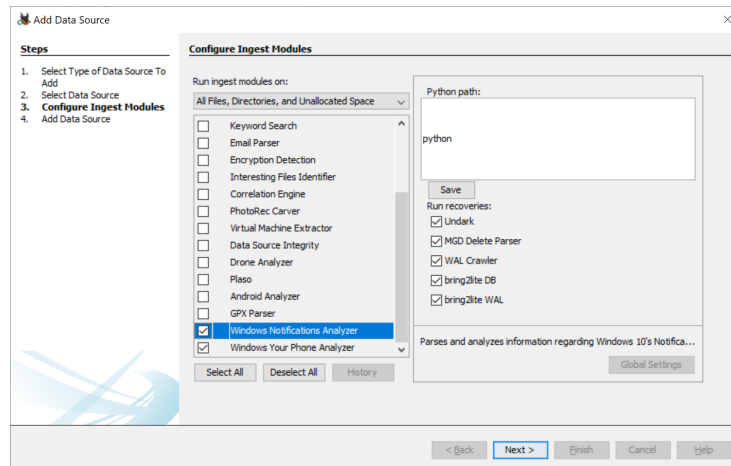


Figure 46: NotifAnalyzer’s ingest module GUI.

Table 19: WhatsApp notification artifact from NotifAnalyzer.

ATTRIBUTE	VALUE
App name	WhatsApp
Type	toast
Content format	Xml
Payload	<toastlaunch="ms-phone://phonenotifications/?dev(...)
Expiry time	2020-10-09 18:55:14 BST
Arrival time	2020-10-06 18:55:14 BST

The data recovery tools are very important for NotifAnalyzer, as [WNS](#)’ local database is even more prone to the deletion of data, and the amount of recovered data is much higher for notifications than Your Phone, as is evidenced by the difference in artifacts seen in [Figures 27](#) and [45](#), which are from the same computer using Your Phone and Windows notifications.

5.3.1 Integration with [YPA](#)

Since Your Phone can be very tied to Windows 10 notifications, a small integration between the two Autopsy modules was created. When running [YPA](#)’s report, notifications are tied to a phone app in the *Apps* page of the report. [Figure 47](#) shows the button in the *Apps* table (inside the red rectangle), and when the button is clicked, the notifications of that app appear in a modal, also shown in the [Figure](#). This is where the formatting of the payload has value, as it is much easier to read it when formatted. It also has an horizontal scrollbar.

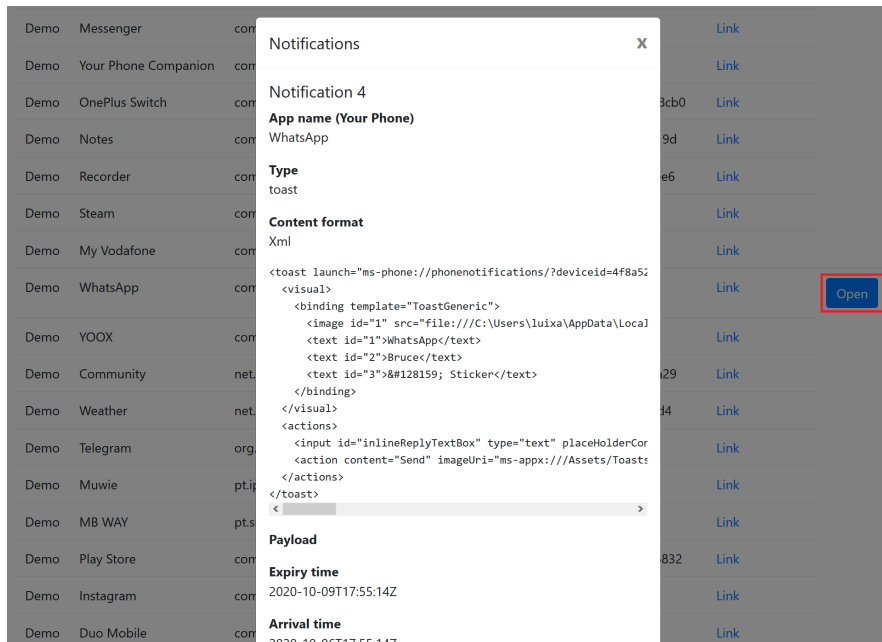


Figure 47: YPA’s integration with notifications from NotifAnalyzer.

5.3.2 Running in a digital forensics image

To test NotifAnalyzer on a normal environment, it was ran on a digital forensics image. James (2020)’s *Desktop Disk Image (E01)* image has a Windows 10 installation, with version 10.0.19041.1⁴. The test was performed in a Windows 10 desktop, with Autopsy 4.17.0, and all NotifAnalyzer’s data recovery tools enabled. Figure 48 shows the artifacts found by NotifAnalyzer.

As can be seen in Figure 48, this Windows installation contains several users: Admin, Administrator, mortysmith, and ricksanchez. The systemprofile user is the system’s Notification database, and is the only database that does not contain any active notifications, and that had a WAL file present. The user databases do not have any standout notification handler, in this case. Some of the notifications are weather tiles (from Washington D.C.), others are ads to a game and a notes application. While there seem to be no relevant notifications in this image, as it was an image set for digital forensic training purposes, therefore without real usage. In real cases, it is possible to see a vast amount of recovered data artifacts from bring2lite, and if any user in this image had a messaging app such as Facebook’s *Messenger*, it would probably be an important source of information, and the recovery tools could play a relevant part getting older notifications.

⁴ <https://thecollectionbook.info/windows/10-v2004/4762>

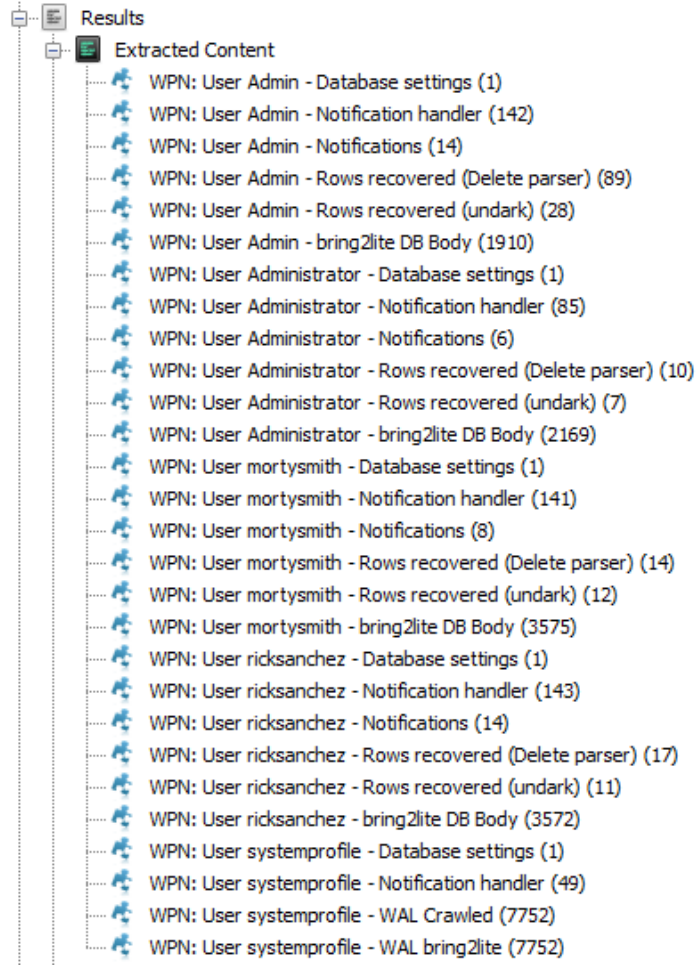


Figure 48: NotifAnalyzer artifacts ran on a forensic disk image.

CONCLUSIONS

SQLite has been gaining importance within the Microsoft software ecosystem. These databases are usually left in a user's computer, as it usually functions as a local data source, and Windows 10 leaves a lot of these behind, both from applications and services.

UWP is a platform inside Windows 10. It contains a plethora of applications, from Microsoft and third-party developers. Some of these applications come bundled with the **OS**, others can be obtained from the Microsoft Store. Updates for these applications are also from the Store.

Your Phone is an **UWP** application. It can be an important asset for digital forensics when a smartphone cannot be accessed, either because the phone data is inaccessible, or the phone is missing. The user's Your Phone data can be found in SQLite databases that the application leaves behind. These databases can have some meaningful data, such as SMS, call log, and MMS from the last 30 days, up to 2000 photos/screenshots, and the address book with all contacts. Your Phone can also be used by multiple users if a computer is shared, further increasing the possibility of valuable data. This dissertation goes in depth on how Your Phone works, including its Android counter-part, **YPC**. **YPA** is a module developed to extract and import into Autopsy the digital forensic artifacts of Your Phone. **YPA** can then create dynamic HTML reports to display the artifacts in a user-friendly manner. Since multiple Your Phone features keep data only for the last 30 days, having the ability to recover deleted records can extend this 30-day period. **YPA** uses several data recovery tools in order to accomplish this. The same occurs with the capability of recovering data from the unallocated space of the databases.

WNS are comprised of two services, one for the user and another for the system. They are responsible to deliver various types of notifications - badges, tiles, and toasts - to applications. **UWP** applications can easily integrate with these services, but it is not restricted to just these applications. The services rely on two identical databases, `wpnatabase.db`, which likewise are for the user and system. The value of these notifications depend heavily on the installed applications, and on the usage of the aforementioned applications. For instance, a messaging application

such as Facebook's [UWP Messenger](#) application might hold some relevant received messages. The `NotifAnalyzer.py` and associated Autopsy module analyze all Windows 10 `wpnatabase.db` found, parsing all notifications and notification handlers. The lifespan of the notifications is rather short, up to just three days. Just like with Your Phone, it can be very valuable to recover the records of these databases. The Autopsy module uses the same data recovery tools as [YPA](#) to overcome this. The limited scope and timespan of notifications are not expected to yield a large volume of meaningful forensic artifacts. Nonetheless, in situations where other more traditional sources of data are not available, or have failed to deliver useful data, notifications can still provide valuable artifacts.

These applications and services are in constant evolution, and as such it is necessary to keep updating the developed software. A solution was created for this, `UWPAppsScanner`, which additionally keeps a history of the databases and schemas found for the specified applications.

6.1 FUTURE WORK

Using the [UWP](#) scanner, it is possible to keep the developed software up-to-date. For future work, the software should be updated to account for any updates each application or service might receive that impact the functionality of the software. New features and database schemas should also be studied and added to the current solutions.

Some of these new functionalities from Your Phone support screen mirroring and using Android applications directly in Windows 10 when paired with Samsung devices¹. It is important to study how these interactions are done, and if it leaves any data on the Android app or the Windows 10 computer. This was not possible in this dissertation due to the lack of Samsung hardware.

The `NotifAnalyzer` Autopsy module is planned to have a report module of its own, other than the integration with [YPA](#).

Other than the continuous development and maintenance for the already developed software, additional scripts could be written to support other Windows 10 applications and services, such as *Timeline*. The work done on the SQLite data

¹ <https://twitter.com/AnalyMsft/status/1291551347435692033>, <https://www.onmsft.com/news/windows-10-your-phone-app-can-now-run-multiple-android-apps-simultaneously-on-select-devices>

recovery tools can be applied to all future Autopsy modules, and applications can be added to be tracked in the scanner with minimal setup.

BIBLIOGRAPHY

- Abiteboul, Serge, Richard Hull, and Victor Vianu (1995). *Foundations of Databases*. Addison-Wesley. ISBN: 0-201-53771-0. URL: <http://webdam.inria.fr/Alice/>.
- Andrade, Luís Miguel (2020). *UWP Apps Scanner*. Version 1.0. DOI: 10.5281/zenodo.4291091. URL: <https://doi.org/10.5281/zenodo.4291091>.
- Andrade, Luís Miguel et al. (2020). *Your Phone Analyzer for Autopsy and NotificationAnalyzer*. Version 1.1. DOI: 10.5281/zenodo.4291064. URL: <https://doi.org/10.5281/zenodo.4291064>.
- Android Developers (2019a). *Data and file storage overview*. [Online; accessed 3-November-2019]. URL: <https://developer.android.com/guide/topics/data/data-storage#db>.
- (2019b). *Room Persistence Library*. [Online; accessed 2-November-2019]. URL: <https://developer.android.com/topic/libraries/architecture/room>.
- (2020). *Bluetooth overview*. [Online; accessed 06-November-2020]. URL: <https://developer.android.com/guide/topics/connectivity/bluetooth>.
- Apple Developer Documentation (2019). *Core Data*. [Online; accessed 3-November-2019]. URL: <https://developer.apple.com/documentation/coredata>.
- Årnes, André (2017). *Digital forensics*. John Wiley & Sons.
- Autopsy (2015). *Python Autopsy Module Tutorial #1: The File Ingest Module*. [Online; accessed 17-November-2020]. URL: <https://www.autopsy.com/python-autopsy-module-tutorial-1-the-file-ingest-module/>.
- Basis Technology (2016). *Autopsy User Documentation: UI Layout*. [Online; accessed 29-December-2019]. URL: https://sleuthkit.org/autopsy/docs/user-docs/4.1/uilayout_page.html.
- (2017). *Autopsy: Module Development Overview*. [Online; accessed 09-November-2020]. URL: https://www.sleuthkit.org/autopsy/docs/api-docs/4.2/platform_page.html.
- (2018). *Autopsy 4.7 Includes Link Analysis, Database Viewers, Triage, and More*. [Online; accessed 29-December-2019]. URL: <https://www.autopsy.com/autopsy-4-7-includes-link-analysis-database-viewers-triage-and-more/>.

- Basis Technology (2019). *Autopsy User Documentation: Ingest Modules*. [Online; accessed 29-December-2019]. URL: https://sleuthkit.org/autopsy/docs/user-docs/4.13.0/ingest_page.html.
- Bulterman, Dick CA and Lloyd W Rutledge (2008). *SMIL 3.0: flexible multimedia for Web, mobile devices and daisy talking books*. Springer Publishing Company, Incorporated.
- Carrier, Brian (2018). *Messaging App Forensics with Autopsy*. [Online; accessed 29-December-2019]. URL: <http://www.osdfcon.org/presentations/2018/Brian-Carrier-Messaging-App-Forensics-with-Autopsy.pdf>.
- Casey, Eoghan, Alex Nelson, and Jessica Hyde (2019). “Standardization of file recovery classification and authentication”. In: *Digital Investigation* 31, p. 100873.
- Cash App (2019). *SQLDelight - Generates typesafe Kotlin APIs from SQL*. [Online; accessed 2-November-2019]. URL: <https://github.com/cashapp/sqldelight>.
- Chang, Fay et al. (2008). “Bigtable: A distributed storage system for structured data”. In: *ACM Transactions on Computer Systems (TOCS)* 26.2, p. 4.
- Chopade, Rupali and Vinod Kesharao Pachghare (2019). “Ten years of critical review on database forensics research”. In: *Digital Investigation* 29, pp. 180–197.
- Daniels, Paul L. (2020). *Undark - a SQLite deleted and corrupted data recovery tool*. Website (access on 2020-08-08). <http://pldaniels.com/undark/>.
- DeGrazia, Mari (2015). *SQLite-Deleted-Records-Parser: recovering deleted entries in SQLite database*. Website (access on 2020-08-08). <https://github.com/mdegrazia/SQLite-Deleted-Records-Parser>.
- Distefano, Alessandro, Gianluigi Me, and Francesco Pace (2010). “Android anti-forensics through a local paradigm”. In: *digital investigation* 7, S83–S94.
- Domingues, Patricio et al. (2019). “Digital forensic artifacts of the Your Phone application in Windows 10”. In: *Digital Investigation* 30, pp. 32–42. URL: <https://www.sciencedirect.com/science/article/pii/S1742287619301239>.
- Facebook (2019). *More Resources - Popular libraries*. [Online; accessed 3-November-2019]. URL: <https://facebook.github.io/react-native/docs/more-resources#popular-libraries>.
- Fingas, Jon (2014). *Windows 10 brings Windows Phone’s notification center to the desktop*. [Online; accessed 04-November-2020]. URL: <https://www.engadget.com/2014-10-21-windows-10-action-center.html>.
- Flutter (2019). *Flutter for Android developers - How do I access SQLite in Flutter*. [Online; accessed 3-November-2019]. URL: <https://flutter.dev/docs/get-started/flutter-for/android-devs#how-do-i-access-sqlite-in-flutter>.

- Horsman, Graeme, Alex Caithness, and Costas Katsavounidis (2019). "A Forensic Exploration of the Microsoft Windows 10 Timeline". In: *Journal of forensic sciences* 64.2, pp. 577–586.
- Ionic Documentation (2019). *Data Storage*. [Online; accessed 3-November-2019]. URL: <https://ionicframework.com/docs/building/storage>.
- Jacobs, Bart (2016). *What is the difference between CoreData and SQLite*. [Online; accessed 23-November-2019]. URL: <https://cocoacasts.com/what-is-the-difference-between-core-data-and-sqlite/>.
- James (2020). *Case 001 - The Stole Szechuan Sauce - DFIR Madness*. [Online; accessed 14-November-2020]. URL: <https://dfirmadness.com/the-stolen-szechuan-sauce/>.
- Kline, Kevin, Daniel Kline, and Brand Hunt (2004). *SQL in a nutshell: a desktop quick reference*. O'Reilly Media, Inc.
- Kreibich, Jay (2010). *Using SQLite*. " O'Reilly Media, Inc."
- Li, Yishan and Sathiamoorthy Manoharan (2013). "A performance comparison of SQL and NoSQL databases". In: *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, pp. 15–19.
- Meng, Christian and Harald Baier (2019). "bring2lite: A Structural Concept and Tool for Forensic Data Analysis and Recovery of Deleted SQLite Records". In: *Digital Investigation* 29, S31–S41. ISSN: 1742-2876. DOI: <https://doi.org/10.1016/j.diin.2019.04.017>. URL: <http://www.sciencedirect.com/science/article/pii/S1742287619301677>.
- Microsoft Docs (2015). *Windows Push Notification Services (WNS) overview (Windows Runtime apps)*. [Online; accessed 04-November-2020]. URL: [https://docs.microsoft.com/en-us/previous-versions/windows/apps/hh913756\(v=win.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/apps/hh913756(v=win.10)?redirectedfrom=MSDN).
- (2016). *SQLite Database Provider - EF Core*. [Online; accessed 16-November-2019]. URL: <https://docs.microsoft.com/en-us/ef/core/providers/sqlite/?tabs=dotnet-core-cli>.
- (2018a). *Use a SQLite database in a UWP app*. [Online; accessed 16-November-2019]. URL: <https://docs.microsoft.com/en-us/windows/uwp/data-access/sqlite-databases>.
- (2018b). *What's a Universal Windows Platform (UWP) app*. [Online; accessed 23-January-2020]. URL: <https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide>.

- Microsoft Docs (2018c). *Xamarin.Forms - Local databases*. [Online; accessed 3-November-2019]. URL: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/databases>.
- (2019a). *What's new in Windows 10, build 18362 - UWP apps*. [Online; accessed 21-November-2019]. URL: <https://docs.microsoft.com/en-us/windows/uwp/whats-new/windows-10-build-18362>.
- (2019b). *Windows 10 - Apps*. [Online; accessed 21-November-2019]. URL: <https://docs.microsoft.com/en-us/windows/application-management/apps-in-windows-10>.
- (2020a). *Send a local tile notification - UWP applications*. [Online; accessed 04-November-2020]. URL: <https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/sending-a-local-tile-notification>.
- (2020b). *Toast content - UWP applications*. [Online; accessed 10-November-2020]. URL: <https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/adaptive-interactive-toasts>.
- (2020c). *Windows Push Notification Services (WNS) overview - UWP applications*. [Online; accessed 04-November-2020]. URL: <https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/windows-push-notification-services--wns--overview>.
- Miller, Preston and Chapin Bryce (2019). *Learning Python for Forensics: Leverage the power of Python in forensic investigations*. Packt Publishing Ltd.
- NetMarketShare (2020a). *Operating system market share*. [Online; accessed 20-November-2020]. URL: <https://netmarketshare.com/operating-system-market-share.aspx?options=%7B%22filter%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Desktop%22%24Laptop%22%25%7D%7D%2C%22dateLabel%22%3A%22Trend%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22platform%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22platformsDesktop%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222019-11%22%2C%22dateEnd%22%3A%222020-10%22%2C%22segments%22%3A%22-1000%22%7D>.
- (2020b). *Operating system market share (mobile)*. [Online; accessed 20-November-2020]. URL: <https://netmarketshare.com/operating-system-market-share.aspx?options=%7B%22filter%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Mobile%22%25%7D%7D%2C%22dateLabel%22%3A%22Trend%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22platform%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22platformsMobile%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222019-11%22%2C%22dateEnd%22%3A%222020-10%22%2C%22segments%22%3A%22-1000%22%7D>.

- 7B%22share%22%3A-1%7D%2C%22id%22%3A%22platformsMobile%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222019-11%22%2C%22dateEnd%22%3A%222020-10%22%2C%22segments%22%3A%22-1000%22%7D.
- Olivier, Martin (2020). “Digital Forensics and the Big Data Deluge—Some Concerns Based on Ramsey Theory”. In: *IFIP International Conference on Digital Forensics*. Springer, pp. 3–23.
- Oracle (2019). *What is a database*. [Online; accessed 3-November-2019]. URL: <https://www.oracle.com/database/what-is-database.html>.
- Quick, Darren and Kim-Kwang Raymond Choo (2014). “Impacts of increasing volume of digital forensic data: A survey and future research challenges”. In: *Digital Investigation* 11.4, pp. 273–294.
- SQLite (2019a). *Appropriate uses for SQLite*. [Online; accessed 3-November-2019]. URL: <https://www.sqlite.org/whentouse.html>.
- (2019b). *File locking and concurrency in SQLite version 3*. [Online; accessed 23-January-2020]. URL: <https://www.sqlite.org/lockingv3.html#rollback>.
- (2020a). *About SQLite*. [Online; accessed 23-January-2019]. URL: <https://www.sqlite.org/about.html>.
- (2020b). *Atomic commit in SQLite*. [Online; accessed 23-January-2020]. URL: <https://www.sqlite.org/uri.html>.
- (2020c). *CREATE VIRTUAL TABLE*. [Online; accessed 17-November-2020]. URL: https://www.sqlite.org/lang_createvtab.html.
- (2020d). *FTS3, FTS4, and FTS5 extensions*. [Online; accessed 28-October-2020]. URLs: <https://www.sqlite.org/fts3.html> and <https://sqlite.org/fts5.html>.
- (2020e). *Most widely deployed SQL database engine*. [Online; accessed 23-November-2019]. URL: <https://www.sqlite.org/mostdeployed.html>.
- (2020f). *PRAGMA statements*. [Online; accessed 23-November-2019]. URL: <https://www.sqlite.org/pragma.html>.
- (2020g). *Temporary files used by SQLite*. [Online; accessed 10-November-2020]. URL: <https://www.sqlite.org/tempfiles.html>.
- (2020h). *Uniform Resource Identifiers*. [Online; accessed 23-January-2020]. URL: <https://www.sqlite.org/uri.html>.
- (2020i). *Write-Ahead Logging*. [Online; accessed 23-November-2019]. URL: <https://www.sqlite.org/wal.html>.
- StatCounter GlobalStats (2020a). *Desktop Operating System Market Share Worldwide*. [Online; accessed 20-November-2020]. URL: <https://gs.statcounter.com/os-market-share/desktop/worldwide>.

- StatCounter GlobalStats (2020b). *Mobile Operating System Market Share Worldwide*. [Online; accessed 20-November-2020]. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- The Sleuth Kit (2019). *The Sleuth Kit (TSK) & Autopsy: Open Source Digital Forensics Tools*. [Online; accessed 23-November-2019]. URL: <https://www.sleuthkit.org/>.
- Vaish, Gaurav (2013). *Getting started with NoSQL*. Packt Publishing Ltd.
- Wikipedia contributors (2019). *Bigtable — Wikipedia, The Free Encyclopedia*. [Online; accessed 2-November-2019]. URL: <https://en.wikipedia.org/w/index.php?title=Bigtable&oldid=921053442>.
- Windows support (2018). *Get help with timeline*. [Online; accessed 21-November-2019]. URL: <https://support.microsoft.com/en-us/help/4230676/windows-10-get-help-with-timeline>.

APPENDICES

APPENDIX A

In this appendix are some JSON examples. Listing 4 shows the database schema of the UWP scanner server database, and Listing 5 shows a WhatsApp notification's JSON.

Listing 4: Database schema example.

```
1 {
2   "apps" : {
3     "App1" : {
4       "dbs" : {
5         "Database1" : {
6           "tables" : [ "Table1", "Table2" ],
7           "user_version" : 214
8         }
9       },
10      "exe" : "AppExecutable.exe",
11      "file_count" : 10,
12      "history" : {
13        "Entry1" : {
14          "app_version" : [ 2019, 19081, 22010, 0 ],
15          "dbs" : { ... },
16          "updated_at" : "2020-01-09 22:40:08.915149",
17          "user" : "Luis",
18          "windows_ver" : "Windows-10-10.0.18362-SP0"
19        }
20      },
21      "path" : "AppPath",
22      "updated_by" : {
23        "app_version" : [ 2020, 19081, 28230, 0 ],
24        "dbs" : { ... },
25        "updated_at" : "2020-02-16 04:26:00.367377",
26        "user" : "Luis",
27        "windows_ver" : "Windows-10-10.0.18362-SP0"
28      },
29      "version" : [ 2020, 19081, 28230, 0 ]
30    }
31  },
32  "schema_version" : 4
33 }
```

Listing 5: Example JSON of WhatsApp notification.

```
1 {
2   "id":1,
3   "key":"0|com.whatsapp|1|3fu8LLvJtKLhn8VIWAQJ8bOKh+RBjudquZ/c4+D7JQY=\n|10250",
4   "groupKey":"0|com.whatsapp|g:group_key_messages",
5   "tag":"3fu8LLvJtKLhn8VIWAQJ8bOKh+RBjudquZ/c4+D7JQY=\n",
6   "packageName":"com.whatsapp",
7   "appName":"WhatsApp",
8   "isClearable":true,
9   "isGroup":true,
10  "isOngoing":false,
11  "featureFlags":3,
12  "platform":0,
13  "version":"29",
14  "flags":8,
15  "eventCount":6,
16  "priority":0,
17  "postTime":1604011916905,
18  "timestamp":1604011908000,
19  "notificationClass":0,
20  "notificationActions":[
21    {
22      "actionName":"Reply",
23      "isActionInlineReply":true,
24      "actionIndex":0
25    },
26    {
27      "actionName":"Mark as read",
28      "isActionInlineReply":false,
29      "actionIndex":1
30    }
31  ],
32  "template":"android.app.Notification$MessagingStyle",
33  "text":"Sticker",
34  "title":"Luis ZeC",
35  "showWhen":true,
36  "messages":[
37    "Sticker",
38    "Sticker",
39    "Sticker",
40    "Sticker",
41    "Sticker",
42    "Sticker"
43  ],
44  "senderNames":[
45    "Luis ZeC",
46    "Luis ZeC",
47    "Luis ZeC",
48    "Luis ZeC",
49    "Luis ZeC",
50    "Luis ZeC"
51  ],
52  "importance":2
53 }
```


B

APPENDIX B

In this Appendix are the diagrams of the local databases found in a user's [YPC](#).

room_master_table [table]	
id	INTEGER
identity_hash	TEXT

content_view [table]	
content_type	INTEGER NOT NULL
id	INTEGER NOT NULL
created_seq_no	INTEGER NOT NULL
modified_seq_no	INTEGER NOT NULL
checksum	INTEGER

android_metadata [table]	
locale	TEXT

generated by SchemaCrawler 16.8.1
generated on 2020-06-14 00:01:15

Figure 49: YPC's content.db.

room_master_table [table]	
id	INTEGER
identity_hash	TEXT

FcmNotificationEvent [table]	
uid	INTEGER NOT NULL auto-incremented
"timestamp"	INTEGER NOT NULL
fcm_message_id	TEXT
original_priority	INTEGER
received_priority	INTEGER
app_standby_bucket	INTEGER
is_doze_mode_active	INTEGER

content_access_event [table]	
uid	INTEGER NOT NULL auto-incremented
start_time	INTEGER NOT NULL
duration	INTEGER NOT NULL
content_type	INTEGER
access_was_useful	INTEGER NOT NULL

android_metadata [table]	
locale	TEXT

agent_service_event [table]	
uid	INTEGER NOT NULL auto-incremented
"timestamp"	INTEGER NOT NULL
agent_service_state	INTEGER NOT NULL
event_id	INTEGER NOT NULL
details_json	TEXT
instance_id	TEXT NOT NULL
version	INTEGER NOT NULL

generated by SchemaCrawler 16.8.1	
generated on 2020-06-14 00:03:40	

Figure 50: YPC's eventstore.

settings [table]	
setting_group_id	TEXT NOT NULL
setting_key	TEXT NOT NULL
setting_type	INTEGER NOT NULL
setting_value	TEXT NOT NULL

room_master_table [table]	
id	INTEGER
identity_hash	TEXT

android_metadata [table]	
locale	TEXT

generated by SchemaCrawler 16.8.1	
generated on 2020-06-14 00:01:54	

Figure 51: YPC's YourPhoneSettings.

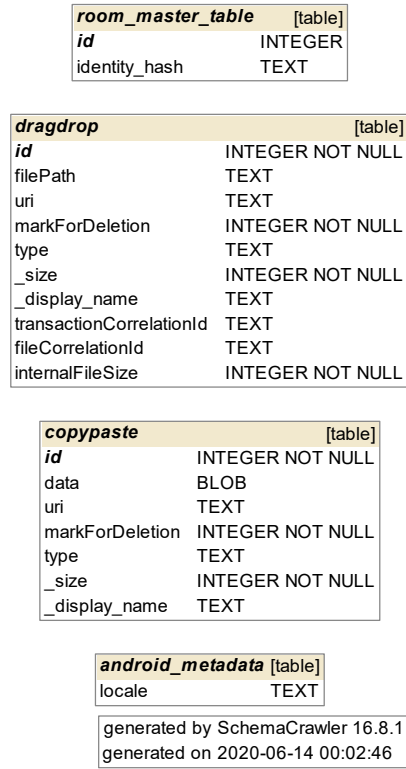


Figure 52: YPC's ContentTransferDatabase.

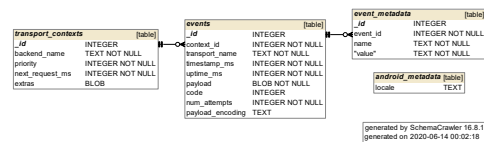


Figure 53: YPC's com.google.android.datatransport.events.

logs [table]	
oid	INTEGER auto-incremented
priority	INTEGER
target_token	TEXT
log	TEXT
type	TEXT
target_key	TEXT
persistence_group	TEXT

android_metadata [table]	
locale	TEXT

generated by SchemaCrawler 16.8.1
generated on 2020-06-14 00:03:06

Figure 54: YPC's com.microsoft.appcenter.persistence.

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *“Digital Forensic Artifacts of SQLite-based Windows 10 Applications”*, é original e foi realizado por Luís Miguel António Andrade (2180234) sob orientação de Professor Patrício Rodrigues Domingues e Professor Miguel Monteiro de Sousa Frade.

Leiria, Novembro de 2020



Luís Miguel António Andrade