Mini BlocklyChain
By OpenQbit

Installation, configuration & administration.

# User Handbook

version 1.0 Beta

July 2020.

# Content

# 1. Introduction.

The blockchain is generally associated with Bitcoin and other crypto currencies, but these are just the tip of the iceberg since it is not only used for digital money, but can be used for any information that may have a value for users and/or companies. This technology, which has its origins in 1991, when Stuart Haber and W. Scott Stornetta described the first work on a chain of cryptographically secured blocks, was not noticed until 2008, when it became popular with the arrival of bitcoin. But currently its use is being demanded in other commercial applications and is projected to grow in the medium future in several markets, such as financial institutions or the Internet of Things IoT among other sectors.

The blockchain, better known by the term blockchain, is a single, agreed upon record distributed over several nodes (electronic devices such as PCs, smartpohones, tablets, etc.) in a network. In the case of crypto-currencies, we can think of it as the accounting book where each of the transactions is recorded.

Its operation can be complex to understand if we go into the internal details of its implementation, but the basic idea is simple to follow.

It is stored in each block:

1.- a number of valid records or transactions,

2.- information concerning that block,

3.- its link with the previous block and the next block through the hash of each block –un unique code that would be like the fingerprint of the block.

Therefore, **each block** has a **specific and unmovable place within the chain,** as each block contains information from the hash of the previous block. The entire chain is stored on each network node that makes up the blockchain, so **an exact copy of the chain is stored on all network participants**.

As new records are created, they are first checked and validated by the network nodes and then added to a new block that is linked to the chain.

Now imagine that this network of devices that communicate among themselves, has the ability to interact without the intervention of a person, ie the great advantage of the blockchain is that it can make autonomous decisions, which benefits in response times of service to users, available 24 hours a day, minimizes costs in business and first of all has a level of security already tested, for this and other reasons has become so popular in use in different public and private sectors.

## 2. What is a public or private network in the blockchain scheme?

Public network. - It is a network of computers or mobile devices that communicate with each other and maintain anonymity, it is not known who interacts in this network in a formal way in their transactions, in this type of network any person or company can interact and connect at any time because you do not need permissions to connect, an example is the blockchain of Bitcoin, anyone can enter to buy or sell. Normally this type of networks are oriented or directed to the purchase and sale of digital monetary assets or its synonym "cryptomonies", examples: DogCoin, Ethereum, LiteCoin, BitCoin, Waves, etc.

Private network. - It is a network of computers or mobile devices that communicate with each other. However, unlike public networks, private networks require prior authorization from some entity (company or person) in order to connect and be part of this type of network. Normally, private blockchain networks are used in companies or corporations to carry out transactions or operations with a variety of different types of information that can have a tangible value in the form of documents, processes, authorizations and/or business decisions applied and supervised by blockchain, examples: financial sector, insurance sector, government, among others.

## 3. What is Blockly programming?

Blockly is a visual programming language composed of a simple set of commands that we can combine as if they were the pieces of a puzzle. It is a very useful tool for those who want to learn how to program in an intuitive and simple way or for those who already know how to program and want to see the potential of this type of programming.

Blockly is a form of programming where you don't need any background in any kind of computer language, this is because it is just joining graphic blocks as if we were playing lego or a puzzle, you just need to have some logic and that's it!

Anyone can create programs for mobile phones (smartphones) without messing with those programming languages difficult to understand, just put together blocks in a graphical way in a simple, easy and fast way to create.

## 4. What is Termux?

Termux is an Android terminal emulator and a Linux environment application that works directly without the need for routing or configuration. A minimal base system is automatically installed.

We will use Termux for its stability and easy installation and management, however, you can use an installed environment of Ubuntu Linux for Android.

In this Linux environment you will have the "core" of the communication processes of the MiniBlocklyChain.

## 5. What is Mini BlocklyChain?

Mini BlocklyChain is a fully functional blockchain is a technology developed for mobile phones with OS (Operating System) Android that will be the nodes that will perform in sending and receiving transactions. We created the first blockchain technology that is structured in a "modular" way through Blockly programming where anyone with minimal knowledge and without knowing how to program can create and develop programs for mobile phones and create their own blockchain either in public or private network mode. If you want to create your own digital currency you can do it or a solution to use it in a company, the possibilities are based on the needs of each real case and the business logic that the user adopts or creates according to his requirements.

Mini BlocklyChain is the first modular blockchain for mobile phones where a transactional system can be implemented in a short time.

Before entering the definition and use of "module" blocks we need to have the basic concepts of the Mini BlocklyChain components to know when, how and where to apply them according to the real case we want to implement.

The following concepts emphasize what type of user they are targeting, so it is not important for people who do not have programming skills that the concepts for development users are fully understood.

Basic concepts:

Node. - Each mobile device (phone, tablet, etc.) with an Android operating system that is part of Mini BlocklyChain's public or private network is named as a node of this network.

Hash. - It is a digital signature that is associated with a set of data, string of characters, documents or some kind of digital information, this digital signature is unique and unrepeatable which helps to send or receive information so that the initial content of information sent can not be altered.

Active. - Any type of digital data or information that can be weighted with some tangible or intangible value for people or companies (documents, digital coins, music, video, images, digital authorizations, etc).

Transaction. - Exchange of information between nodes occupying some kind of asset.

UXTO Transaction - It is a type of transaction that packs one or several transactions from the nodes and all the unspent transactions from each node (UXTO: Unspent *Transaction* Outputs) can be spent as input in a new transaction. These transactions are grouped in a queue to be processed every certain time that can be regulated according to the requirements of each information flow.

Transaction (input). - It is a type of transaction that is based on UXTO transactions. When a UXTO transaction queue enters, it is processed by an input transaction that classifies the transaction as a deposit or an expense and thus be able to have a balance of the final result for each user.

Source address. - This is the address of the person who generates or sends the transaction to be processed in the SQLite Master queue. It is an alphanumeric number of 64 characters that is created and verified by the system.

Destination address. -This is the address of the person who receives the transaction and adds it to his balance or stores the sent asset. It is a 64-character alphanumeric number that is created and verified by the system.

SQLite Master. - API REST database that receives the transactions and creates a queue to be processed every certain variable or fixed time according to the business need.

DataBase transaction. - Are the transactions that are reflected in the SQLite database that reserves or stores Mini BlocklyChain transaction information.

AES (*Advanced Encryption Standard)* - It is a security process that encrypts the data that is stored in Mini BlocklyChain's SQLite database.

Balance. - After making any transaction process in the Mini BlocklyChain, a balance of the operation is obtained either as a tangible value (cryptomoney) or intangible value (business processes between people or companies).

Business process. - It is any process that involves some kind of information flow to obtain a result to an end user, the end user can be a person or company from a variety of sectors in the private or public sector.

Public and private key. - It is a type of information encryption where two alphanumeric keys associated with each other are generated and used to send sensitive information through public or private networks. The private key is used to encrypt information and when sending it through the network it cannot be altered or be legible at first sight, the public key is the

one that is shared and is seen by any person or company and this one will help to verify the information sent, as well as to be able to read it only by the valid addressee.

Digital signature. - It is a signature that can be created with the private key, with this signature the recipient ensures that the information has not been altered and confirms that the information is valid for the recipient.

Digital address (Mini BlocklyChain address). - It is an address that is composed of alphanumeric characters unique to each user of Mini BlocklyChain, this address is used to perform transactions between users and whether public or private network.

Magic Number. - This is a random number defined by business rules for each running UXTO transaction that serves to authorize the node to be a candidate to execute the UXTO transaction and create a new block the Mini BlocklyChain.

Creation of a new block. - A new block in Mini BlocklyChain is a hash created and attached by the node that came out as the winning candidate to process the UXTO transaction.

PoW (Proof of Work) consensus protocol. - It is a test that executes all the nodes and the first node that finishes the test successfully is the one chosen to execute the UXTO transaction. It is an algorithm that is composed of mathematical calculations to obtain a result (hash) according to rules given in each transaction executed by Mini BlocklyChain is based on the wear or demand of the information processing resources of computers, in the case of Mini BlocklyChain has been structured and modified to obtain a "magic number" this is a number to be able to have authorization or consensus of the majority of the nodes to be the one that can execute the UXTO transaction. The PoW has a maximum difficulty level of 5 since it is only used to obtain the "magic number".

PoQu consensus protocol (quantum test) - It is a test that runs all the nodes and that is composed initially by the PoW to obtain the "magic number" and later it executes a probability algorithm based on a QRNG (Quantum Random Number Generator) a quantum random number generator, this process ensures the equality of probability for all the nodes and so choose which node will be the one that executes the UXTO transaction and the creation of the new block.

Merkle tree. - This is a security method to assure Mini BlocklyChain that transactions are valid or have not been modified by any external entity. A hash merkle tree is a binary or non-binary data tree structure in which each node that is not a sheet is tagged with the hash of the concatenation of the labels or values of its child nodes. They are a generalization of hash lists and hash strings.

Redis DB. - Real time transaction database used to process and send to the nodes the new transactions that have been requested.

SQlite DB. - Database where the balances and new blocks for Mini BlocklyChain are stored to ensure the integrity of the network.

Sentry. - Security and data integrity connector between Redis and SQLite. This connector or program is in charge of reviewing, processing, validating, distributing and translating the transactions to the nodes.

OpenSSH. - Security connector to execute tasks within the Android operating system.

Termux Shell Terminal. - Program where you can find dependencies of third parties to process the transactions of Mini BlocklyChain, in this version 1.0 it is used by facility of installation nevertheless in future versions it will be replaced by the system BlocklyShell that is at the moment in development.

Wallet. - It is the digital repository where two fundamental data are kept in any transaction; the public and private keys that will be used as the source address and digital signature respectively for any transaction performed, as well as the balance of the transactions sent or received. It is a repository where the Mini BlocklyChain addresses are kept, as well as the results of the UXTO transactions (Balance).

Application developer or programmer:

Object Oriented Programming (Java) - Mini BlocklyChain is made in Java.

BlocklyCode. - It is the term of the intelligent contracts that can be executed in the Mini BlocklyChain environment, the BlocklyCode is created in java programming.

OpenJDK for Android. - It is the suite of JDK and JRE for Android where the BlocklyCode are created and executed.

QRNG (Quantum Random Number Generator). - It is a quantum random number generator, Mini BlocklyChain currently has two APIs for the generation of these.

rsync. - It is a free application for Unix and Microsoft Windows type systems that offers efficient transmission of incremental data, which also operates with compressed and encrypted data.

ffsend. - It is an application to share files easily and securely from the command line.

NTP. - Network Time Protocol, is an Internet protocol for synchronizing the clocks of computer systems through packet routing in networks with variable latency.

Termux (development). - Shell terminal with a wide range of opensource applications and libraries.

Blockly modules (data). - Developers can integrate modules to enhance the features of Mini BlocklyChain.

Peer to Peer. - This term is referred to the communication between nodes in a direct way, that is to say, the information update does not depend on a central server but each node works as a central server that communicates between all of them having the same information which helps to avoid failure points.

Syncthing. - tool for synchronizing data or files between two devices using the "Peer to Peer" communication type.

Red Tor. - This is a low-latency distributed communications network overlaid on the Internet, in which the routing of messages exchanged between users does not reveal their identity, i.e. their IP address (network-wide anonymity).

Mobile routing. - Installation process of external software on your phone to enter as a system administrator in Linux operating systems (Android) the administrator user is called "root" to be able to rotate your phone will have access to any process. It is important to note that some mobile phone manufacturers (Smartphones) say that the warranty is lost due to any fault in the mobile phone.

## 6. Process architecture in Mini BlocklyChain

Mini BlocklyChain is formed by three processes that form its architecture, the first is the business processes, the second is the communications processes and the third is the development environment for complementary modules and/or the creation of BlockyCode "intelligent contracts".

The business processes are the blocks that form a series of routines to create a transaction either in public or private network, this type of process is the planning of the business, the how, when, what, who, where and more attributes will be ordered, planned and distributed so that the main functionality of each transaction sent, received, stored and/or rejected is achieved. The first process is composed of the following types of blocks divided into four categories:

1. Data Entry Blocks
2. Data processing blocks
3. Security blocks.
4. Modular data blocks (developers)
5. Communication bots.

The data entry blocks are those that receive the transaction with a minimum of four input parameters (source address, destination address, active, data) and can have more depending on the variable "data", this can be a block developed by third parties or with a null value (NULL).

The data process blocks develop the logistics, calculation, data normalization, logical decisions and flow checks for each transaction. These types of blocks are used to give intelligence to the business process and are based mainly as its name indicates on processing all types of information, as well as its conversion from different types of data.

The security blocks are used to validate the information and ensure that the transactions have not been altered from their origin to their destination, they are always processed with various security algorithms used in blockchain technologies, among the most used tools are (hash signature, digital signature, AES data encryption, creation and validation of digital addresses, among others).

The modular data blocks, these are the blocks that are developed by third parties, remember that Mini BlocklyChain was created in a modular way to be enriched by new blocks according to the needs of each sector whether public or private. To know more about how to create modules check the Developers Annex.

As we commented previously the architecture of Mini BlocklyChian in its second component are the processes of communications, these processes are the ones in charge of the communication channels via TCP and Sockets of communication to give flexibility of sending and receiving transactions either by the different means that are used at the moment the most used are: Mobile Internet, Wifi currently working to include the medium of communication Bluethtooth.

The communication blocks are basically based on the exchange of data with security implemented in the transfer channel and this is based on a communication through the SSH (Secure Shell) protocol with the different stages that a sent or received transaction goes through.

The communications part is fundamental since these processes have the function of updating the information in all the nodes via TCP and the connection called **"Peer to Peer" the** blocks that intervene in the communication are based on the exchange of information between the nodes without the intervention of intermediary servers so each node makes it independent being able to create a network of nodes where the points of failure are minimum or almost null. Likewise, this independence of each node helps them to make decisions individually or as a whole according to the needs of the business.

The "Peer to Peer" architecture consists of three parts that form the public or private network that you decide to create, in both cases the communication channel is encrypted from node to node.

The first component for communication between mobile devices (smartphones) or wifi is to provide the nodes or devices with a network where they can be found anywhere in the world, the communication network where MiniBlocklyChain is mounted is the **"Tor" network.**

What is the Tor network? - (https://www.torproject.org)

"**Tor** (acronym for Te **Onion Router** - in Spanish) is a project whose main objective is the development of a low-latency distributed communications network superimposed on the Internet, in which the routing of messages exchanged between users does not reveal their identity, i.e. their IP address (anonymity at network level) and which, moreover, maintains the integrity and secrecy of the information that travels through it".

The second component and no less important task is to get all the nodes in MiniBlocklyChain to have the same data at any time or have their databases and files synchronized to perform this task between the nodes will be implemented **"syncthing".**

What is the Syncting network? - (https://syncthing.net)

Syncthing is a free open source peer-to-peer file syncing application available for Windows, Mac, Linux, Android, Solaris, Darwin and BSD. You can sync files between devices on a local network or between remote devices over the Internet.

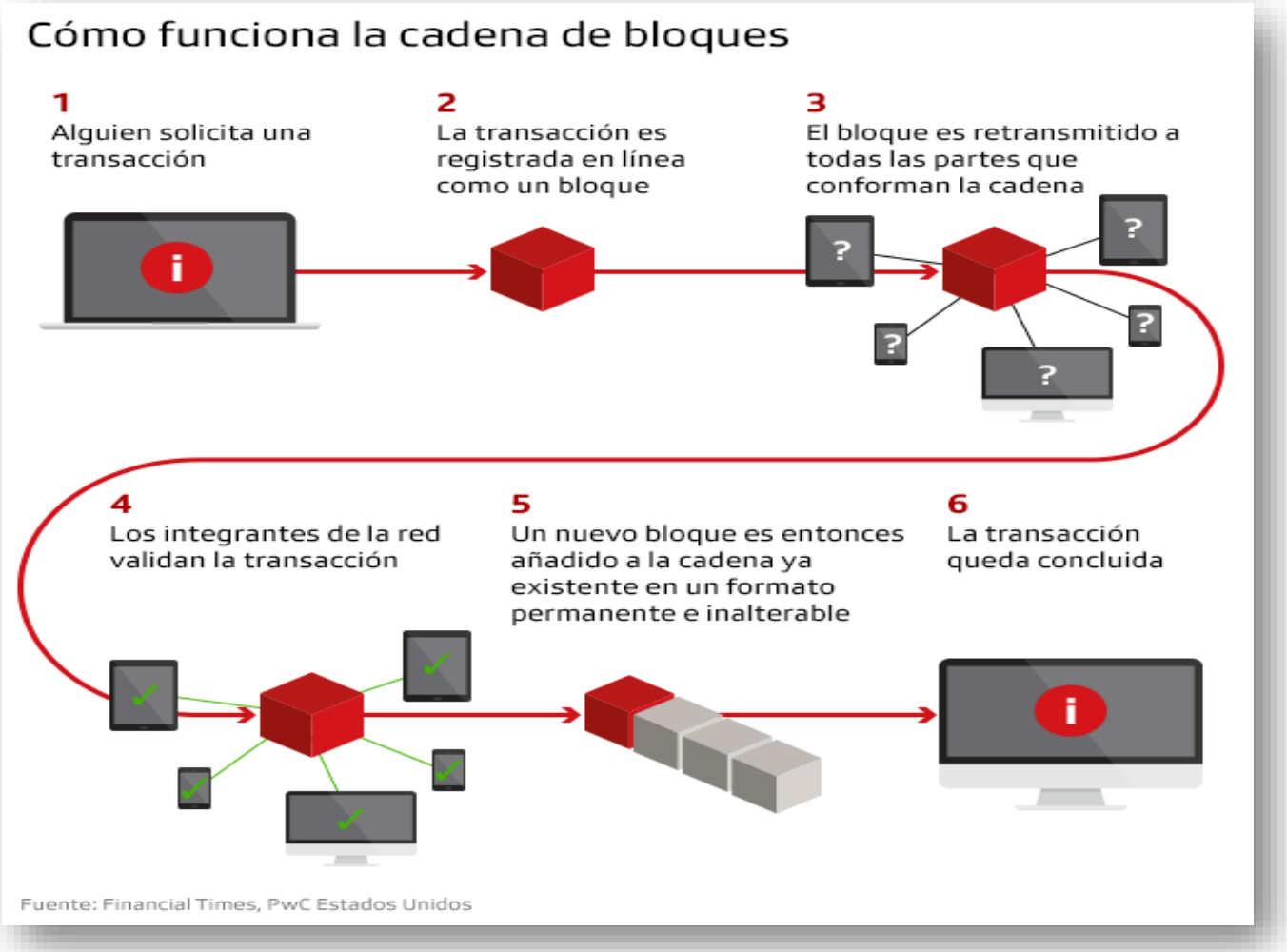
Based on the two previous communication components we can start to develop a trust network between nodes and with the level of data synchronization that will be the backbone or "core" of the business processes in order to meet every need of the user or company.

The third component in the communications network is the Redis database, which will notify all the nodes in real time of new transactions that are received and new transactions that are sent.

What is the Redis DB network? - (https://redis.io)

**Redis** is an in-memory database engine, based on storage in hash tables (key/value) but which can optionally be used as a durable or persistent database.

7. BlocklyChain functionality diagram (Mini BlocklyChain).



## 8. What is Mini BlocklyCode?

Mini BlocklyCode are programs created in the Java language and are stored in a repository independent of the nodes, they are normally named as intelligent contracts, these programs are designed with logical conditions for when these conditions are met they are executed automatically without depending on any authorization or external human intervention.

"A smart contract is a computer program that facilitates, secures, enforces and executes registered agreements between two or more parties (e.g., individuals or organizations). As such they would assist them in negotiating and defining such agreements that will cause certain actions to happen as a result of a number of specific conditions being met.

A smart contract is a program that lives in a system not controlled by either party, or their agents, and that executes an automatic contract which works like an if-then sentence of any other computer program. With the difference that it is done in a way that interacts with real assets. When a pre-programmed condition, not subject to any human judgment, is triggered, the intelligent contract executes the corresponding contract clause.

They aim to provide greater security than traditional contract law and reduce transaction costs associated with contracting. The transfer of digital value through a system that does not require trust (e.g. bitcoins) opens the door to new applications that can make use of intelligent contracts. "

Mini BlocklyCode is aimed at developers with experience in java programming. In Mini BlocklyChain some types of libraries are restricted for security of the same system, however, libraries to create transactions to send or receive some contractual value created or agreed by two or more parties can be created.

Every Mini BlocklyChain complies with the following premises:

- ✓ Any Mini BlocklyChain created is based on running only messages and not executions on the system, however, these messages may contain authorizations, physical contract approvals or physical actions.
- ✓ Every Mini BlocklyChain created has to go through the "auditor" communication block. This block is in charge of monitoring malicious code or forbidden code to review sentences or unauthorized orders in the system.
- ✓ All Mini BlocklyChain is executed only on the JVM of the source node, programs are not executed globally for system protection.

For more details see the "Mini BlocklyChain for Developers" appendix.

# 9. Mini BlockyChain communications network installation

1. Installation and configuration of Mini SQLSync network

The Mini SQLSync network is in charge of receiving the transactions from the nodes so that they can process those transactions. This network is formed by a main network that is through "Peer to Peer" between nodes and a backup network called Mini Sentinel RESTful.

We will start with the installation of the RESTful Mini Sentinel backup network, this service is the one that will receive the transactions from the nodes, this service is based on storing the transactions for a determined time to later convert the information through a connector that will inject a queue of all the encrypted transactions to a Redis service. Later the Redis database service will execute in real time the release of the encrypted transaction queue to all the nodes that integrate the Mini BlocklyChain network.

A RESTful type service or design is a simple and easy way to consult, modify, delete and/or insert information into a database through a URL or internet address, in our case we will use the SQLite database because of its characteristics of being all the information in one file. For a use of requirements with needs of major scaling we will be able to use another type of database as MySQL, Oracle, DB2 or another commercial database, simply we will have to change the connector of Mini BloclyChain of SQLite (free version) for the connector of Mini BlocklyChain DB others (commercial version).

IMPORTANT NOTE: RESTful Mini Sentinel configuration does not process any kind of transactions at any time, it is only the service that formats "shapes the information" so that the nodes can perform the transaction processing properly and in a planned and synchronized time for all the nodes.

RESTful service installation is based on a SQLite database. This installation will be done in Windows environment for practical purposes, however, this model can be easily replicated in a Linux type operating system.

For those who wish to review the performance and support of SQLite queries and insertions, please refer to these performance tests:

https://www.sami-lehtinen.net/blog/sqlite3-performance-testing

https://sqlite.org/src4/doc/trunk/www/lsmperf.wiki

RESTful Mini Sentinel backup network installation. We will be installing and configuring this service in a Windows 10 computer, however, the process has also been installed in an Ubuntu 18.09 Linux environment easily.

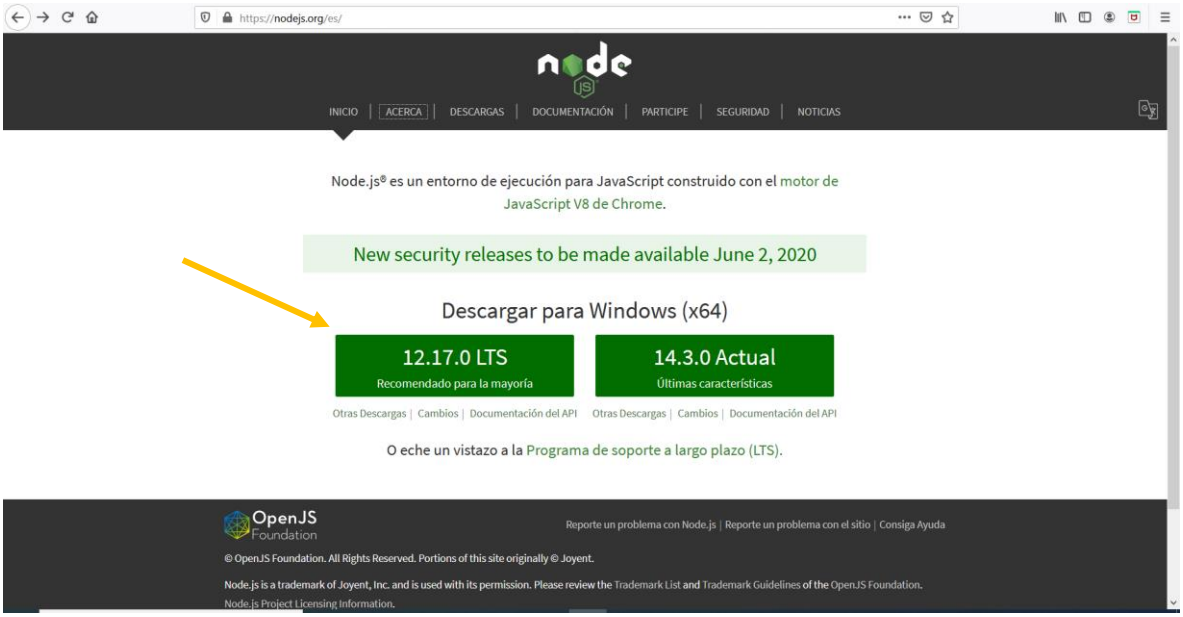
Requirements:

- ✓ SQLite database server in RESTful mode ( https://github.com/olsonpm/sqlite-to-rest ).
- ✓ SQLite-Redis Sentinel Connector
- ✓ Redis database server in Master-Slave mode (https://redis.io/topics/replication).

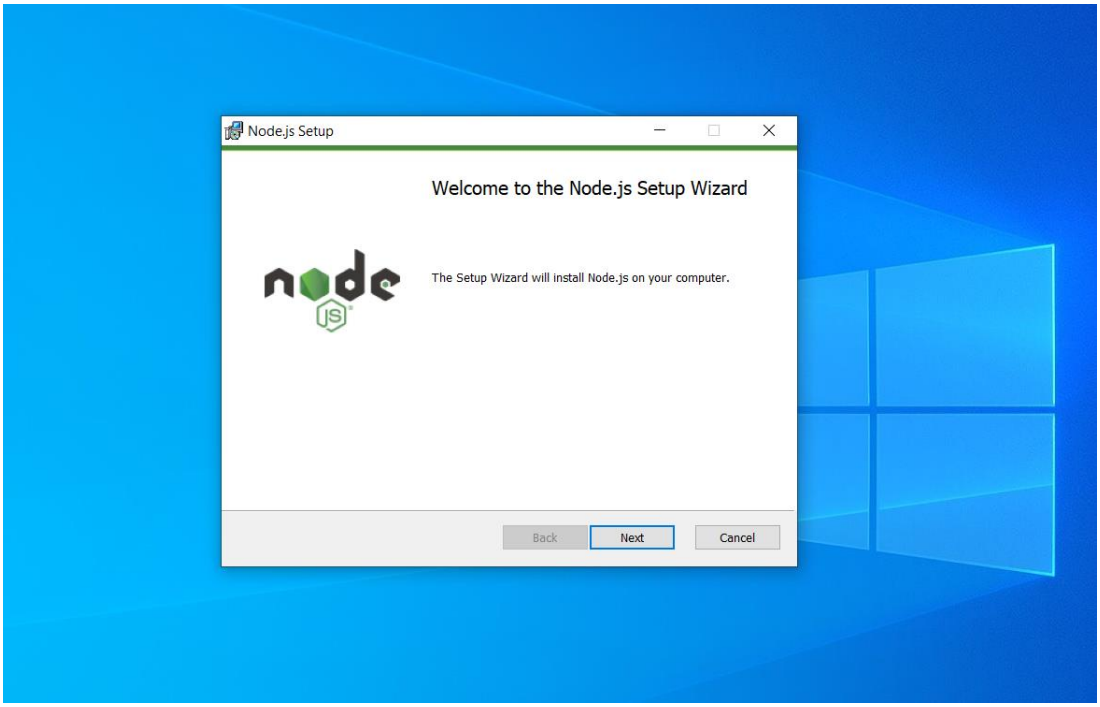
Installation of SQLite database server in RESTful mode

For the installation we need to start with the following software packages, Nodejs, sqlite3 and Git Bash for Windows.

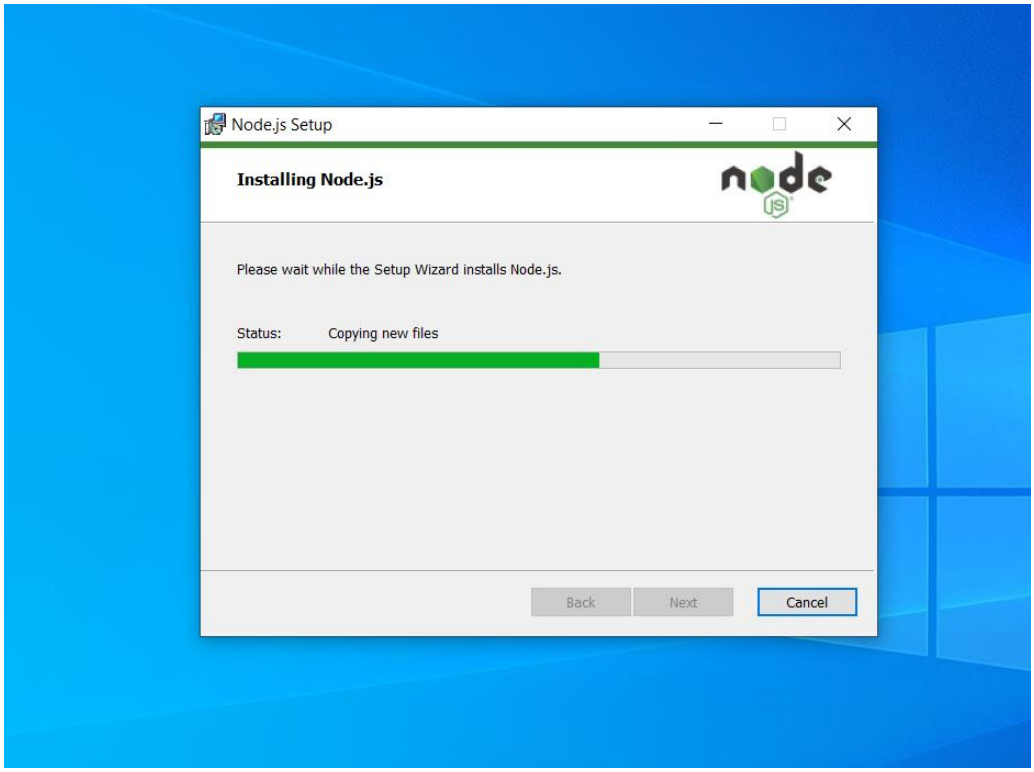
To obtain Nodejs we consulted their official site https://nodejs.org/es/ and chose the "Recommended for Most" version.

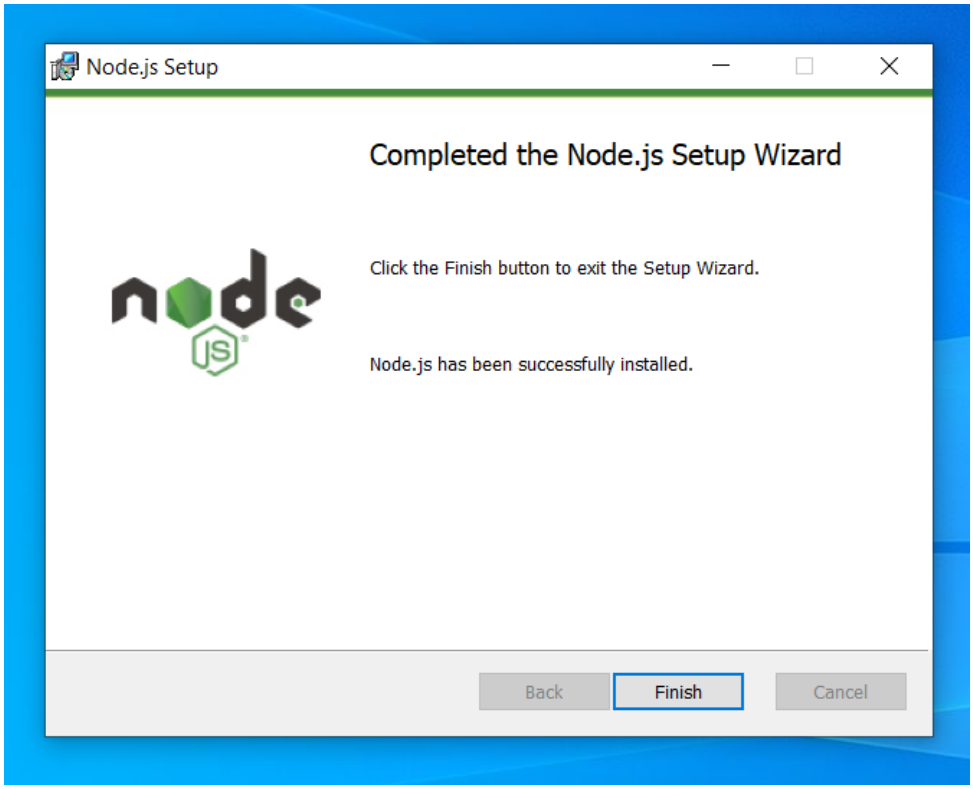



After downloading the file with extension **. msi** we double click to install it.




We perform the installation by default, simply click "Next" without choosing any additional options you ask.

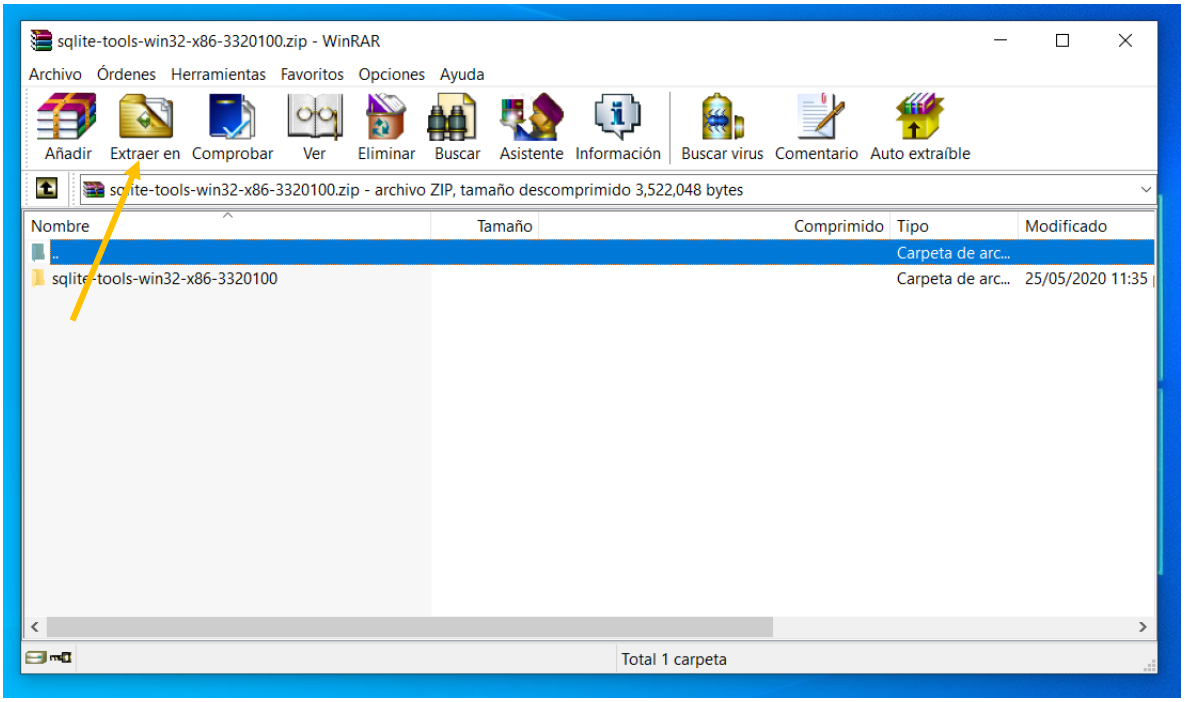




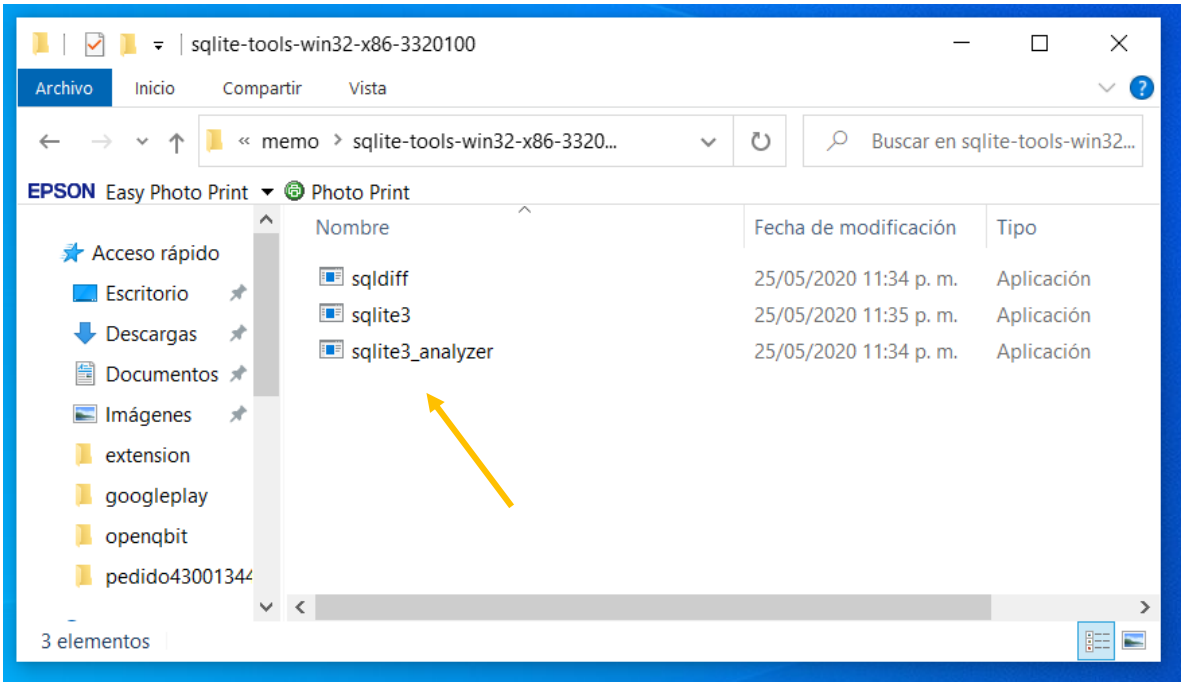

Since we have finished the installation of Nodejs we continue with the installation of the SQLite database.

We go to their official site, https://www.sqlite.org/index.html and click on the part where you "download".

Next we choose the option where it says **"Precompiled Binaries for Windows"** and click on the software version **"sqlite-tools-win32-x86-3320100.zip"** when the download is finished we proceed to decompress the file in a simple way, we open the folder where the file was downloaded and double click on the file to decompress it.

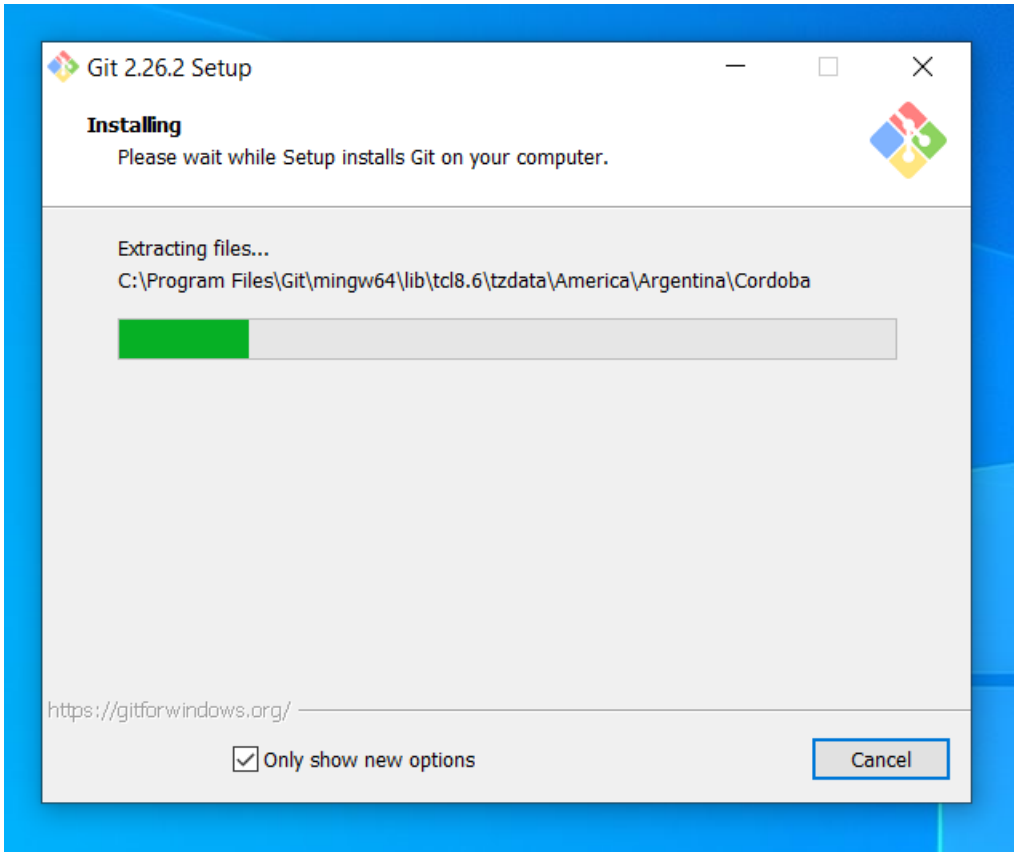



After decompressing it we will have three files, these are our environment to create our SQLite database that we will use later.



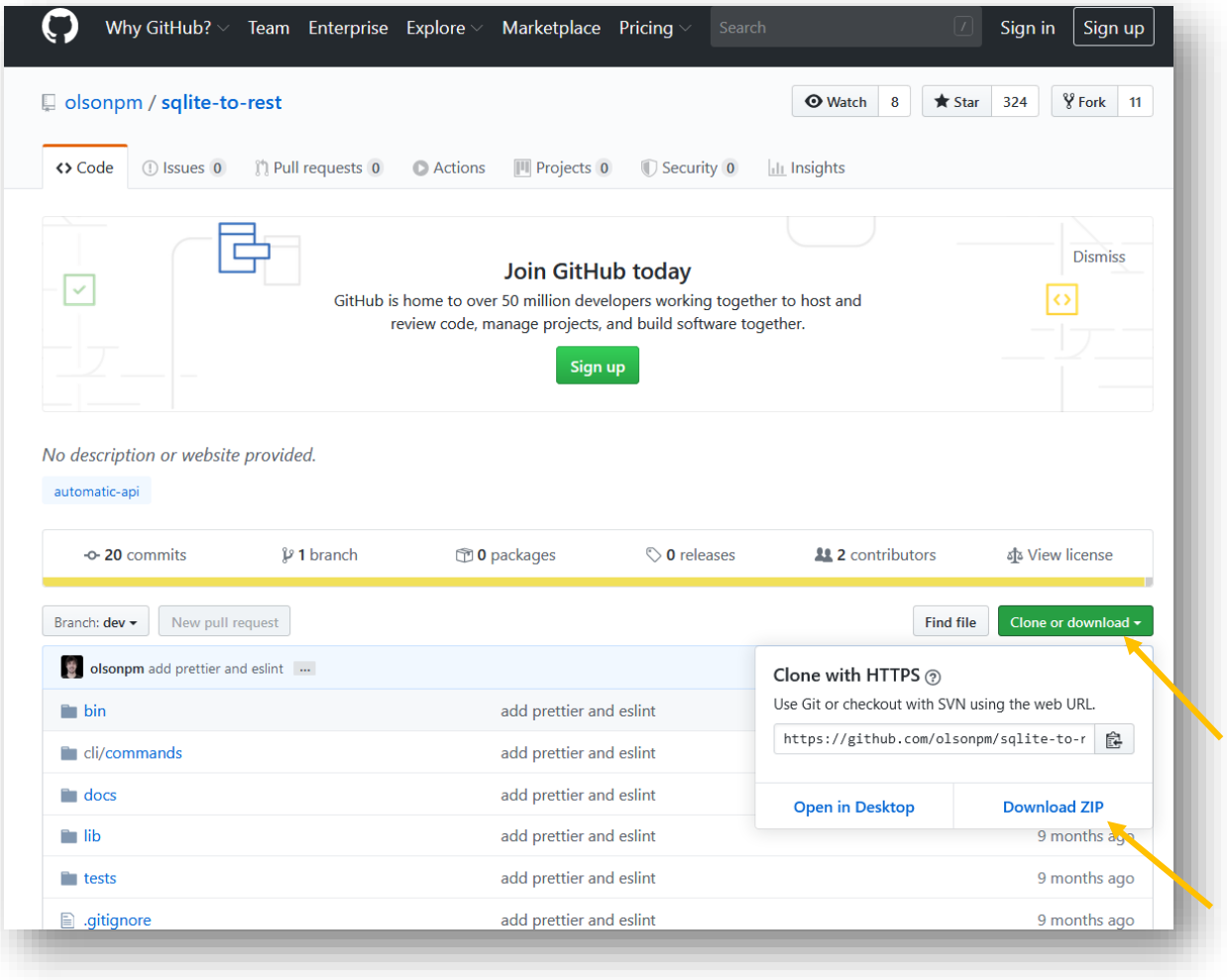
We'll start by installing Git Bash, a Linux-like terminal emulator that will help us run the RESTful backup service properly.

We will download it from their official site, https://gitforwindows.org/ and proceed to install it with the default options that it indicates.



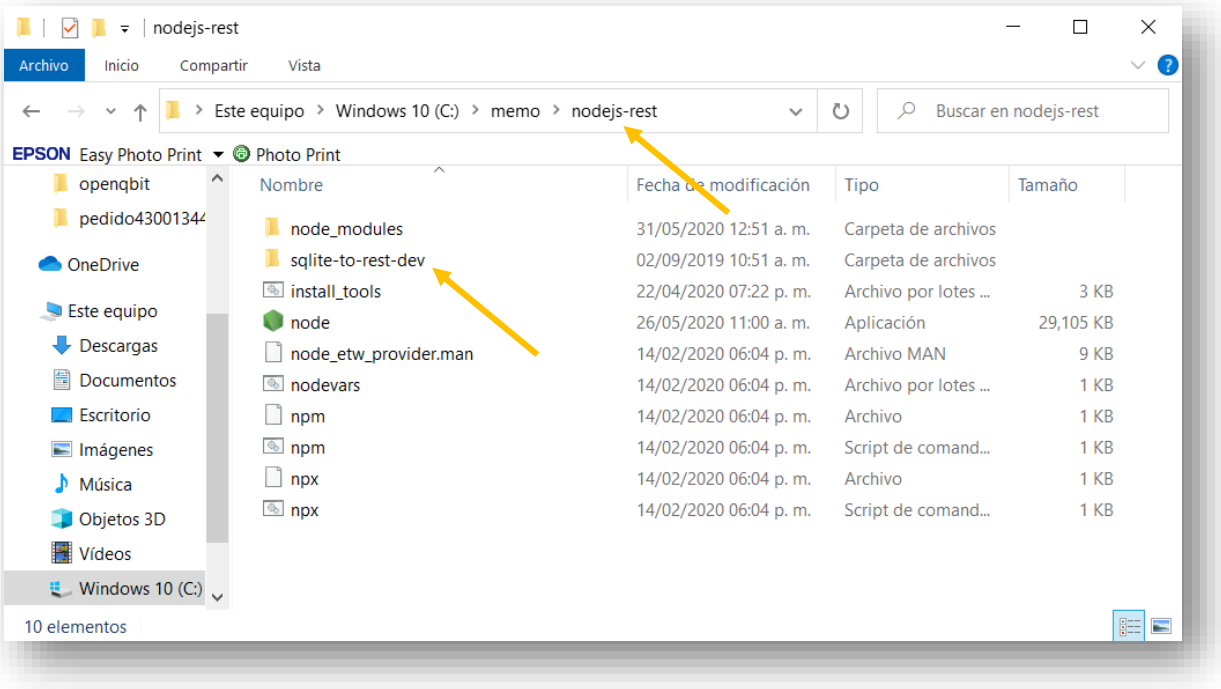

Now that we have nodejs, sqlite and Git Bash installed on our Windows 10 machine we proceed to install the environment that will support our SQLite database in RESTful mode.

At the time of downloading the software that will give the functionality of RESTful to SQLite. For this we must go to the following site, https://github.com/olsonpm/sqlite-to-rest in this we will click on the green right button "Clone or download" and choose the option "Download ZIP" this will download the software in our computer.




After downloading it in our computer we proceed to decompress it inside the directory or folder where we installed the **nodejs** program and we will have a directory with the name **"sqlite-to-rest-dev"**.

**NOTE:** It is important that the **sqlite-to-rest-dev directory is** inside the directory where **nodejs** was installed.

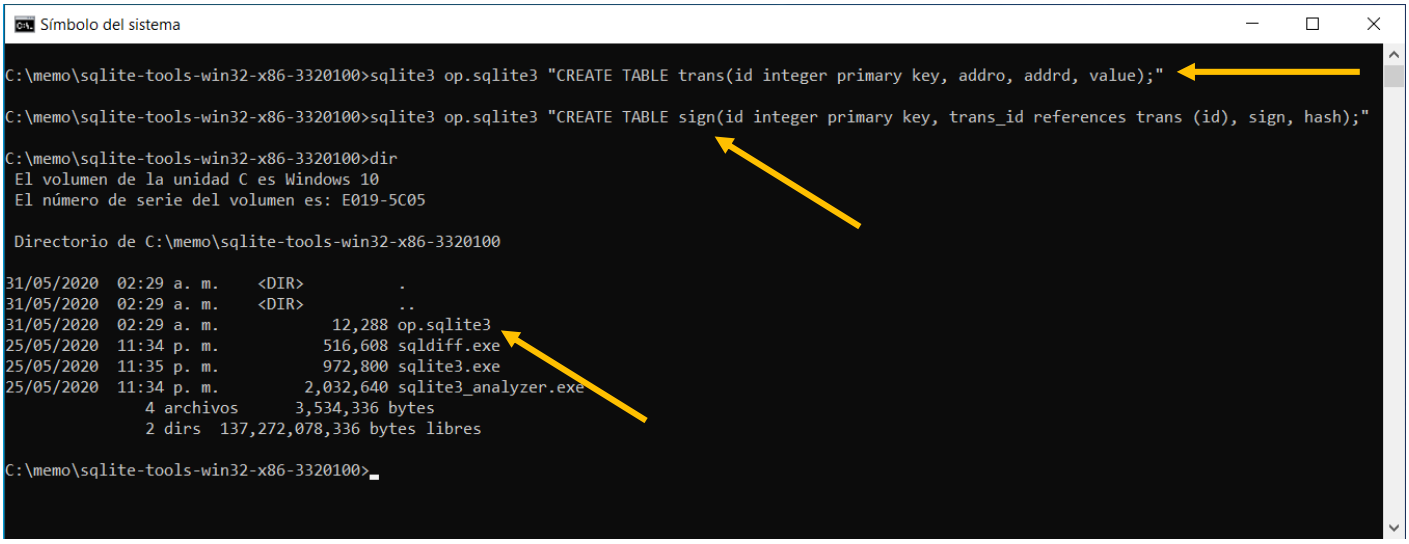



Having everything installed we proceed with the configuration of the SQLite database that we will use to store the transactions of the nodes in the RESTful backup environment.

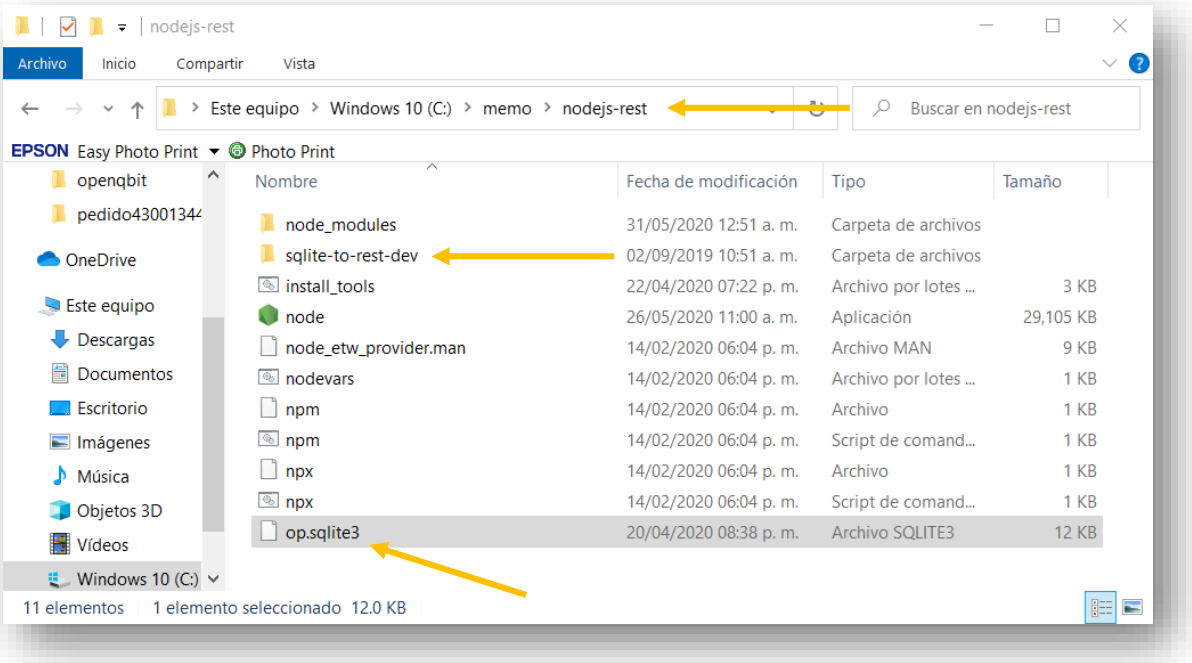
Table design and data structure. Commands to be executed online in CMD command list

sqlite3 op.sqlite3 "CREATE TABLE trans (id integer primary key, addro, addrd, value);"

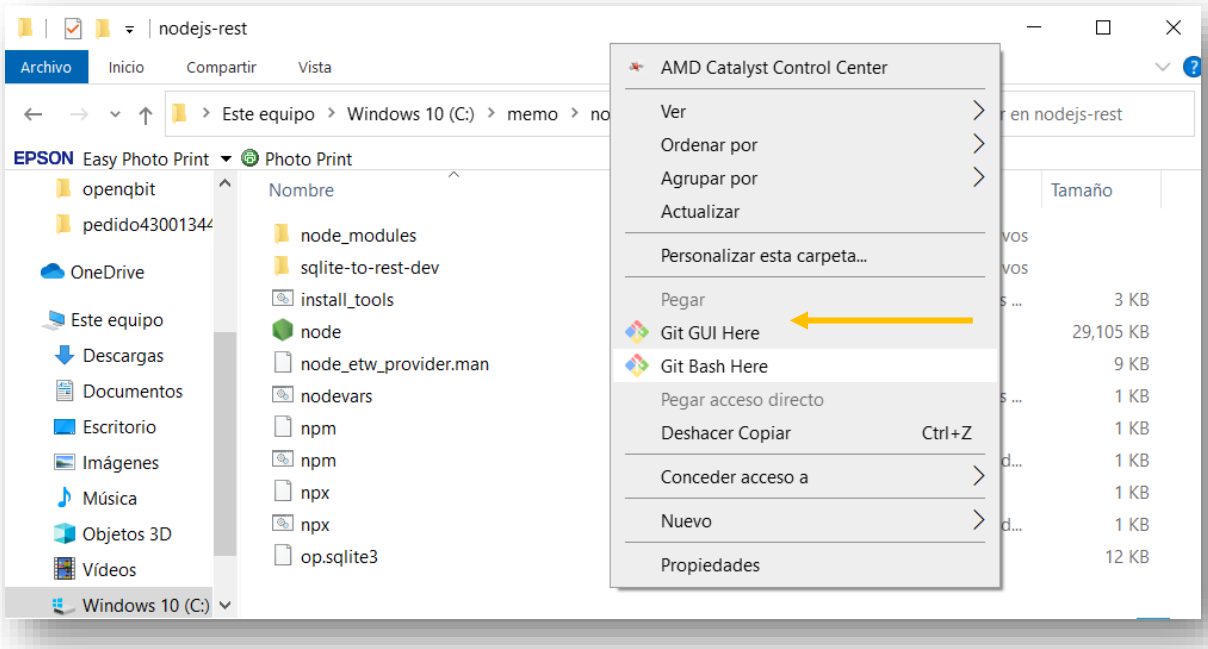
sqlite3 op.sqlite3 "CREATE TABLE sign (id integer primary key, trans_id references trans (id), sign, hash);"



After creating the database op.sqlite3 we must make a copy in the directory where the **nodejs** was installed. In the **nodejs directory** must also be the copy of **"sqlite-to-rest-dev".**

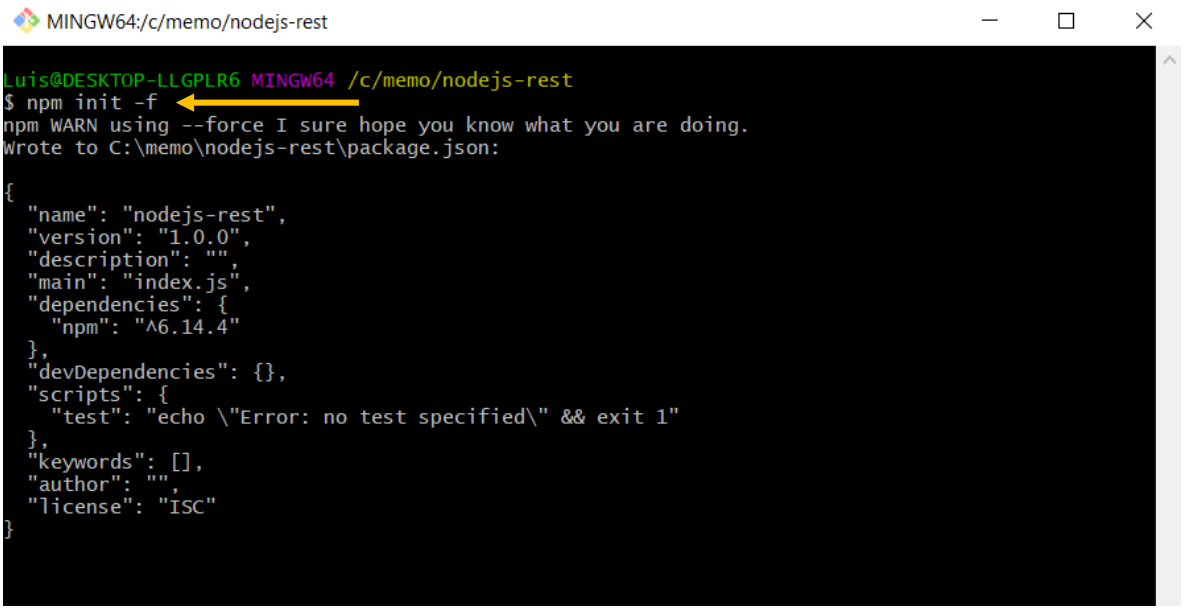



We place ourselves in the folder where the **nodejs** installation is and where the **"sqlite-to-rest-dev"** software should also be. Point to the folder with the pointer and right-click to display the menu and choose where it says **"Git Bash"** to open a terminal.

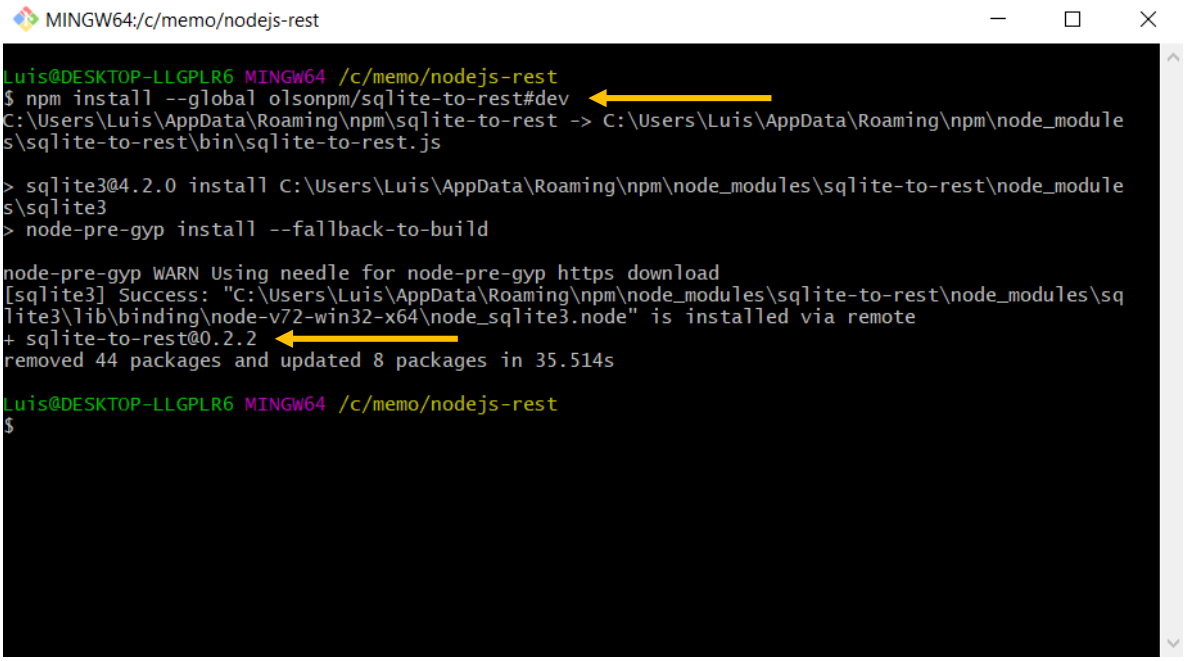


In the new open Git Bash terminal, we execute the following commands:

$ npm init -f




$ npm install --global olsonpm/sqlite-to-rest#dev

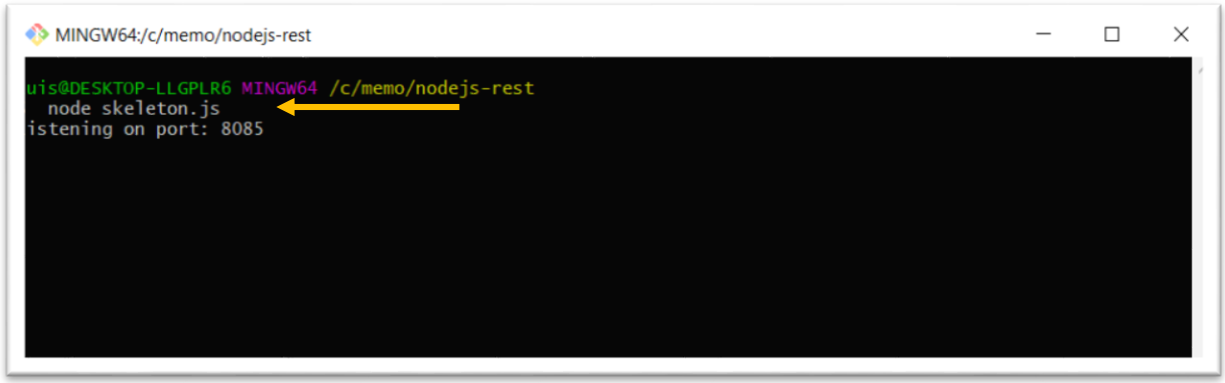



After the execution of the command, the installation of the "**sqlite-to-rest**" package will appear.

We generated RESTful environment for SQLite with the **op.sqlite3** base we created before.

We execute the command**: $ sqlite-to-rest generate-skeleton --db-path ./op.sqlite3**


We started RESTful SQLite service on default port 8085. We run the following command: **$ node skeleton.js**




We tested the RESTful service with adding data and querying data.


To test the SQLite RESTful service we use the following commands:

To insert data in the table trans that is within the database op.sqlite3:

$ curl -s -H "Content-Type: application/json" -d '{"addro":"QWERTY1234", "addrd":"ASDFG4567", "value": "999"}' http://localhost:8085/trans

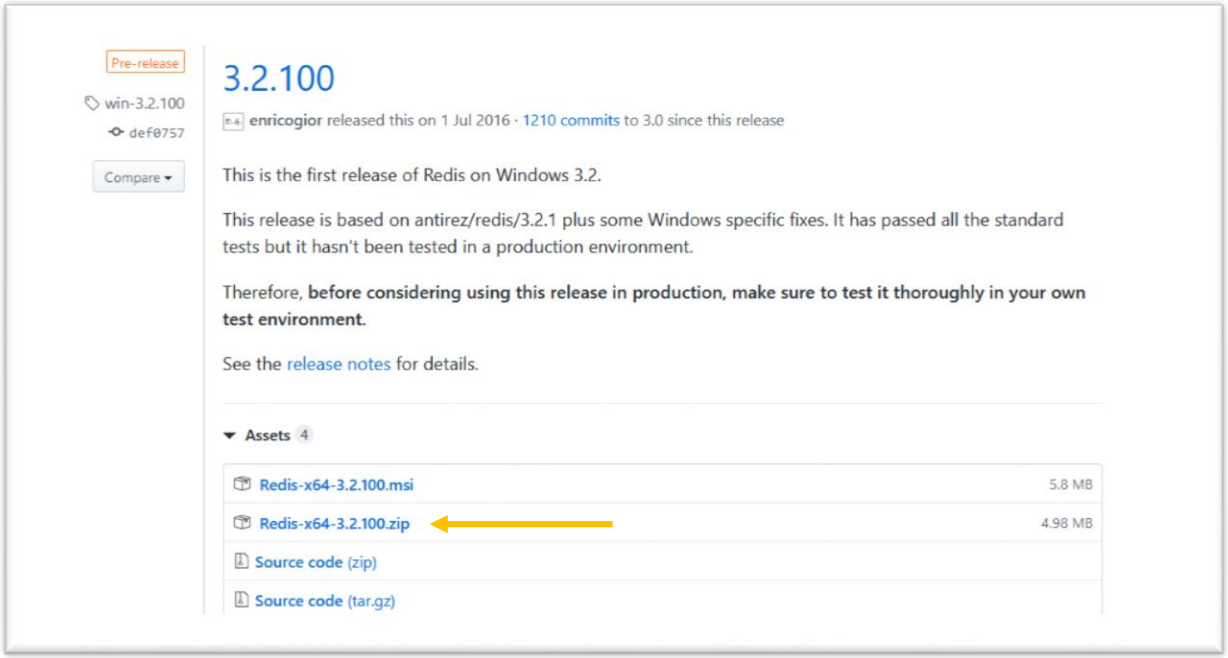
To make a query of all the data in the trans table:

$ curl -s -H 'range: rows=0-2' http://localhost:8085/trans

To review how to use in detail all RESTful enabled services (query, insert, update and/or delete data) review the **Appendix "Restful SQLite GET/POST commands".**

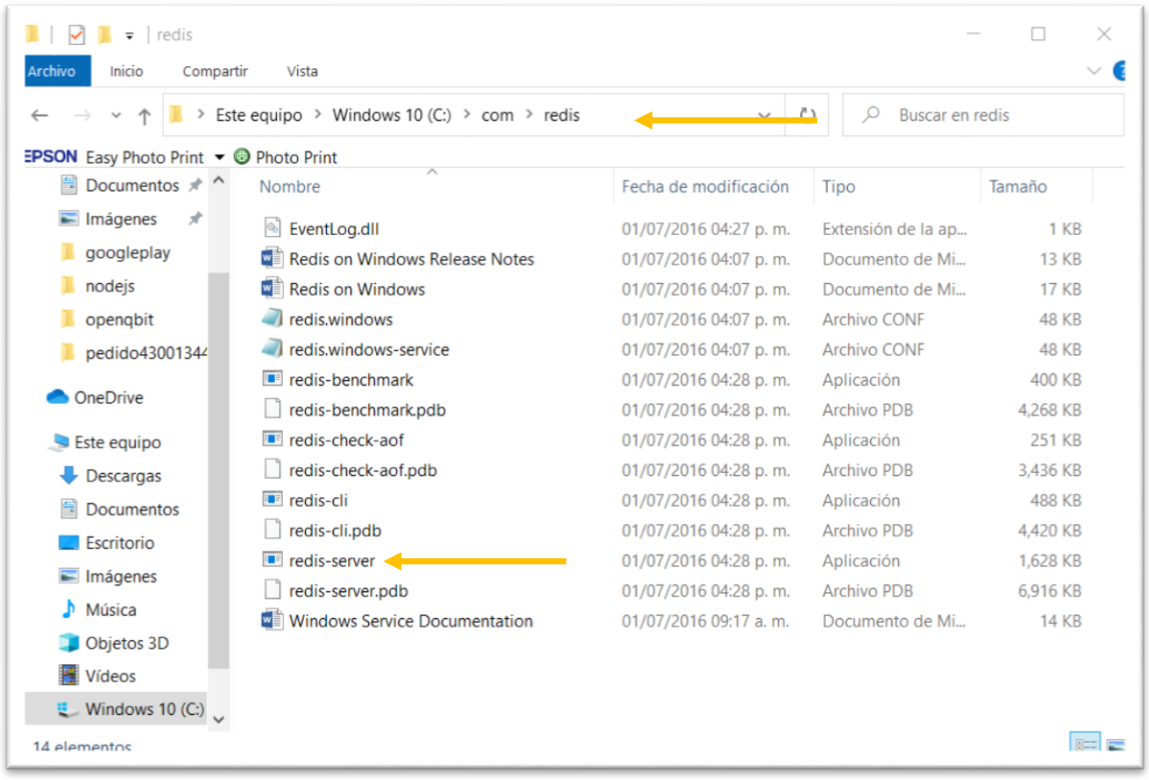
NOTE: In the previous installation, all the queries are being made in the URL with the address "localhost", however, when the server is exposed with a public IP (internet) or private IP (Wifi) it will work without any problem. We will test this when we are doing the communication tests between the SQLite RESTful service and the communication network nodes that form Mini BlocklyChain.

Redis database installation for backup network service, to download the redis software for Windows we have to go to the site, https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100 and choose the ZIP package.

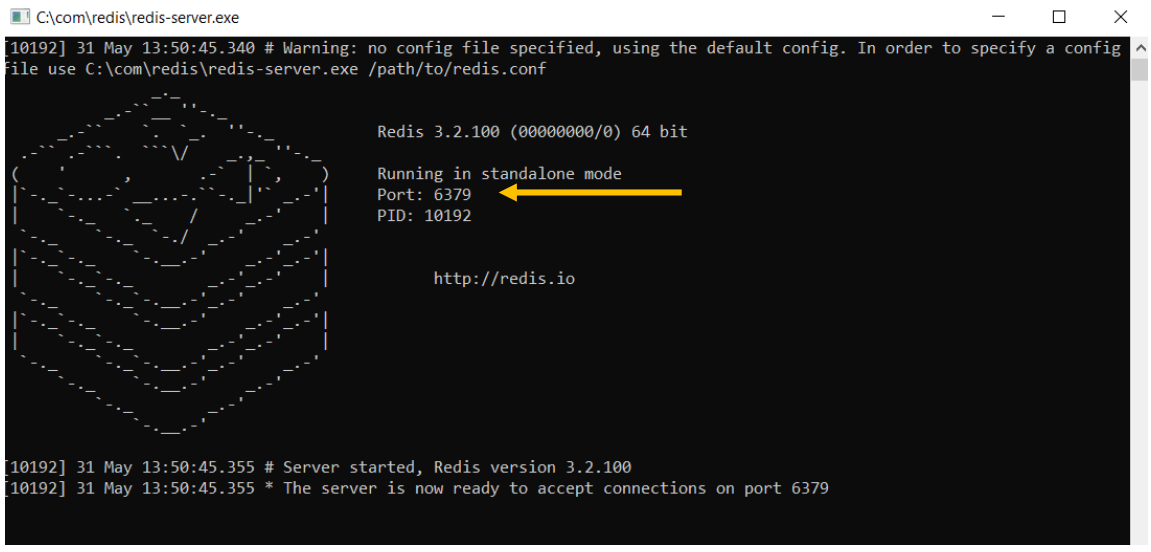



To locate the installation we will create a directory called "redis" within Windows and download the file with a ZIP extension, we decompress it in the directory created previously, we already have the installation completed redis.

We test the installation by running the server by double clicking on the command **"redis-server"**.




After executing the command "redis-server" we will see the server running in a Windows CMD command terminal on the default port 6379:



In this test we have a version of Redis 3.2.100 for Windows 10, in the case of Linux operating system we have the version 6.0.4 (released May 2020) however for our Mini BlocklyChain configuration the Windows version has enough features of functionality that we need.

To stop the execution we make the key combination Ctrl + C. We proceed to configure the Redis 3.2.100 database for Windows 10 with a configuration between Redis and the Mini SQLSync communications network nodes, type Master(Master)-Slave(Slave) is distributed as follows:

The Master is the Redis server for Windows 10 that we are configuring and this will replicate the data in real time and synchronized with the same information to all the nodes (mobile phones). These nodes will have a Redis server installed in Slave mode.

At this point we start to see how the backup network we are installing and configuring works. This configuration will serve us to communicate with all the nodes in real time, it will help us to communicate to all the nodes of the network when there is a new transaction queue to be processed by the nodes, in an equality in the information transfer to all the nodes so that any node has the same probability of being able to process the "transaction queue" information.

We start the configuration of the SQLite-Redis Sentinel connector.

This connector is a program developed in Java language and as its name indicates it connects the SQLite and Redis (Master) databases.

The function of the connector is to transmit the information (transactions) from SQLite to Redis (Master) and this sends the "transaction queue" to the nodes (Slaves).

For more details on the Java code of the **SQLite-Redis Sentinel** connector see the Annex "SQLite-Redis Java Code Connector".

Configuration of the Redis **(Master)** database **redis.conf** file for Windows 10.

Add the following changes or directives to the file, save changes and start Redis server

Start by finding the **tcp-keepalive** setting and setting it to 60 seconds as suggested by the comments. This will help Redis to detect network or service problems:

**tcp-keepalive 60**

Find the **requirepass** directive and configure it with a strong passphrase. While your Redis traffic must be protected from third parties, this provides authentication to Redis. Since Redis is fast and does not rate borderline passphrase attempts, choose a strong, complex passphrase to protect against brute force attempts:

**requirepass type_your_network_master_password**

**example:**

**requirepass** FPqwedsLMdf76ass7asddfd2g45vBN8ty99

Finally, there are some optional settings you may want to adjust depending on your usage scenario.

If you do not want Redis to automatically put in the oldest and least used keys as it fills up, you can disable automatic key removal:

**maxmemory-policy noeviction**

For enhanced durability guarantees, you can enable add-on-only file persistence. This will help minimize data loss in the event of a system failure at the expense of larger files and slightly slower performance:

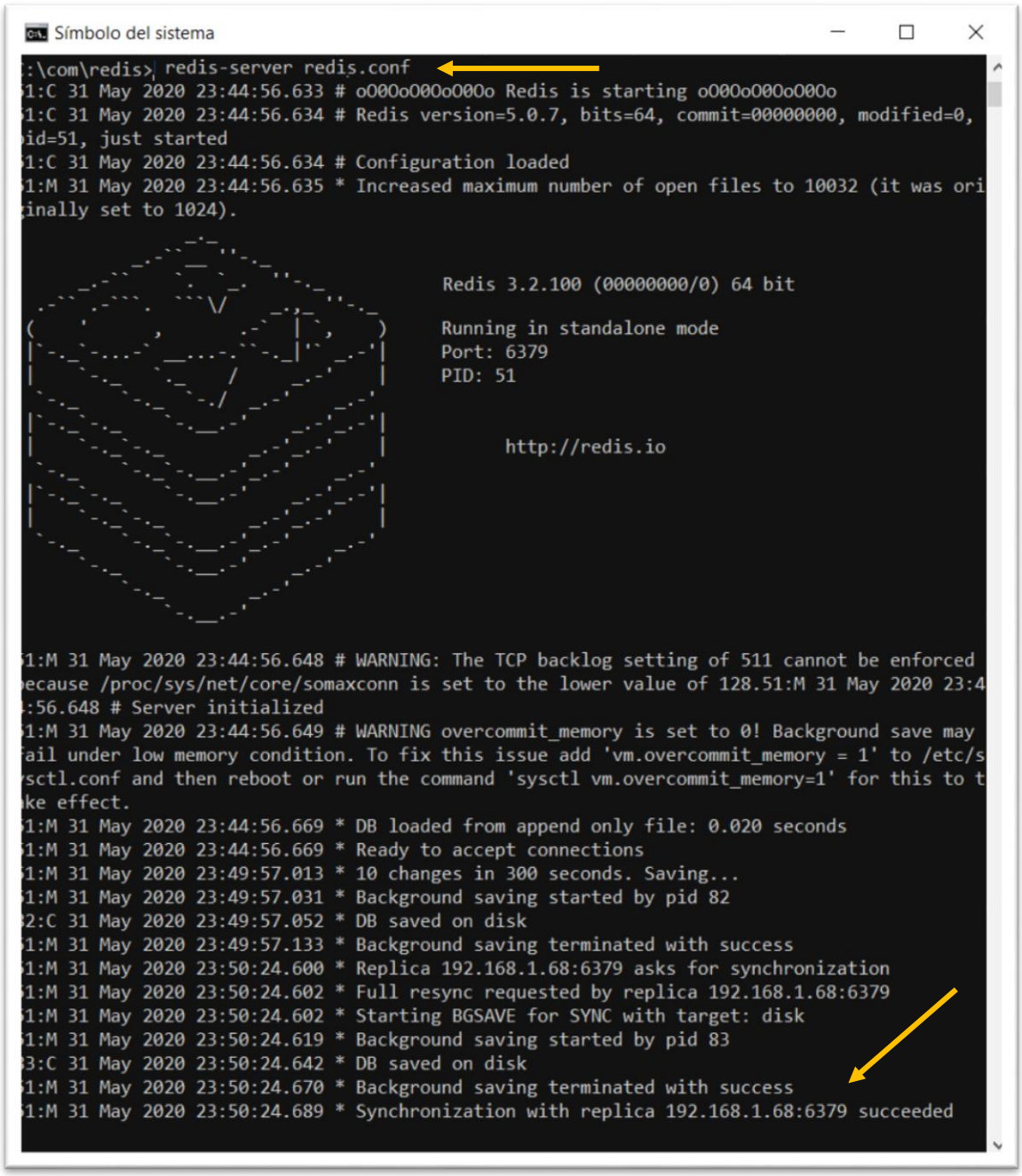
**appendonly yes**

**appendfilename "redis-staging-ao.aof"**

Proceed to save the changes and restart the Redis service for Windows 10, stop with the Ctrl + C keys and run the Windows CMD command line again:

C: \redis_redis_directory redis_server

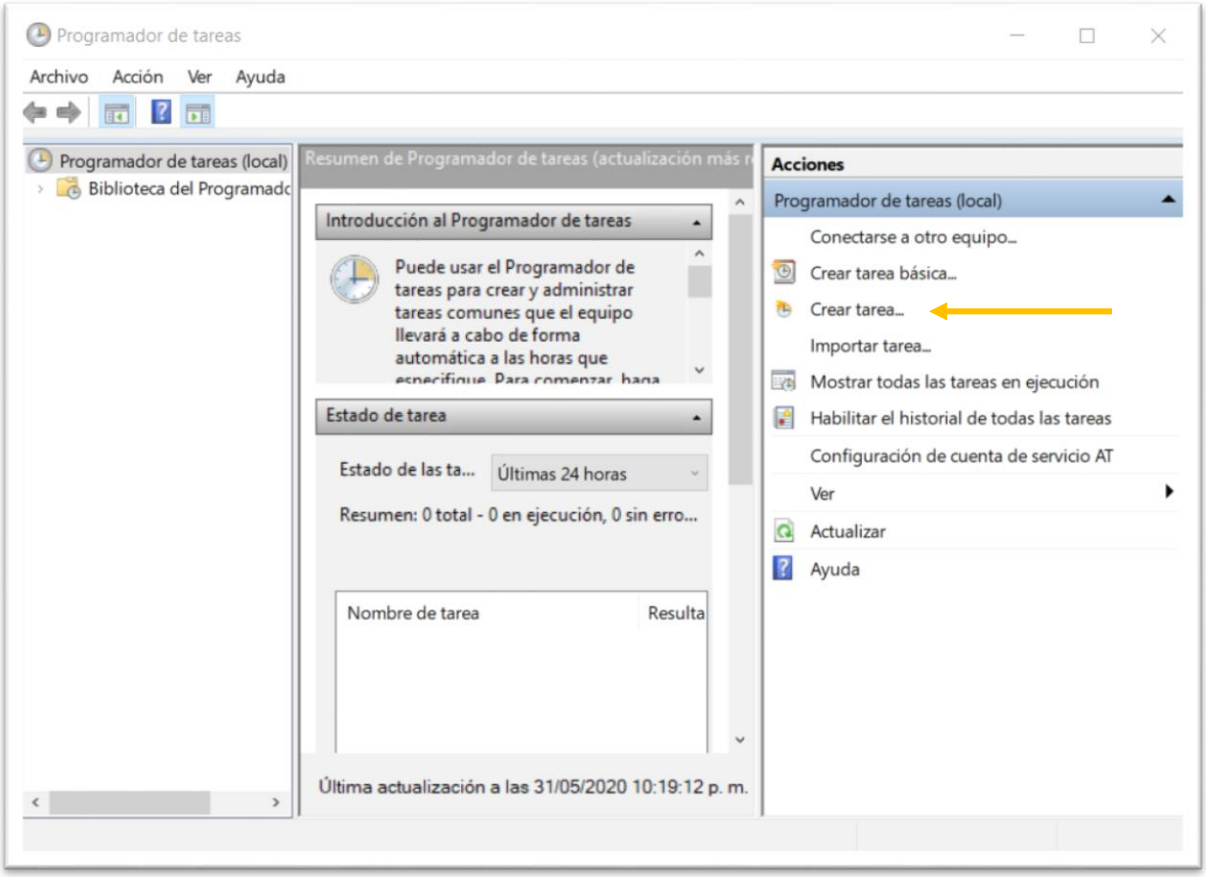
In our example we can see that we have a **(Slave)** node connected.




We can see that we have synchronized a node with the IP 192.168.1.68 in the default port 6379.

Now we need to schedule in the Window 10 operating system the task of executing automatically the **SQLite-Redis Sentinel** connector**.** We do this with the Windows 10 tool is we execute it from the bottom left by typing "Task Scheduler".


We will create a new task "Create task" where we will include the execution of the connector.

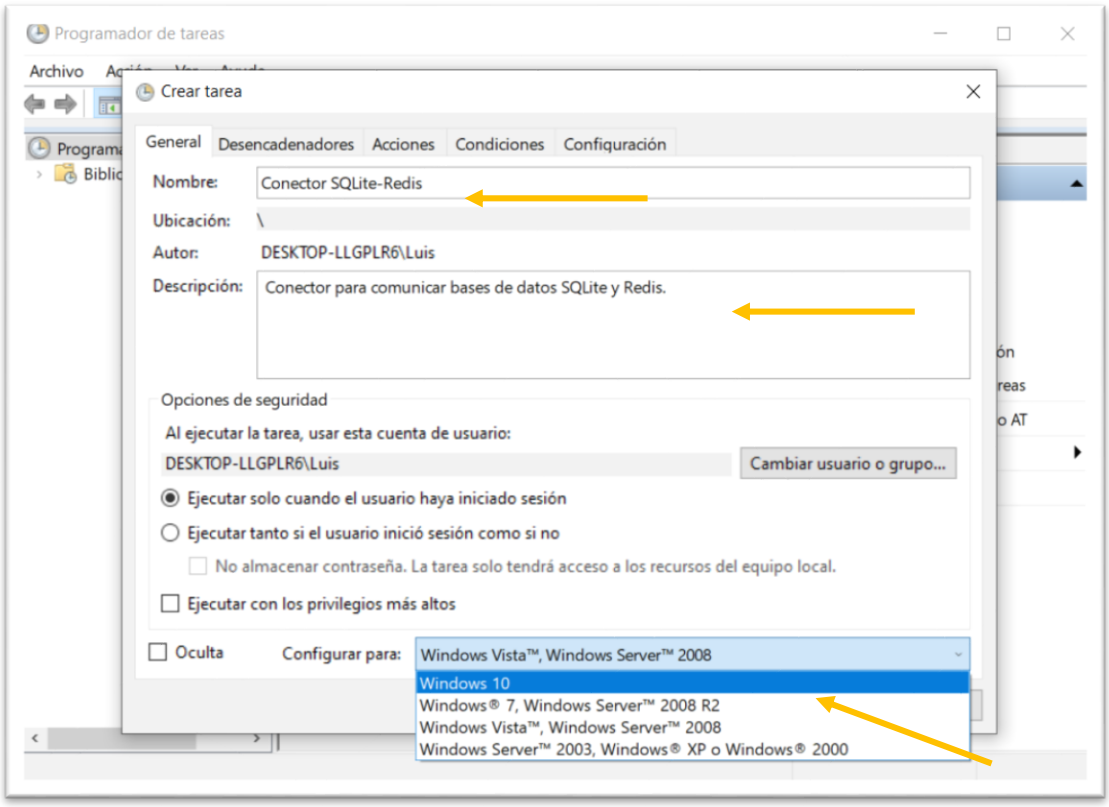



Clicking on "Create task" will open an additional window where we have to give the following parameters in the "General" tab:

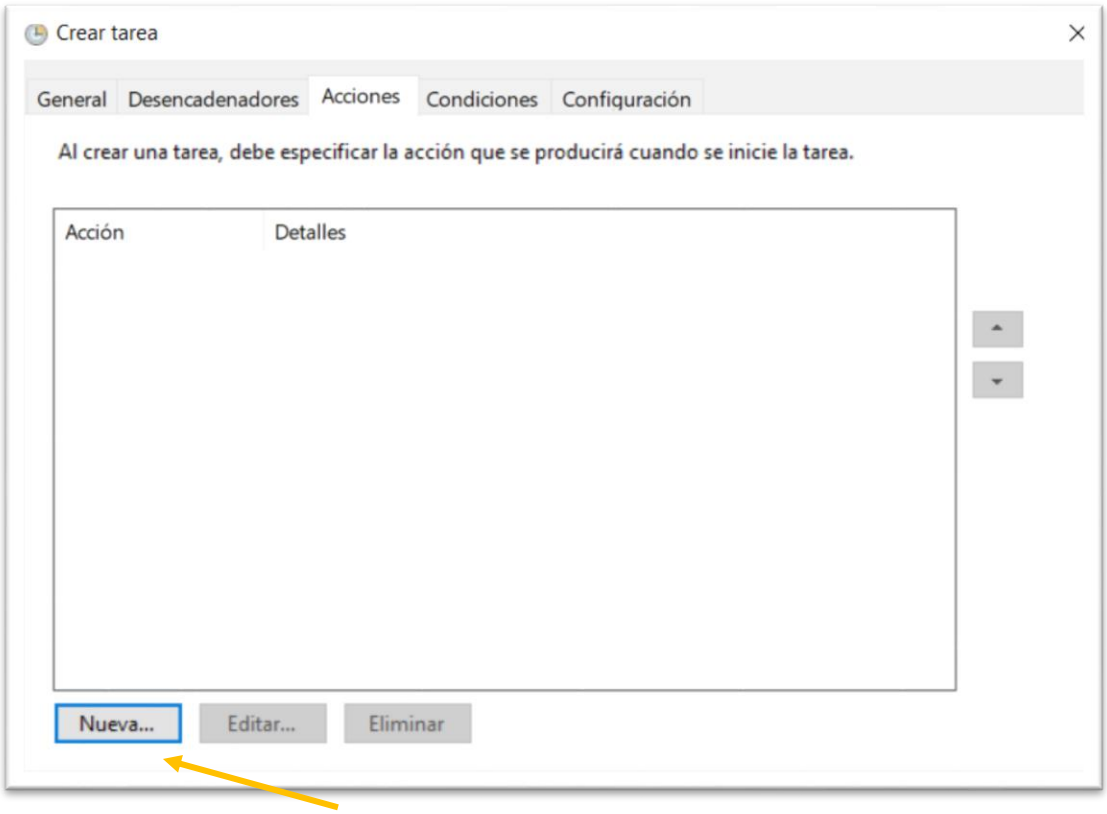
**Name:** task_name

**Description:** optional

**Configure for:** select_operating_system_windows_version



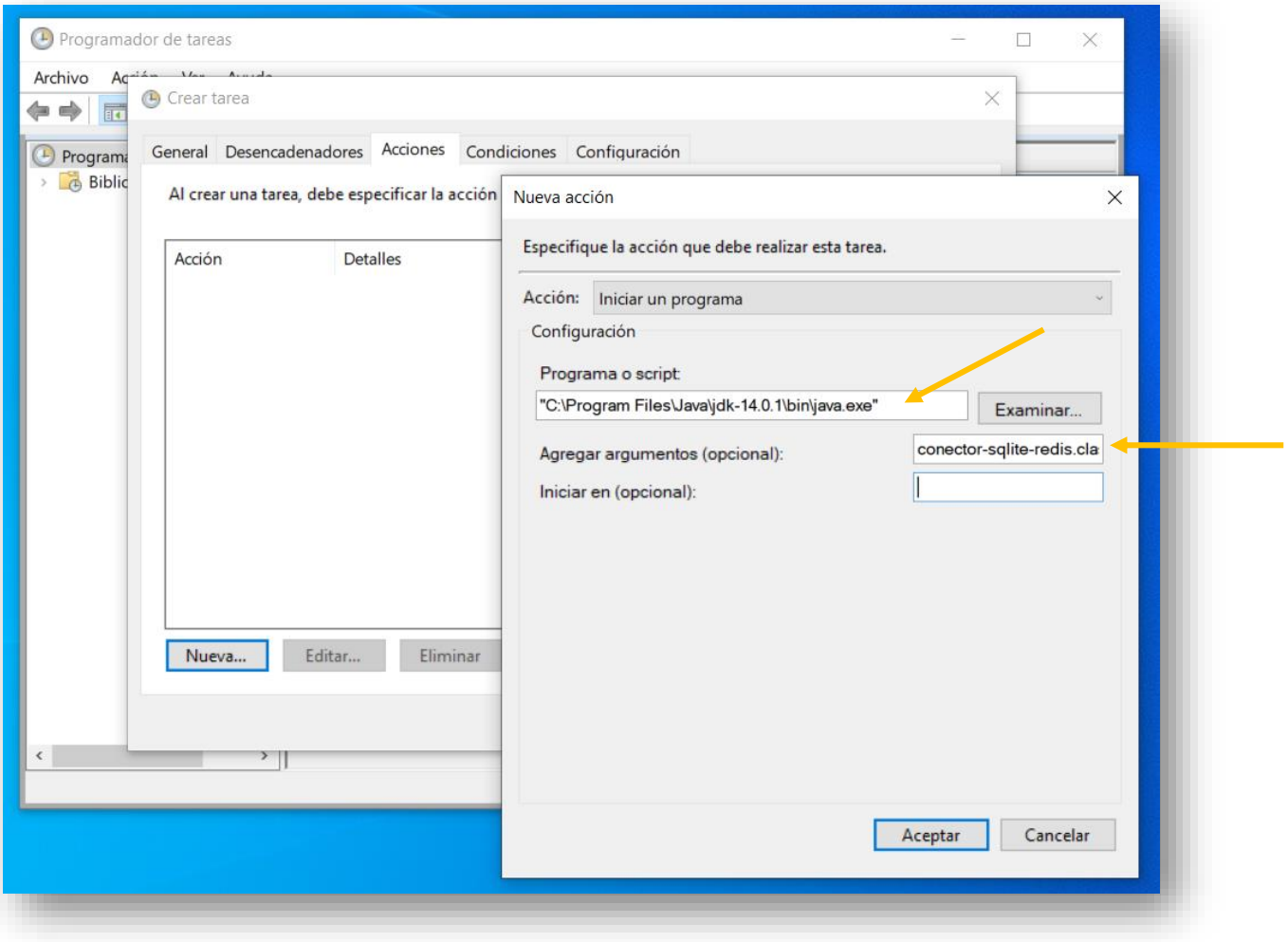

Change by clicking on the "Actions" tab and click on the "New" button:



We give the following parameters:

Program or script: connector_path

Adding arguments: connector name




The above parameters may vary depending on the location of the connector. The main idea is the execution of the **connector-sqlite-redis-v1.class** program as it is normally done on the command line:

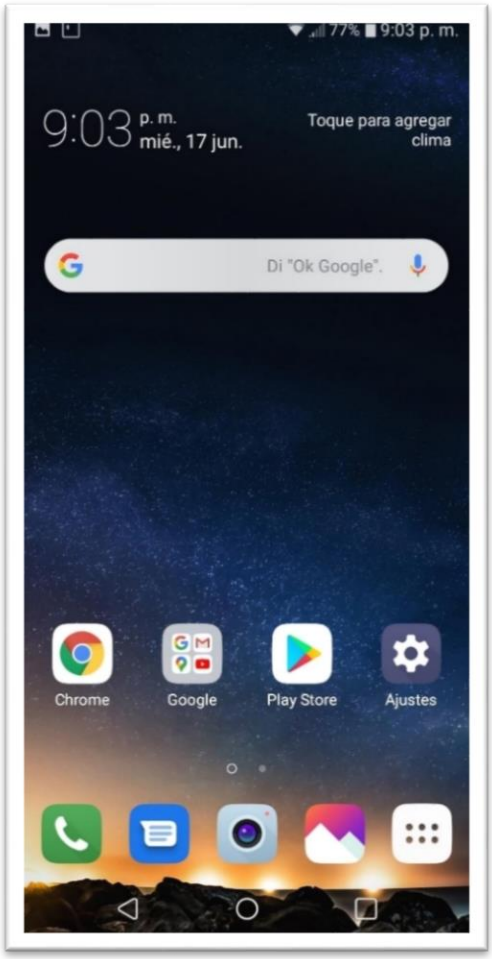
C: \jdk_directory java connector-sqlite-redis-v1

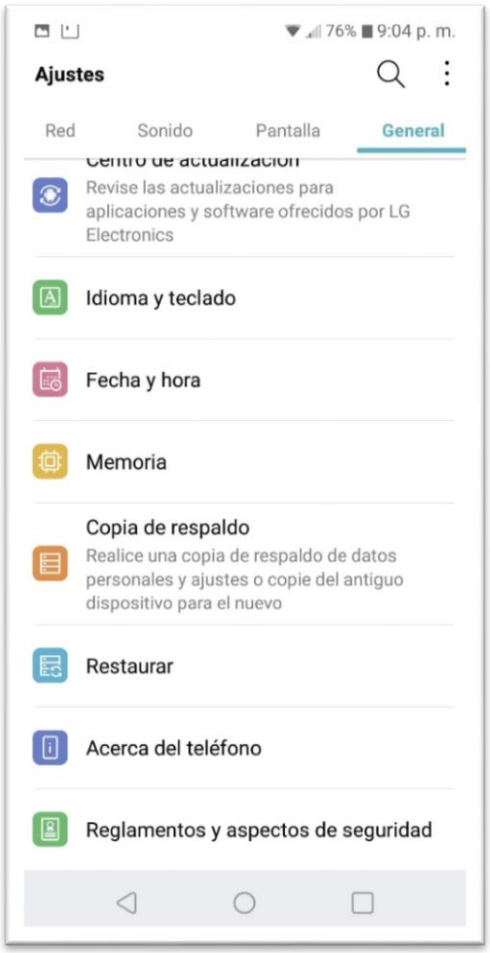
To finish we must choose the tab "Triggers", in this we will give the parameters of when (day, hour, minutes) we want the task to be executed, these parameters are based on the business rules that the Mini BlocklyChain system will have to be created.

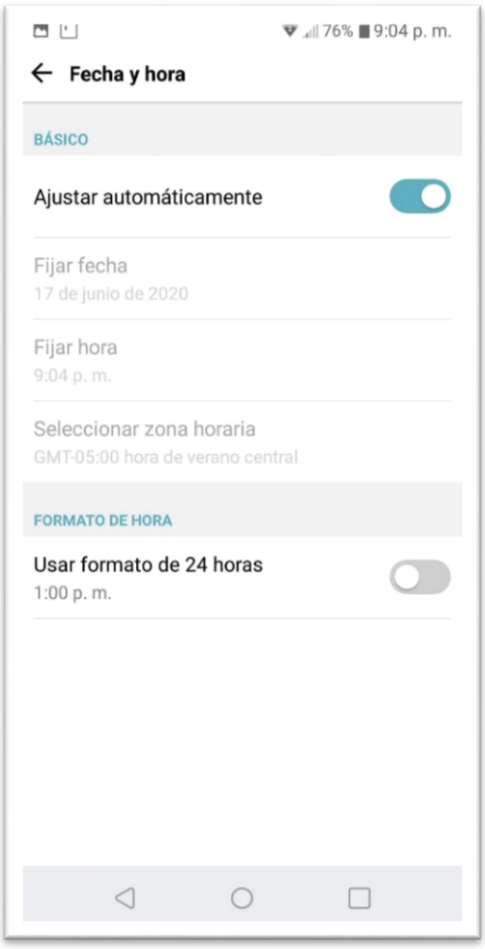
## 10.Synchronization in system nodes (mobile phone) minutes and seconds.

It is very important that all the nodes are synchronized mainly in the minutes and seconds part. Since when it is done the sending or publication in a determined time of the transaction queue all the nodes should be synchronized since this will depend on the task manager that will be established in the local system with the CRON tool to be executed at the same time in all the nodes, this will make that all the nodes have the same probability to be able to gain the right to be the chosen one to be able to process the transaction queue and to be able to generate the new block to add it to the block chain of the system. To synchronize the nodes we have two options.

The first option of the synchronization of the node, we will be able to do it in a simple way. In our device is through the internal option that includes the Android system, we will have to go to the part of **Settings > Date and Time > Adjust automatically**



With the previous configuration we will have synchronized in minutes and seconds all the nodes of the system no matter what country in the world are already synchronized is based on each geographical area possibly what changes is the time but depending on minutes and seconds should be synchronized this is enough for us to run the process of tasks on a scheduled basis with the cron tool in all the nodes to run every certain time in minutes, ie we can create a task in crontab to run every 10 minutes or 30 minutes depending on each system design.

This will be useful when using the "Peer to Peer" communication network, in case of using the backup network this process will not apply since the distribution of the transaction queue is done in a client-server model and the server is the one that controls the cron tool.

Reference: https://appinventor.mit.edu/explore/blogs/karen/2016/08.html

The second option is to use an external API where we will execute the Curl command through the extension (ConnectorSSHClient).

The place where we will be using the external services of NTP (Network Time Protocol) is:

http://worldtimeapi.org/

Now we'll look at a way to get the time from the NTP servers that are located worldwide and that will help us to get all the nodes to have a query at a certain time on the same date and time.

Example of a query with extension (ConnectorSSHClient).

$ curl "http://worldtimeapi.org/api/timezone/America/Mexico_City"

We made the connection to the Termux terminal:


We execute the Curl command:


We must take into account that the result of the Curl command will be in JSON format, something similar to the following result:


Also the result could be in JSON format without the data formatted in or JSON in linear form as shown below:


```
{"abbreviation":"CDT","client_ip":"200.77.16.151","datetime":"2020-06-
18T14:19:07.216800-
05:00","day_of_week":4,"day_of_year":170,"dst":true,"dst_from":"2020-04
05T08:00:00+00:00","dst_offset":3600,"dst_until":"2020-10-
25T07:00:00+00:00","raw_offset":-
21600,"timezone":"America/Mexico_City","unixtime":1592507947,"utc_datetim
e":"2020-06-18T19:19:07.216800+00:00","utc_offset":"-
05:00","week_number":25}
```


Either of the two previous ways we will have to filter the information through an existing JSON extension like "JSONTOOLS" or use filters in App Inventor in the text processing to only obtain the hour, day or seconds according to the need of each system. After processing the result, a logical comparison can be made and based on that comparison we can execute a task already programmed with the "cron" service that later on we will see its configuration in each node.

JSONTOOLS extension reference:

https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html

We have now reviewed two options for synchronising the time of the nodes. We will continue to configure the "cron" service on each node.

Configuration of automated task scheduling for execution with the CRON service on Android systems (system nodes).

First we need to understand how the automatic scheduler works.

The cron service normally all systems have this pre-installed and where all tasks to be executed automatically are scheduled are in a file called crontab.

We edit crontab file with $ crontab -e, we will use Vi editor, inside we use the format:

```
# m h dom mon dow user command
```

where:

- m corresponds to the minute in which the script will be executed, the value goes from 0 to 59
- h the exact time, the format is 24 hours, the values go from 0 to 23, being 0 12:00 midnight.
- dom refers to the day of the month, e.g. you can specify 15 if you want to run every 15th
- dow means the day of the week, it can be numeric (0 to 7, where 0 and 7 are Sunday) or the first 3 letters of the day in English: mon, tue, wed, thu, fri, sat, sun.
- user defines the user who will execute the command, it can be root, or a different user as long as he has permissions to execute the script.
- command refers to the command or absolute path of the script to be executed, example: /home/user/scripts/update.sh, if you call a script it must be executable

To make it clear a few examples of cron tasks explained:

```
15 10 * * * user /home/user/scripts/update.sh
```
You will run the update.sh script at 10:15 a.m. every day

```
15 22 * * * user /home/user/scripts/update.sh
```
You will run the update.sh script at 10:15 p.m. every day

```
00 10 * * 0 root apt-get - and update User root
```
You will run an update every Sunday at 10:00 a.m.

```
45 10 * * sun root apt-get - and update
```
Root user will run an update every Sunday (sun) at 10:45 a.m.

We saved changes in the editor and with this we finished the configuration of the cron service. In case you don't have the cron installed in the system you can do it with the following command:

$ apt install cron

2. Installation and configuration of network Nodes - Mobile phones.

Let's start with the communications network for nodes that will use Mini BlocklyChain.

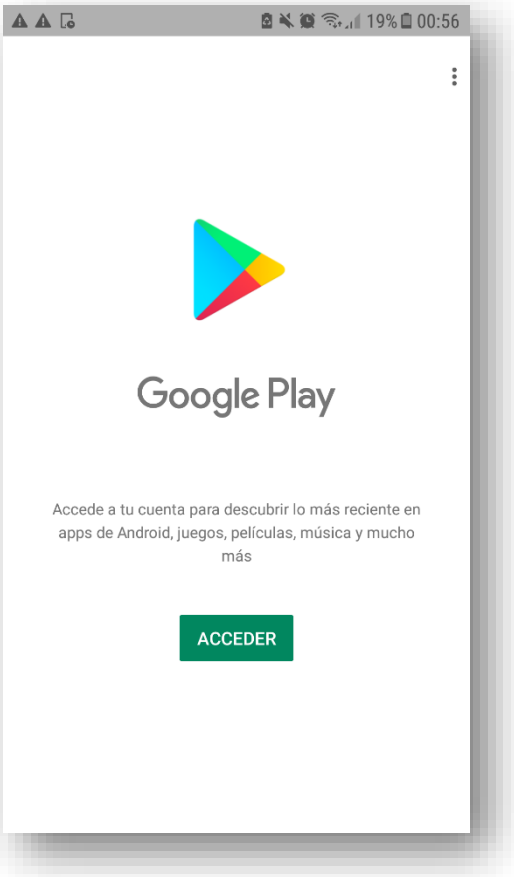
First we need a Linux environment since every Android system is based on Linux for security and flexibility in tools, we will use the "Termux" terminal that contains that environment where we will install the communications network.

Termux is a Linux emulator where we will install the necessary packages to create our communication network between nodes.

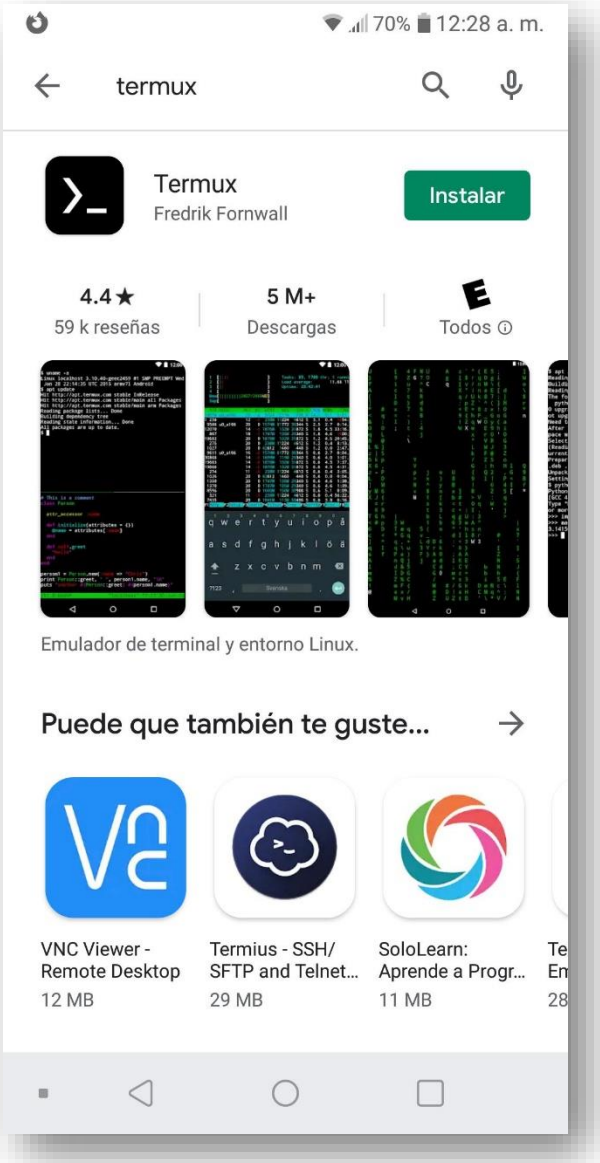
One of the main advantages of using Termux is that you can install programs without having to "rotate" the mobile phone (Smartphone). This ensures that no manufacturer's warranty is lost due to this installation.

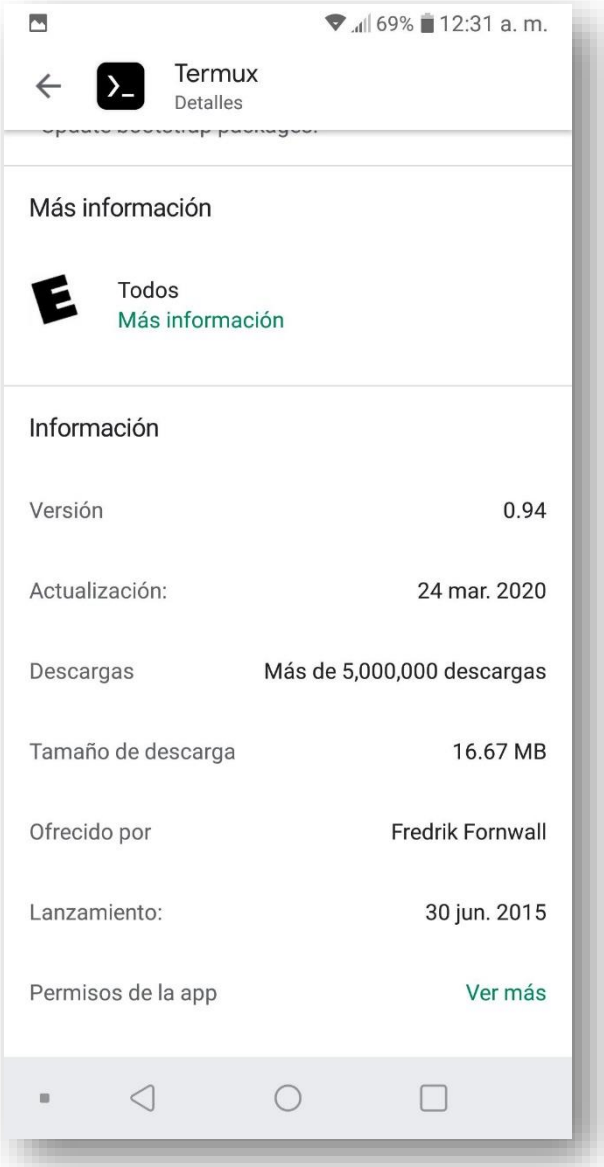
Termux installation.

From your mobile, go to the Google Pay icon application (play.google.com).



Search by application "Termux", select it and start the installation process.



Start of the Termux application.

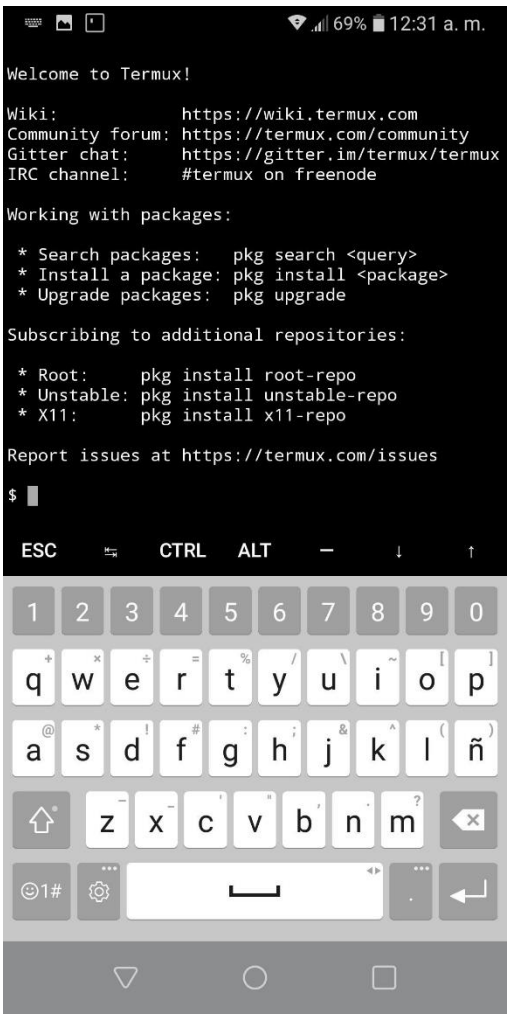
After starting we will have to execute the following two commands to perform updates of the Linux operating system emulator:

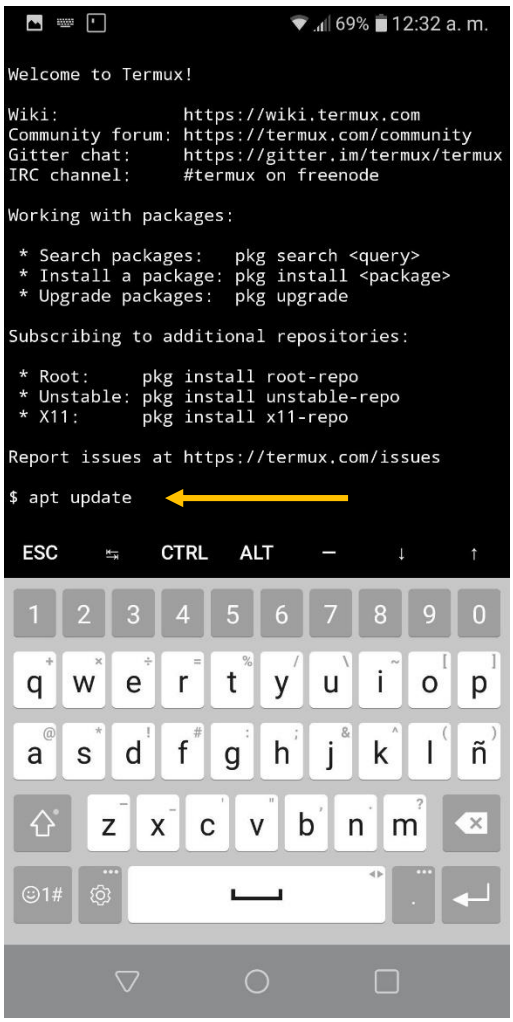
$ apt update

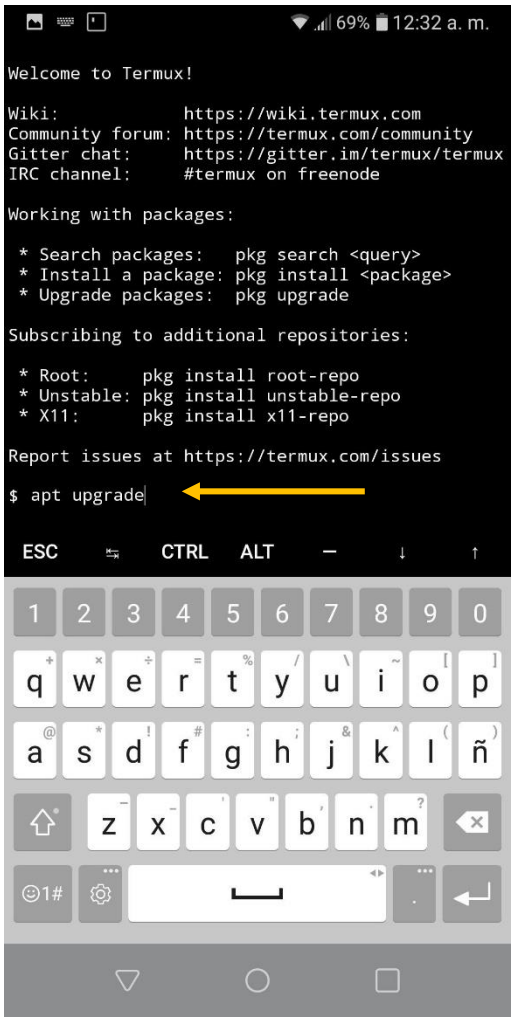
$ apt upgrade

Confirm all options Y(Yes)...

Termux Home $ apt update $ apt upgrade



## 11. Storage configuration within Termux.

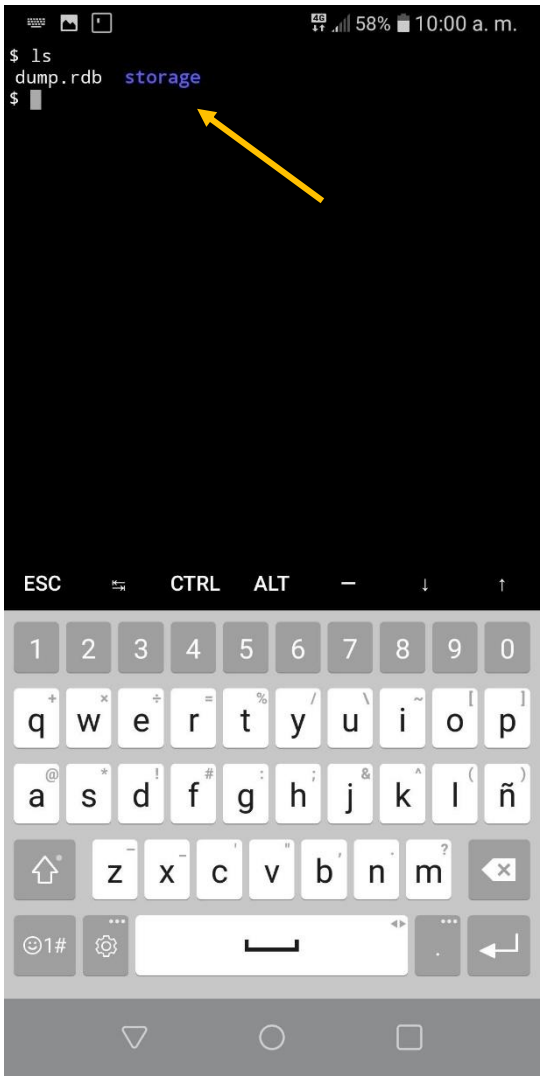
After you have updated and upgraded the Termux system, we will start configuring how to view the internal storage of the phone in the Termux system. This will help you to be able to exchange information between Termux and our information in the phone.

This can be done simply and quickly by running the following command on a Termux terminal.

$ termux-setup-storage

When you execute the previous command, a window appears asking you to confirm the creation of a virtual **storage** (directory) in Termux. We verify by giving the command:

$ ls

## 12. “Tor" network installation and "Syncthing" installation.

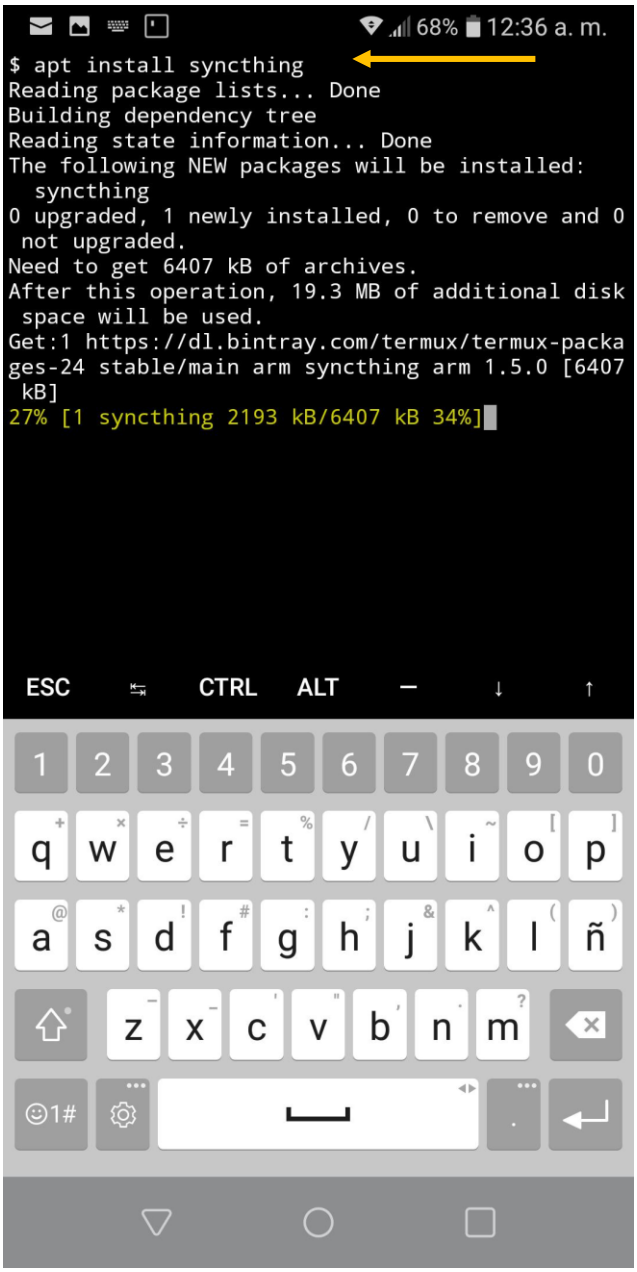
$ apt install tor

$ apt install syncthing

Accept installation by typing capital Y in both cases if requested...

$ apt install tor


$ apt install syncthing



## 13. Installation of "Redis" database and SSH (Secure Shell) server.

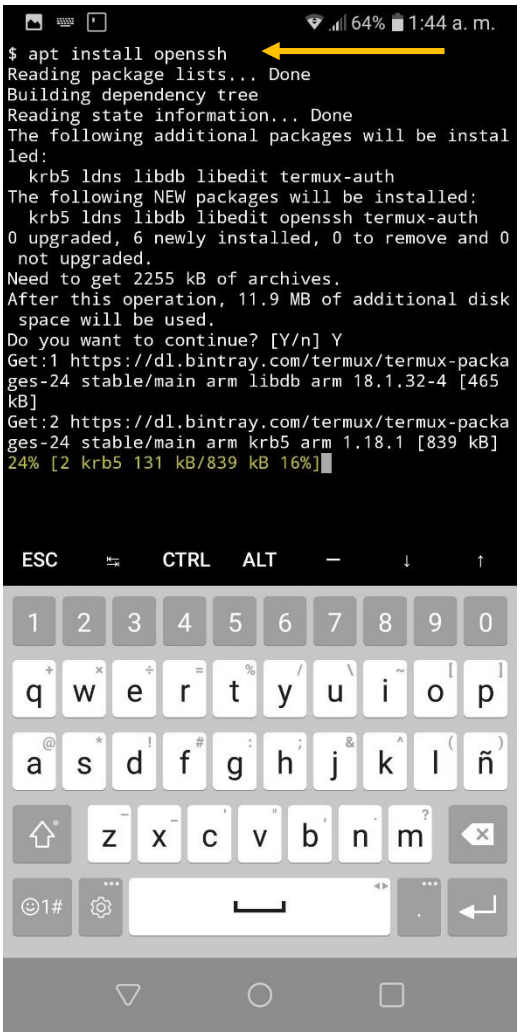
$ apt install redis

$ apt install openssh

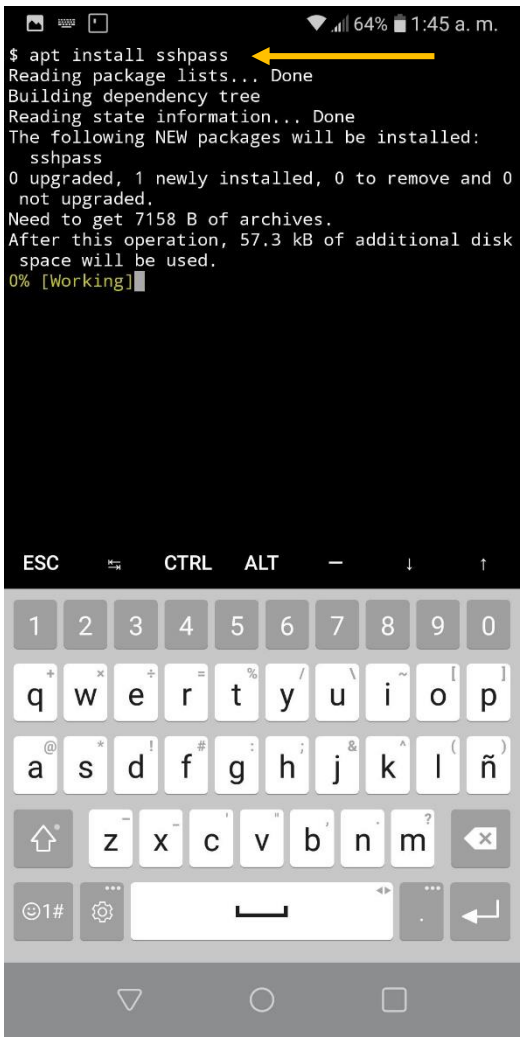
$ apt install sshpass

$ apt install redis


$ apt install openssh




$ apt install sshpass




We have finished with the installation of the communication network, we continue with the configuration of the packages: Tor, Syncthing and Redis DB.

## 14. SSH server configuration on mobile phone (smartphone).

We will enable the SSH server in the mobile phone to be able to connect from our PC to the mobile and to be able to work in a faster and more comfortable way, also it will serve to check that the service of the SSH server in the mobile works correctly since we will use it in the communication network in the Mini BlocklyChain.

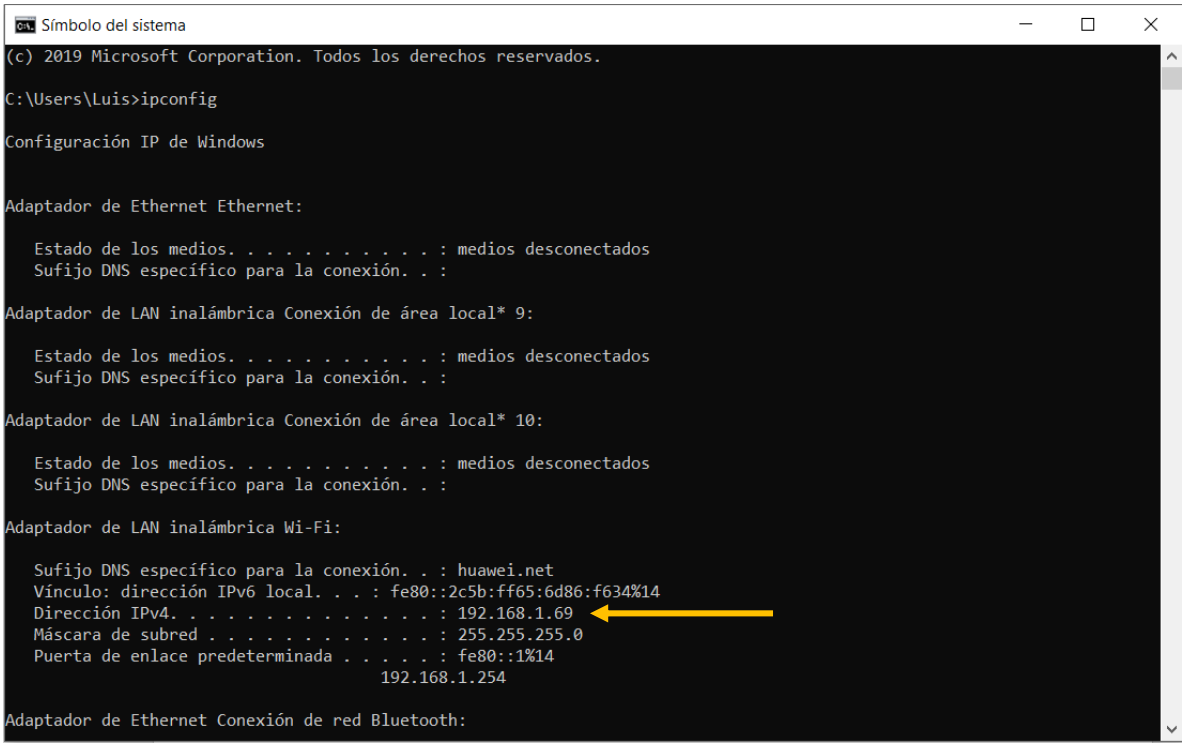
The first thing we have to do is connect the mobile and the PC to the same WiFi network so that they can see each other. The IPs or addresses must be similar to 192.168.XXX.XXX the XXX values are variable numbers that are assigned randomly in each computer.

This example was tested on an LG Q6 mobile phone and a PC with Windows 10 Home.

Check the IP or address that the PC has connected to the WiFi we must open a terminal in Windows.

In the bottom panel where the search magnifier is write cmd and press the Enter key. A terminal will open and in it we write the command:

C:User_Name> ipconfig




It will show us the IP assigned to the PC in this is 192.168.1.69 but this is most likely to be different in each case. NOTE: The address where it says "IPv4 address" should be taken, not to be confused with the Gateway.

Now in the case of the mobile phone in the Termux terminal we must type the following command to know the name of our user that we will use to connect to the SSH server that has our phone, we execute the following command:

Now in the case of the mobile phone in the Termux terminal we must type the following command to know the name of our user that we will use to connect to the SSH server that has our phone, we execute the following command:

$ whoami

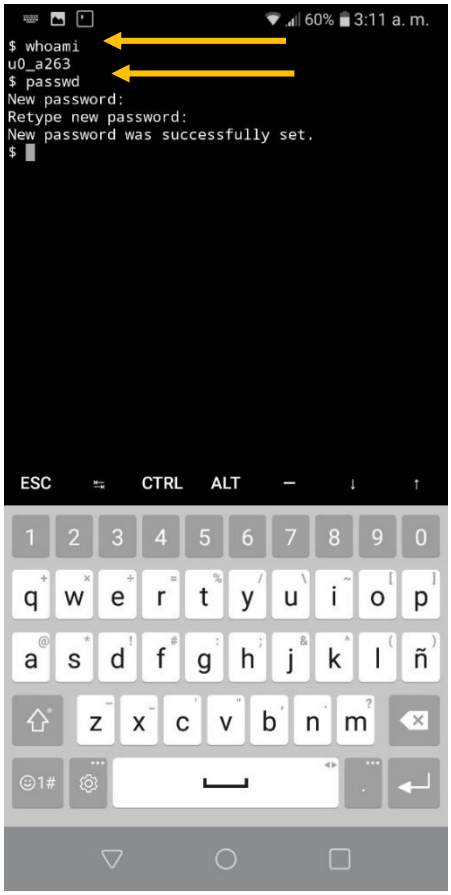
Later we must give a password to this user so we have to execute the following command:

$ passwd

It will ask us to type a password and hit Enter, again it asks us for the password we confirm it and hit Enter, if it has been successfully "New password was successfully set" in case of marking an error is possible that the password has not been typed correctly. Perform the procedure again.

And then to know what IP we have in Termux we type the following command, the IP is after the word "inet":

$ ifconfig -a



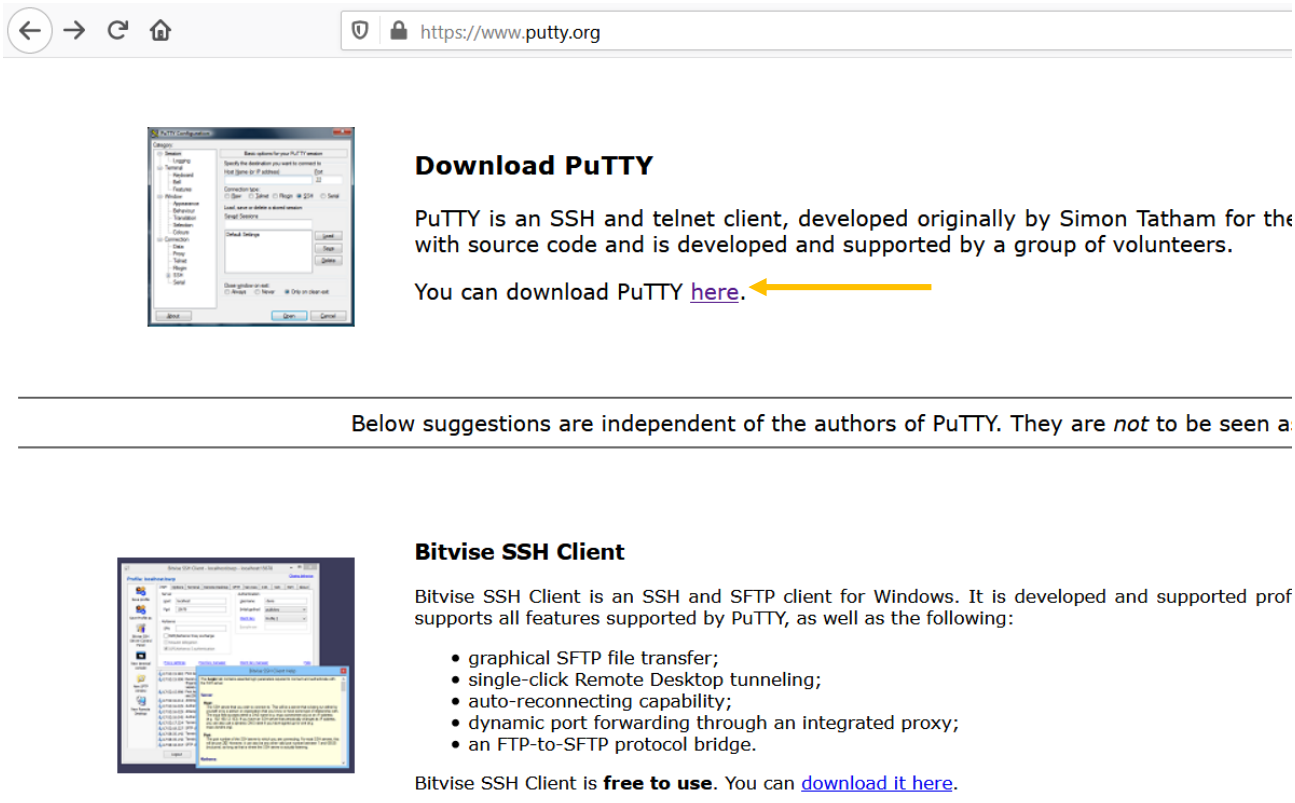

Now it's time to start the SSH server service on your phone so you can receive sessions from your PC. We execute the following command in the Termux terminal, this command does not give any result.

$ sshd

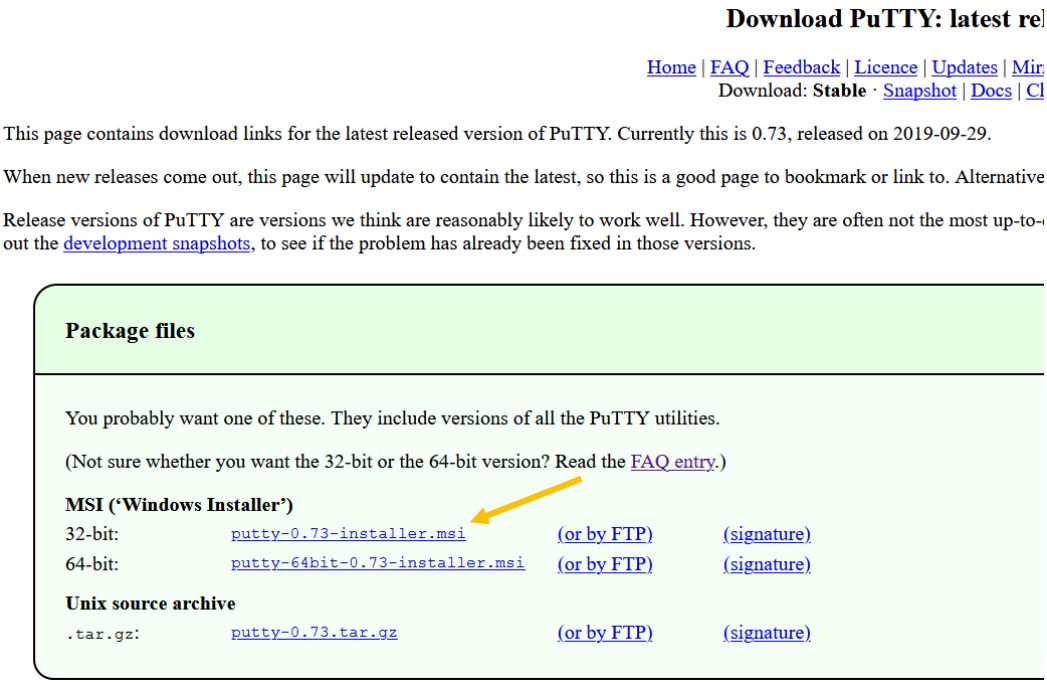

Now we will have to install a program on the PC that will communicate with the phone's SSH server from the PC.

We have to go to [https://www.putty.org](https://www.putty.org)

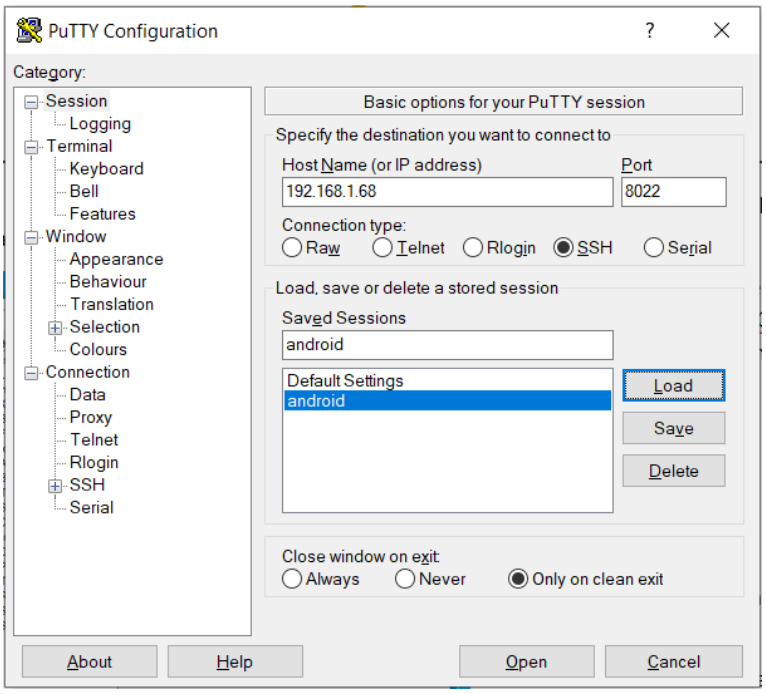
Select where the link "You can download PuTTY here" is



Choose the 32-bit version, it doesn't matter if your system is 64-bit will work fine.




Once it has been downloaded to your PC, run it and install it with the default options. Then start the PuTTY application.




In this session we will enter the data from our Openssh server that we installed in the mobile phone.

Enter the IP of the mobile phone.

HostName or IP address:

192.168.1.68 (IP example)

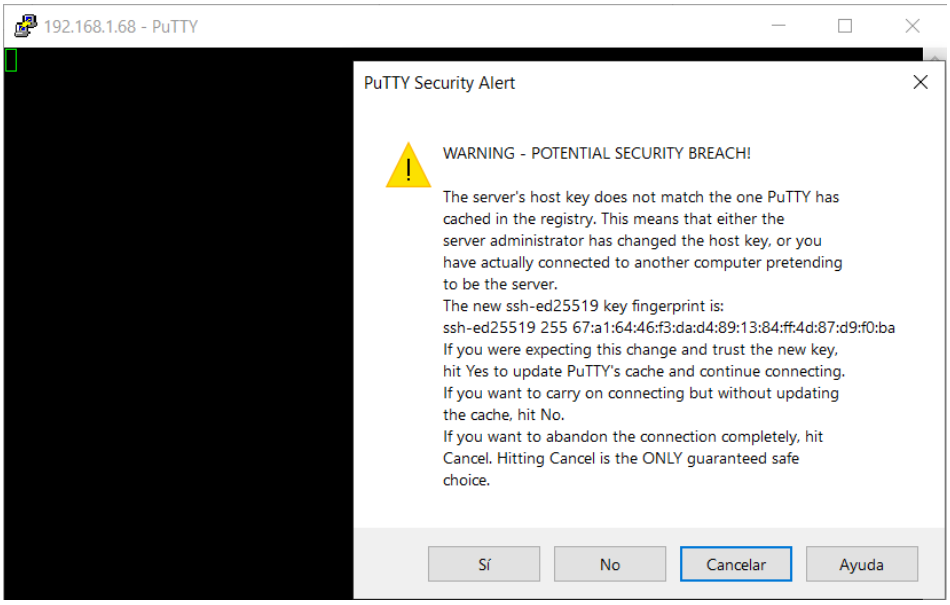
Port:

8022 (Default port of the mobile SSH server).

We can give a name to the session in "Saved Sessions" and click on the Save button.

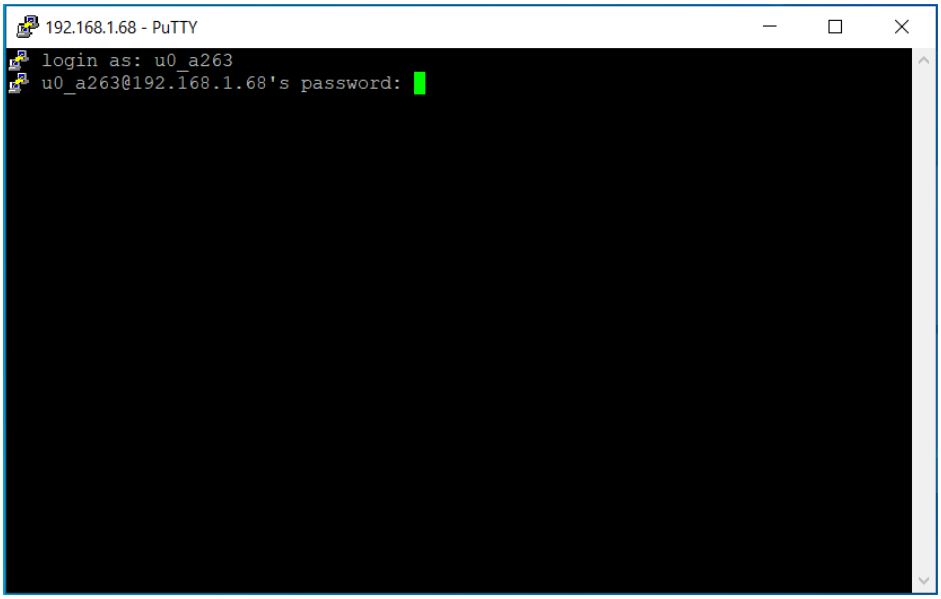
Later on in the lower part we press to open a connection to the server giving the button "Open".

On the PC when you connect for the first **time you will be asked for confirmation of** the information encryption key by clicking on the "Yes" button.

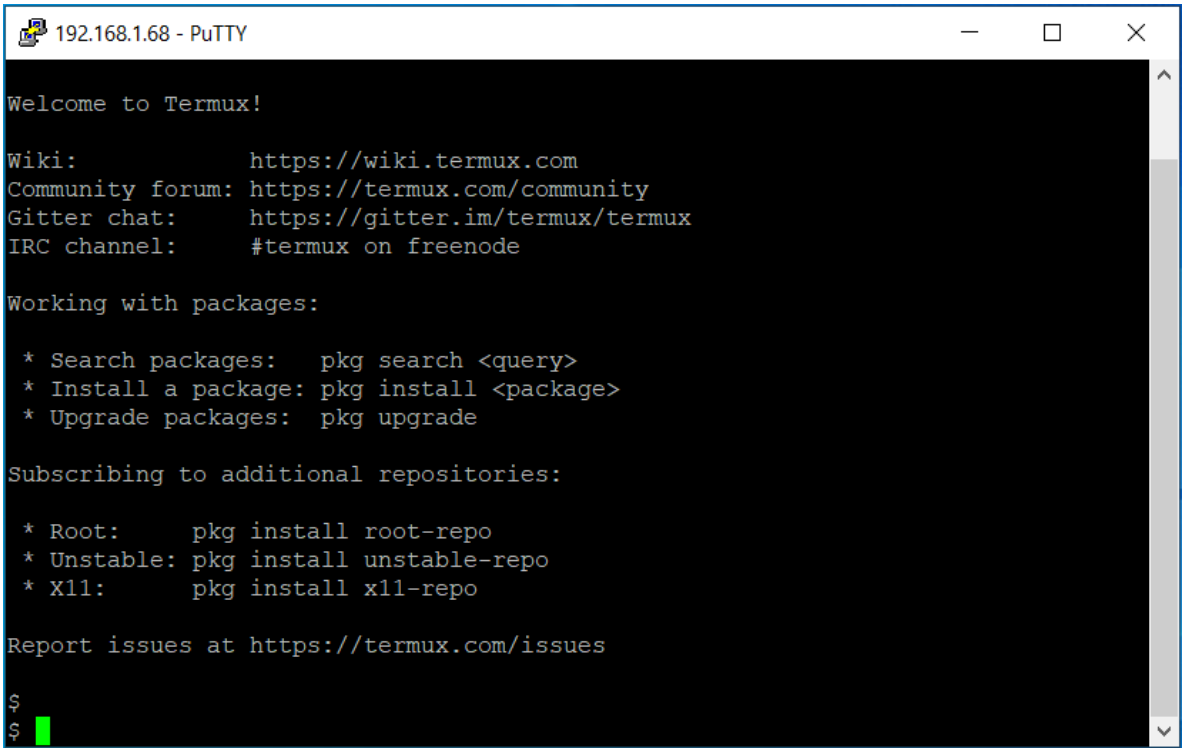



Later on, we will be asked for the user we are going to connect with. We will use the information that we previously obtained (user and password).

In the **Login as:** we must enter our user and give Enter, then we will ask for the password again give the Enter button.



If the data was correct, we will be in an SSH (Secure Shell) session performed from the PC (Client) on the phone (SSH Server).




IMPORTANT NOTE: Remember that the IP (address) of the PC and the IP (address) of the mobile phone connected to the same WiFi will probably be changing every time we disconnect and reconnect so we have to double check what addresses each device has, this will ensure the success of the connection between devices through the phone's SSH server and PC (Client).

So far we've only been able to connect to the same WiFi network, but if we move our phone outside the same network as the PC, we won't be able to connect because there are different networks involved in going through other more complicated communication devices. We'll solve this when we set up the "Tor" network.

Remember that this connection is made only to verify the service of the server we installed on the phone and to have a more comfortable working environment with a remote session from a PC to the phone.

## 15. “Tor" network configuration with SSH (Secure Shell) service.

With the remote session from the PC we will start configuring the "Tor" network with the SSH service enabled.

The importance of having the "Tor" network is to give the devices the property of being able to communicate anywhere in the world through the Internet without being in the same WiFi network, no matter where we are we will be able to connect and form the "Peer to Peer" network between nodes that is an essential functionality of the Mini BlocklyChain architecture.

The "Tor" network has a lot of flexibility in its configuration since it has several parameters in its configuration file "torrc" this file is in the path (`$PREFIX/etc/tor/torrc`) in our case with the Termux session from our PC we will know by typing the following command:

$ echo $PREFIX

/data/data/com.termux/files/usr

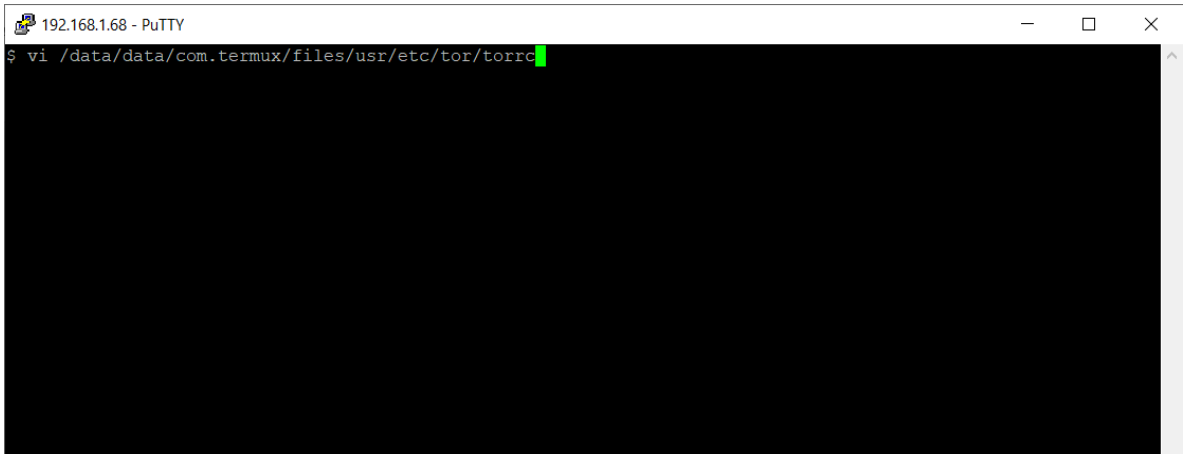
Therefore the file we need to edit will be in the path:

PREFIX/etc/tor/torrc which is equal to /data/data/com.termux/files/usr/etc/tor/torrc

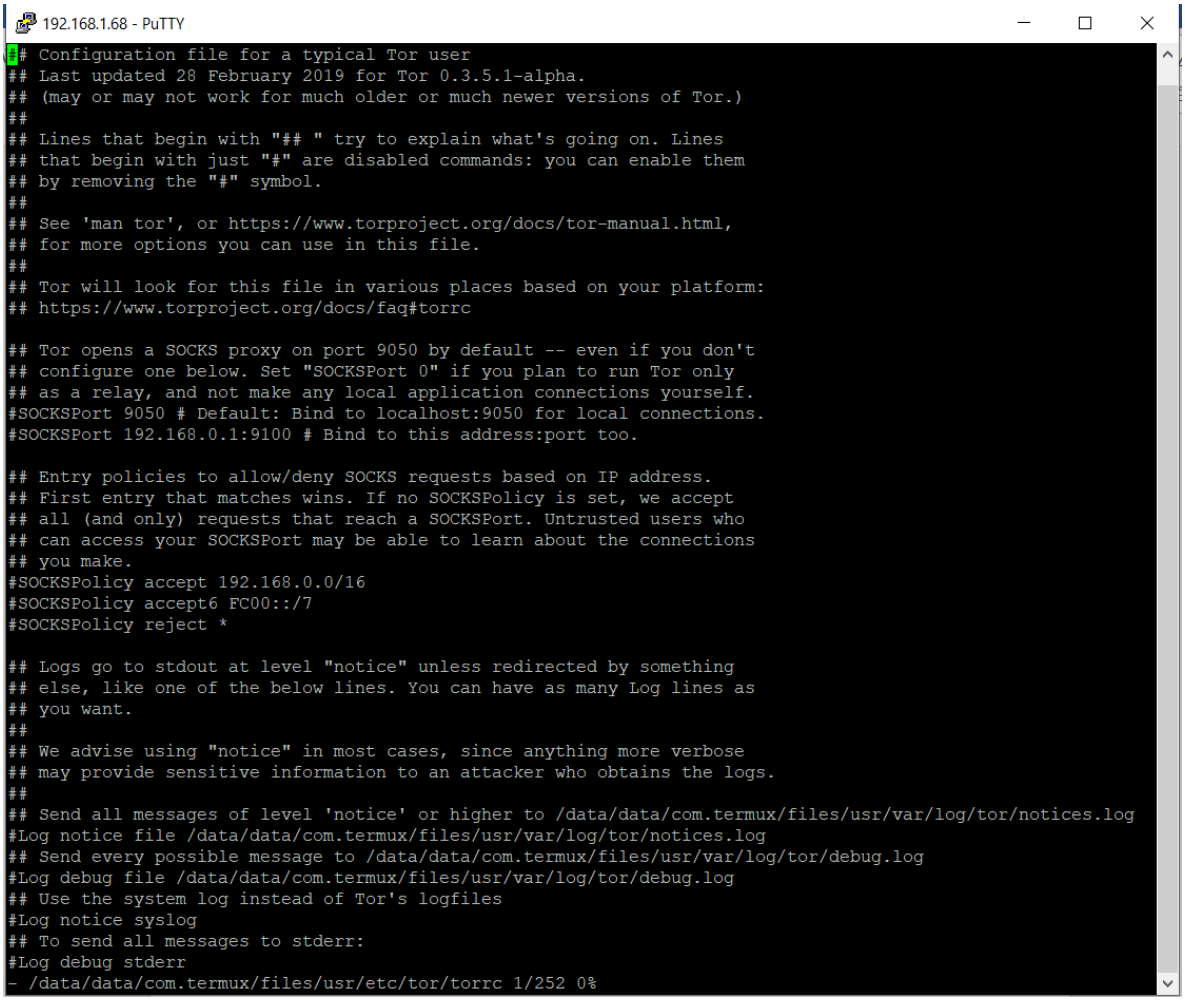
We proceed to edit the configuration file using the command line editor "vi" by executing the following command:

vi /data/data/com.termux/files/usr/etc/tor/torrc

He will edit that file for us, as an example:



Edited **"torrc"** file:




In this "torrc" file we will have to add or use the lines that the file has by making the following changes, three lines that are the following:

Syntax: SOCKSPort <application port number>

Example: SOCKSPort 9050

The **SOCKSPort** variable tells us that this communication socket over the TCP-IP protocol will use the mobile device (phone) by default and port 9050 will be opened or in use.

Syntax: HiddenServiceDir <Directory where the application's configuration will be saved>

Example: HiddenServiceDir /data/data/com.termux/files/

The HiddenServiceDir variable tells us that this will be the directory where the configuration of the service that will be used through the Tor network will be stored. In this directory you will find the configuration and security file for the service, and in this directory you will find a file called hostname.

We can see the address assigned by the Tor network for the specific service created in our case is creating an SSH service that will be implemented on the Tor network, the directory we're naming as hidden_ssh will contain the hostname file. This directory does not need to be created, because when we start the Tor network service it will be created automatically.

To see the address provided by the Tor network, we can use the "more" command. Before we use this command, we must finish configuring the "torrc" file and register the Tor network service so that the hidden_ssh directory and the files inside it are created, as is the case with the hostname file.

more /data/data/com.termux/files/

The above command will result in an address with an alphanumeric string with an extension . onion similar to :

uqwthf6ojdmoybyzvudoq3x2weq7k7rjitblhba3pjbawk2nvjmx3wer.onion

Finally we need to add the HiddenServicePort variable to the "torrc" configuration file. This parameter tells the Tor network which port the application we're signing up for will use over the Tor network.

Syntax: HiddenServicePort <Port of departure> <Local or public IP>: <Port of departure>

O

HiddenServicePort <Port of departure>

Examples:

HiddenServicePort 22 127.0.0.1:8022

O

HiddenServicePort 8022

With the above we have finished configuring the Tor network for generic services and SSH (Secure Shell) service. This service will be used for SQLite database update communication where we will be saving the validation processes of our Mini BlocklyChain system.

We continue with the configuration of the Mini BlocklyChain communications network.

## 16. Peer to Peer system configuration with manual syncthing.

A "Peer to Peer" architecture is fundamental in a blockchain technology system because this architecture gives two central points in the system communication processes, one is to provide the same updated (synchronized) information at any time in all the nodes no matter if the nodes are in a private network (Wifi) or a public network (Internet) or a hybrid of these two and a second point of this type of architecture is that they do not depend on an intermediary (server) to transfer, update or consult information between nodes. All this is due to the fact that the communication is done directly between the nodes, in our case Mini BlocklyChain the communication is done directly between the phones that form the network without an intermediary.

For this task we will use the opensource Syncthing tool that works based on a "Peer to Peer" architecture.

The configuration of Syncthing for devices (mobile phones) will be seen manually and automatically. The manual form is applied in synchronization with up to 5 nodes, in case of having a large number of automatic configuration is optimal, the process focuses on the registration of the nodes with which we want to perform the synchronization of the selected information.

The requirements to start with are:

1. Have initiated service of the Tor network.
2. (optional - recommended) Have installed an application that can read QR codes, we suggest the App inventor Android application that is easy and simple to install from Google Play and later in this manual we will use it in the section of "Definition and use of blocks in Mini Blocklychain".
3. To have initiated the service of Synthing.


We show the App Inventor application that we will use for reading QR codes that will serve us in the future for the registration of nodes in Syncthing.

The following example was made between two mobile devices (phones) using the following models:

Mobile device #1: Model LG Q6

Mobile device #2: Sansumg model

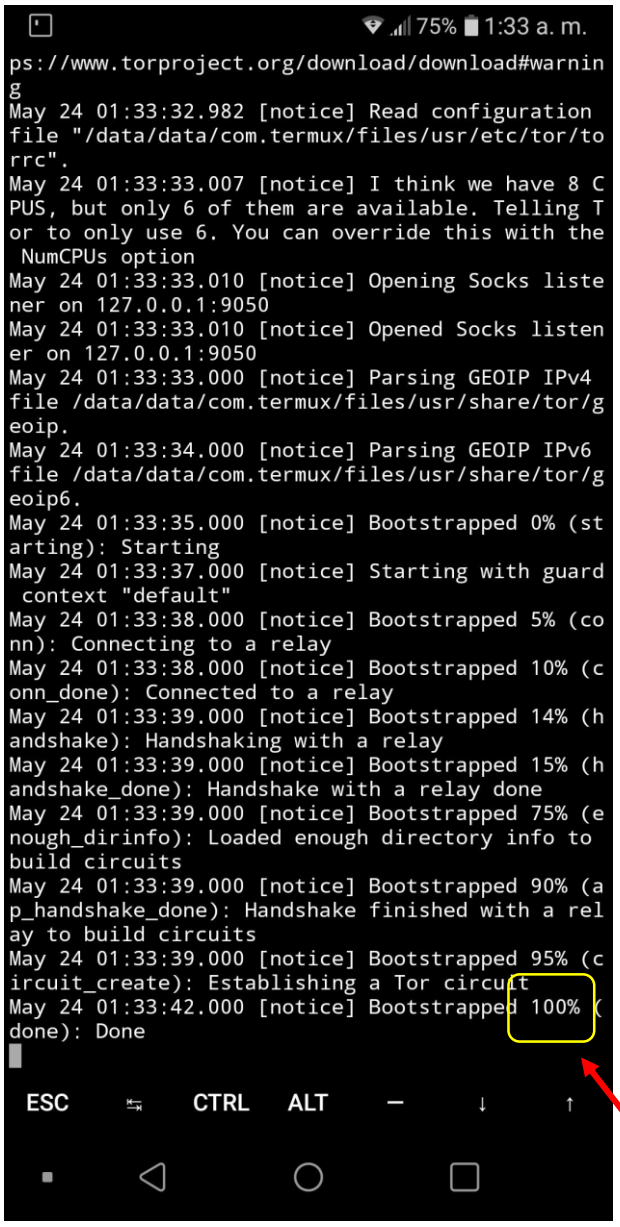
We'll start with starting Tor network services and syncthing services using the following commands, open two terminals inside Termux, and run each command separately:

$ tor

After you have typed the Tor network program startup command, you should check that the execution was successful, by checking that it was done 100%.

Running the Tor program on a Termux terminal, we verify that it is running at 100%.

$ tor

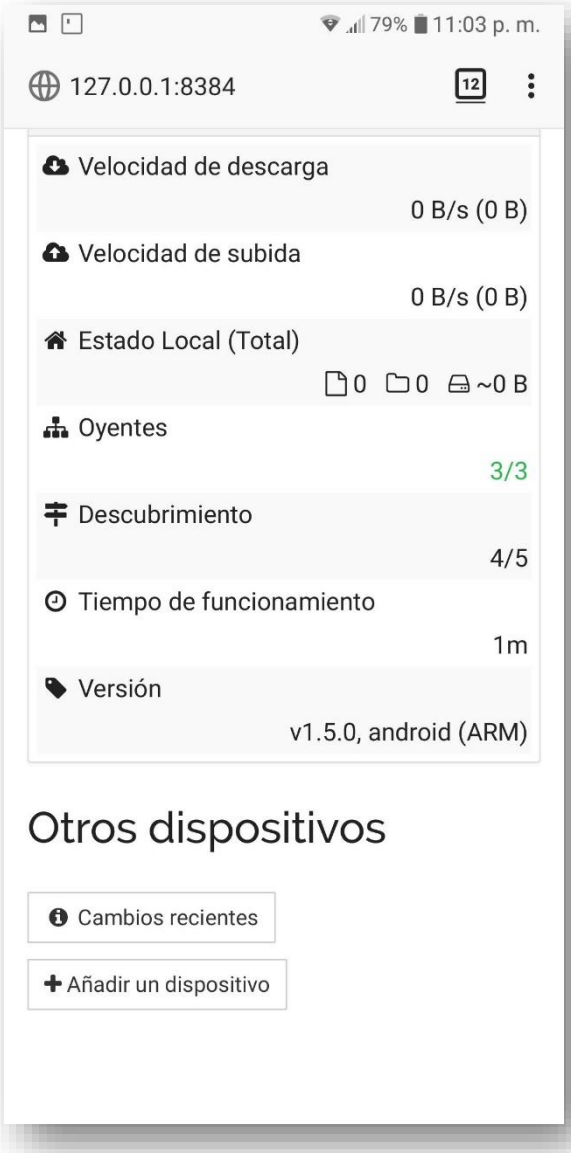

Then we execute the syncthing command:

$ syncthing

After executing this command it will open an administration page in the browser of our phone if it does not open automatically, we can go to any browser that we have installed where we normally browse the internet and we can put the following l

http://127.0.0.1:8384

It will open the administration screen of the tool that will help us to synchronize our information between all the nodes (phones) of the system.

We have the "syncthing" administration screen open, we proceed to register the node or nodes that we want to synchronize the information. At this point is where we will occupy the program that reads QR codes.

The syncthing program when it starts for the first time creates a unique identifier of the phone that is composed of a group of eight sets of alphanumeric characters in capital letters, this identifier (ID) is the one that we will register in the node or nodes that we want to synchronize the information.

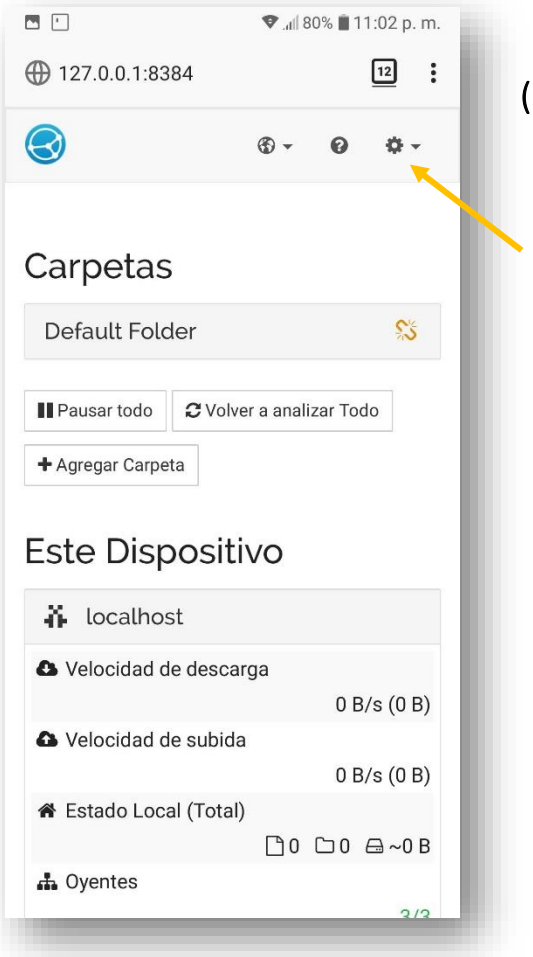
In our case the LG Q6 phone ID will have to be registered to the Sansumg phone and the Sansumg phone ID will have to be registered to the LG Q6. They must be in both phones in order to work properly.

We will perform the steps of the Sansumg mobile phone registration on the LG Q6 phone.

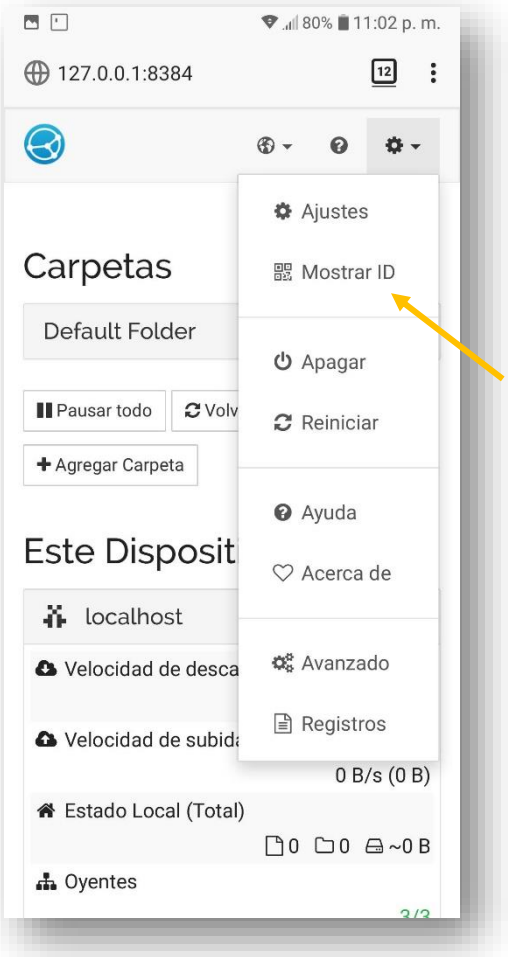
First (1) on the top of the administration screen (internet browser) of the Sansumg phone with syncthing we will click on the menu tab on the top right side.

Second (2) step we will see a menu, in this we click on "Show ID" of the Sansumg.

(1)




(2)

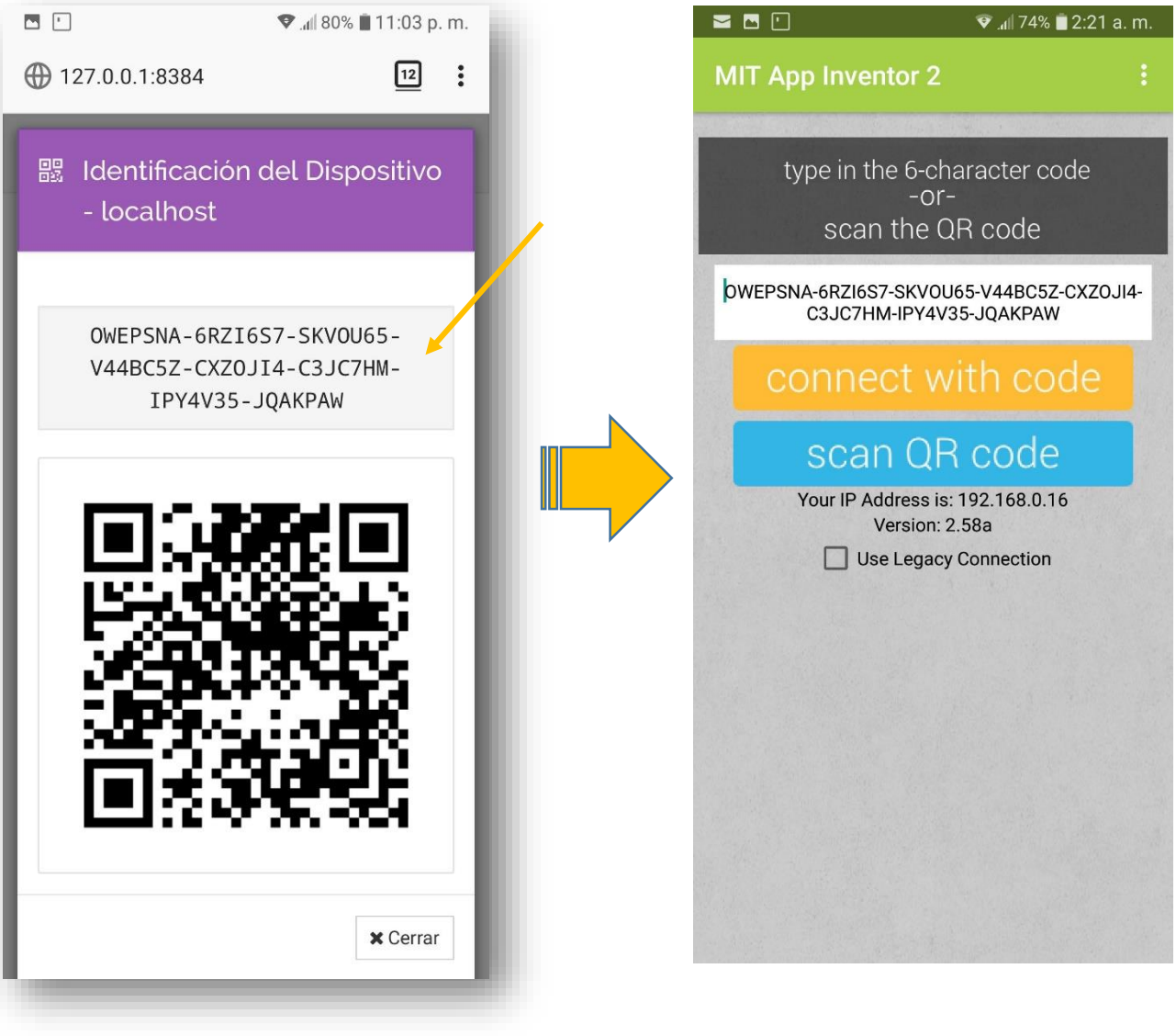


When you click on "Show ID" the following screen will appear which is a QR code of the Sansumg phone in our case the ID that identifies the phone is

OWEPSNA-6RZI6S7-SKVOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW

Then in the LG Q6 phone, the program that will help us to capture this Sansumg's QR code is the one we suggest from the App Inventor application (optional), although in case we don't have it we can also simply introduce it manually when we start the registration in the LG Q6, however, to avoid any error we suggest to use it.

Using App program inventor installed in the LG Q6 mobile to capture QR code from the remote Sansumg phone we want to synchronize.



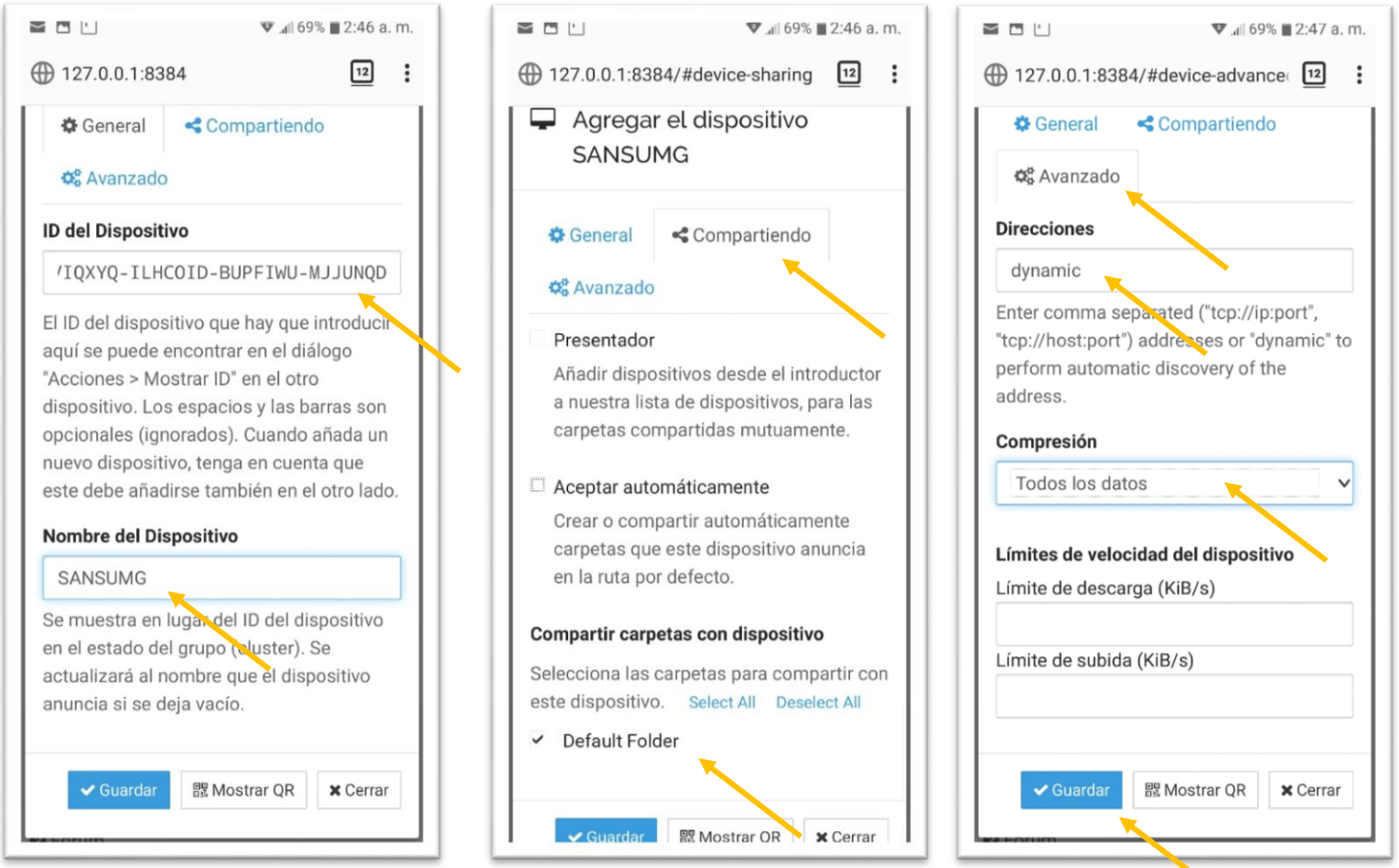
Then we copy to the clipboard the QR code in the App inventor program by clicking on the captured code, we choose "SELECT ALL" →"COPY".

With this information we proceed to register the Sansumg in the LG Q6. In the administration screen we move to the lower part where it says "Other devices" and click on the button "Add a device", we introduce the Sansumg's ID and give it a registration name.

Then in the upper tab "Share" we click and in this we mark the lower option in folder "Default" with this we will be sharing a directory that was created by Default called Sync in our device.

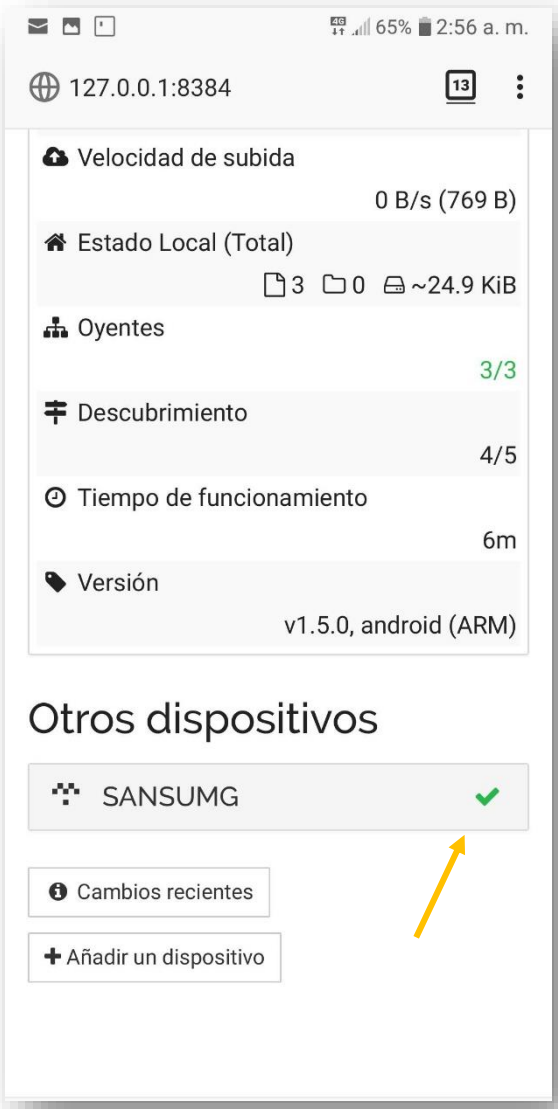
Then at the top we click on the "Advanced" tab and select the data compression option under "All Data".

Finally we click on the lower save button, we finish the registration in a node, this same process must be done in the remote phone or phones that we want to synchronize.

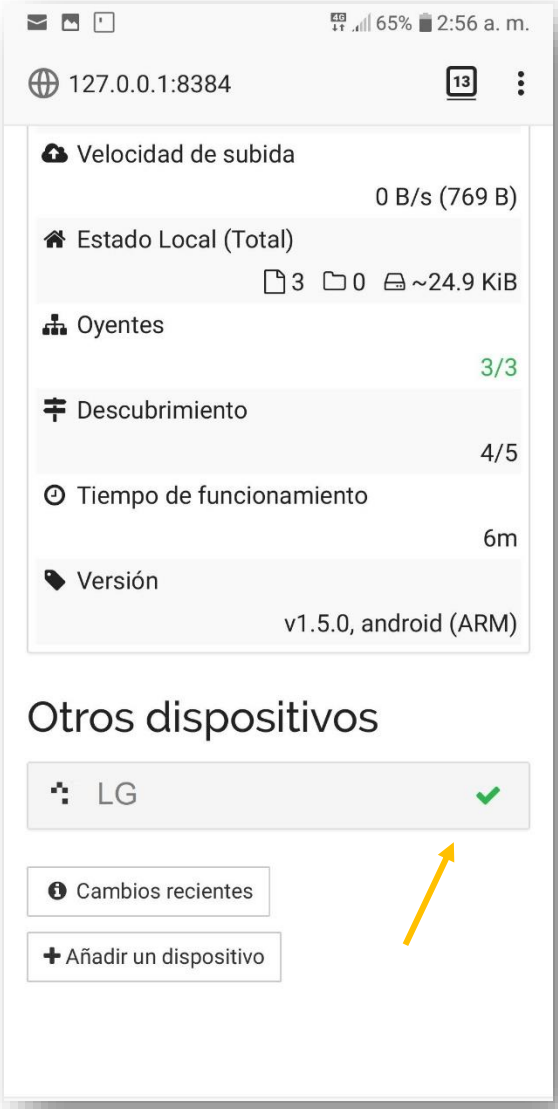



When saving and registering in both phones we will wait a maximum period of 30 seconds in which the devices are located and the connection between devices will appear as good giving a confirmation in green.

Device #1 LG Q6




Device #2 Sansumg




All information in the **Sync** directory located in the path **/data/data/com.termux/file/home/Sync** will be synchronized, any changes will be copied and synchronized.

## Peer to Peer system configuration with Syncthing automatically for use in Mini BlocklyChain.

Now we will make the configuration in an automatic way, previously we made the manual process this is useful if we handle a minimum amount of nodes, however, in the case of having a great amount of nodes it would be inefficient so we will have the SyncthingManager tool later to configure it in an automatic way using automated online commands.

## Redis database configuration for (Slave) mode nodes.

Configuration of the **/data/data/com.termux/files/usr/etc/redis.conf** file of the Redis **(Slave)** database for Android mobile phones.

Add the following changes or directives to the file, save changes and start Redis server.

First, find and remove the # character from the **slaveof** line. This directive takes the IP address and port that you use to securely contact the Redis master server, separated by a space. By default, the Redis server listens on 6379 on the local interface, but each of the network security methods changes the default in some way for others.

The values you use will depend on the method you used to protect your network traffic:

Isolated network: use the IP address of the isolated network and the Redis port (6379) of the master server (e.g., slave of simple IP_address 6379).

stunnel or spiped: use the local interface (127.0.0.1) and the configured port to tunnel the traffic

PeerVPN: Use the IP VPN address of the master server and the normal Redis port.

The general would change it:

**slaveof ip_contact_server** master_contact_port

Example: **slaveof** 192.168.1.69 6379

**masterauth** your_network_master_password

Example: **masterauth** sdfssdfsd12WqE34Rfgthtdfd

**requirepass** your_network_slave_password

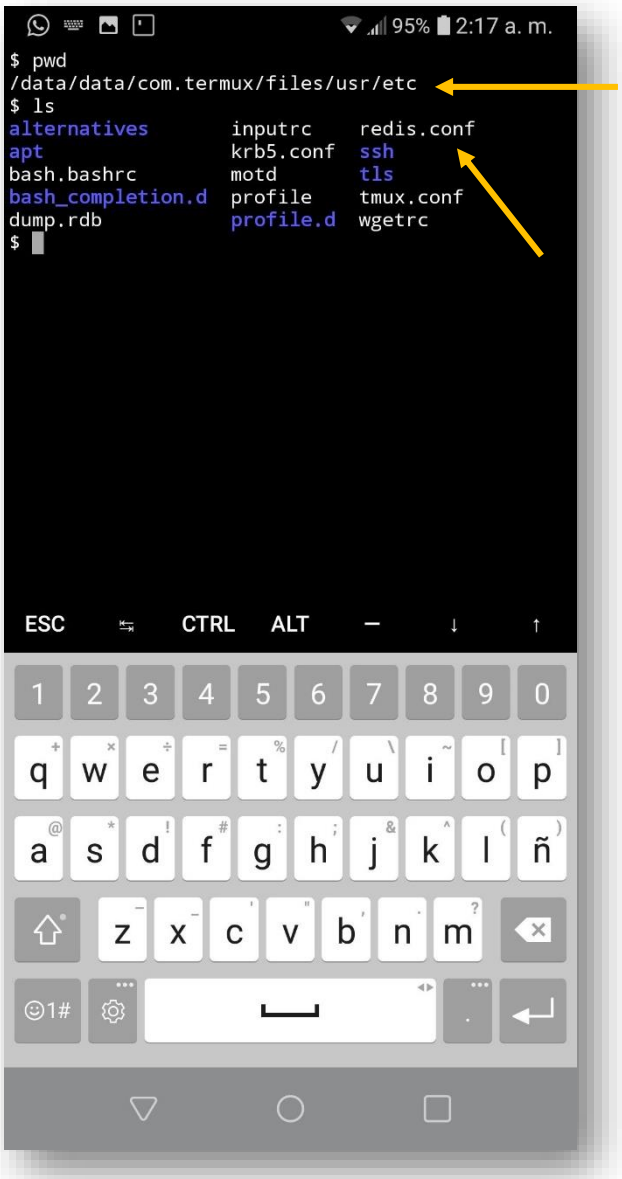
Example: **requirepass** asdsjdsh34sds67sdFGbbnh

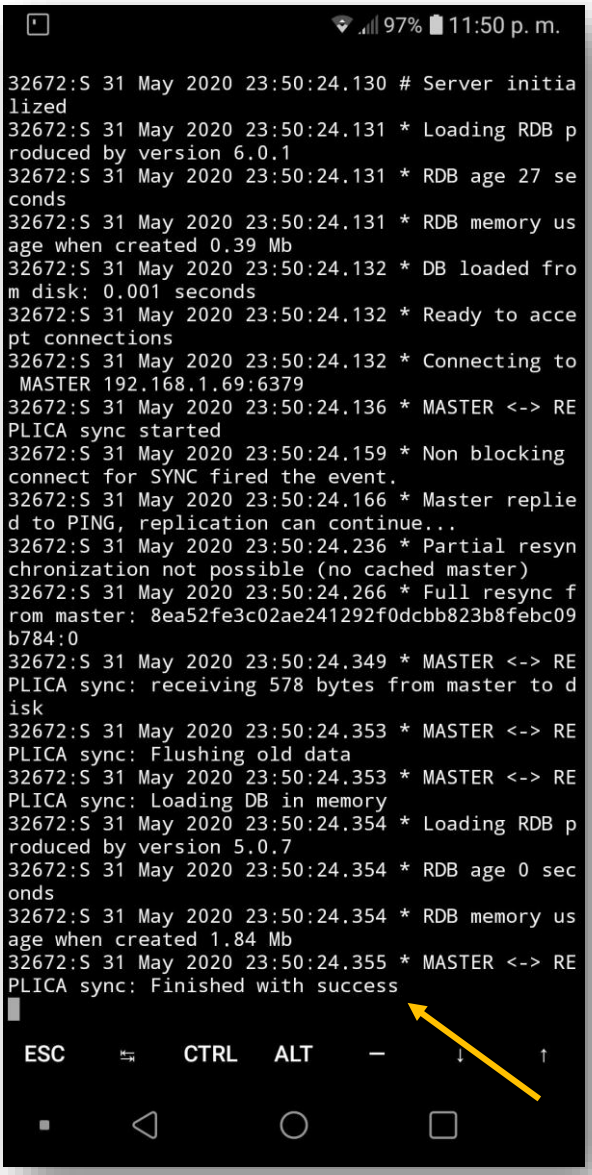
Save and run the server from Termux terminal with the following command.

**$ redis-server redis.conf**

After the execution we can see how it synchronizes with the Windows 10 server **(Master).**

Configuration file redis.conf $ redis-server redis.conf



Installation of syncthing management tool for nodes. We proceed to the installation of **SyncthingManager.**

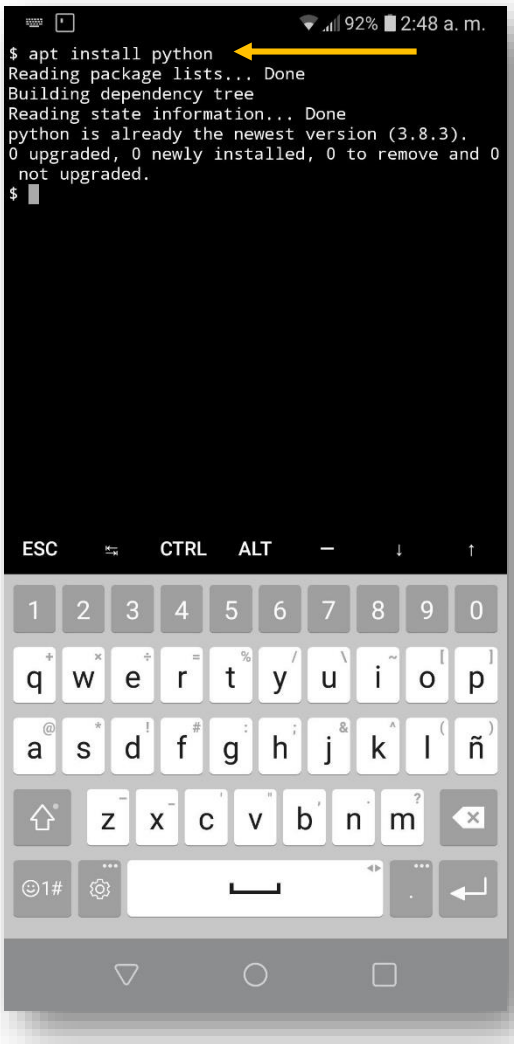
First we install what you will need for SyncthingManager to work properly.

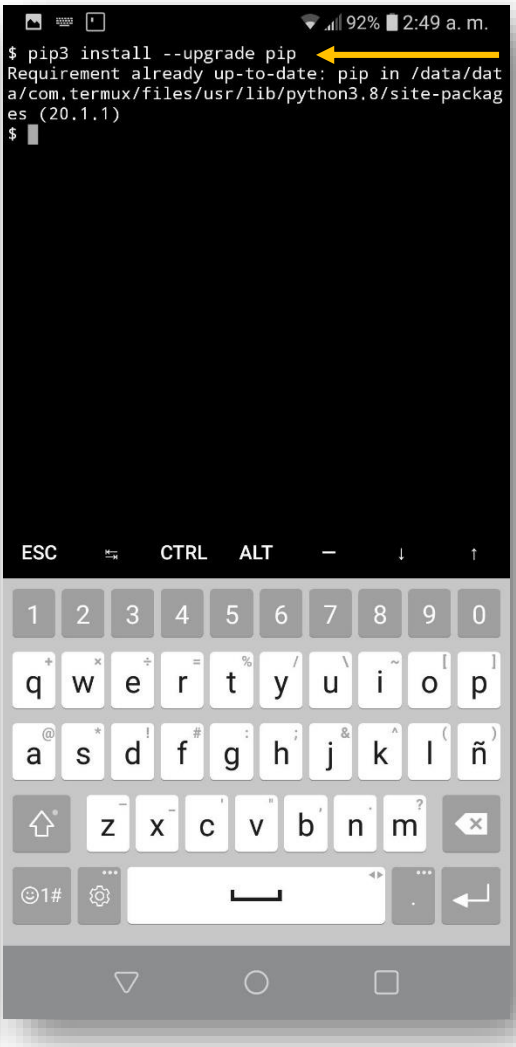
**$ apt install Python**

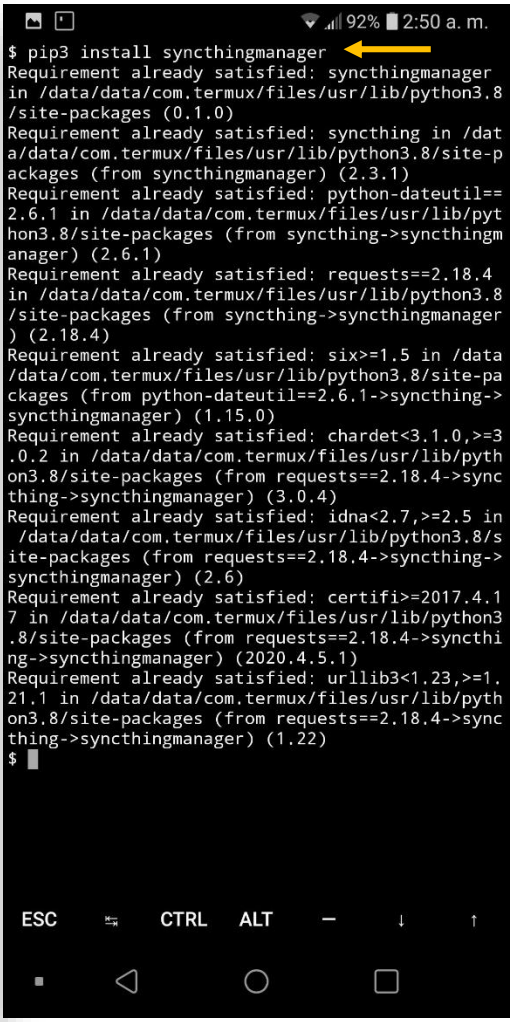
**$pip3 install -upgrade pip**

**$ npm install syncthingmanager**

The SyncthingManager tool will help us to synchronize "peer to peer" in an automatic way and not manually for new and existing nodes.



## 17. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor is a software development environment created by Google Labs to build applications for the Android operating system. The user can, visually and from a set of basic tools, link a series of blocks to create the application. The system is free and can be easily downloaded from the web. Applications created with App Inventor are very easy to create because no knowledge of any programming language is required.

All the current environments that use Blockly's technology such as AppyBuilder and Thunkable among others have their free version, their way of use can be through the internet in their different sites or it can also be installed at home.

The blocks that make up the Mini BloclyChain architecture have been tested in App inventor and AppyBuilder but because of their code optimization they should work on the other platforms.

Online versions:

App Inventor.

https://appinventor.mit.edu/

AppyBuilder.

http://appybuilder.com/

Thunkable.

https://thunkable.com/

Version to be installed on your computer (PC):

https://sites.google.com/site/aprendeappinventor/instala-app-inventor

Environment for developers of Blockly blocks.

https://editor.appybuilder.com/login.php

## 18. What is Proof of Quantum (PQu)?

PoQu. - "Proof of Quantum" is a consensus algorithm developed for Mini BlocklyChain, this test is a variant of the Test of Work (PoW) that works as follows.

The Test of Quantum (PoQu) at startup is executed with the same algorithm as the "Test of Work" (PoW) is based on putting the processor of the device (PC, Server, Tablet or Mobile Phone) to work to obtain a string of characters that is a mathematical puzzle called a "hash".

Remember that a "hash" is an algorithm or mathematical process that when introducing a phrase or some type of digital information such as text files, program, image, video, sound or other diverse type of digital information gives us as a result an alphanumeric character that represents the digital signature that represents it in a unique and non-repeatable way of the data, the hash algorithm is unidirectional, this means that when you enter a data to obtain its signature "hash" its reverse process can not be performed, having a signature "hash" we can not know what information was obtained this property gives us a security advantage to process the information we send over the Internet. How does it work? Imagine sending any kind of information through non-secure channels and accompany it with its respective "source hash", the receiver when receiving the information can get the "hash" of the information received we will call it "destination hash" and check it with the "source hash" if both "hashes" are the same we can confirm that the information has not been altered in the channel that was sent, is just an example where this type of information security process is currently used.

Currently there are different types of algorithms or hash processes that differ in the level of security. The most used or known are: MD5, SHA256 and SHA512.

Example of SHA256:

We have a chain or sentence as follows: "Mini BlocklyChain is modular.

If we apply a SHA256 hash to the previous string it will give us the next hash.

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9of97cb2afd827e8804db8

The above alphanumeric string is the signature that represents the sentence in the above example

For more example we can use the site on the internet:

https://emn178.github.io/online-tools/sha256.html

In the case of the "Test Work" (PoW) algorithm, it works by using computing power to obtain a predefined hash.

Let's imagine that we have the previous "hash" that we took from the "Mini BlocklyChain is modular" chain.

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9of97cb2afd827e8804db8

To this "hash" in its beginning we put the parameter of difficulty that is simply to put zeros "0" in the beginning, that is to say if we say that the difficulty is of 4 it will have "0000" + "hash" to this we will call it "seed hash"

0000 f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

Now taking into account that we know the input information that is the string: "Mini BlocklyChain is modular" we add at the end of the string a number starting from zero "0" and we take out its hash to this we will call it "hash nonce":

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db80

We got hash nonce:

7529f3ad273fc8a9eff12183f8d6f886821900750bb6b59c1504924dfd85a7c8

Then we perform a comparison of the new "hash nonce" with the "hash seed" if they are equal the node that first finds the equality will win the execution of processing the current transaction. As we can see this process is based on probability and computational strength of the device which gives the "Proof of Work" test a consensus equity for all nodes.

If the "seed hash" does not coincide with the "hash nonce", the difficulty is increased by one and the "hash nonce" is removed again, the number that is being increased is called the "nonce" number, it is compared with the "seed hash" until they coincide or are the same.

As we can see the number "nonce" or increase is the one that will help to obtain the "hash" of equality.

Based on the "Test of Work" (PoW) algorithm, the Test of Quantum (PoQu) algorithm is based on obtaining the number "nonce" as PoW does and using a minimum level difficulty ranging from 1 to 5, this serves only to the mobile device to gain the right to be a candidate to win the consensus.

The Quantum Test (PoQu), is activated when the mobile phone has finished the minimum PoW and wins the pass to obtain a probability number in the QRNG system.

The QRNG (Quantum Random Number Generator) is a Quantum Random Number Generator, this system is based on generating true random numbers based on quantum mechanics is the safest system today to generate such numbers. For more details see Annex "Quantum Computation with OpenQbit".

Mini BlocklyChain can implement both minimum PoW and PoQu concession types.

The PoQu test is based on obtaining the number "nonce" this number in the PoQu test is known as "Magic Number" with this the "Peer to Peer" system will confirm if the number is correct and then a random number will be obtained with the QRNG server pool. This random number will be registered in all the nodes, a list will be created containing **((Node Sum /2)) +1** and from this list the one with the highest percentage of probability to be the winner candidate of the consensus (PoQu) will be chosen and this one will execute the current transaction queue.

The PoQu algorithm also uses **NIST (**National Institute of Standards and Technology) testing to assure us that the random numbers in the QRNG are truly random numbers.

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

In Mini BlocklyChain we have implemented a block for PoW and a block for PoQu.

These blocks use a type of hash: SHA256 for free use, for commercial use you have a SHA512 and other types of hash as required.

For more details on the concept of HASH see:

https://es.wikipedia.org/wiki/Funcion_hash

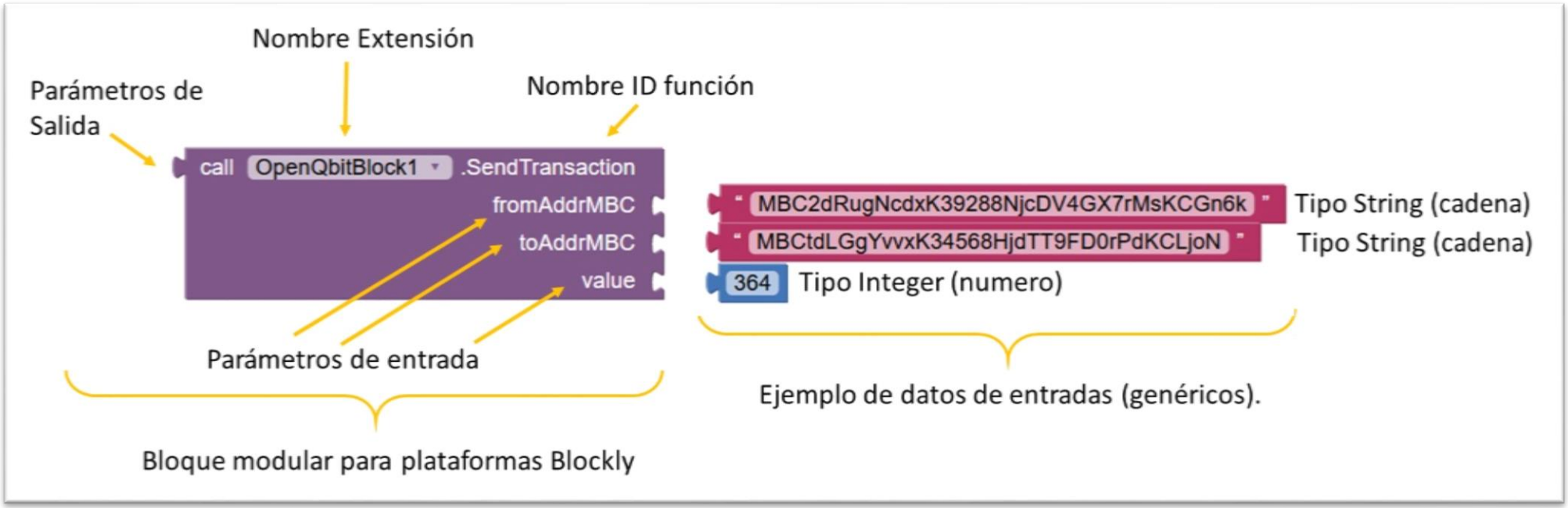
NOTE: The Test of Work (PoW) used in mobile phones can only use a maximum difficulty of 5 since the mathematical processing of these devices is not dedicated like servers or PCs. We only use the PoW algorithm to obtain the opportunity to obtain your pass or permission to enter the Quantum Random Number Generator (QRNG) system and with it to execute the Quantum Random Number Generator (PoQu) algorithm.

On mobile phones do not use a maximum difficulty of 5 as the system may lock up and not respond properly.

## 19. Definition and use of blocks in Mini BlocklyChain

We will start by explaining the distribution of the data that all the blocks will have, their syntax of use and configuration.

In the following example we can see a modular block and its input and output parameters, as well as the types of input data, these data can be of type String (string of characters) or Integer (integer or decimal). We show how it is used and configure it for its proper functioning.




Each module block will have its description and will be named in case it has any mandatory or optional dependency(s) of other blocks used as input parameters, the integration process will be announced.

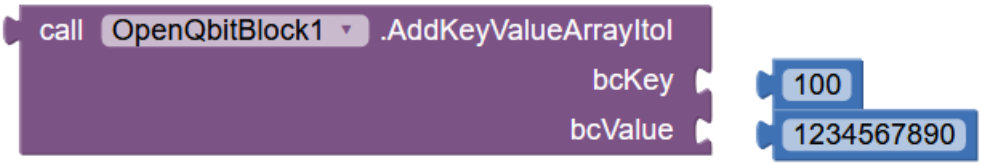
Block to create a temporary string list in a predefined array called internally "chain" - (**AddHash)**.


Input parameters: **block** <string>

Output parameters: Not applicable.

Description: Block for storing a temporary array of hashes or strings in a predefined array called internally "chain".

Block to create key-value arrays (**Integer-Integer**) - (**AddKeyValueArrayItoI**)




Input parameters: **bcKey** <Integer>, bcValue **<Integer>**

Output parameters: Returns the values entered at the input.

Description: It's a temporary key-value arrangement, it's predefined with internal name "**ItoI_UTXO**" this is useful to process UTXO transactions.

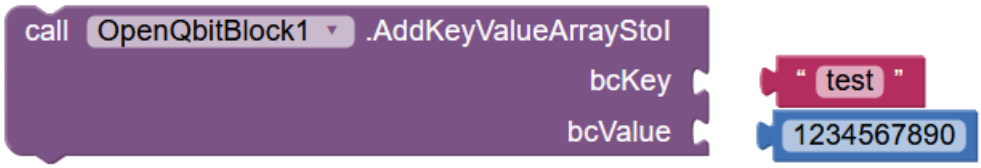
Block to create key-value arrays (**Integer-String)** - (**AddKeyValueArrayItoI**)


Input parameters: **bcKey** <Integer>, bcValue **< String>**

Output parameters: Returns the values entered at the input.

Description: It's a temporary key-value arrangement, it's predefined with internal name "**ItoS_UTXO**" this is useful to process UTXO transactions.

Block to create key-value arrays (**String-Integer)** - (**AddKeyValueArrayItoI**)

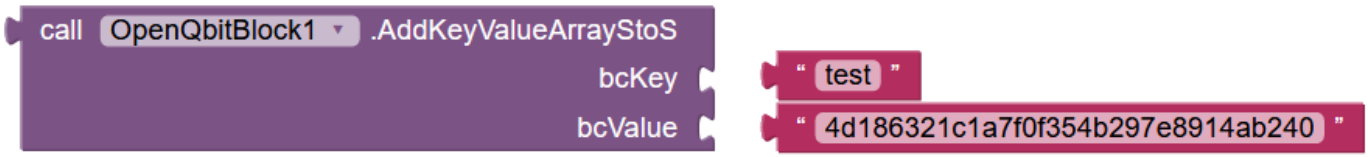



Input parameters: **bcKey** <String>, bcValue **<Integer>**

Output parameters: Returns the values entered at the input.

Description: It is a temporary key-value arrangement, it is predefined with internal name "**StoI_UTXO**" this is useful to process UTXO transactions.

Block to create key-value arrays (**String-String)** - (**AddKeyValueArrayStoS**)




Input parameters: **bcKey <String**>, bcValue **<String>**

Output parameters: Returns the values entered at the input.

Description: It is a temporary key-value arrangement, it is predefined with internal name "**StoS_UTXO**" this is useful to process UTXO transactions.

Block for generating decimal random quantum numbers - (**ApiGetQRNGdecimal**)


Input parameters: **qty <** Integer>

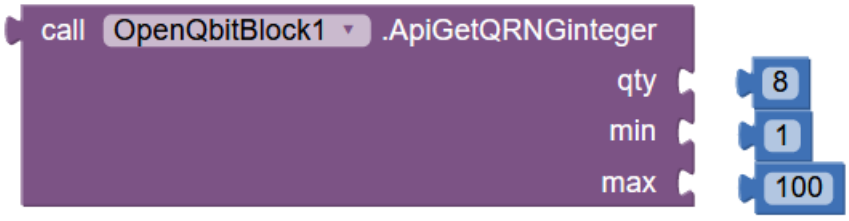
Output Parameters: Gives the amount "qty" of random quantum decimal numbers entered in the input numbers are within the range of 0 and 1 in JSON format.

Example:

qty = 5; output: {"result": [0.5843012986202495, 0.7746497687824652, 0.05951126805960929, 0.1986079055812694, 0.03689783439899279]}

Description: Quantum Random Number Generator (QRNG) API

Block for generating decimal random quantum numbers - (**ApiGetQRNGdecimal**)




Input parameters: **qty <Integer>**, min <Integer>, max <max>

Output Parameters: Gives the amount "qty" of random quantum integers entered in the input the numbers are within the range of min and max in JSON format.

Example:

qty = 8, min = 1, max = 100; output: {"result": [3, 53, 11, 2, 66, 44, 9, 78]}

Description: Quantum Random Number Generator (QRNG) API

Hash block "SHA256" for character strings (**ApplySha256**).


Input parameters: **input <String>**

Output parameters: Calculates the "SHA256" hash of a character string.

Example:

Input = "Mini BlocklyChain is modular"

output: f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2afd827e8804db8

Description: Function to remove hash "SHA256". *Sha* or *SHA* refer to: Secure Hash Algorithm, a set of hash functions designed by the United States National Security Agency. The SHA256 uses a 256-bit algorithm.

Block to clean the predefined internal array "chain" (**ClearBlockList**).


Input and output parameters: Not applicable

Description: Delete all the elements that have the internal temporal arrangement "chain".

Block to clean the predefined internal array (**Integer-Integer**) - (**ClearItoI)**.


Input and output parameters: Not applicable

Description: Delete all the elements that have the internal temporary arrangement "ItoI_UTXO".

Block to clean the predefined internal array (**Integer-String)** - (**ClearItoS)**.


Input and output parameters: Not applicable

Description: Delete all the elements that have the internal temporary arrangement "ItoS_UTXO".

Block to clean the predefined internal array (**String-Integer)** - (**ClearStoI)**.


Input and output parameters: Not applicable

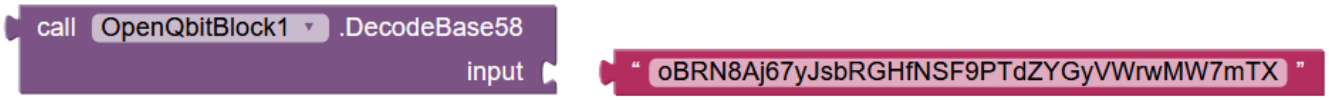
Description: Delete all the elements that have the internal temporary arrangement "StoI_UTXO".

Block to clean the predefined internal array (**String-String)** - (**ClearStoS)**.


Input and output parameters: Not applicable

Description: Delete all elements that have the internal temporary arrangement "StoS_UTXO".

Block to decode a string to Base10. (**DecodeBase58**)




Input parameters: **input<String>**

Output parameters: It delivers the original string that was used in the block (**EncodeBase58**).

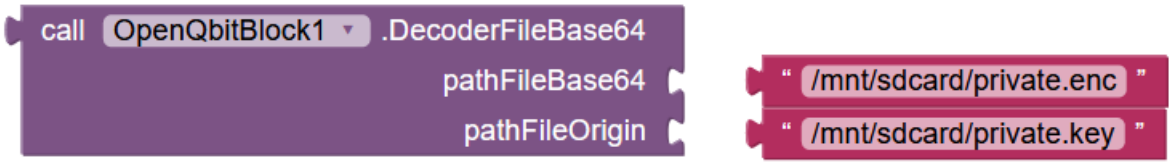
Example:

Input = oBRN8Aj67andJsbRGHfNSF9PTdZYGandVWrwMW7mTX

Output: "Mini BlocklyChain is modular".

Description: Converts a Base58 string to the original text given in the block (**EncodeBase58**)

Block to decode a file with Base64 algorithm (**DecoderFileBase64**).




Input parameters: **pathFileBase64 <String>** , pathFileOrigin **<String>**

Output parameters: Source file that was entered in the block (**EncoderFileBase64**)

Description: A Base64 file is converted to the original file that was inserted in the block (**EncoderFileBase64**).

Block to know if the predefined internal array "chain" is empty. (**EmptyBlockList**)


Input parameters: Not applicable.

Output parameters: Returns "True" if empty or returns "False" if you have data.

Description: Block to ask if the predefined temporary internal array "chain" has elements.

Block to encode a character string to Base58. (**EncodeBase58**).


Input parameters: **input<String>**

Output parameters: It delivers the original string that was used in the block (**EncodeBase58**).

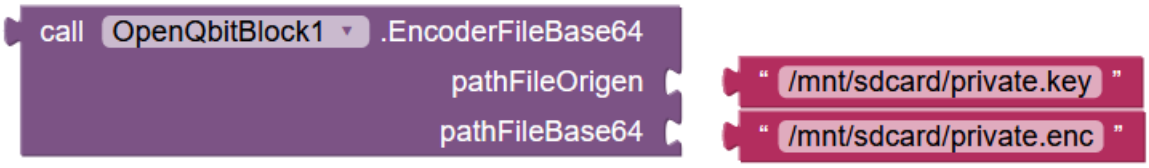
Example:

Input = "Mini BlocklyChain is modular".

Ourput: oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX

Description: Converts a breast chain to a chain in Bae58. The Base58 algorithm is a group of binary to text encoding schemes used to represent large integers as alphanumeric text, introduced by Satoshi Nakamoto for use with Bitcoin.

Block for encoding a file with Base64 algorithm (**EncoderFileBase64**).




Input parameters: **pathFileOrigin <String>** , pathFileBase64 **<String>**

Output parameters: Base64 encoded file.

Description: Converts a source file of any format to a Base64 file. File names can be arbitrary and chosen by the user.

Block Generator of user addresses (**GenerateAddrBitcoin**).


Mandatory unit: Block (**ApiGetQRNGinteger**),

Dependencies (optional): **OpenQbitFileEncription extension**, **OpenQbitFileDecryption extension** and **OpenQbitSQLite extension.**

Input parameters: **qrng <** mandatory dependence>

Output parameters: 34-character alphanumeric transaction address and keyStore use address

Description: Block to create a new Bitcoin generic transaction address for user and private key generator (Digital signature for sending transactions) and public key (Public address for making transactions). This key generator is basically the address generator for use in a digital wallet or commonly called "Wallet".

This block is used in conjunction with the Quantum Random Number Generator (QRNG) blocks and the two output parameters must be inserted into the KeyStore.

KeyStore is a database that stores private keys in hexadecimal format:

Example of a hexadecimal address that is stored in the KeyStore

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

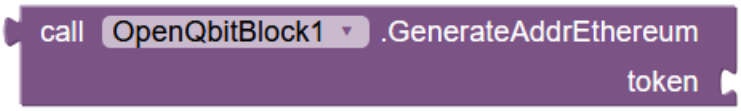
This base is used only by the local user and the information is encrypted, this process is through the use of OpenQbitSQLite extension and OpenQbitAESEncryption and OpenQbitAESDecryption information encryption extensions.

To create a KeyStore see the "Creating a KeyStore" appendix. The generated addresses use the same Bitcoin address algorithm, with the initial bitcoin address identifier being "1".

The addresses generated by the block (**GenerateAddrBitcoin**) are 34 alphanumeric characters are composed of 33 alphanumeric characters and 1 of the Bitcoin identifier as follows:

12dRugNcdxK39288NjcDV4GX7rMsKCGn6k

User Address Generator Block (**GenerateAddrEthereum**).




Compulsory dependency: Obtain a token at: https://accounts.blockcypher.com/signup

Dependencies (optional): **OpenQbitFileEncription extension**, **OpenQbitFileDecryption extension** and **OpenQbitSQLite extension.**

Input parameters: **token <** mandatory dependence - String>

Output parameters: 40 alphanumeric character transaction address does not include the initial Ethereum indicator "0x" which would give us the 42 characters of a common address. It also returns the public key and the private key.

Example:

{

"**private**": "227ac59f480131272003c2d723a7795ebd3580acaab62b5c537989e2ce4e08ef",

"**public**": "04e2d55ebccd32a7384e096df559cc36b856c64a16e5b402e10585dc3ea055672aafa84df8a859531570a650a8ab1e7a22949100efa1aa5f072c035551cac1ce",

"**address**": "14e150399b0399f787b4d6fe30d8b251375f0d66"}

Description: Block to create a new transaction address for user and private key generator (Digital signature to send transactions) and public key (Public address to make transactions). This key generator is an external API and you must have a Wifi or mobile connection, it is basically the address generator to use it in a digital wallet or commonly called "Wallet".

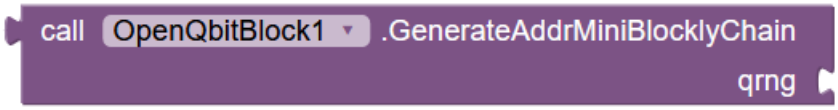
You can create aKeyStore is a database that stores private keys in hexadecimal format, See Appendix "KeyStore Creation".

Example of a hexadecimal address that is stored in the KeyStore

**0x14e150399b0399f787b4d6fe30d8b251375f0d66**

The private key is used only by the local user and the information is encrypted, this process is through the use of OpenQbitSQLite extension and OpenQbitAESEncryption and OpenQbitAESDecryption information encryption extensions.

User Address Generator Block (**GenerateAddrMiniBlocklyChain**).




Mandatory unit: Block (**ApiGetQRNGinteger**),

Dependencies (optional): **OpenQbitFileEncription extension**, **OpenQbitFileDecryption extension** and **OpenQbitSQLite extension.**

Input parameters: **qrng <** mandatory dependence>

Output parameters: 36 alphanumeric character transaction address and keyStore use address. Review Block Method (**PairKeysMBC**).

Description: Block to create a new transaction address for user and private key generator (Digital signature to send transactions) and public key (Public address to make transactions). This key generator is basically the address generator to use it in a digital wallet or commonly called "Wallet".

This block is used in conjunction with the Quantum Random Number Generator (QRNG) blocks and the two output parameters must be inserted into the KeyStore.

KeyStore is a database that stores private keys in hexadecimal format:

Example of a hexadecimal address that is stored in the KeyStore

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

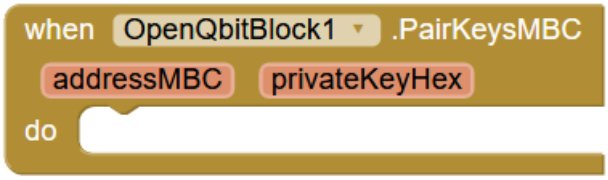
This base is used only by the local user and the information is encrypted, this process is through the use of OpenQbitSQLite extension and OpenQbitAESEncryption and OpenQbitAESDecryption information encryption extensions.

To create a KeyStore see the "Creating a KeyStore" appendix. The generated addresses use the same Bitcoin address algorithm, the only difference is that the bitcoin address identifier which is "1" or "3" is changed to the Mini BlocklyChain identifier "MBC".

The addresses generated by the block (**GenerateAddrMiniBlocklyChain) are** 36 alphanumeric characters and are composed of 33 alphanumeric characters and 3 of the Mini BlocklyChain identifier capital letters "MBC" as follows:

**Mbc2dRugNcdxK39288NjcDV4GX7rMsKCGn6k**

Block method (**PairKeysMBC**) is block output (**GenerateAddrMiniBlocklyChain**).




Input parameters: Not applicable.

Output parameters: **addressMBC** <String>, **privateKeyHex <String**>.

Description: Delivered public user address in Mini BlocklyChain format and private key in hexadecimal format.

example:

**addressMBC**: MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

**privateKeyHex**:
024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

Block Generator of user addresses (**GenerateKeyPair**).


Mandatory unit: Block (**ApiGetQRNGinteger**),

Dependencies (optional): **OpenQbitFileEncription extension**, **OpenQbitFileDecryption extension** and **OpenQbitSQLite extension.**

Input parameters: **qrng <** mandatory dependence>

Output parameters: Provides the public key and primary key for local user

Description: Generates public and primary keys using ECC algorithm (1) and with hexadecimal format.

(1) Elliptic curve cryptography (ECC) is a variant of asymmetric or public-key cryptography based on the mathematics of elliptic curves.

Block to sign transactions (**GenerateSignature**).

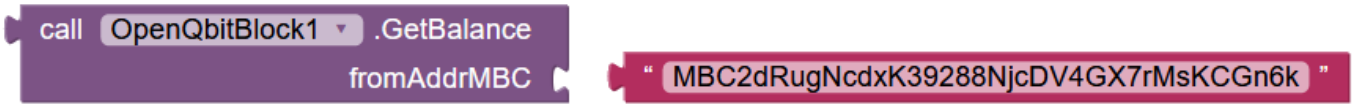

Required dependency: Block (**GenerateKeyPais**), Block (**SenderLoadKeyPair),** Block (**ReciepientLoadKeyPair**). Preface these blocks before use.

Input parameters: **<** Mandatory check dependencies>

Output parameters: No output.

Description: It generates a digital signature from the Sender applied to the asset sent in the transaction to the Recipient, this signature will be confirmed when the transaction is being processed by some node of the network, with this signature the Mini BlocklyChain system ensures that the asset has not been modified in it sent and that it complies with the sender-recipient relationship and is not diverted or applied to another digital address.

Block to obtain the total balance of assets of a user (**GetBalance**).




Mandatory dependency: Block (**ConnectorTransactionTail**).

Input Parameters: **fromTransTail <Array String>, <** Mandatory Boards dependencies>

Output parameters: Checks the incoming and outgoing asset expenses for each transaction.

Description: Checks the balance to approve if the user has assets to send or if the assets he receives are added to his balance.

Block to get q22 from mobile phone. (**GetDeviceID**)


Input parameters: **Not applicable**

Outbound parameters: It delivers the internal identifier of the mobile phone.

Description: Provides the IMEI mobile phone identifier unique to each device.

Block to remove the RIPEMD-160 hash from a string. (**Rimpemd-160**)


Input parameters: **str <String>**

Output parameters: Calculates the "RIPEMD-160" hash of a character string.

Example:

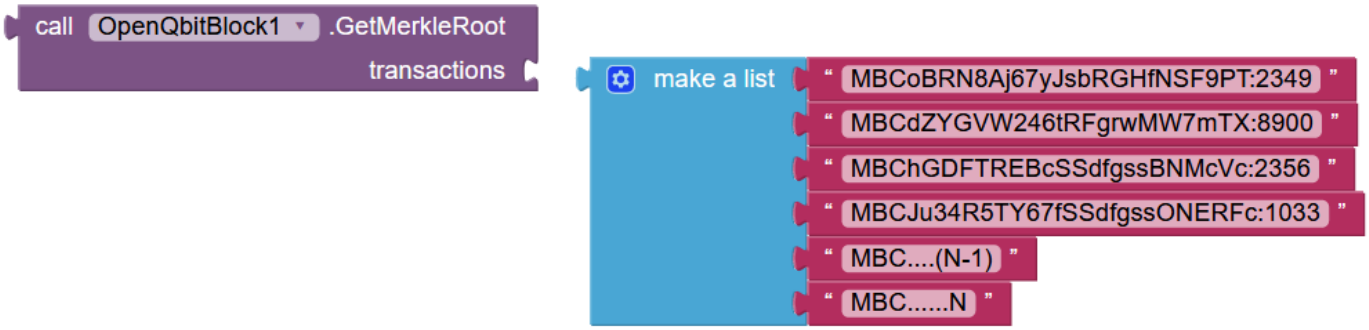
Input = "Mini BlocklyChain is modular"

output: ae29436e4b8ea8ed6143f3f92380dfa2f4f47336

Description: Obtain the "RIPEMD-160" hash. This hash is used in the process of generating a valid address for Bitcoin and Mini BlocklyChain.

RIPEMD-160 (acronym for RACE Integrity Primitives Evaluation Message Digest) is a 160-bit message digest algorithm (and cryptographic hash function).

Block to calculate Merkler's tree. (**GetMerklrRoot**)




Input parameters: **transactions <Array** String>

Output parameters: It delivers a hash type "SHA256".

Example:

Input = transaction queue, an arrangement of all transactions to be processed.

output: b4a44c42b6070825f763cd118d6ab49a8e80bbb7cdc0225064f8e042b94196bd

Description: Obtain the "SHA256" hash using Merkle's tree algorithm

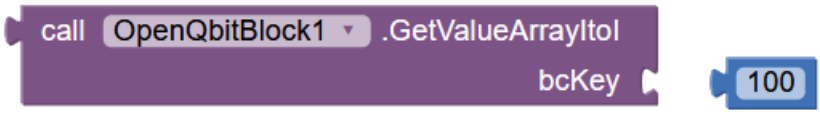
A Merkle Hash Tree is a binary or non-binary data tree structure in which each node that is not a leaf is tagged with the hash of the concatenation of the labels or values (for leaf nodes) of its child nodes. These are a generalization of hash lists and hash strings.

It allows a large number of separate data to be linked to a single hash value, the hash of the tree root node. This provides a secure and efficient method of verifying the contents of large data structures. In its practical applications, the hash of the root node is usually signed to ensure its integrity and that the verification is totally reliable. Demonstrating that a leaf node is part of a given hash tree requires an amount of data proportional to the logarithm of the number of nodes in the tree.

Currently the major use of Merkle trees is to make safe the data blocks received from other peers in the peer-to-peer networks, to ensure that they are received without damage and without being altered.

It is used to verify that a queue that has the transactions to be processed has not been modified and to ensure its integrity for dispersion in all the nodes of the Mini BlocklyChain network. All nodes have to execute this algorithm to ensure the integrity of each transaction that will be applied.

Block to obtain a value from the predefined array "ItoI_UTXO" (**GetValueArrayItoI**).

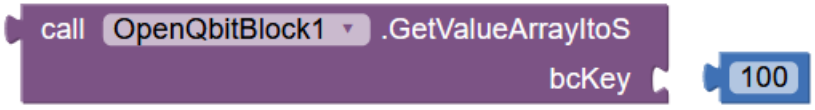



Input parameters: **bcKey** <Integer>

Output parameters: Returns the value <Integer> stored in the label with the given number as input.

Description: It is a request to the temporary key-value arrangement, which is predefined with internal name "**ItoI_UTXO**" this is useful to process UTXO transactions.

Block to obtain a value from the predefined array "ItoS_UTXO" (**GetValueArrayItoS**).

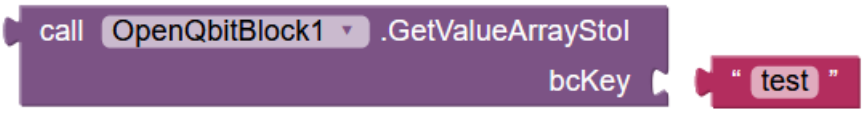



Input parameters: **bcKey** <Integer>

Output parameters: Returns the value <String> stored in the label with the given number as input.

Description: It is a request to the temporary key-value arrangement, which is predefined with internal name "**ItoS_UTXO**" this is useful to process UTXO transactions.

Block to obtain a value from the predefined array "StoI_UTXO" (**GetValueArrayStoI**).




Input parameters: **bcKey** < String>

Output parameters: Returns the value <Integer> stored in the label with the given name as input.

Description: It is a request to the temporary key-value arrangement, which is predefined with internal name "**StoI_UTXO**" this is useful to process UTXO transactions.

Block to obtain a value from the predefined array "StoS_UTXO" (**GetValueArrayStoS**).


Input parameters: **bcKey** < String>

Output parameters: Returns the value <String> stored in the label with the given name as input.

Description: It is a request to the temporary key-value arrangement, which is predefined with internal name "**StoS_UTXO**" this is useful to process UTXO transactions.

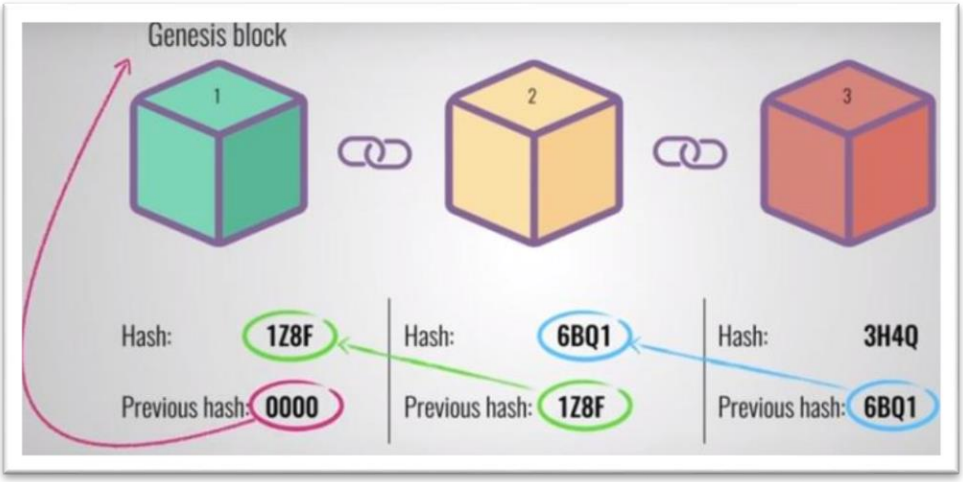
Block validator of the block chain. (**IsChainValid**)


Required dependency: Block (**GenerateKeyPais**), Block (**SenderLoadKeyPair),** Block (**ReciepientLoadKeyPair),** Block (**GenerateSignature)**. Preface these blocks before use.

Input parameters: Not applicable.

Output parameters: Delivery "True" if the validation of the block string is correct or "False" in case the validation fails.

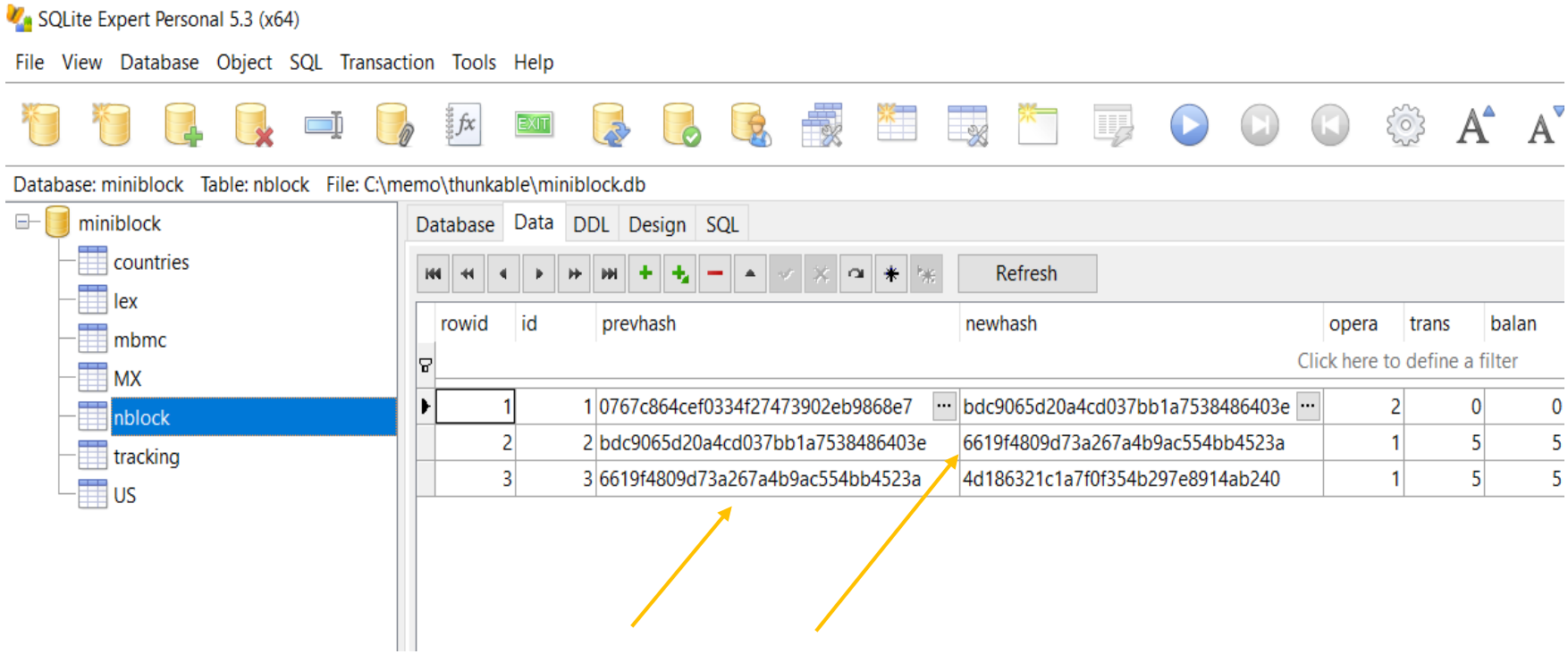
Description: It provides us with the validation of the components that have been previously inserted in the block chain system, this validation is the most important of the whole system. How does the block chain validation work or how it is commonly known in a generic way (BlockChain). It is the core part of every system.


Genesis block
1
2
3
Hash: 1Z8F
Previous hash: 0000
Hash: 6BQ1
Previous hash: 1Z8F
Hash: 3H4Q
Previous hash: 6BQ1

As we see in the previous image each block that is added in the storage system is related to the previous block through the hash algorithms.

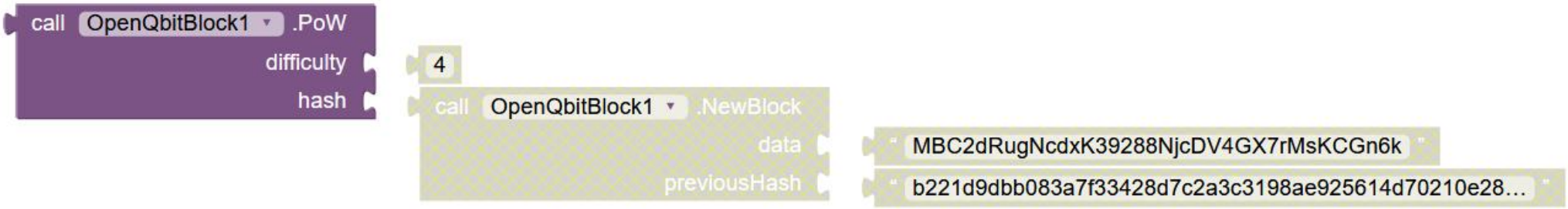
For example, when creating a new block to add, the hash representing the new information is obtained by taking the hash of the new information block + the previous hash. With this type of link any minimal modification in the storage of block chains will be detected which allows a very high and effective data security.

Below is an example of how a string of blocks is stored in a SQLite database, we'll see how the previous hash is linked to the new hash of the last block (last queue of processed transactions) that was inserted. Each hash in both the "prevhash" and "newhash" columns represents the information of the transaction queue that was processed at the time.




Previous hash is related to the new hash.
They're always the same.

Block to perform PoW Work Test and to mine new blocks. (**Pow**)




Mandatory dependency: **NewBlock**. Preface this block before using it.

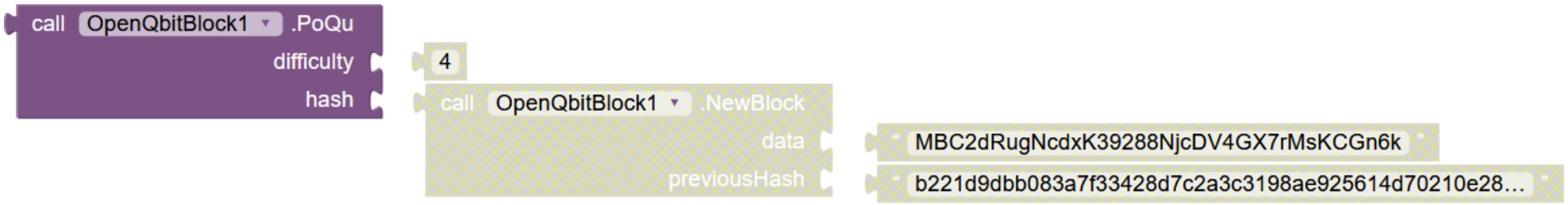
Input parameters: **difficulty** <Integer>, hash <Compulsory dependence>

Output parameters: Gives as output a hash "SHA256" composed with the #Number of "zeros" given in the input parameter difficulty.

Description: This block performs an activity called "mining" a new block is what we have previously described as the process of PoW (Proof of Work), see section What is Proof of Quantum (PQu)?

Mining or running the PoW is a concept where the nodes are given a task to solve a mathematical puzzle. Whoever solves it first in the case of Mini BlocklyChain gets the opportunity to continue with the process in order to be chosen to be the winner of being able to process the transaction queue that is waiting to be processed.

Block to perform PoW Work Test and to mine new blocks. (**PoQu**)




Mandatory dependency: **NewBlock**. Preface this block before using it.

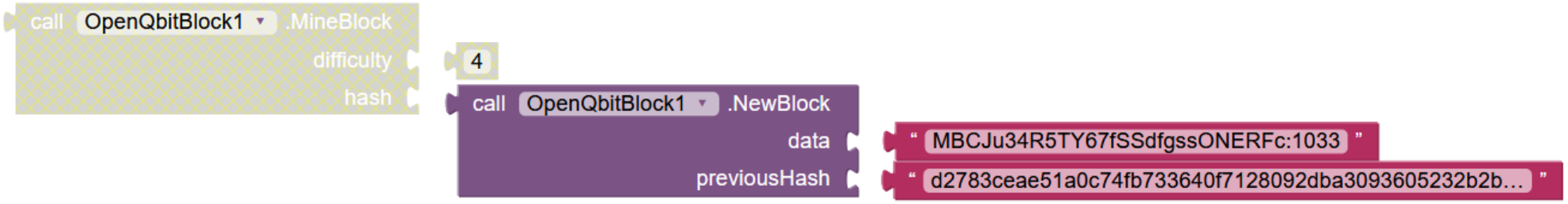
Input parameters: **difficulty** <Integer>, hash <Compulsory dependence>

Output parameters: Results in the number "Magic Number" and random quantum number that is within the range of 0 and 1 16-digit decimal type, example (**0. 5843012986202495**) and a hash "SHA256" composed with the #Number of "zeros" given in the input parameter difficulty.

Description: This block performs the process of "mining" a new block is what we have previously described as the process of PoW (Proof of Work), but this process calls the function of random quantum number generated after calculating the number "nonce" suitable to meet the level of difficulty that can be in the range of minimum 1, maximum 5. See section What is Proof of Quantum (PQu - Proof Quantum)?

Mining or running the PoW or PoQu is a concept where the nodes are given a task to solve a mathematical puzzle. The node who solves it first in the case of any blockchain system gets the opportunity to continue with the process in order to be chosen to be the winner of being able to process the transaction queue that is waiting to be processed.

Block for the creation of **new** blocks (**NewBlock**).

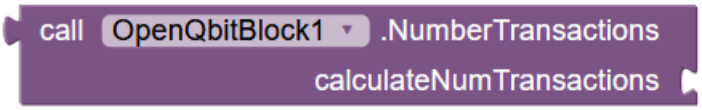



Input parameters: **data** <String>, **previousHash <String>**

Output parameters: It delivers a hash calculated as: **SHA256 (data (**input parameters**) + previousHash (**input parameter**) + IMEI (**internal parameter) + **MerkleRoot (**internal parameter**)).**

Description: This block performs the creation of a new block to be processed. It gives as output a "SHA256" hash composed of the string of input parameters in the case of data varies depending on each system design and the previous hash is based on the hash of the previous transaction string that was already processed and is stored in the Mini BlocklyChain block string system, these two are variable parameters, there are two internal system parameters that are fixed and ensure the integrity of the information and the system, these two internal parameters are the unique ID of the mobile phone and the hash of the merklet tree of the transaction queue that is being processed.

Block of the total local transactions of the node (**NumberTransactions**)




Mandatory dependency: Block (**AddKeyValueArrayStoS**). Preface this block before using it.

Input parameters: < Mandatory dependence>.

Output parameters: Returns the total of local transactions to the node.

Description: Provides the total of transactions that will be applied only to the local accounts of the node.

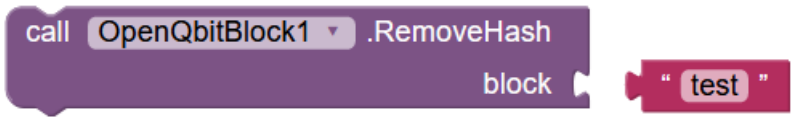
Block to read address of recipient in binary file. (**ReciepientLoadKeyPair**)


Input parameters: **Not applicable.**

Output parameters: Returns private key and public key in a Base64 format.

Description: Gets the private and public address of the recipient from a binary file as an input parameter.

Block to delete element in internal temporary arrangement "chain" (**RemoveHash**).

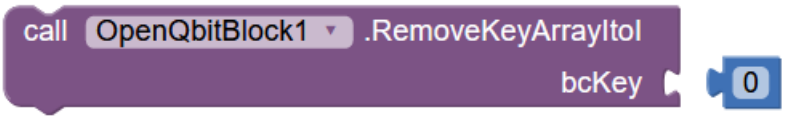



Input parameters: **block < String**>.

Output parameters: Not applicable.

Description: Gets the private and public address of the recipient from a binary file as an input parameter.

Block to delete element in internal temporary arrangement "ItoI_UTXO" (**RemoveKeyArrayItoI**)

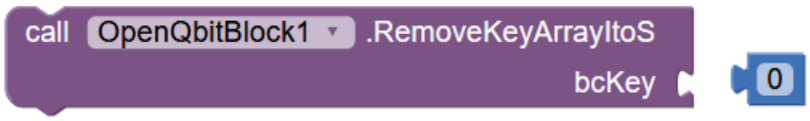



Input parameters: **bcKey < Integer**>.

Output parameters: Not applicable, type of element delete <Integer>

Description: The element associated to the numerical label of the internal temporary arrangement "ItoI_UTXO" is deleted.

Block to delete element in internal temporary arrangement "ItoS_UTXO" (**RemoveKeyArrayItoS**).

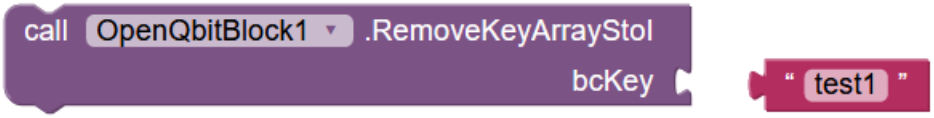



Input parameters: **bcKey < Integer**>.

Output parameters: Not applicable, type of element to be deleted <String>

Description: The element associated to the numerical label of the internal temporary arrangement "ItoS_UTXO" is deleted.

Block for deleting element in internal temporary arrangement "StoI_UTXO" (**RemoveKeyArrayStoI**)

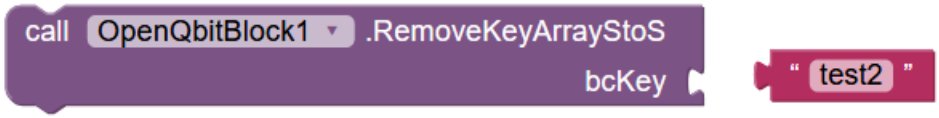



Input parameters: **bcKey < String**>.

Output parameters: Not applicable, type of element delete <Integer>

Description: The element associated with the numerical label of the internal temporary arrangement "StoI_UTXO" is deleted.

Block for deleting element in internal temporary arrangement "StoS_UTXO" (**RemoveKeyArrayStoS**)

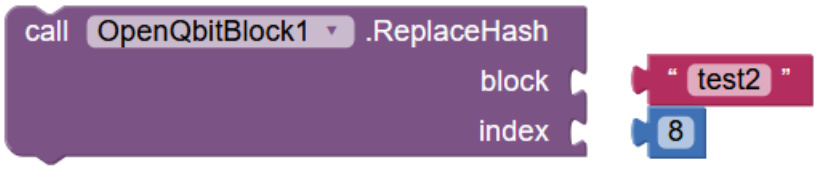



Input parameters: **bcKey < String**>.

Output parameters: Not applicable, type of element to be deleted <String>

Description: Delete element of the internal temporary arrangement label "StoS_UTXO".

Block to replace a value of the internal temporary arrangement "chain" (**ReplaceHash**).

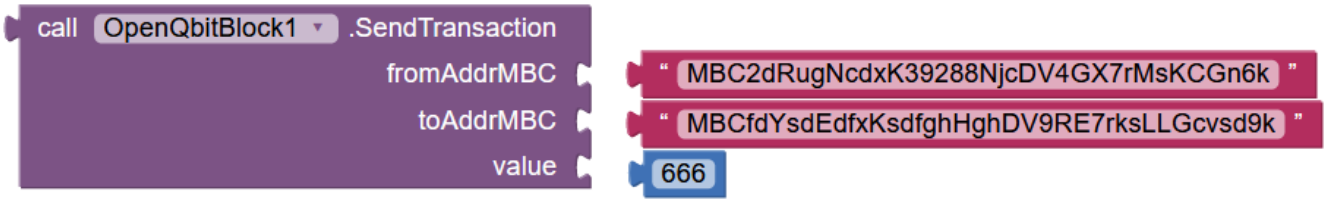



Input parameters: **block < String**>, **index < Integer>**

Output parameters: Not applicable.

Description: The element associated to the index "index" is replaced with the value of the label "block" in the temporary internal arrangement "chain".

Block to start (**send)** new transaction (**SendTrasaction**).




Input parameters: **fromAddrMBC < String**>, **toAddrMBC < String>,** value < **Integer>**

Output parameters: Returns value "True" if the transaction was sent successfully or "False" if it was not successful he sent.

Description: Block that transforms the sender and recipient address into a binary format and is attached to the transaction queue to be processed.

Block to read sender's address in binary file. (**SenderLoadKeyPair**)


Input parameters: **Not applicable.**

Output parameters: Returns private key and public key in a Base64 format.

Description: Gets the private and public address of the sender from a binary file as an input parameter.

Block to obtain element number of the temporary arrangement "chain" (**SizeBlockList**).


Input parameters: **Not applicable.**

Output parameters: Returns the element number of the internal array "chain".

Description: Gets the element number of the internal array "chain".

Block to obtain element number of the temporary arrangement "ItoI_UTXO" (**SizeItoI**)


Input parameters: **Not applicable.**

Output parameters: Returns the element number of the internal array "ItoI_UTXO".

Description: Gets the element number of the internal array "ItoI_UTXO"

Block to obtain element number of the temporary arrangement "ItoS_UTXO" (**SizeItoS**)


Input parameters: **Not applicable.**

Output parameters: Returns the element number of the internal array "ItoS_UTXO".

Description: Gets the element number of the internal array "ItoS_UTXO".

Block to obtain element number of the temporary arrangement "StoI_UTXO" (**SizeStoI**)


Input parameters: **Not applicable.**

Output parameters: Returns the element number of the internal array "StoI_UTXO".

Description: Gets the element number of the internal array "StoI_UTXO".

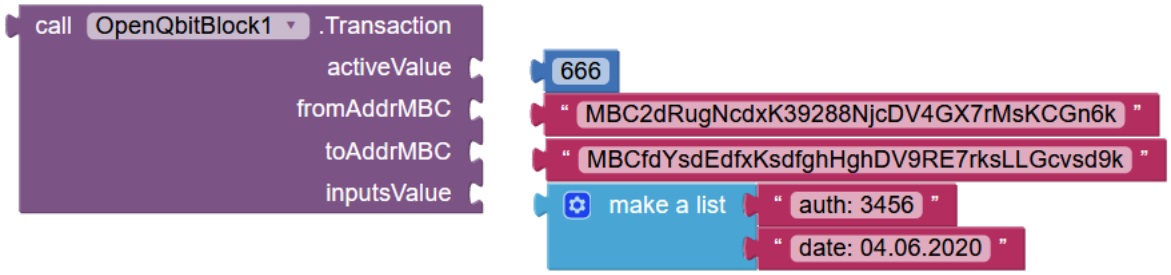
Block to obtain element number of the temporary arrangement "StoS_UTXO" (**SizeStoS**)


Input parameters: **Not applicable.**

Output parameters: Returns the element number of the internal array "StoS_UTXO".

Description: Gets the element number of the internal array "StoS_UTXO".

Block creation of transaction with additional values (**Transaction**).




Input parameters: **activeValue < Integer>**, **fromAddrMBC < String>,** toAddrMBC < **String>, inputsValue** < **Array String>.**

Output parameters: It gives us, the addresses in binary format and the value inpustValue converted into a string 'String', and gives us the hash "SHA256" of the ActiveValue value.

Description: Prepares a new transaction to be processed by the nodes.

Mini BlocklyChain address validation block (**ValidateAddMiniBlocklyChain**)


Input parameters: User addresses in Mini BlocklyChain format.

Output parameters: Returns "True" if the address is in the correct format or "False" if the address is invalid.

Description: Validates if the entered Mini BlocklyChain address is correct, this block applies an algorithm to verify if the address was created using the address creation mechanism to be used in the Mini BlocklyChain system.

Bitcoin address validation block. (**ValidateAddBitcoin**)


Input parameters: **addr** < String> , Bitcoin formatted user addresses (accepts Bitcoin addresses with initial identifier "1", addresses with identifier "3" are not applicable.

Output parameters: Returns "True" if the address is in the correct format or "False" if the address is invalid.

Description: Validates if the entered Bitcoin address is correct, this block applies an algorithm to verify if the address was created using the address creation mechanism to be used in the Bitcoin system.

Digital signature verification block for the current transaction. (**VerifySignature**).


Input parameters: **Not applicable**.

Output parameters: Returns "True" if the check is valid or "False" if the check is invalid.

Description: Gets the verification of the digital signature that the sender should have seen done in the process of sending the transaction you want to make. In this verification process is performed to check the source value sent has not been altered in the channel where the transaction was sent, as well as verifying the recipient to which the transaction should be applied.

## 20. Use of blocks for SQLite database (MiniSQLite version).

In this section we will see how to use the blocks to perform two main operations that we are interested in for the functionality of the Mini BlocklyChain system which is to "INSERT" data into the block chain and query it.

NOTE: The transactions in the SQLite database are different from the transactions sent by the nodes to be applied in the Mini Blocklychain system.

The transactions in the database are based on a CRUD (Create, Read, Update and Delete) model and are the processes that can be executed with external or internal data only in the SQLite database. In the blockchain systems are used mainly the processes of Create and Read, for security are discarded the processes of updating or deleting and more where it is stored the chain of blocks that gives the integral security of the system.

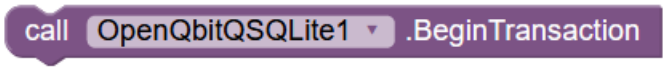
On the other hand, the transactions sent by the nodes refer to all the processes that involve making the action of sending some type of asset between the members (nodes) of the Mini BlocklyChain system, this type of transactions include diverse processes for its application, these can be from the creation of a digital address, a digital signature, a signature validation, a process within the SQLite database, among other processes.

We will start with the definition and use of the blocks of the SQLite database MiniSQLite version this version is integrated only by 8 blocks. You have a full version to manipulate the data in the SQLite database, however, for practical purposes the Mini BlocklyChain system with the MiniSQLite version is sufficient.

In case you want to review all the components their use and description see Annex "Extended blocks for SQLite database".

### MiniSQLite version blocks:

Block to start some kind of transaction in the SQLite database (**BeginTransaction**)



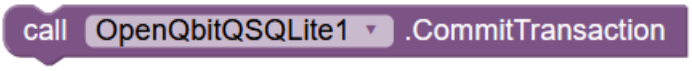

Mandatory unit(s): Block (**ImportDatabase), Block** (**OpenDatabase**).

Input Parameters: **Use before <** Mandatory Dependency(s)>

Output parameters: Not applicable, it starts a transaction in a SQLite database.

Description: Block that starts a process in the SQLite database, you must first have executed the database import block (**ImportDatabase)** and the database open block (**OpenDatabase**).

Block to make Commit in SQLite. (**CommitTrasaction**)



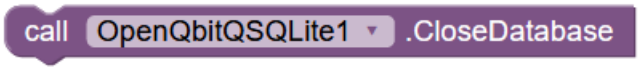

Required dependency(s): Block (**BeginTransaction)**, Block (**ImportDatabase), Block** (**OpenDatabase**).

Input Parameters: **Use before <** Mandatory Dependency(s)>

Output parameters: Not applicable, it performs a **commit of a** transaction in a SQLite database.

Description: Block that starts a **commit** process on the SQLite database, you must first have executed the database import block (**ImportDatabase)** and the database open block (**OpenDatabase**).

Block to close the SQLite database that was imported or exported (**CloseDatabase**)



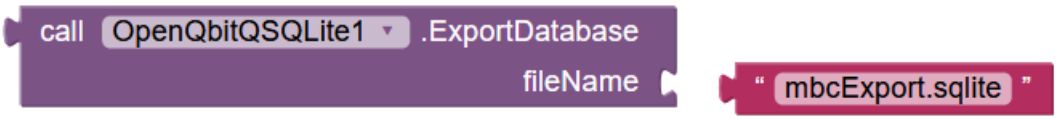

Mandatory unit(s): Block (**ImportDatabase), Block** (**OpenDatabase**).

Input Parameters: **Use before <** Mandatory Dependency(s)>

Output parameters: Not applicable, it closes a SQLite database.

Description: Block that closes the SQLite database, you must first have executed the database import block (**ImportDatabase)** and the database open block (**OpenDatabase**).

Block to export a SQLite database (**ExportDatabase**).




Mandatory unit(s): Block (**ImportDatabase), Block** (**OpenDatabase**).

Input parameters: **fileName < String>** enter a path where you can find a database already created with SQLite format.

**Use before <** Compulsory Dependency(s) >

Output parameters: Export to a SQLite database.

Description: Block that exports a SQLite database, you must first have executed the database import block (**ImportDatabase)** and the database open block (**OpenDatabase**).

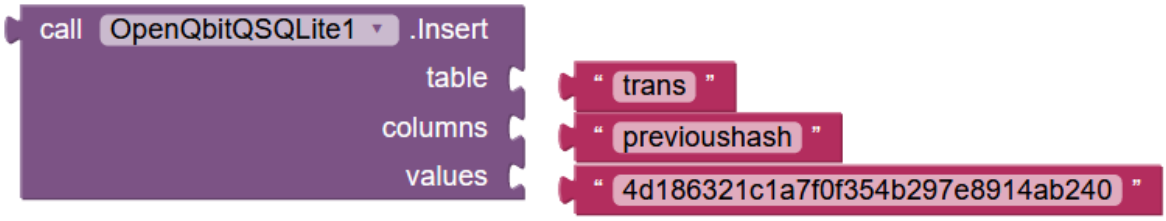
Block to import SQLite database. (**ImportDatabase**)


Input parameters: **fileName < String>** enter a path where you can find a database already created with SQLite format.

Output parameters: Starts a SQLite database to execute transactions.

Description: Block that starts a process in the SQLite database, you must first have executed the database import block (**ImportDatabase)** and the database open block (**OpenDatabase**).

Block to insert data in SQLite database. (**Insert**)



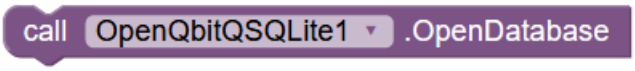

Mandatory unit(s): Block (**ImportDatabase), Block** (**OpenDatabase**).

Input parameters: **table** < String> , colums < **String> , values <** String> **, Use** before < Mandatory **dependence(s**)>

Output Parameters: Inserts a transaction into a SQLite database.

Description: Block that inserts data into the SQLite database, you must first have executed the database import block (**ImportDatabase)** and the database open block (**OpenDatabase**).

Block to open SQLite database (**OpenDatabase**)



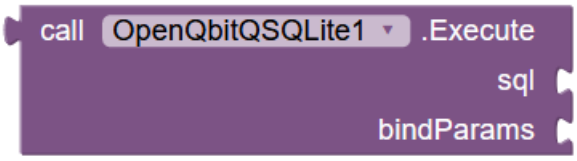

Mandatory unit(s): Block (**ImportDatabase).**

Input Parameters: **Use before <** Mandatory Dependency(s)>

Output parameters: Not applicable, start or open a SQLite database to perform transactions.

Description: Block that starts a SQLite database, there must be before.

Block for data query in SQLite (**Execute**).




Mandatory unit(s): Block (**ImportDatabase), Block** (**OpenDatabase**).

Input Parameters: **sql** < String> , **bindParams < String> ,** Use before < Mandatory **Dependency**(s)>

Output parameters: The SQL statement is executed to create a transaction in a SQLite database.

Description: Block that execute SQL statements in the SQLite database, must first have executed the block of importing database (**ImportDatabase)** and the block of opening the database (**OpenDatabase**).

## 21. Definition and use of security blocks.

In this session we will review the use of the blocks that provide us with levels of security to save, validate and transfer transactions from the Mini BlocklyChain network nodes.

The security blocks are based on the following extension:

I. OpenQbitAESEncryption extension.
II. OpenQbitAESDecryption extension.
III. OpenQbitAEStoString extension.
IV. OpenQbitEncDecData extension.
V. OpenQbitFileHash extension.
VI. OpenQbitRSA extension.
VII. OpenQbitSSHClient extension (would require)
VIII. OpenQbitStringHash extension.

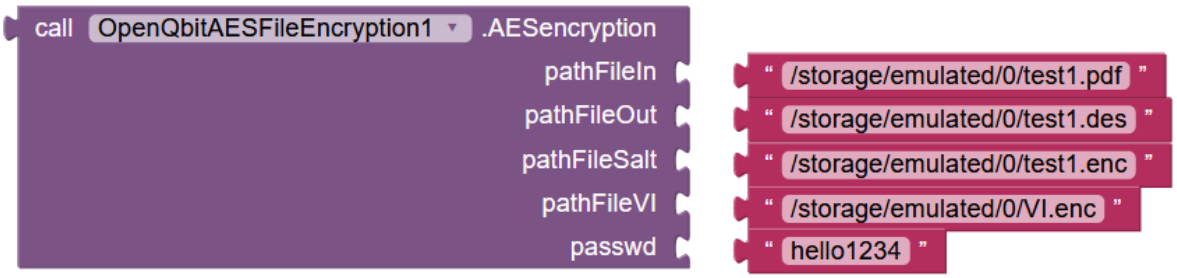
Except for the OpenQbitSSHClient extension which is mandatory, the above extensions are optional to use in the creation of a public or private Mini BlocklyChain network.

However, in order to have a secure system for your transactions depending on each business case, the use of the extensions that are optional should be applied at your discretion.

For example, in the case of using hash we can use a number of algorithm options such as MD5, SHA1, SHA128, SHA256, SHA512 for a string of characters as well as being applied to any type of file depending on the information flow of each system created with Mini BlocklyChain.

OpenQbitAESEncryption extension.

Block to encrypt file with AES security. (**AESEncryption**).



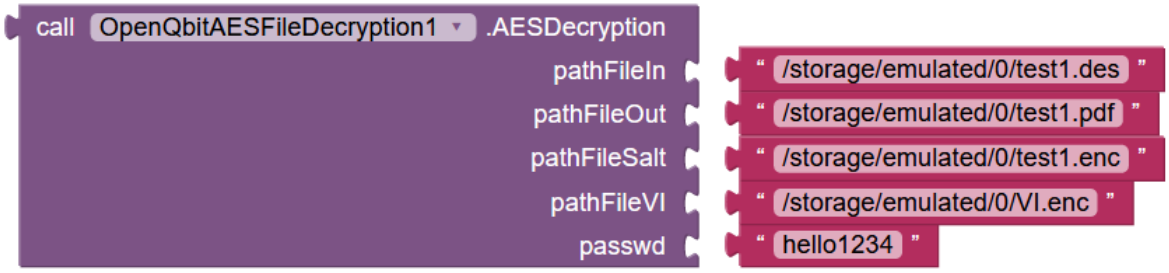

Input parameters: **pathFileIn** < String> , pathFileOut < **String> ,** pathFile **<** String**>, pathFileVI < String> and passwd** < String>. File names are arbitrary and at the discretion of each system design.

Output parameters: AES-encrypted file entered in the input parameter **pathFileIn**.

Description: Block that gives us three output files one of these is the source file already encrypted by the AES algorithm with a 256-bit key, the other two files are used to control the extension to decrypt and recover the original file.

OpenQbitAESDecryption extension.

Block to decrypt AES file. (**AESDecryption**).




Input parameters: **pathFileIn** < String> , pathFileOut < **String> ,** pathFile **<** String**>, pathFileVI < String> and passwd** < String>. The names of the files are the same as those obtained as a result in the block (**AESEncryption**).

Output parameters: Original file decrypted with AES entered in input parameter **pathFileIn,** in this case to decrypt the file it is entered in **pathFileIn the** encrypted file and in **pathFileOut it** will give us the original file (decrypted).

Description: Block that gives us a file in the pathFileOut parameter that will be the output decrypted by the AES algorithm with a 256-bit key.

OpenQbitAEStoString extension.

This extension is single-use per device session, i.e. it only works when the device is not rebooted (mobile phone), as the VI encryption value is generated temporarily.

NOTE: If the device is rebooted, the encrypted string cannot be recovered.

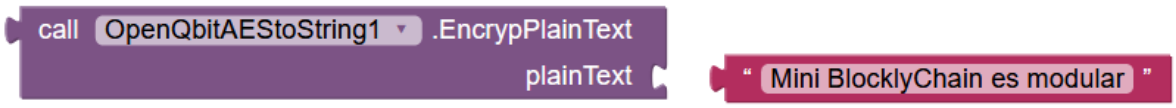
Block for temporary character string encryption (**DecrypSecreText**)


Input parameters: **secretText** < String>

Output parameters: Original character string decrypted with AES with 256-bit key

Description: Block that gives us a string of characters, is the input parameter that was introduced in the block (**EncrypPlainText**).

Block to decode a string (**EncrypPlainText**)




Input parameters: **plainText** < String>

Output parameters: AES-encrypted character string using 256-bit key.

Description: Block that gives us an alphanumeric character string encrypted with AES using a 256-bit digital key.

OpenQbitEncDecData extension.

Specialized encryption block for generic databases (**EncryptionData**)


Input parameters: **plainText** < String>

Output parameters: AES-encrypted character string using 256-bit key.

Description: Block that gives us an alphanumeric character string encrypted with AES using a 256-bit digital key.

Specialized encryption block for generic databases (**DescryptionData**)

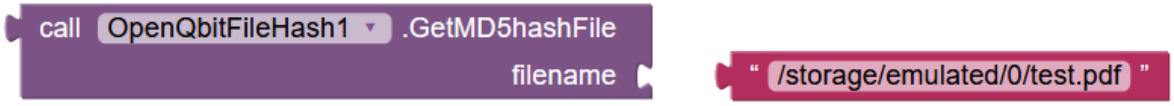

Input parameters: **plainText** < String>

Output parameters: AES-encrypted character string using 256-bit key.

Description: Block that gives us an alphanumeric character string encrypted with AES using a 256-bit digital key.

OpenQbitFileHash extension.

Block to generate MD5 from a file (**GetMD5hashFile**)

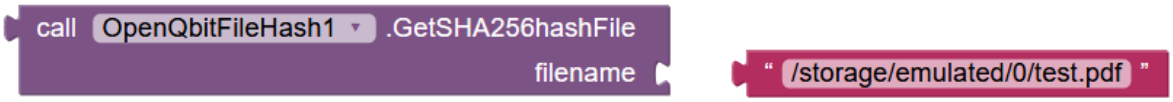



Input parameters: **filename** <string>

Output parameters: Delivers the MD5 hash file.

Description: Block to create the MD5 hash of the file given in the input parameter.

Block to generate SHA256 from a file (**GetSHA256hashFile**)

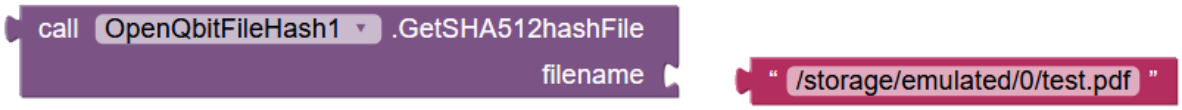



Input parameters: **filename** <string>

Output Parameters: Delivers the SHA256 archive hash.

Description: Block to create the SHA256 hash of the file given in the input parameter.

Block to generate SHA512 from a file (**GetSHA512hashFile**)

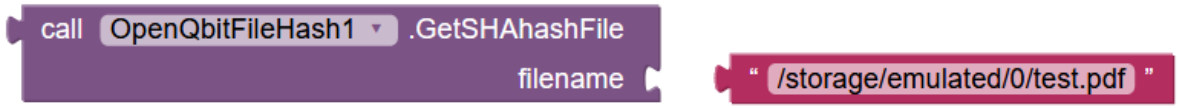



Input parameters: **filename** <string>

Output Parameters: Delivers the SHA256 archive hash.

Description: Block to create the SHA256 hash of the file given in the input parameter.

Block to generate SHA1 from a file (**GetSHA1hashFile**)



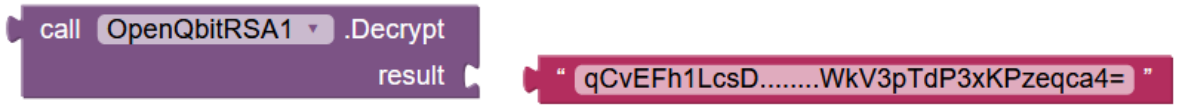

Input parameters: **filename** <string>

Output Parameters: Delivers the SHA1 archive hash.

Description: Block to create the SHA1 hash of the die in the input parameter.

OpenQbitRSA extension.

Block for **decrypting** string with RSA (**Decrypt**)




Required dependency(s): Block (**Encrypt),** Block (**OpenFromDiskPrivateKey**), Block (**OpenFromDiskPublicKey**).

Input parameters: **result** < String>

Output parameters: Character string decoded with RSA.

Description: Block that gives us a string of alphanumeric characters deciphered using key of the size that was used in the block (**GenKeyPair**).

Block for encrypting string with RSA (**Encrypt**)

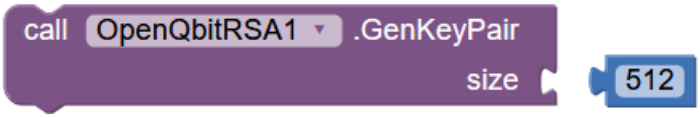

Required unit(s): Block (**GenKeyPair),** Block (**SaveFromDiskPrivateKey**), Block (**SaveFromDiskPublicKey**)

Input parameters: **plain** < String>

Output parameters: RSA-encrypted character string

Description: Block that gives us a string of alphanumeric characters deciphered using key of the size that was used in the block (**GenKeyPair**).

Block for **decrypting** string with RSA (**Decrypt**)

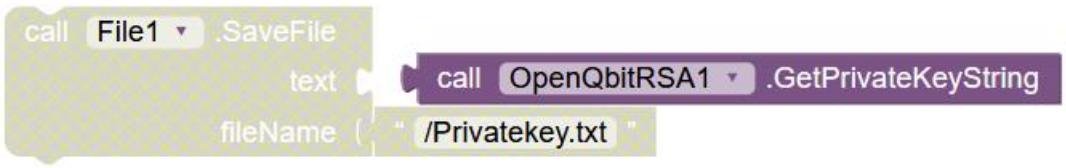



Input parameters: **size** < **Integer>**

Output parameters: Not applicable.

Description: Block to generate private key and public key based on the chosen size.

Block to obtain the private key (**GetPrivateKeyString**)



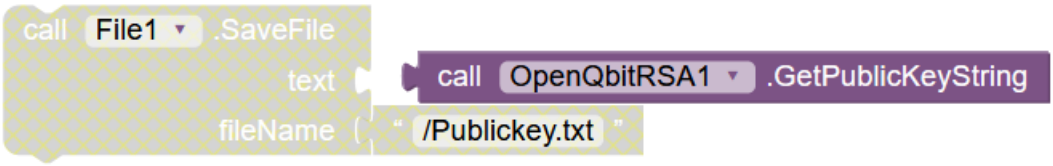

Mandatory Unit(s): Block (**GenKeyPair),** Block (**GenKeyPair**), Block (**File**)

Input parameters: **Not applicable.**

Output parameters: File with RSA encrypted character string (private key)

Description: Block that gives us an alphanumeric string representing the private key encrypted using the size key that was used in the block (**GenKeyPair**).

Block to obtain the private key (**GetPublicKeyString**)




Mandatory Unit(s): Block (**GenKeyPair),** Block (**GenKeyPair**), Block (**File**)

Input parameters: **Not applicable.**

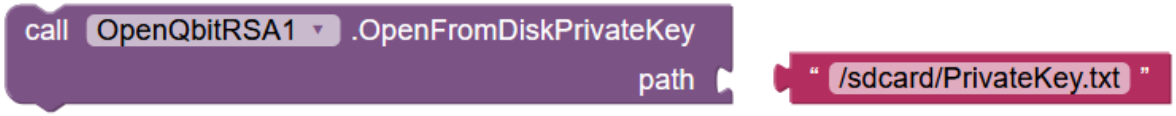
Output parameters: File with RSA encrypted string (public key)

Description: Block that gives us an alphanumeric string representing the public key encrypted using key size used in the block (**GenKeyPair**).

NOTE: In the previous blocks (**GetPrivateKeyString)** and (**GetPublicKeyString) in the** dependencies we will use the generic block (**File) of the** App Inventor application of the "Storage" pallet session.

This way of storing the private and public key by means of the block (**file**) can help us to have a better manipulation of the information.

Block to read private key from a file (**OpenFromDiskPrivateKey**).



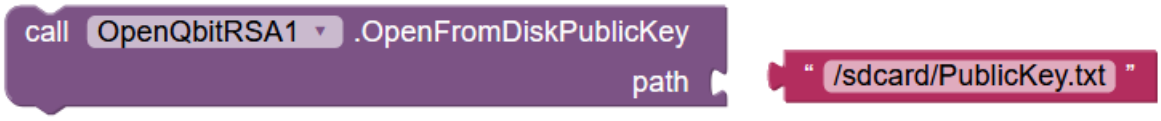

Required dependency(s): Block (**SaveFromDiskPrivateKey**) or Block (**GetPrivateKeyString)**.

Input parameters: **path** < String>

Output parameters: System load RSA private key encrypted character string.

Description: Block that gives us an encrypted alphanumeric character string of the private key stored in the provided file path.

Block to read public key from a file (**OpenFromDiskPublicKey**)



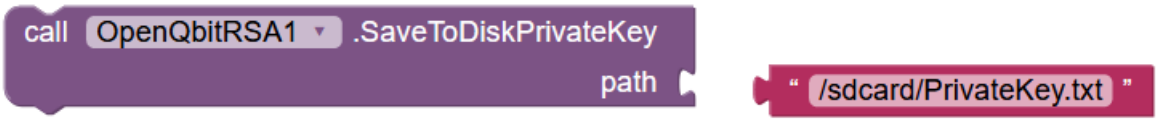

Mandatory Unit(s): Block (**SaveFromDiskPublicKey**) or Block (**GetPublicKeyString)**.

Input parameters: **path** < String>

Output parameters: Load in system RSA public key encrypted character string.

Description: Block that gives us an encrypted alphanumeric string of the public key stored in the provided file path.

Block for saving the private key to a file (**SaveToDiskPrivateKey**).



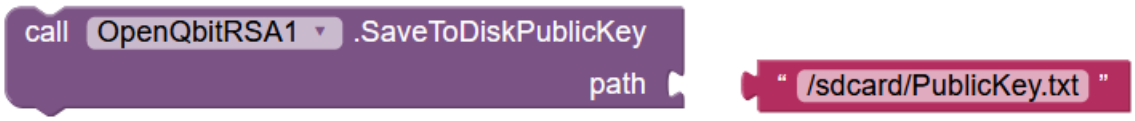

Mandatory unit(s): Block (**GenKeyPair).**

Input parameters: **plain** < String>

Output parameters: File with RSA encrypted string. (private key)

Description: Block that gives us a file with an alphanumeric string encrypted using private key of the size used in the block (**GenKeyPair**).

Block for saving the private key to a file (**SaveToDiskPublicKey**).




Mandatory unit(s): Block (**GenKeyPair).**

Input parameters: **plain** < String>

Output parameters: File with RSA encrypted string. (public key)

Description: Block that gives us a file with an alphanumeric string encrypted using public key of the size that was used in the block (**GenKeyPair**).

OpenQbitSSHClient extension.

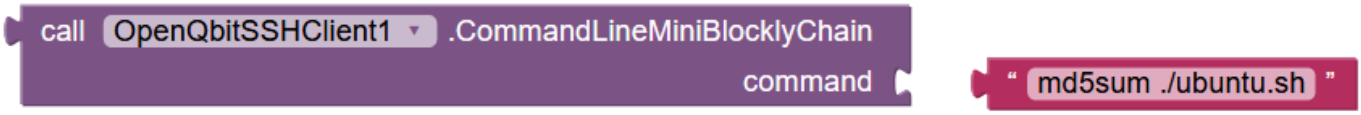
**Connector** Block SSH Client (**ConnectorMiniBlocklyChain**)


Input parameters: **username** <string>, **password <string>,** host <string>, port<integer>

Output parameters: If the connection with the ssh server of the Termux terminal is successful, it gives us a message; **"Connect SSH"**, if it is not successful, it gives us a **NULL** message.

Description: Communication block to connect Mini BlocklyChain to Termux terminal, via SSH (Secure Shell) communication protocol.

Block for Executing Commands in Termux Linux Terminal (**CommandLineMiniBlocklyChain**).

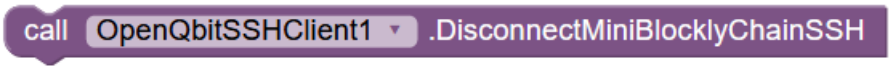



Input parameters: **command** <string>

Output parameters: Variable data, depending on the executed command or program.

Description: Command execution block in Termux terminal pre-requisite to make a connection with the block (**ConnectorMiniBlocklyChain**) can execute all kinds of commands online and/or get specific execution data from scripts or programs that have a CLI (Command-Line Interface) online.

**DisconnectMiniBlocklyChainSSH**.

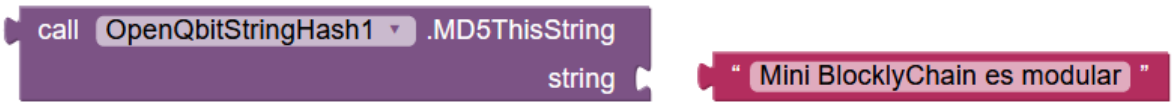



Input and output parameters: Not applicable (none)

Description: Block to close SSH session.

OpenQbitStringHash extension.

Block to generate MD5 string (**MD5ThisString**)

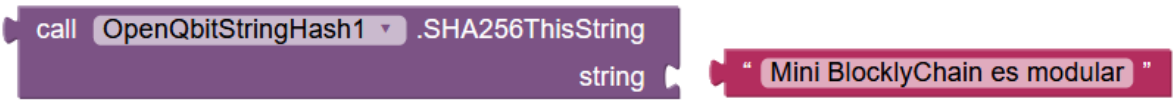



Input parameters: string

Output Parameters: Delivers the MD5 hash.

Description: Block to create the MD5 hash of the string given in the input parameter.

Block to generate SHA256 character string (**SHA256ThisString**)

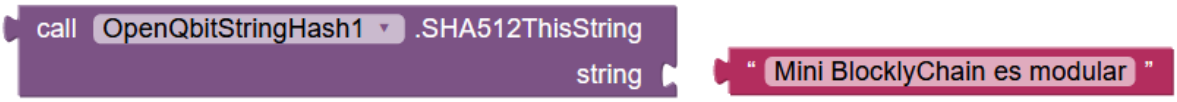



Input parameters: string

Output Parameters: Delivers the SHA256 hash.

Description: Block to create the SHA256 hash of the string given in the input parameter.

Block to generate SHA512 string (**SHA512ThisString**)

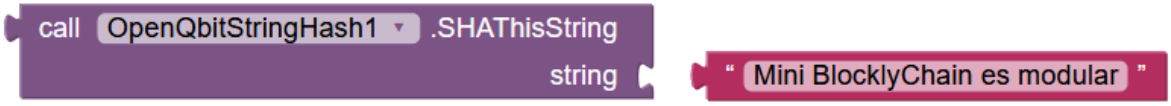



Input parameters: string

Output Parameters: Delivers the SHA512 hash.

Description: Block to create the SHA512 hash of the string given in the input parameter.

Block to generate SHA1 character string (**SHAThisString**)



Input parameters: string

Output Parameters: Delivers the SHA1 hash.

Description: Block to create the SHA1 hash of the string given in the input parameter.

## 22. Setting security parameters in Mini BlocklyChain.

The safety parameters are divided into three components of any system that is designed and applied in the following components:

a. Redis database (backup communication network)
b. Peer to Peer Syncthing System
c. Integrate security to protect the SQLite database
d. Developer environment for module integration

a. Redis database (backup communication network)

Renaming dangerous commands, Redis' additional built-in safety feature involves renaming or disabling some commands that are considered dangerous.

When executed by unauthorized users, these commands can be used to reconfigure, destroy or delete their data. Like the authentication password, renaming or disabling commands is configured in the same `SECURITY` section of the `/etc/redis/redis.conf` file.

Some of the commands considered dangerous: FLUSHDB, FLUSHALL, KEYS, PEXPIRE, DEL, CONFIG, SHUTDOWN, BGREWRITEAOF, BGSAVE, SAVE, SPOP, SREM, RENAME, and DEBUG. This is not a complete list, but renaming or disabling all the commands on that list is a good start to improving the security of your Redis server.

Depending on your specific needs or those of your site, you must rename or disable a command. If you know that you will never use a command that can be manipulated, you can disable it. On the other hand, you may want to rename it.

To enable or disable Redis commands, reopen the configuration file:

```
$ vi /etc/redis/redis.conf
```

Warning: The following steps to disable and rename commands are examples. You should only choose to disable or rename the commands that apply to you. You can review the complete list of commands and determine how they can be misused in redis.io/commands.

To disable a command, simply rename it so that it becomes an empty string (symbolized by a pair of quotes with no characters between them), as shown below:

/etc/redis/redis.conf

```
. . .
# It is also possible to completely kill a command by renaming it into
# an empty string:
#
rename-command FLUSHDB ""
rename-command FLUSHALL ""
rename-command DEBUG ""
. . .
```

To rename a command, give it another name as shown in the examples below. Renamed commands should be hard for others to guess, but easy for you to remember.

/etc/redis/redis.conf

```
. . .
# rename-command CONFIG ""
```

```
rename-command SHUTDOWN SHUTDOWN_MENOT
rename-command CONFIG ASC12_CONFIG
. . .
```

Save the changes and close the file.

After renaming a command, apply the change by restarting Redis:

- `sudo systemctl restart redis.service`

To test the new command, enter the Redis command line:

- `redis-cli`

Then, perform the authentication:

- `auth your_redis_password`

```
Output
OK
```

As in the previous example, suppose you renamed the CONFIG command to ASC12_CONFIG. First, try using the original CONFIG command. Because you renamed it, it shouldn't work:

- `config get requirepass`

```
Output
(error) ERR unknown command 'config'
```

However, the renamed command can be called up successfully. It is not case sensitive:

- `asc12_config get requirepass`

```
Output
1) "requirepass"
2) "your_redis_password"
```

Finally, you will be able to close the redistricting:

- `exit`

Note that if you already use the Redis command line and restart Redis, you will need to authenticate again. Otherwise, if you type a command this error will be displayed:

```
Output
NOAUTH Authentication required.
```

b. Peer to Peer Syncthing System

In the use of the "Peer to Peer" system between nodes the system has the following three elements applied in the communication network

-Private: there is no information stored anywhere other than your computers. There is no central server that can be compromised (legally or illegally).

-Encrypted: All communication is secured through the TLS (Transport Layer Security) protocol; a cryptographic protocol that includes a perfect sequence to prevent anyone outside your confidence from accessing your information.

-Authenticated: Each node is identified with a strong cryptographic certificate. Only the nodes that you have explicitly allowed, can connect to your information.

https://docs.syncthing.net/users/security.html

Using the online command SyncthingManager:

https://github.com/classicsc/syncthingmanager

c. Integrate security to protect the SQLite database

When using the extension (OpenQbitSQLite) or in its case the extension (ConnectorSSHClient) to use the SQLite CLI both extensions can be combined with the security extension AES (OpenQbitEncDecData).

See Appendix "Example of Mini BlocklyChain system creation".

d. Developer environment for module integration

To implement security in the development environment can be done with two processes:

- Restrict use of OpenJDK libraries.
- Use restricted to authorized system nodes.

## 23. Annex "Creation of KeyStore & PublicKeys databases".

A KeyStore is a repository of security certificates, either authorization certificates or public key certificates, passwords or generic security keys such as the corresponding private keys (addresses) of a user, which is used to create or process transactions.

It is an encrypted database so that only the owner can make use of it. Normally it is a local type, however, in the Mini BloclyChain system it is a secure database but it is shared and distributed in all the nodes, this is basically due to a simple reason, all the nodes must know the addresses of all the users (public addresses), the private addresses are always local and are of unique and exclusive use of each user, however when making an entrance(deposit) or exit(expense) transaction every transaction always has at least three components when being created: Origin address, destination address and asset or value that is being sent.

To create our KeyStore we will have to follow the following steps and requirements.

A first requirement is to have a storage type where they will be stored, in our case we will use the SQLite database and we will create two KeyStore one with the user's private keys that will be local and it will be secured "encrypted" the information and another database that will store the public keys this base will be shared in all the nodes of the network, the base that will be shared in the network will also be encrypted, however this one will have a password that only the members (nodes) of the network will be able to share.

Second fundamental requirement is the process of information encryption, this will be done with the specialized extension for use in generic database called (OpenQbitEncDecData) that uses an AES algorithm.

The extension (OpenQbitEncDecData) is functional for the verification of unitary elements of the block chain, however, in case of having to check the total integrity of the block chain it will not be efficient so we will also be performing an encryption of a copy, but in this case it will be through the extensions: (OpenQbitAESEncryption) and (OpenQbitAESDecryption) that work on a file, not by referenced data.

NOTE: The SSH server must be running on the Termux terminal for the KeyStore database to function properly. Execute the command in the terminal:

$ sshd

Extensions based on the AES algorithm can be used for any type of data or file and this algorithm was chosen as it is the only one that has been proven to be proof against quantum computing based attacks.

| Criptosistema | Categoría | Tamaño de clave | Parámetro de seguridad | Algoritmo cuántico estimado que rompa el criptosistema | Nº de qubits lógicos necesarios | Nº de qubits físicos necesarios | Tiempo necesario para romper el sistema | Estrategias de reemplazo cuántico-resilientes |
|---|---|---|---|---|---|---|---|---|
| AES-GCM | Cifrado simétrico | 128<br>192<br>256 | 128<br>192<br>256 | Algoritmo de Grover | 2.953<br>4.449<br>6.681 | $4{,}61 \times 10^{6}$<br>$1{,}68 \times 10^{7}$<br>$3{,}36 \times 10^{7}$ | $2{,}61 \times 10^{12}$ años<br>$1{,}97 \times 10^{22}$ años<br>$2{,}29 \times 10^{32}$ años | |
| RSA | Cifrado asimétrico | 1024<br>2048<br>4096 | 80<br>112<br>128 | Algoritmo de Shor | 2.290<br>4.338<br>8.434 | $2{,}56 \times 10^{6}$<br>$6{,}2 \times 10^{6}$<br>$1{,}47 \times 10^{7}$ | 3,58 horas<br>28,63 horas<br>229 horas | Migrar a un algoritmo PQC seleccionado por el NIST |
| ECC Problema del logaritmo discreto | Cifrado asimétrico | 256<br>386<br>512 | 128<br>192<br>256 | Algoritmo de Shor | 2.330<br>3.484<br>4.719 | $3{,}21 \times 10^{6}$<br>$5{,}01 \times 10^{6}$<br>$7{,}81 \times 10^{6}$ | 10,5 horas<br>37,67 horas<br>95 horas | Migrar a un algoritmo PQC seleccionado por el NIST |
| SHA256 | Minado de Bitcoin | N/A | 72 | Algoritmo de Grover | 2.403 | $2.23 \times 10^{6}$ | $1{,}8 \times 10^{4}$ años | |
| PBKDF2 con 10.000 iteraciones | Hashing de contraseñas | N/A | 66 | Algoritmo de Grover | 2.403 | $2.23 \times 10^{6}$ | $2.3 \times 10^{7}$ años | Abandonar la autenticación basada en contraseñas |

The above table is referenced to National Academies of Science, Engineering and Medicine.

https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects

Description

Quantum mechanics, the subfield of physics that describes the behavior of very small particles (quantums), provides the basis for a new computing paradigm. First proposed in the 1980s as a way to improve the computational modeling of quantum systems, the field of quantum computing has recently attracted significant attention due to progress in the construction of small-scale devices. However, significant technical advances will be required before a practical quantum computer can be achieved on a large scale.

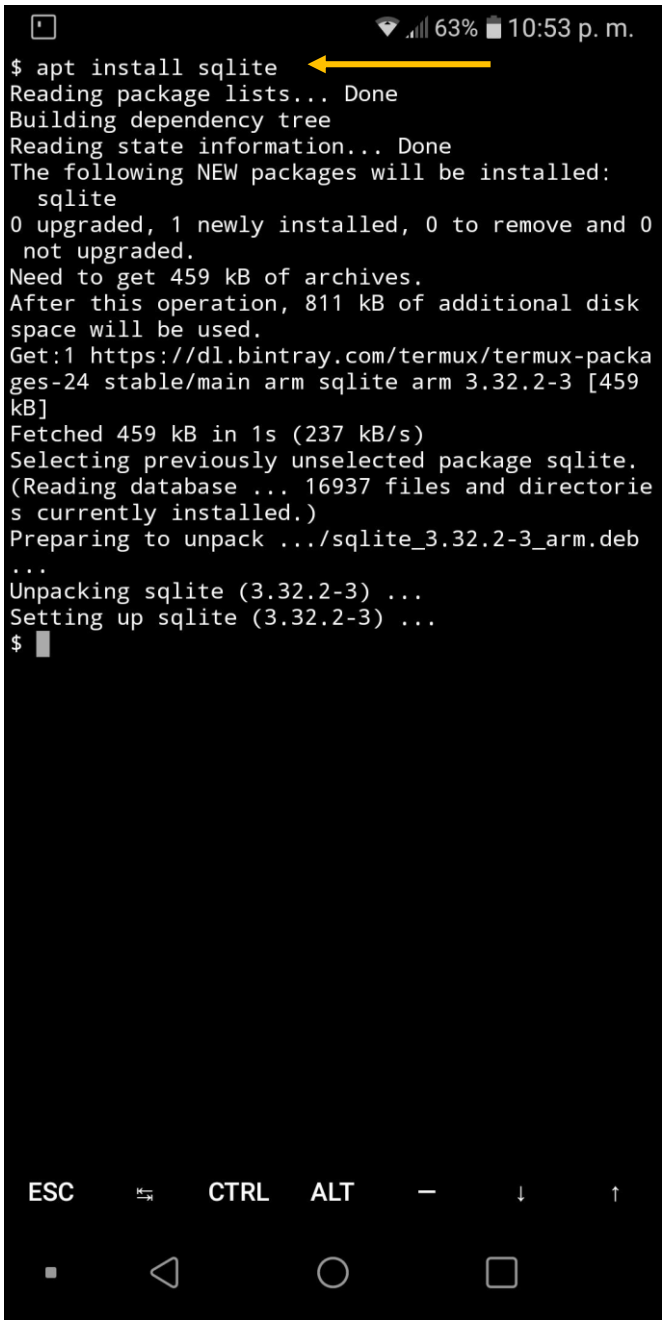
Quantum Computing: Progress and Prospects provides an introduction to the field, including the unique characteristics and limitations of the technology, and assesses the feasibility and implications of creating a functional quantum computer capable of addressing real-world problems. This report considers hardware and software requirements, quantum algorithms, drivers of advances in quantum computing and quantum devices, benchmarks associated with relevant use cases, the time and resources needed, and how to assess the probability of success.

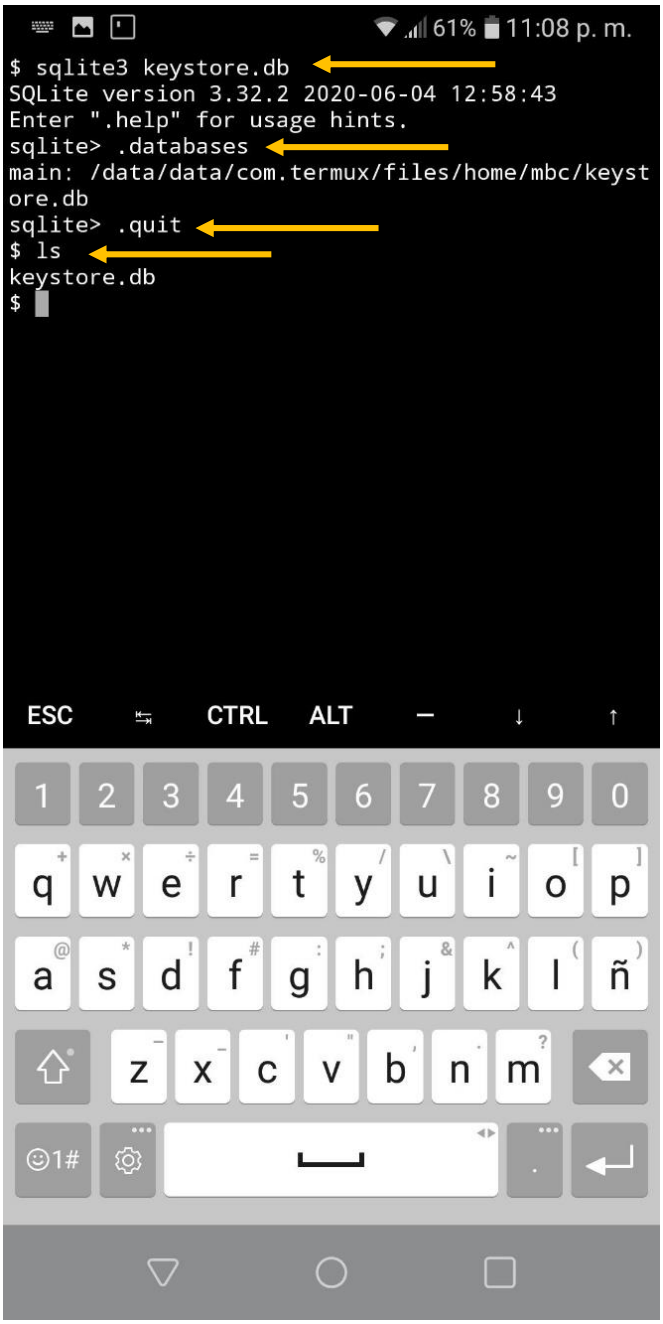
Installation of SQLite database handler in TERMUX terminal and test the CLI (Command-Line) of the **sqlite3** command by creating a database called **keystore.db**

We use the commands:

$ apt install sqlite

$ sqlite3 keystore.db




Then inside the **sqlite>** handler we execute the .databases sentence to check in which path is the database we are creating, then to exit and save the database we give the sentence of . quit.

NOTE: In both statements or commands within the sqlite> handler you must first put a dot "." in the syntax.

Finally we check that the (empty) database has already been created by giving the command:

$ ls

We proceed to create a table where the primary keys will be stored in three different formats: hexadecimal, binary and the Mini BlockyChain user reference address which is the public key of its respective primary key.

AES data encryption will be applied only in hexadecimal and binary formats. In the case of the user's address addrMBC and the alias not because it is the public key which can and should be shared across the network in order to receive transactions, as well as the alias to perform the search by this field.

Created a table called "privatekey" in the database in SQLite (keystore.db).

```
CREATE TABLE privatekey (
 id integer primary key
 a.k.a.              VARCHAR(50) NOT NULL
 addrHexVARCHAR      (65) NOT NULL
 addrMBCVARCHAR      (65) NOT NULL
 addrBinBLOB    NOT NULL
);
```

Let's execute the sentences in the sqlite CLI to create the keystore.db database.

Let's use the sqlite3 command line again with the following command:

$ sqlite3

This will send us inside the sqlite> database manager. In this first one we will open the database already created to be able to work on it with the following sentence inside the manager:

Sqlite> .open keystore.db

Then we will enter the SQL statement "CREATE TABLE" to create the privatekey table.

Next, two options are shown to create the same table, one is through a single line where all the SQL statement of the elements that integrate it are included. The second example is through the introduction of each element that integrates the structure in a segmented way.

```
$ sqlite3

SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> CREATE TABLE privatekey (id integer primary key AUTOINCREMENT NOT NULL, aka VARCHAR(50) NOT NULL, addrHex VARCHAR(65) NOT NULL, addrMBC VARCHAR(65) NOT NULL, addrBin BLOB NOT NULL);
sqlite> .quit
```

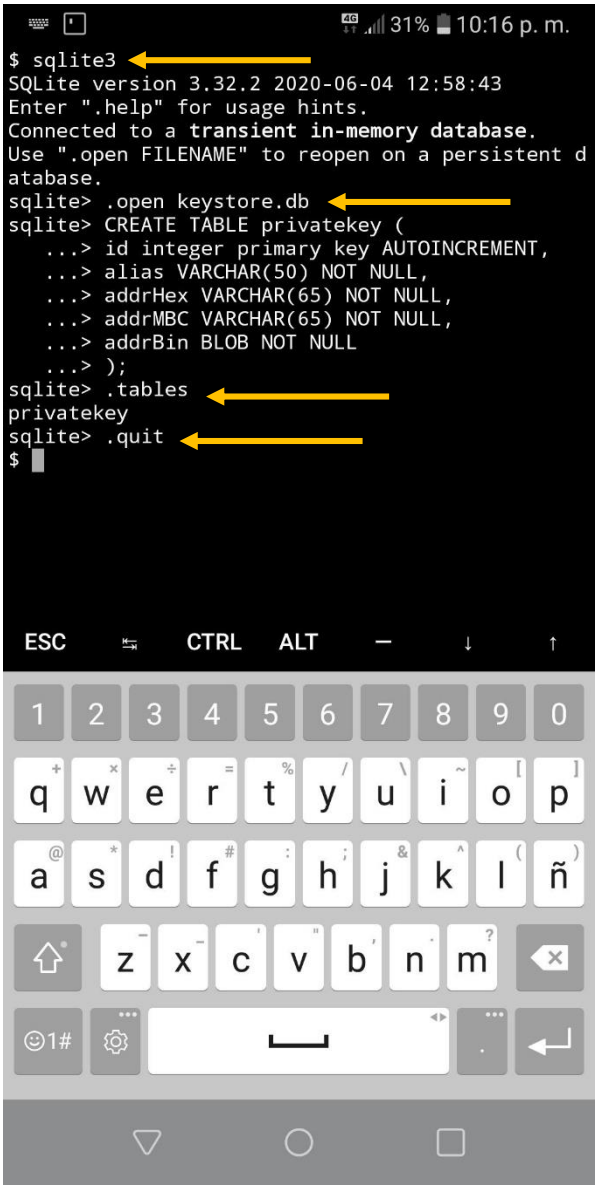
The creation of the same **privatekey** table is shown next, but it is by introducing the SQL statement in a segmented way:

```
sqlite> CREATE TABLE privatekey (
   ...> id    integer primary key AUTOINCREMENT
   ...>    aka VARCHAR(50) NOT NULL,
   ...> addrHex    VARCHAR(65) NOT NULL,
   ...> addrMBC    VARCHAR(65) NOT NULL,
   ...> addrBin    BLOB NOT NULL
 ...> );
sqlite> .tables
privatekey
sqlite> .quit
```

The two examples above give the same result. Later we execute the **.tables** sentence to check that the privatekey table has been created, at the end we will give the **.quit sentence** with this process we have already created the **privatekey** table inside the SQLite **keystore.db** database.

Until this moment we already have the structure of the keystore.db database and its privatekey table where the private keys will be stored in an encrypted way.

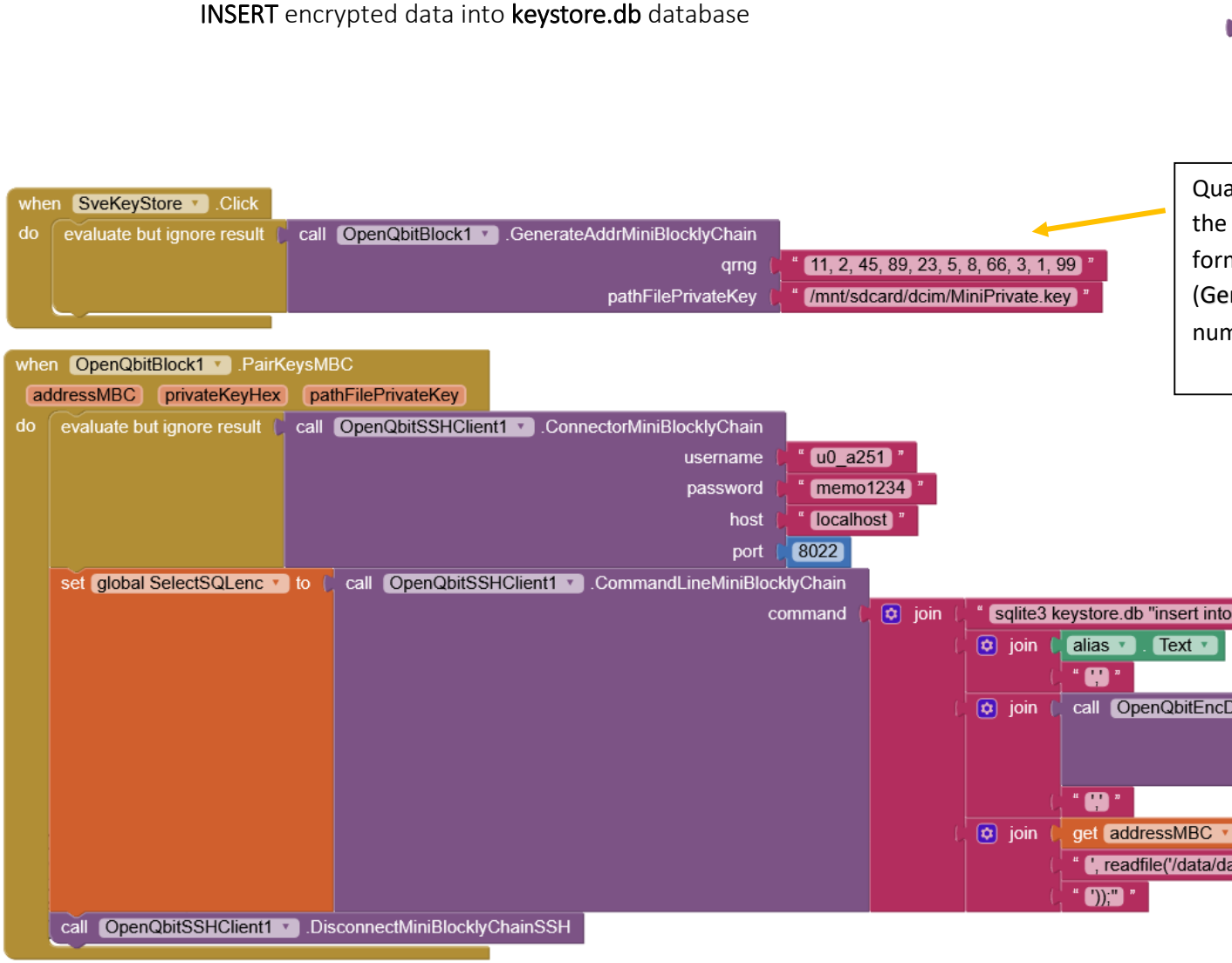
This should show something similar in the node (mobile phone) with the TERMUX terminal.




We will now use some security extensions to insert "private key" data and then consult this data.

We will use the block to generate a new user address (GenerateAddrMiniBlocklyChain), this block will give us the private address in two formats (hexadecimal and binary), as well as the public address in MBC generic address format. We will also use the extension (OpenQbitSSHClient) and the extension (OpenQbitEncDecData) to see more details of these check the section "Definition and use of security blocks**".** In the Termux terminal you should be running the SSH service.

**INSERT** encrypted data into **keystore.db** database

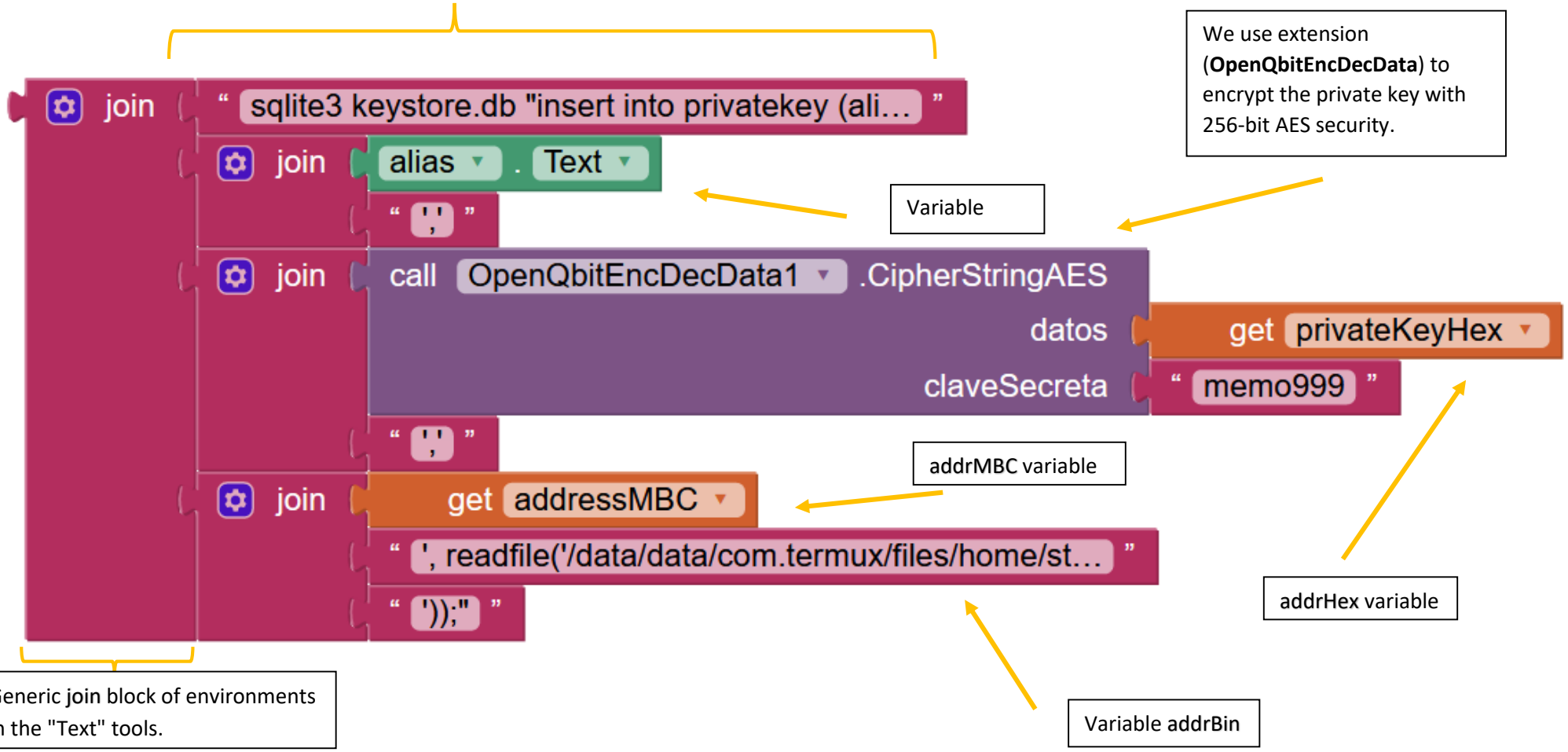



Quantum random number series generated by the block (**ApiGetQRNGInteger**) check how to format the JSON result, since the block entry (**GenerateAddrMiniBlocklyChain**) needs a series of numbers only separated by commas ",".

The syntax of the command is very important, we must introduce the following command in the right format in the Blockly environment.

The values of values will always be the variables, example:

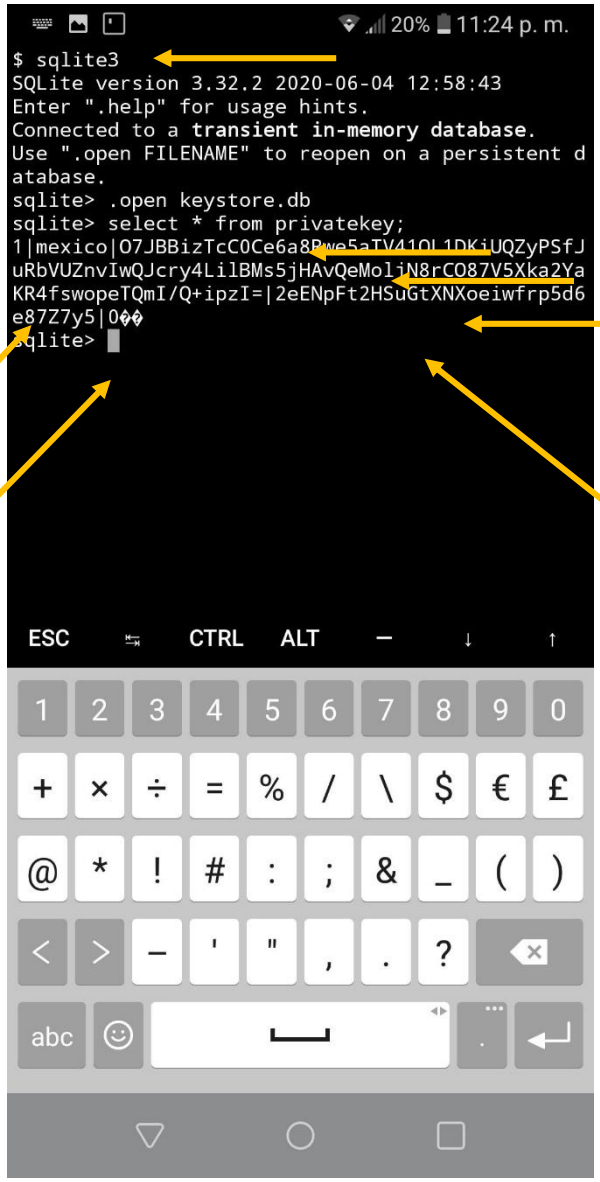
sqlite3 keystore.db "insert into privatekey (alias, addrHex, addrMBC, addrBin) values ('memo', 'QWERTY', 'MBC12345', readfile('/data/data/com.termux/files/home/storage/dcim/MiniPrivate.key'));"



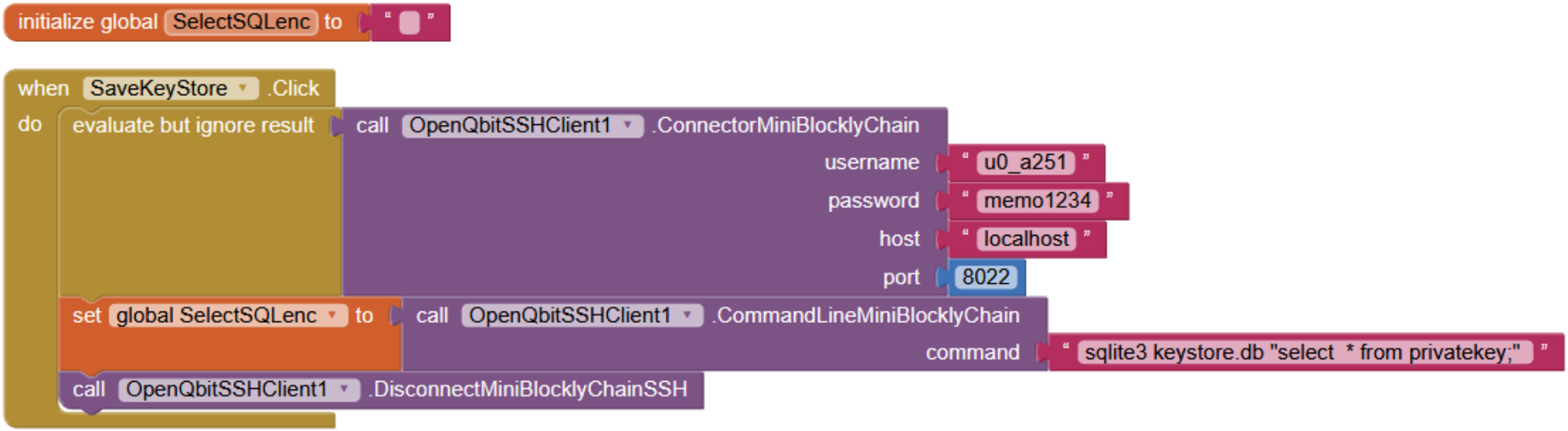
After executing the blocks we will have a result in the privatekey table of the keystore.db database similar to the following:

We enter the sqlite handler in the Termux terminal, open the keystore.db base and execute the sentence:

$ sqlite3
SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> select * from privatekey;



**CONSULT** all data in the **privatekey** table of the SQLite **keystore.db** database




**CONSULT** the aliases of the data in the **privatekey** table of the SQLite **keystore.db** database

sqlite3 keystore.db “select alias from privatekey;”

**Query** all data in the addrHex column encrypted in the **privatekey** table of the SQLite **keystore.db** database

sqlite3 keystore.db “select addrHex from privatekey;”

**CONSULT** to retrieve the addrBin private key in the **privatekey** table of the SQLite **keystore.db** database

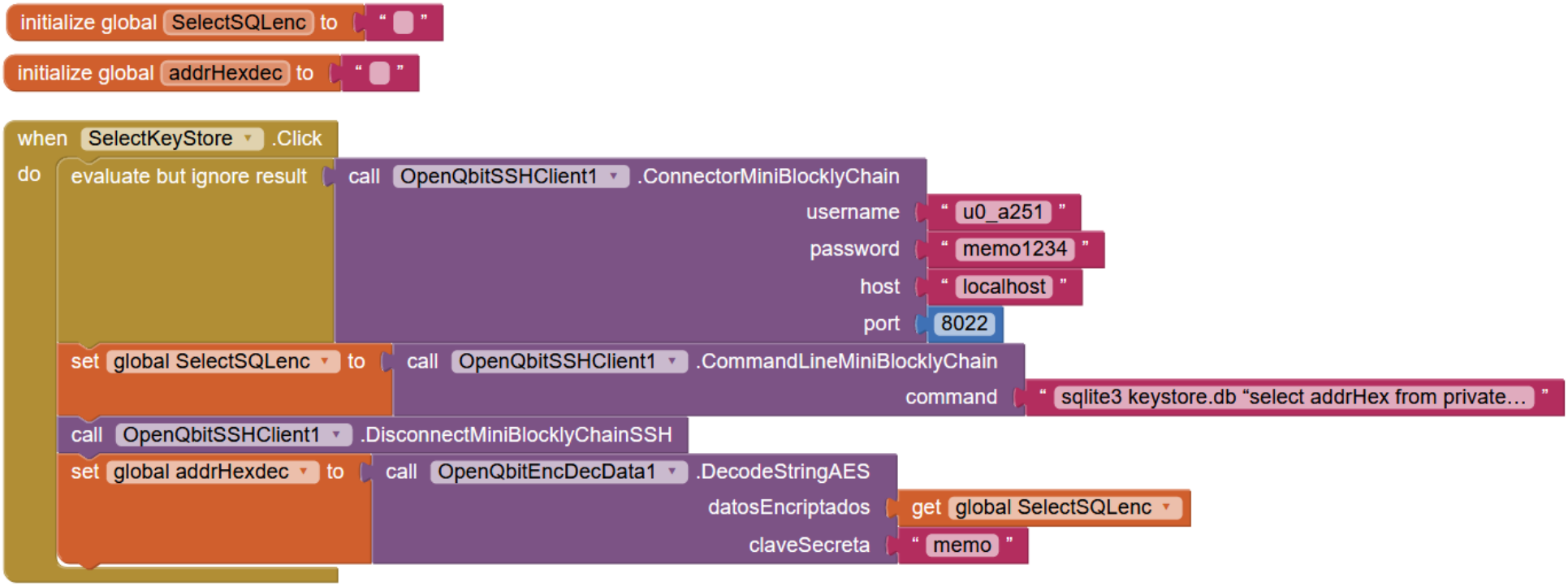
sqlite3 ***writefile****('PrivateKey.key', addrBin) FROM privatekey WHERE alias='mexico';* (2)

(1) This type of query will be the main one to consult the private key to perform the process of signing the transactions.

**Query for Decryption of Data** by Alias in the addrHex column of the **privatekey** table of the SQLite **keystore.db** database

In this case we will use the block (**DecodeStringAES**) to decode the data stored in the "addrHex" column.

sqlite3 keystore.db “select addrHex from privatekey where='mexico';”



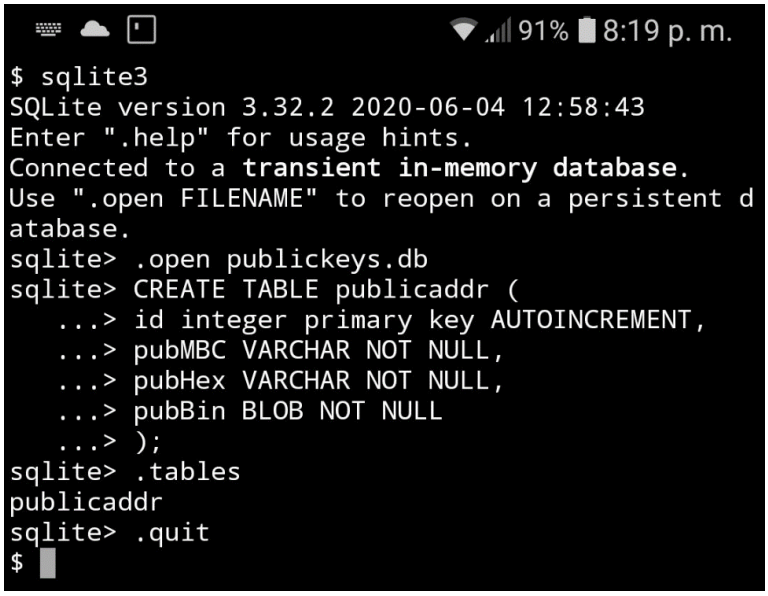

This consultation gives us the private key in hexadecimal format, this is the fundamental part of any reception or sending of transactions.
It is repeatedly recommended that a backup of this keystore.db database be kept.

Database design **publickeys.db** with table publicaddr:

```
$ sqlite3

SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .open publickeys.db
sqlite> CREATE TABLE publicaddr (id integer primary key AUTOINCREMENT NOT NULL, pubMBC VARCHAR NOT NULL, pubHex VARCHAR NOT NULL, pubBin BLOB NOT NULL);
sqlite> .quit
```

Next, the creation of the **published** table is shown with the following SQL sentence in segmented form:

```
sqlite> CREATE TABLE publishedaddr (
   ...> id    integer primary key AUTOINCREMENT
   ...> pubMBC    VARCHAR NOT NULL,
   ...> pubHex    VARCHAR NOT NULL,
   ...> pubBin    BLOB NOT NULL
   ...> );
sqlite> .tables
publishedaddr
sqlite> .quit
```



## 24. Annex "RESTful SQLite GET/POST commands".

Next, the different commands that we can use in the Restful SQLite of the backup network are shown. These have been consulted from the original development of GITHUB (https://github.com/olsonpm/sqlite-to-rest).

CRUD operations (Create, Read, Update and Delete) RESTful.

The following is a list of available operations made available by the RESTful API in the form of pseudo examples. We will assume a base "op.sqlite" and two associated "trans" tables for transactions and another "sign" for source address, destination address and asset value with two columns id INTEGER PRIMARY KEY.

As noted in the limitations, please note that the methods (DELETE and POST) can only affect one row at a time.

OBTAIN This allows for the greatest variation. Later on we will see all the available query operators.

Headings can be specified just below the URLs.

**/transRequests**
for all rows

**/trans?id=1Where**
id = 1

- **/trans**

```
range: rows=0-2
```

First three rows

**/trans**

```
range: rows=-5Last
```

five rows

**/trans**

```
range: rows=0-
```

As many rows as the server can provide, which in practice will be the lesser of maxRange and total row count.

**/trans**

```
range: rows=1-
```

As many rows as the server can provide, starting from row 1.

• **/trans**
`order: name`
Ordered by name ascending

• **/trans**
`order: name desc`
Ordered by name descending

**/trans**
`order: name desc,id`

Contributed, but first sorted by descending name, and in the case of a tie by ascending id.

**/trans?id>1**
Where id > 1

**/trans?id>=2&id<5**
Where id >= 2 and id < 5

**/trans?name_NOTNULL**
Where name is not null

**/trans?name_ISNULLWhere**
name is null

**/trans?id!=5&name_LIKE'Spotted%'.**
Where id ! = 5 and name is LIKE "Spotted%" (ignore quotes)

**/trans?id>=1&id<10&name_LIKE'Avery%'.**
`order: name desc,id range: rows=2-4`

Contributed for the sake of example.

Obtain transaction with identifiers between 1 and 9 inclusive, with a name like "Avery%", ordered first by descending name and then by ascending identifier, obtaining the third to fifth row of the result. Or in SQL:

DELETE Requires a query string with all primary keys set equal to a value. This requires a maximum single-row deletion.

**/trans?id=1**
Deletes trans with id=1

If the transaction instead had a main key composed of id and name.

**/trans?id=1&name='Avery IPA'.**

POST create

You must not pass a query string. If a query string is passed, a POST update is assumed. All POST requests must pass the header content type: application / json.

Please note that the body must contain all the non-cancelable PRIMARY KEY columns and not INTEGRATE. Otherwise, a 400 response will be sent indicating which fields were lost. The nullable columns will be null and the INTEGER PRIMARY KEY columns will be automatically increased according to the sqlite3 specifications.

The JSON data will be specified just below the URLs.

• **/trans**

```
{"id":1,"name":"Serendipity"}
```

Creates a trans with id = 1 and name = 'Serendipity'

• **/trans**

```
{"id":1}
```

Creates a trans with id = 1 and name = NULL

**/trans**

```
{"name": "Serendipity"}
```

Creates a transaction with id set to the following value increased by sqlite3 Specifications INTEGER PRIMARY KEY.

**/trans**

```
{}
```

Creates a transaction with increased id and the name set to NULL.

POST update

It must contain a query string. Without a query string, POST creation is assumed. As with POST create, the header content type: application / json is required.

The query string must contain all primary keys to ensure that only one row is updated. If incorrect values are passed on, a 400 will be returned with the offending keys.

The body of the request must contain a non-empty object and must contain valid keys corresponding to the column names.

The JSON data will be specified just below the URLs.

**/trans?id=1**

```
{"id":2}
```

Update the transaction with an ID of 1 by setting it to two.

**/trans?id=1**

```
{"name":" MCBza45Rt56cvbgfdR2Swd788kj"}
```

Update the transaction with the ID of 1 by setting its name or value to "MCBza45Rt56cvbgfdR2Swd788kj".

If the trading desk instead had a main key composed of id and name.

**/trans?id=1&name=MCBza45Rt56cvbgfdR2Swd788kj**
`{"name":" MCB3ofFG5Hj678MNb09vLdfaaSx "}`

Update the transaction where id is one and the address is MCBza45Rt56cvbgfdR2Swd788kj, setting the MCB3ofFG5Hj678MNb09vLdfaaSx.

Reference

isSqliteFile

Simply check the first 16 bytes of the file to see if it equals 'sqlite format 3' followed by a null byte.

isDirectory

Returns the result of fs.statsSync followed by .isDirectory

isFile

Returns the result of fs.statsSync followed by .isFile

GET consulting operators

The enquiry conditions must be marked with connection symbols, e.g. id> 5 & name = MCBza45Rt56cvbgfdR2Swd788kj

Binary operators (requires a value after) Man.

**=**
**!=**
**>=**
**<=**
**>**
**<**
**LIKE**

LIKE is special because it must have simple opening and closing quotes. Otherwise, a 400 error will be generated showing where the analysis could not be completed and what was expected. See CRUD RESTful Operations for examples.

Single operators (must follow the name of a column)

IS NULL

NOT NULL

Router configuration object

isLadenPlainObject

The purpose of this object is to provide a generic configuration for the sqlite router. The following properties are allowed:

prefix: isLadenString The string passed to the koa-router prefix builder option. For example, the skeleton server does not specify a prefix, which allows the beer API to be hit directly from the root of the http domain: // localhost: 8085 / trans. If you set the prefix to '/ api', then you should send requests to http: // localhost: 8085 / api / trans.

allTablesAndViews: a tabular configuration object

The settings specified in this object will be applied to all tables and views, optionally overridden by the tablesAndViews property.

tablesAndViews: isLadenPlainObject The past object must have keys that match the database column or view the names. Otherwise, a friendly error message will be issued. The values for each table and view must be a tabular configuration object.

Tabular configuration object

isLadenPlainObject This object represents settings that can be set for views or tables. It allows the following properties:

maxRange: isPositiveNumber

Default application: 1000

This is the maximum range your server will allow requests. If a GET request arrives without a range header, the specification assumes that you want the entire resource. If the number of rows resulting in GET is greater than maxRange, a 416 status is returned with the custom max-range header. The default value of the application is deliberately conservative in the hope that authors will set maxRange according to their needs.

Please note that 'Infinity' is a valid positive number.

flags: isLadenArray

Currently, the only accepted indicator is the string 'sendContentRangeInHEAD'. When set, HEAD requests will return the range of available content in the form content-range: * / <max-range>. The reason it is configurable is that calculating the maximum range may be more work than it is worth, depending on the server load and the size of your tables.

Custom headers

Application

order: this header is only defined for GET, and can be considered as the equivalent of sql ORDER BY. It must contain a comma-delimited column name, each optionally followed by a space and the strings 'asc' or 'desc'. If incorrect order values are sent, a 400 response will indicate which ones.

Answer

Not all are necessarily customized, but all use is outside the specification and therefore needs clarification.

GET

max-range: this header is returned when the number of rows requested exceeds the configured maxRange. Note that the request may not specify the range header, but the number of rows resulting in that resource will still be verified.

range of contents: rfc7233 states

Only status codes 206 (Partial Content) and 416 (Unsatisfactory Range) describe a meaning for Content-Range.

When sqlite-to-rest responds with a status code 200, the content range header is sent with the format 206 of <row start> - <row end> / <row count>.

When a request is sent without a range header and the resulting number of rows exceeds maxRange, a 400 is returned with the range of content set to the 416 format of * / < row count>

Note that this header can be returned in a HEAD response.

accept-order: will be returned if the order of the request header has an incorrect syntax or incorrect column names specified. For more details, see HEAD -> accept-order below.

## 25. Annex "Java Code SQLite-Redis Connector".

A code of the link between the communication of both SQLite-Redis databases is shown below. Basically, as we can see, the connector changes the SQLite format to a generic string of the data (transactions) for Redis to receive.

The above process acts as a transaction queue trigger to all nodes that are connected and are possible candidates for processing the current transaction queue.

When the Redis database receives the transaction queue by the Master-Slave configuration it has, it will distribute the information in real time and equally to all the nodes.

Another fundamental point is that all the nodes must be synchronized in their clock, this will be done as we saw before with the functionality of the query to a server pool NTP (Network Time Protocol).

This connector also performs the first level of data security review by calculating the Merkle Root tree of all transactions.

Base code of the SQLite-Redis Connector.

```
import java.sql.*;
import java.util.*;
import java.util.arrays;
import java.util.list;
import java.util.array;
import java.security.*;
import java.security.MessageDigest;
import redis.clients.jedis.Jedis;
class RediSqlite{
    public String previousHash;
 public String merkleRoot;
    public static ArrayList<String> transactions = new ArrayList<String>();
    public static ArrayList<String> treeLayer = new ArrayList<String>();
    public static String getMerkleRoot() {
        int count = transactions.size();
        int item = 1;
        Odd int = 0;
        ArrayList<String> previousTreeLayer = transactions;
        while(count > 1) {
            treeLayer = new ArrayList<String>();
```

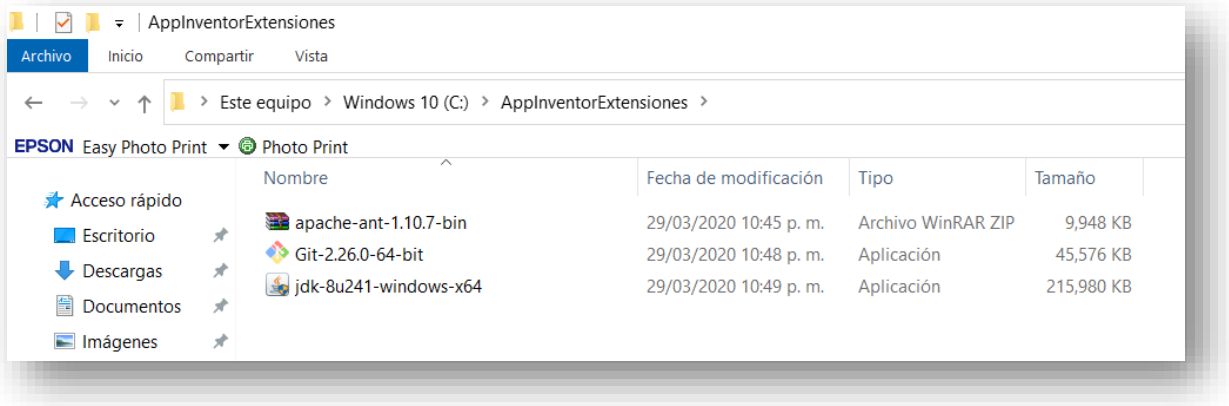
```
            for(int i=1; i <= count ; i=i+2) {

            if (!esPar(count) && i == count) odd = 1;
                  treeLayer.add(applySha256(previousTreeLayer.get(i-item)                    +
previousTreeLayer.get(i-impar)));

            }
            item = 1;
            Odd = 0;
            count = treeLayer.size();
 previousTreeLayer = treeLayer;
        }
 String merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : "";
        return merkleRoot;
    }
 //Define if Merkle Tree is even or odd
 static boolean enPar(int number){
    if (numero%2==0) return true; else return false;
}

public static String applySha256(String input){
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            //Applies sha256 to our input,
            byte[] hash = digest.digest(input.getBytes("UTF-8"));
            StringBuffer hexString = new StringBuffer(); // This will contain hash as hexidecimal
            for (int i = 0; i < hash.length; i++) {
               String hex = Integer.toHexString(0xff & hash[i]);
               if(hex.length() == 1) hexString.append('0');
               hexString.append(hex);
            }
            return hexString.toString();
        }
        catch(Exception e) {
            throw new RuntimeException(e);
 }
   }

    public static void main(String args[]){
    try{
```

```
    Connection con=DriverManager.getConnection("jdbc:sqlite:C://memo/sqlite-tools-win32-x86-3310100/trans.sqlite3");
    //Connecting to Redis server on localhost
    Jedis = new Jedis ("localhost");
    jedis.auth("memo1234");
    Statement stmt=con.createStatement();
    String sql = "SELECT * FROM brewery";
    ResultSet rs=stmt.executeQuery(sql);
    while(rs.next()) {
    transactions.add(rs.getString(2));
jedis.set("LATAM:"+String.valueOf(rs.getInt(1)),"["+"\""+rs.getString(2)+"\""+","+"\""+rs.getString(3)+"+","+"\""+rs.getString(4)+"\""+"));
 }
 jedis.set("LATAM:merkleroot", getMerkleRoot());
    System.out.println("Number of ArrayList elements:: "+treeLayer.size());
    with .close();
    }catch(Exception e){ System.out.println(e);}

    }
 }
```

## 26. Annex "Mini BlocklyChain for developers".

In this annex we will review the configuration, installation and basic use of how to generate external modules based on the Java programming language for Blockly environment and we will be able to create specialized modules for each business case and insert functionalities to the Mini BlocklyChain System or add other functionalities to our mobile application.

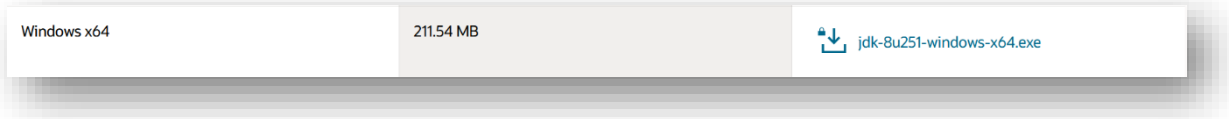
We created a directory in our Windows system called "AppInventorExtensions" and this will download the following public software packages.



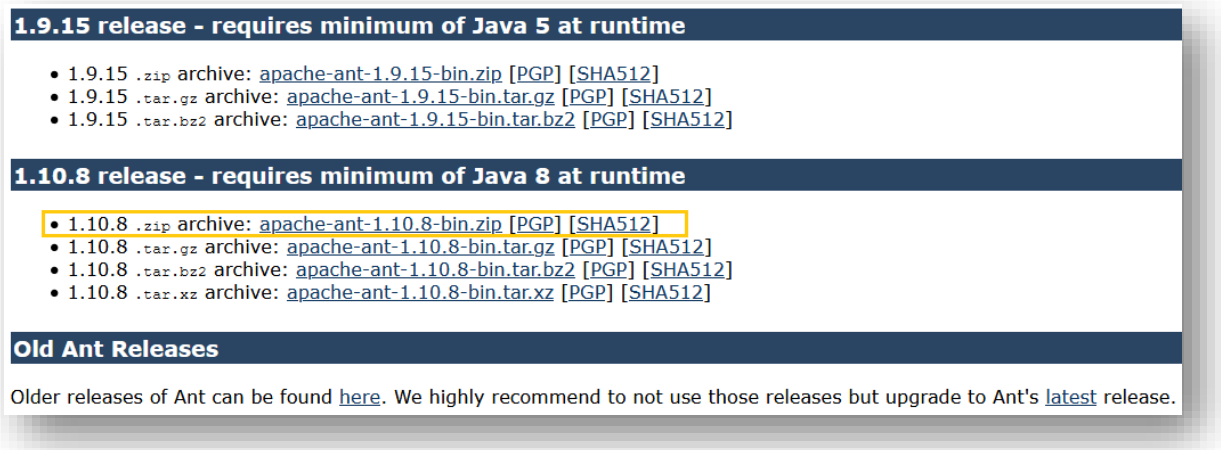

1.- We are going to download the latest version of the **JDK (Java Development Kit)**

example: jdk-8u251-windows-x64.exe (211 MB)

https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html




2.- **Apache Ant** library that uses JAVA to build applications, http://ant.apache.org/bindownload.cgi, in my case I have downloaded Ant 1.10.8 (Binary Distributions) (apache-ant-1.10.8-bin.zip). There may be more advanced versions.



**1.9.15 release - requires minimum of Java 5 at runtime**

- 1.9.15 .zip archive: apache-ant-1.9.15-bin.zip [PGP] [SHA512]
- 1.9.15 .tar.gz archive: apache-ant-1.9.15-bin.tar.gz [PGP] [SHA512]
- 1.9.15 .tar.bz2 archive: apache-ant-1.9.15-bin.tar.bz2 [PGP] [SHA512]

**1.10.8 release - requires minimum of Java 8 at runtime**

- 1.10.8 .zip archive: apache-ant-1.10.8-bin.zip [PGP] [SHA512]
- 1.10.8 .tar.gz archive: apache-ant-1.10.8-bin.tar.gz [PGP] [SHA512]
- 1.10.8 .tar.bz2 archive: apache-ant-1.10.8-bin.tar.bz2 [PGP] [SHA512]
- 1.10.8 .tar.xz archive: apache-ant-1.10.8-bin.tar.xz [PGP] [SHA512]

**Old Ant Releases**

Older releases of Ant can be found here. We highly recommend to not use those releases but upgrade to Ant's latest release.

3.- We installed the Git Bash from your site https://git-scm.com/download/win

# Downloading Git

You are downloading the latest (**2.27.0**) **64-bit** version of **Git for Windows**. This is the most recent maintained build. It was released **9 days ago**, on 2020-06-01.

**Click here to download manually**

## Other Git for Windows downloads

**Git for Windows Setup**
**32-bit Git for Windows Setup.**

**64-bit Git for Windows Setup.**

**Git for Windows Portable ("thumbdrive edition")**
**32-bit Git for Windows Portable.**

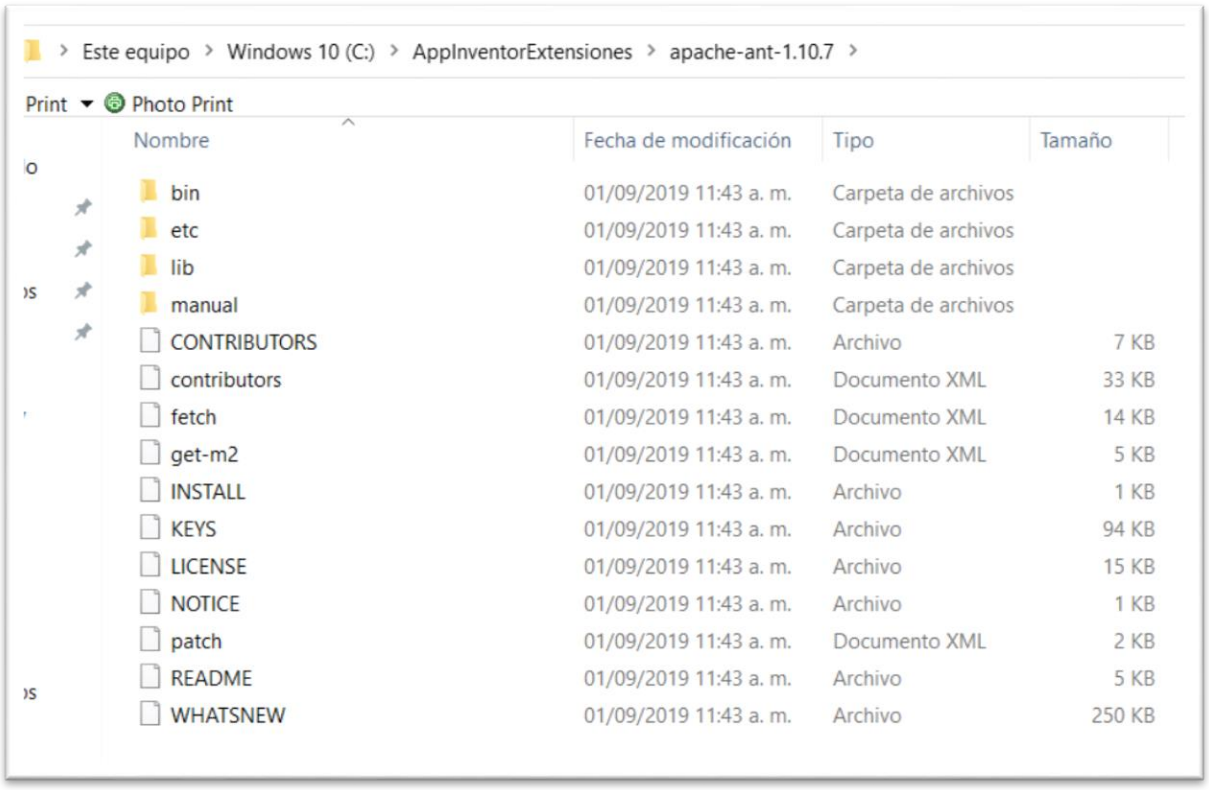
**64-bit Git for Windows Portable.**

The current source code release is version **2.27.0**. If you want the newer version, you can build it from the source code.

4.- Unzip "Apache Ant." When you finish unzipping, you can do it in a double folder, for example:

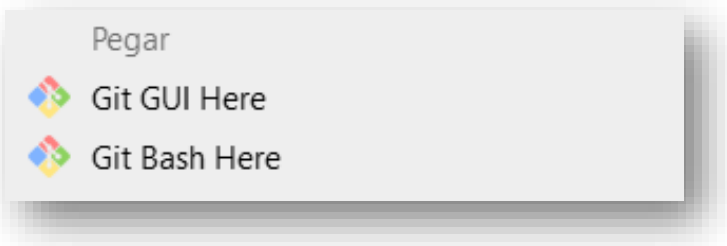
C: AppInventorExtensionsapache-ant-1.10.8-binapache-ant-1.10.8-bin

- Change it to C: Apache-ant-1.10.8-bin




5.- We installed Git Bash. We left everything by default in the installation.

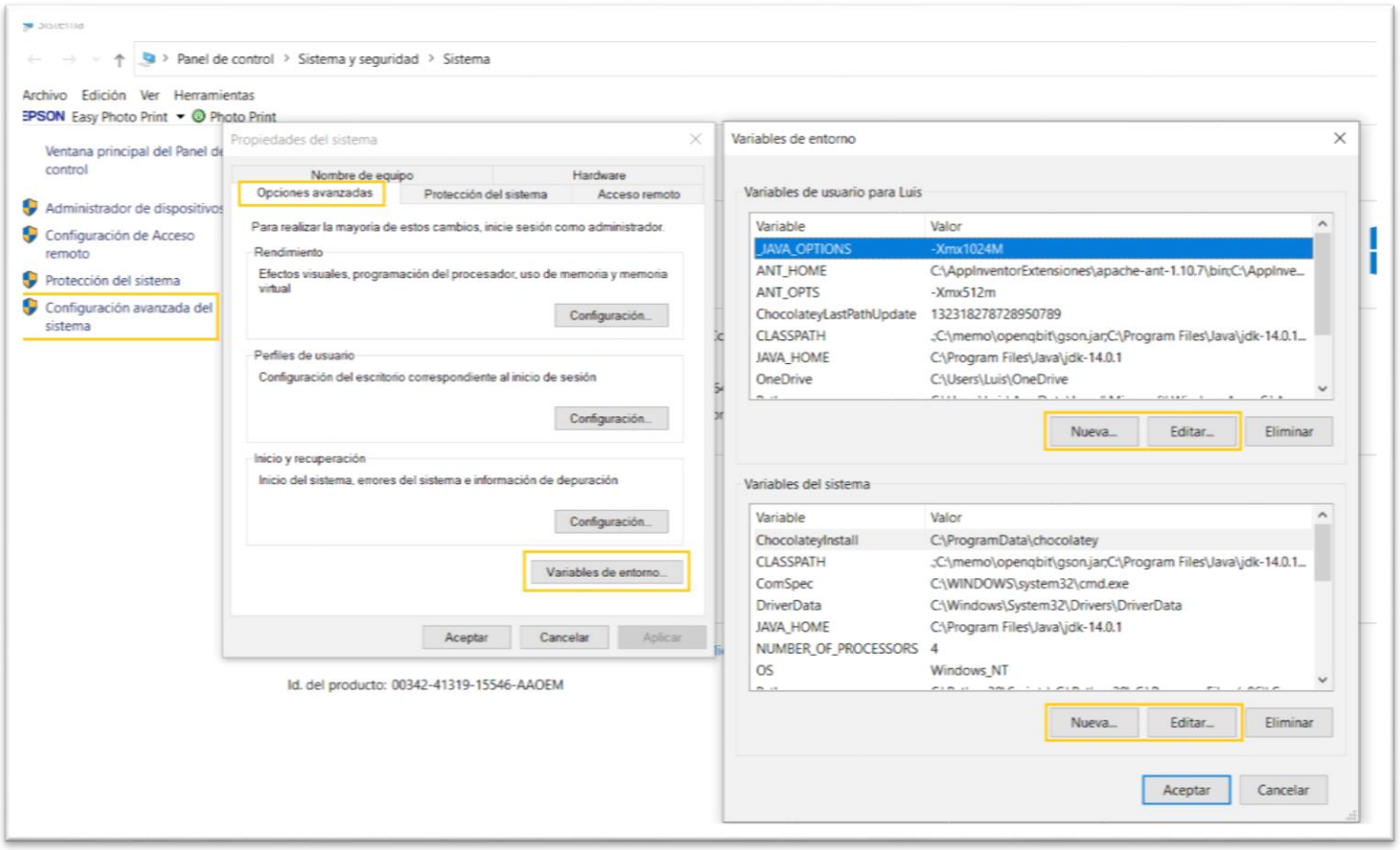
- Later we can see that it has a link in the Windows Start Button, by clicking on the left button in the file browser.



6.- We installed the Java SE Development Kit. Depending on the version of Windows you will probably need to restart your computer.

Windows system environment variables. We will create the environment variables. To do this we will:

Control Panel -> System -> Advanced System Settings -> Advanced Options -> Environment Variables.




Depending on whether we want to put a new variable or edit an existing one, we press the "New..." or "Edit..." button

To add addresses to those already established, we separate them by semicolon;

In the User Variables section for John, we set up these new...

JAVA_OPTIONS we put you of Value -Xmx1024m

ANT_HOME we put of Value C: \AppInventorExtensions\apache-ant-1.10.8-bin [that is to say, the folder where we have decompressed apache-ant]

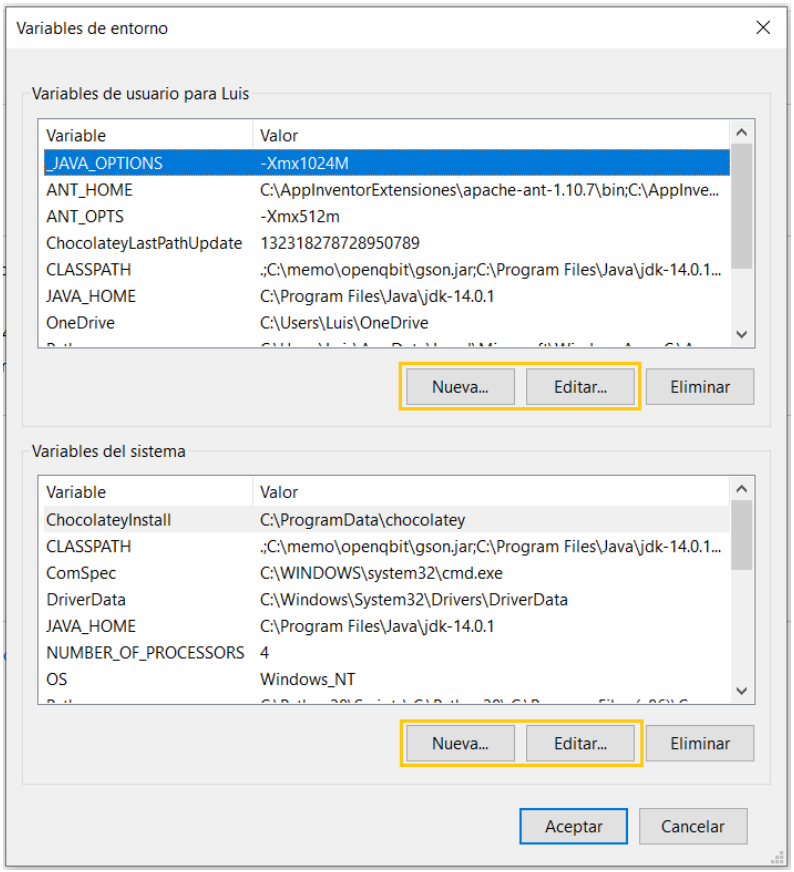
ANT_OPTS we put of Value -Xmx256M

JAVA_HOME is set to C:[333]Program Files[333]jdk1.8.0_131 [If it has another value, change it. Note that it is jdk NOT jdr]

CLASSPATH we put of Value %ANT_HOME%\lib;%JAVA_HOME%\lib

In PATH we added ;%ANT_HOME%\bin;%JAVA_HOME%\bin [Note that ;starts with a semicolon; to add to the ones already there].

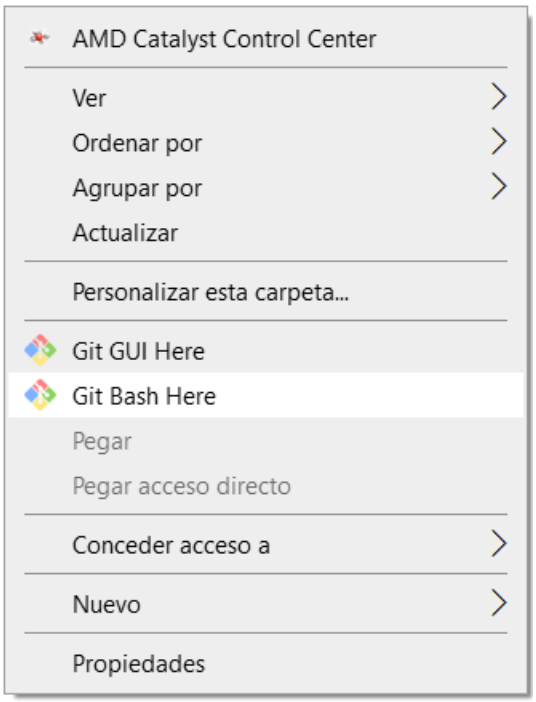
NOTE: Variables are separated from each other by a semicolon: variable-one; variable-n; variable-n+1




Creation of the App Inventor clone in our computer

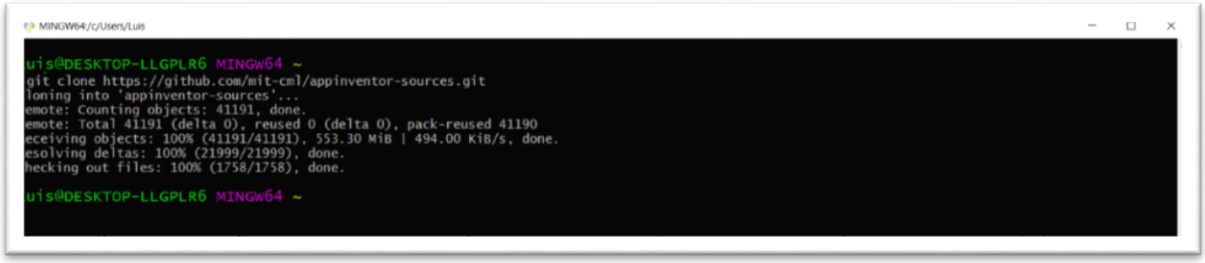
We will create a "clone" copy of App Inventor on our server (PC), download it directly from the Internet and create that copy.

To do this, we'll use the **Git Bash** application and click on it to open a terminal.




We run the command on the Git Bash terminal:

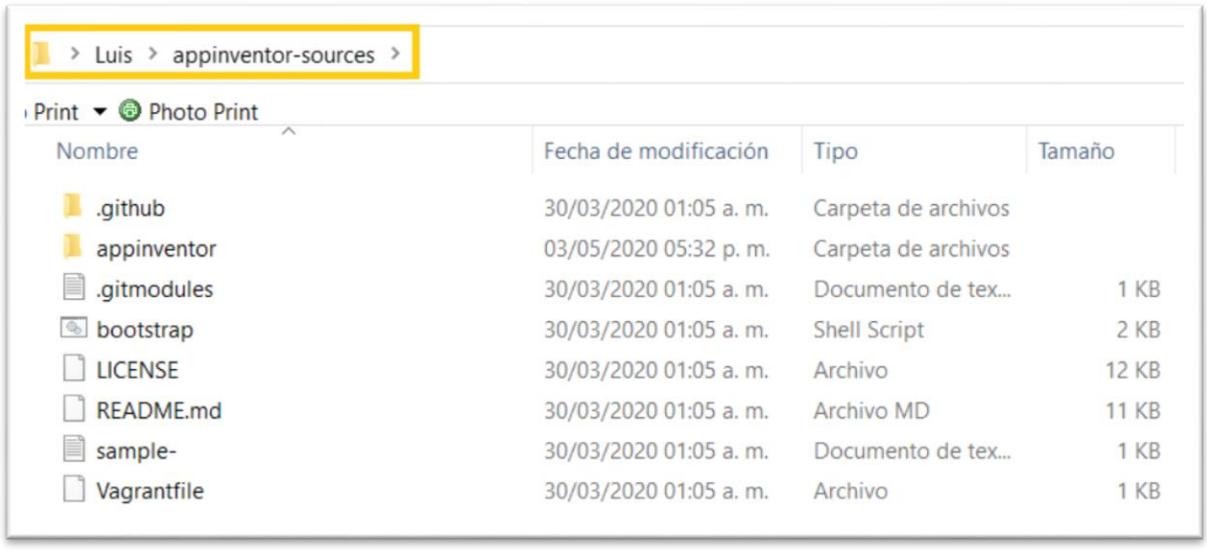
$ git clone https://github.com/mit-cml/appinventor-sources.git




The site where the repository is:

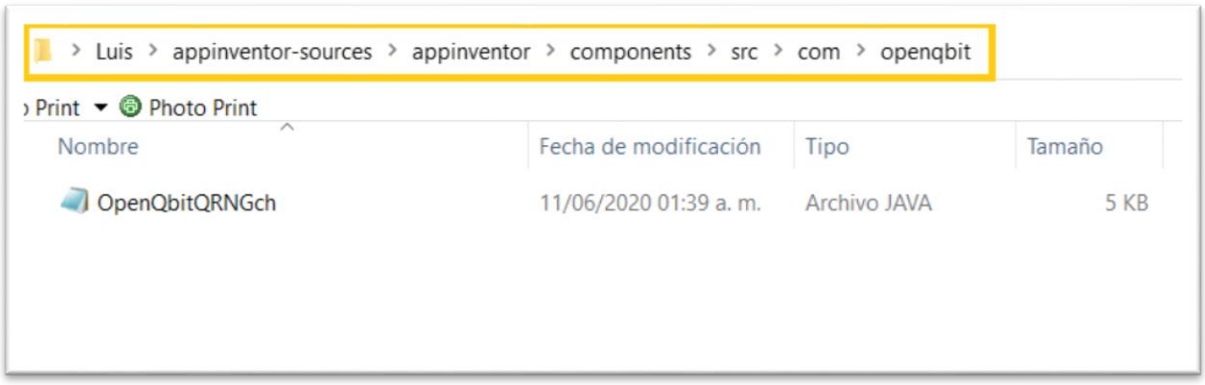
https://github.com/mit-cml/appinventor-sources/

It will create a folder called appinventor-sources with the App Inventor source.




We created the following directory in the path C: Users - Apppinventor-sourcesv-babkup - Apppinventor-components -rc-openqbit

Inside we copied our following test program called OpenQbitQRNGch.java




Creation of the extension.

RESPECT THE CAPS AND MINUSCULES, it's not the same Hello and Hello.

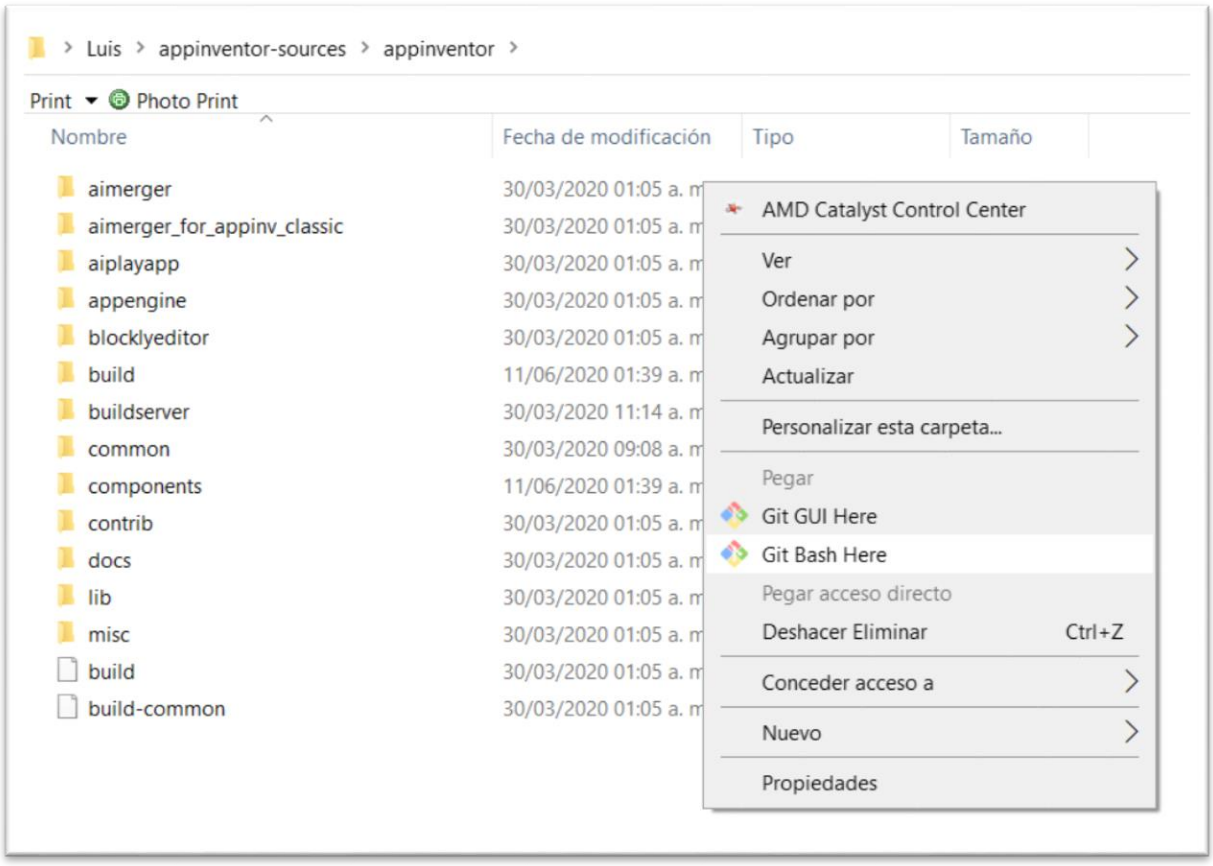
Don't put any accents. We're ready to make our first extension. It will be the quantum random number generator.

We use a Text Editor, Notepad++, we create a file called OpenQbitQRNGch.java that generates random quantum numbers with the following code:

```
// This extension is used to output Quantum Random Number Generator QRNG
Switzerland API.

package com.openqbit. OpenQbitQRNGch;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.runtime.util.MediaUtil;
import com.google.appinventor.components.runtime.*;
import java.io.*;

@DesignerComponent(version = OpenQbitQRNGch.VERSION,
     description = "API Quantum Random Number Generator. Quantum random
numbers as a service, QRNGaaS, web API for the quantum random number
generator of Quantis developed by the Swiss company - " + "Guillermo
Vidal - OpenQbit.com ",
     category = ComponentCategory.EXTENSION
     nonVisible = true,
     iconName = "http://www.pinntar.com/logoQbit.png")
@SimpleObject(external = true)

public class OpenQbitQRNGch extends AndroidNonvisibleComponent implements
Component {

    public static final int VERSION = 1;
    public static final String DEFAULT_TEXT_1 = "";
    private ComponentContainer container;
    private String text_1 = "";

     public OpenQbitQRNGch(ComponentContainer container) {
         super(container.$form());
 this.container = container;
     }

 // Establishment of the Properties.
//@DesignerProperty(editorType =
PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue =
OpenQbitQRNGch.DEFAULT_TEXTO_1 + "")
//@SimpleProperty(description = "API Get Quantum Random Number Generator,
random number between 0 and 1. Enter variable *quantity* of numbers to
generate")

 @SimpleFunction(description = "API Get Quantum Random Number Generator,
random number between 0 and 1. Enter variable integer number *quantity*
of numbers to generate")
 public String APIGetQRNGdecimal(String qty) throws Exception {
     String url = "http://random.openqu.org/api/rand?size=" + qty;
```

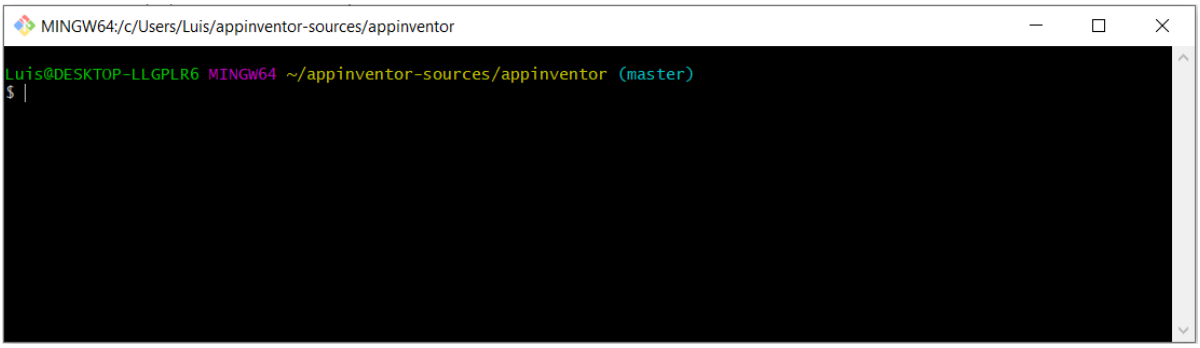
```
     String[] command = { "curl", url };

     ProcessBuilder process = new ProcessBuilder(command);
      Process p;
     p = process.start();
                  BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
                  StringBuilder builder = new StringBuilder();
                  String line = null;
                  while ((line = reader.readLine()) != null) {
                        builder.append(line);

      builder.append(System.getProperty("line.separator"));
                  }
     String result = builder.toString();
      //System.out.print(result);
      return result;

      }
 @SimpleFunction(description = "API Get Quantum Random Number Generator,
random number between a range min to max numbers. Enter variable integer
number *quantity* of numbers to generate a range of minimum *min* number
and maximum *max* number")
 public String APIGetQRNGinteger(String qty, String min, String max)
throws Exception {
     String url = "http://random.openqu.org/api/randint?size=" + qty +
"&min=" + min + "&max=" + max
     String[] command = { "curl", url };

     ProcessBuilder process = new ProcessBuilder(command);
      Process p;
     p = process.start();
                  BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
                  StringBuilder builder = new StringBuilder();
                  String line = null;
                  while ((line = reader.readLine()) != null) {
                        builder.append(line);

      builder.append(System.getProperty("line.separator"));
                  }
     String result = builder.toString();
      //System.out.print(result);
      return result;

      }
}
```

We execute the **Git Bash by** positioning ourselves on the following path: **C: Luis-appinventor-sources-appinventor**. In this directory, we click the left mouse button and select the Git **Bash** terminal.



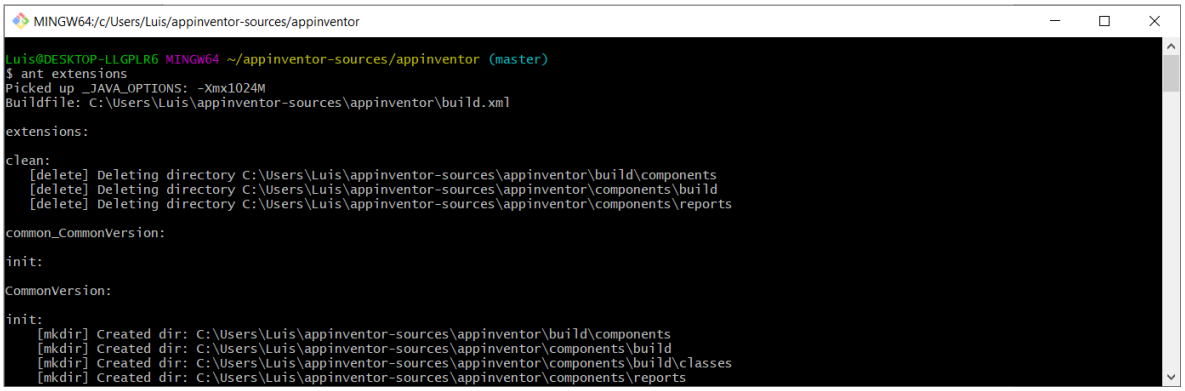

The Git bash terminal will be positioned in the following directory:

**C:\Users\Luis\appinventor-sources\appinventor (master)**



In the Git Bash terminal and execute the following command:

$ ant extensions

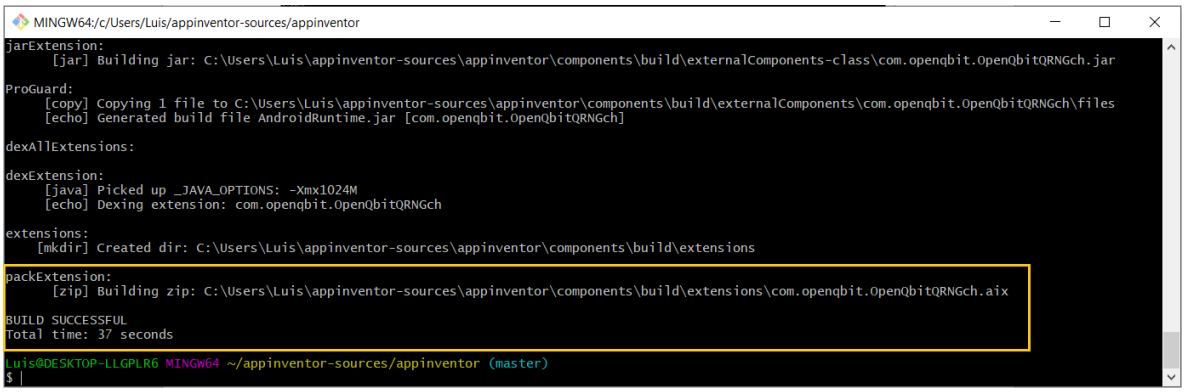



If everything went well, we'll get it: BUILD SUCESSFULLY.

Our extension has been created in...

C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

NOTE: the content of the extensions folder is deleted and recreated every time we make an ant extensions.



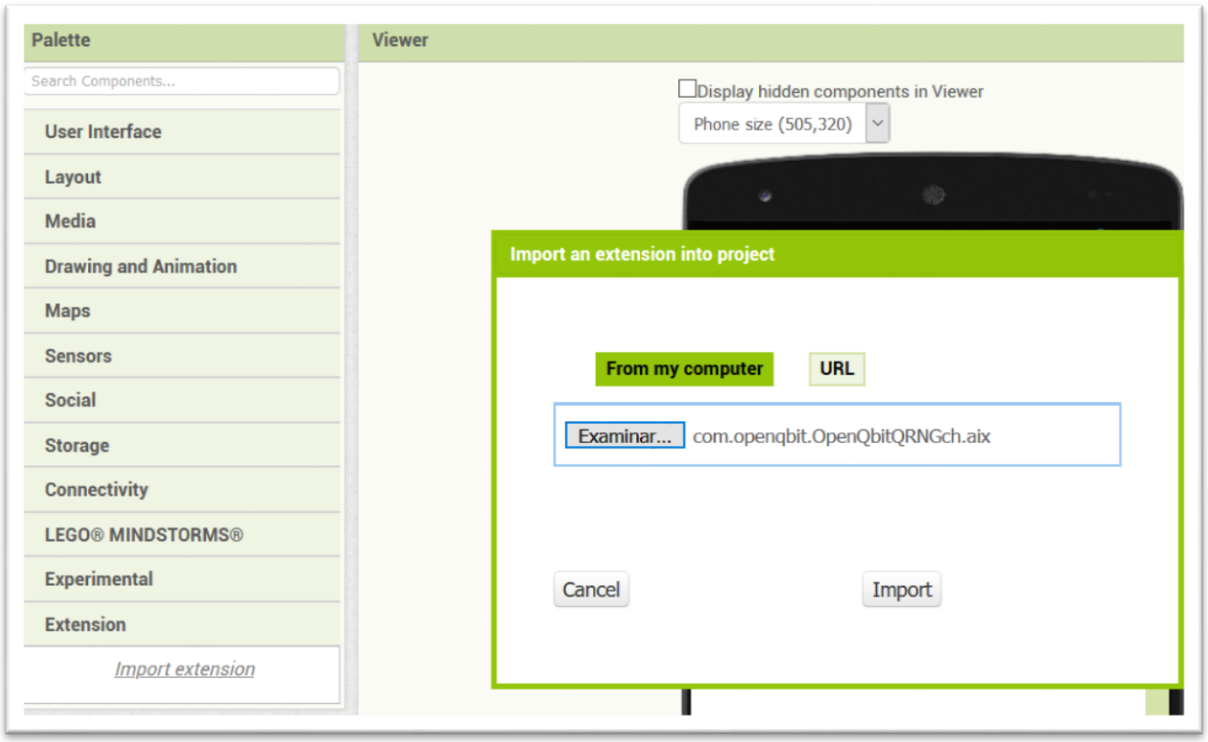

Let's go to that folder:

C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

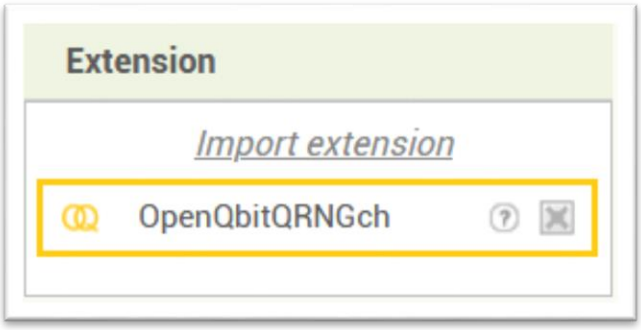
and copy the file com.openqbit.OpenQbitQRNGch.aix, this file is our extension.

Having already an account in App Inventor or another Blockly system we entered to add the new extension and test it.

Go to the bottom where the Extension option is and click on "Import extension":




We hit the "Import" button. Now we have the blocks (**APIGetQRNGdecimal)** and (**APIGetQRNGinteger)** to be used:





We already have our first created and functional extension.

## 27. Annex "BlocklyCode Smart Contracts".

BlocklyCodes are programs made in java language. The extension to create this type of program commonly called "Smart Contracts" is a way to process agreements between users (companies or people).

This block (**BlocklyCode),** is implemented in a program that already has parameters established and that depending on the type of agreements that could be executed by the Mini BlocklyChain system in an automatic way when the premises that govern the "Smart Contract" are fulfilled.

The parameters to consider suggested or input parameters 'inputs' are

Expiration date

Referenced date

Expiration time

Referenced time

Referenced asset

Total fixed assets

Variable assets

Contract members

Confirmed data (String)

Confirmed data (Filename)

Variable event

Fixed event

Validation of document(s)

String Validation

Signature valid

Defined interval (Integer)

Decimal interval

Established minimum

Established maximum

Before starting to use the block (**BlocklyCode**) we must first install the system that will perform the execution of the "Smart Contracts", this is done by installing in the terminal of Termux OpenJDK and OpenJRE.

OpenJDK (Open Java Development Kit). - It is the tool to develop programs in java, it contains the libraries, compiler and the JVM (Java Virtual Machine).

OpenJRE (Open Java Runtime Environment). - This is the tool for running java programs only. It contains the JVM (Java Virtual Machine).

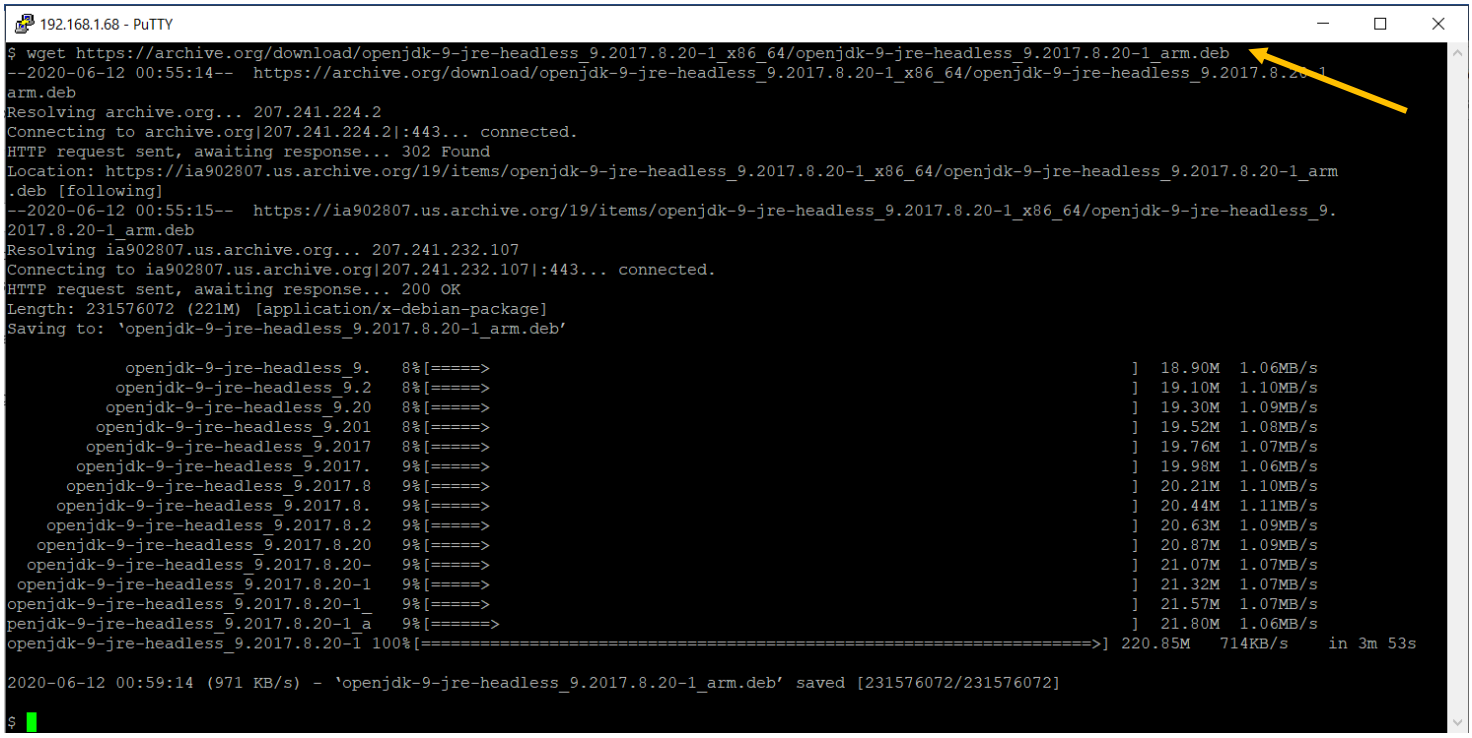
We proceed with the installation of OpenJRE and OpenJDK in the Termux terminal of the nodes that form the Mini BlocklyChain system.

We installed the rooms

**$ apt install - and wget**

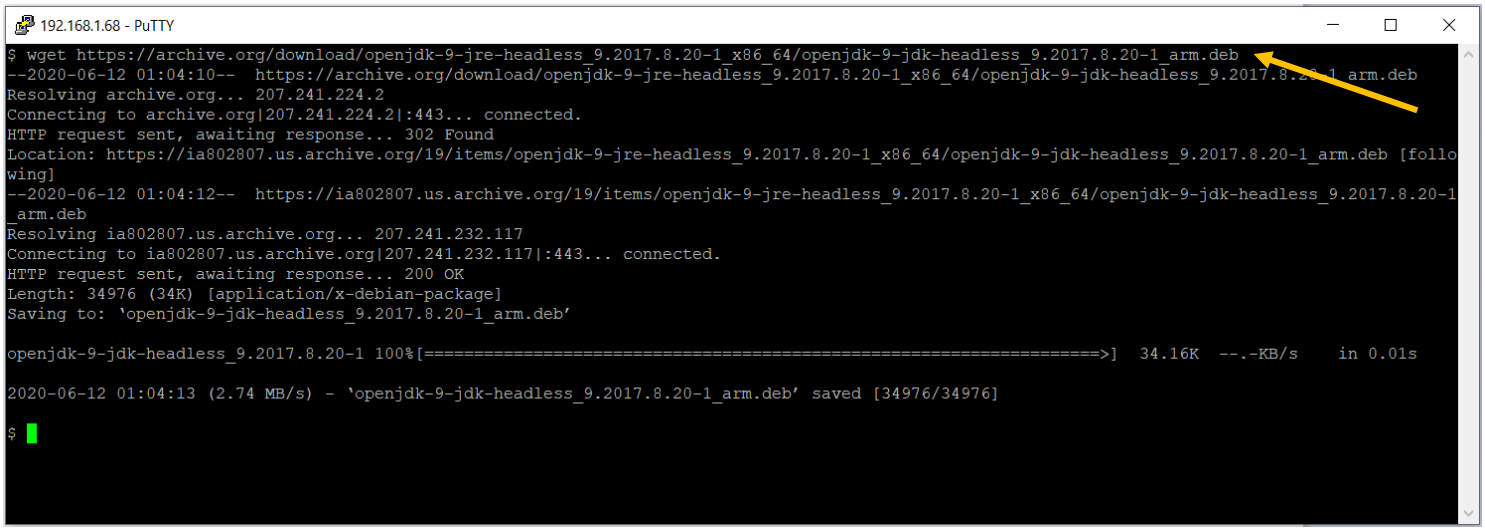
We downloaded the OpenJRE

$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb



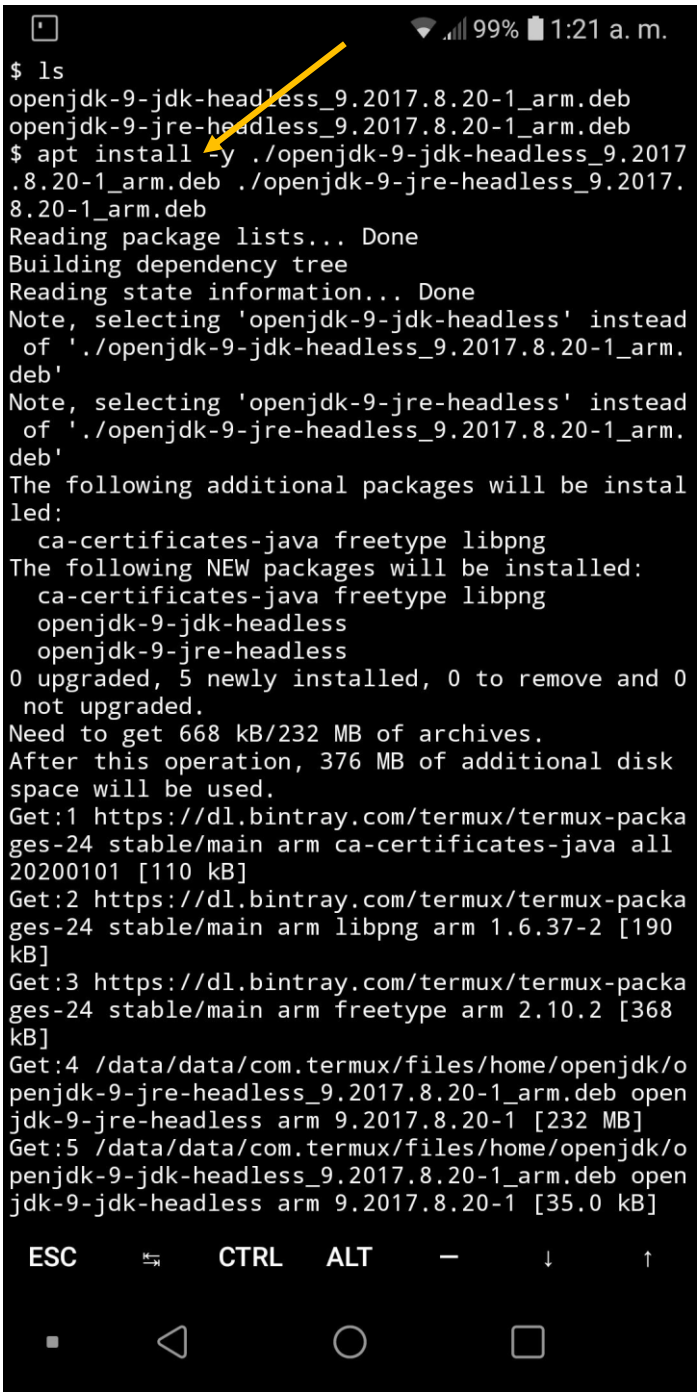

We downloaded the OpenJDK

$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb



We carry out the installation from the Termux terminal of OpenJDK and OpenJRE:

**$ apt install - and** ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb




Since we have finished the installation of the OpenJDK and OpenJRE environment. These installations contain the JVM (Java Virtual Machine) which will be the environment that will run the BlocklyCode.

Now we will install an editor to create file online, we will use the editor called **vi** execute the following command:

**$ apt install vim**

Now let's try compiling a test file and running it. We create in the Termux terminal with the **vi** editor and write the java code of a "Hello World", and save it in the HelloWorld.java file

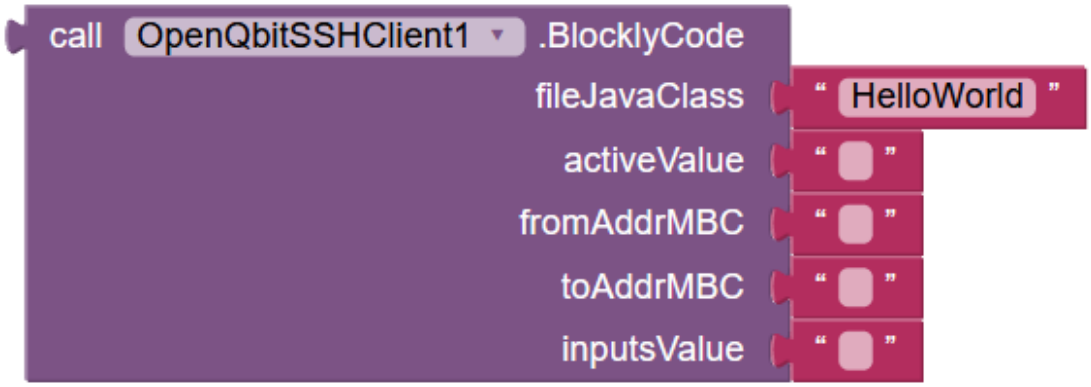
```
public class HelloWorld {

     public static void main(String[] args) {
         // Prints "Hello, World" to the terminal window.
         System.out.println("Hello, World");
 }

}
```

Then we run the following command in the Termux terminal for the compilation and then the execution of the program:

**$ javac HelloWorld.java** (After running, a bytecode HelloWorld.class file is created)

**$ java HelloWorld** (After running and printing the ad, Hello World)

Using the block (**BlocklyCode**) this block is located in the extension (**ConnectorSSHClient**).




In the previous block you can see how the HelloWorld.class file will be executed, which has already been compiled and positioned in the user base directory.

It is a block in which it is possible to execute a program already compiled in java, which is commonly known in the development environment as the bytecode file in its binary form.

This type of file is ready to be executed by the Java Virtual Machine (JVM).

There are two ways to create a bytecode file or with extension .CLASS, the user can create it in his PC by comfort or his like can also be done in the mobile phone, remember that we have already installed the environment to compile JDK (Java Development Kit) before, although it will be less efficient because the limitation of the keyboard will count, also in case of choosing the Android environment will have to take into account that the libraries are limited to OpenJDK that has been installed.

The input parameters are open in <inputsValue> and the other fixed ones are those of activeValue, fromaddrMBC and toAddrMBC.

In the case of execution tests they can be left blank without content and only the bytecode file will be executed without parameters.

The previous block is like the next command line being executed:

$ java HelloWorld

The programs developed for BlocklyCode will be subject to each particular design environment and can be controlled and executed by routine with the "cron" agent and shared with syncthingmanager.

# 28. Annex "OpenQbit Quantum Computing".

## How does quantum computing work? (2)

The digital transformation is bringing about change in the world faster than ever before. Would you believe that the digital era is about to end? Digital literacy has already been identified as an area where open knowledge and accessible opportunities to learn about technology are urgent to address gaps in social and economic development. Learning from the key concepts of the digital age will become even more critical with the imminent arrival of another new technological wave capable of transforming existing models with amazing speed and power: quantum technologies.

In this article, we compare the basic concepts of traditional computing and quantum computing; and we also begin to explore their application in other related areas.

What are quantum technologies?

Throughout history, human beings have developed technology as they have understood how nature works through science. Between 1900 and 1930, the study of some physical phenomena that were not yet well understood gave rise to a new physical theory, Quantum Mechanics. This theory describes and explains the functioning of the microscopic world, the natural habitat of molecules, atoms or electrons. Thanks to this theory, not only has it been possible to explain these phenomena, but it has also been possible to understand that subatomic reality works in a completely counter-intuitive, almost magical way, and that in the microscopic world events take place that do not occur in the macroscopic world.

These quantum properties include quantum superposition, quantum entanglement and quantum teleportation.

- Quantum superposition describes how a particle can be in different states at the same time.
- Quantum entanglement describes how two particles as far apart as desired can be correlated in such a way that, when interacting with one, the other is aware of it.
- Quantum teleportation uses quantum entanglement to send information from one place to another in space without having to travel through it.

Quantum technologies are based on these quantum properties of the subatomic nature.

In this case, today the understanding of the microscopic world through Quantum Mechanics allows us to invent and design technologies capable of improving people's lives. There are many and very different technologies that use quantum phenomena and some of them, such as lasers or magnetic resonance imaging (MRI), have been with us for more than half a century. However, we are currently witnessing a technological revolution in areas such as

quantum computing, quantum information, quantum simulation, quantum optics, quantum metrology, quantum clocks or quantum sensors.

What is quantum computing? First, you have to understand classical computing.

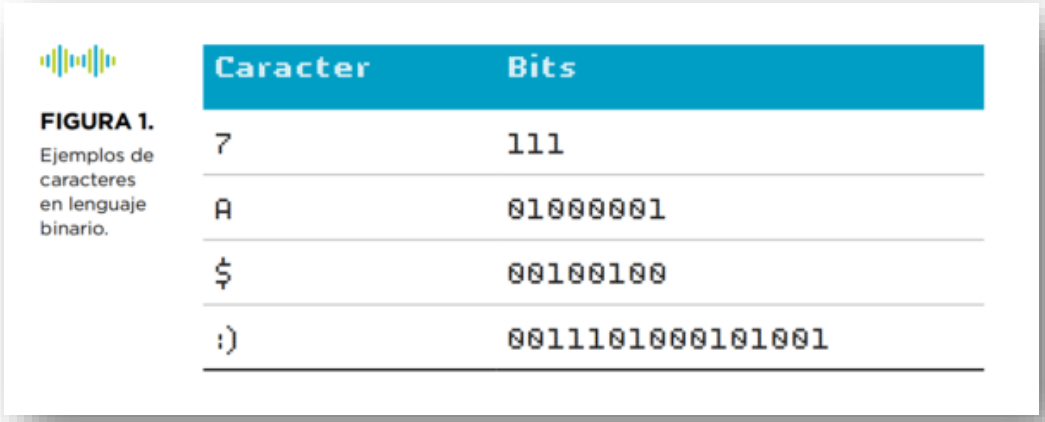


**FIGURA 1.**
Ejemplos de caracteres en lenguaje binario.

| Caracter | Bits |
|---|---|
| 7 | 111 |
| A | 01000001 |
| $ | 00100100 |
| :) | 0011101000101001 |

To understand how quantum computers work, it is convenient to first explain how the computers we use every day, which we will refer to in this document as digital or classic computers, work. These, like the rest of the electronic devices such as tablets or mobile phones, use bits as the fundamental units of memory. This means that programs and applications are encoded in bits, that is, in binary language of zeros and ones. Every time we interact with any of these devices, for example, by pressing a key on the keyboard, strings of zeros and ones are created, destroyed and/or modified inside the computer.

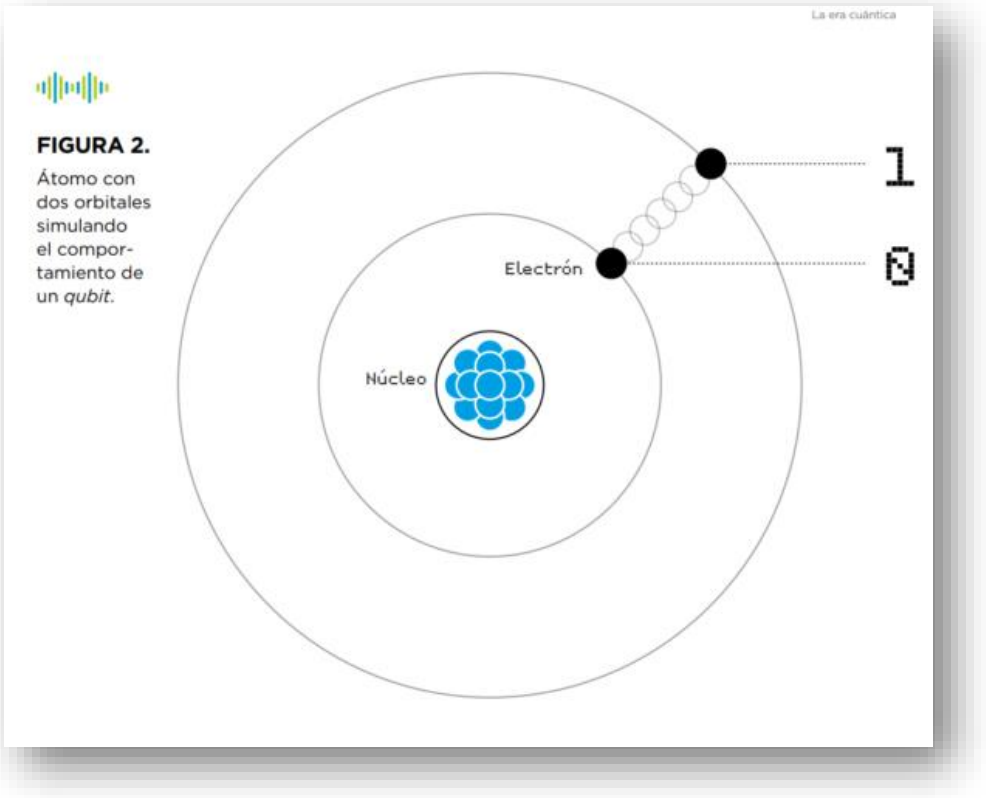
The interesting question is, what are these zeros and ones physically inside the computer? The zero and one states correspond to electrical current that circulates, or not, through microscopic parts called transistors, which act as switches. When no current is flowing, the transistor is "off" and corresponds to bit 0, and when it is flowing it is "on" and corresponds to bit 1.

More simply, it is as if bits 0 and 1 correspond to holes, so that an empty hole is a bit 0 and a hole occupied by an electron is a bit 1. This is why these devices are called electronics. As an example, figure 1 shows the binary writing of some characters. Now that we have an idea of how today's computers work, let's try to understand how quantums work.

### From bits to qubits

The fundamental unit of information in quantum computing is the quantum bit or qubit. Qubits are, by definition, two-level quantum systems -we will see examples here- which, like bits, can be at the low level, which corresponds to a state of low excitation or energy defined as 0, or at the high level, which corresponds to a state of higher excitation or defined as 1.

However, and here lies the fundamental difference with classical computing, qubits can also be in any of the infinite intermediate states between 0 and 1, such as a state that is half 0 and half 1, or three quarters of 0 and a quarter of 1.




Quantum algorithms, exponentially more powerful and efficient computing

The purpose of quantum computers is to take advantage of these quantum properties of *qubits,* as quantum systems that they are, in order to run quantum algorithms that use overlapping and interleaving to provide much greater processing power than the classics. It is important to point out that the real change of paradigm does not consist in doing the same thing that digital or classical computers do -the current ones- but faster, as it can be read in many articles, but that quantum algorithms allow to perform certain operations in a totally different way that in many cases turns out to be more efficient -that is, in much less time or using much less computational resources-.

Let's look at a concrete example of what this involves. Let's imagine we are in Bogotá and we want to know the best route to get to Lima from among a million options to get there (N=1,000,000). In order to use computers to find the optimal route we need to digitize 1,000,000 options, which implies translating them into bit language for the classic computer and into *qubits for* the quantum computer. While a classic computer would need to go one by one analyzing all the paths until finding the desired one, a quantum computer takes advantage of the process known as quantum parallelism that allows it to consider all the

paths at once. This implies that, while the classical computer needs the order of N/2 steps or iterations, that is, 500,000 attempts, the quantum computer will find the optimal path after only √N operations on the registry, that is, 1,000 attempts.

In the previous case the advantage is quadratic, but in other cases it is even exponential, which means that with n *qubits* we can obtain a computational capacity equivalent to 2n bits. To exemplify this, it is common to count that with about 270 qubits we could have more base states in a quantum computer - more different and simultaneous character strings - than the number of atoms in the universe, which is estimated to be around 280. Another example is that it is estimated that with a quantum computer of between 2000 and 2500 *qubits we* could break practically all the cryptography used today (the so-called public key cryptography).

Why is it important to know about quantum technology?

We are in a moment of digital transformation in which different emerging technologies such as blockchain, artificial intelligence, drones, Internet of things, virtual reality, 5G, 3D printers, robots or autonomous vehicles have more and more presence in multiple fields and sectors. These technologies, called to improve the quality of life of the human being accelerating the development and generating social impact, advance nowadays in a parallel way. Only rarely do we see companies developing products that exploit combinations of two or more of these technologies, such as blockchain and IoT or drones and artificial intelligence. Although they are destined to converge, thus generating an exponentially greater impact, the initial stage of development in which they find themselves and the scarcity of developers and people with technical profiles mean that convergence is still a pending task.

Because of their disruptive potential, quantum technologies are expected not only to converge with all these new technologies, but to have a cross-cutting influence on virtually all of them. Quantum computing will threaten authentication, exchange and secure storage of data, having a major impact on those technologies where cryptography has a more relevant role, such as cyber security or blockchain, and a minor negative impact but also to be considered in technologies such as 5G, IoT or drones.

Do you want to practice quantum computing?

Dozens of quantum computer simulators are already available on the net with different programming languages already in use such as C, C++, Java, Matlab, Maxima, Python or Octave. Also, new languages like Q#, launched by Microsoft. You can explore and play with a virtual quantum machine through platforms such as IBM and Rigetti.

Mini BlocklyChain is created by OpenQbit.com company that focuses on developing quantum computing technology for different types of private and public sectors.

***Why Mini BlocklyChain is different from other blockchains, simply because the system was created to be modular and include quantum computing.***

(2) https://blogs.iadb.org/conocimiento-abierto/es/como-funciona-la-computacion-cuantica/

## 29. Annex "Extended blocks for SQLite database".

To see more detail of the proper and timely use of each block that make up the extension OpenQbitSQLite check the original author of the extension in the following link.

OpenQbitSQLite is a clone of the original design with small modifications to be used with an AES security level.

https://github.com/frdfsnlght/aix-SQLite

## 30. Annex "Example of Mini BlocklyChain system creation".

We will make a test system where we can verify the functionalities, advantages and ease of creating a Mini BlocklyChain system.

We will start with the design and development of the storage where the information will reside what we have already defined as a chain of blocks. The design will be based on the SQLite database and depending on each organization or design this can be easily changed by another database such as Microsoft SQL Server, MySQL, Maria DB, Oracle, DB2, PostgreSQL among others. Although again due to the process of transactions and concurrence to the SQLite database can be used at any level and for business or personal application.

Database creation called **minibc** in this will have a table that will be storing the block string. This database will be embedded in the final program code that will be installed in the nodes, this storage place is called "**assets packaged" is an internal species** in Blockly environments such as App Inventor.

$ sqlite3 minibc.db

SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .quit

Designing fields of the **nblock** table.

```
CREATE TABLE nblock (
 id integer primary key AUTOINCREMENT
 , prevhashVARCHAR    NOT NULL
 , newhashVARCHAR     NOT NULL
 ntransINTEGER        NOT NULL
 nonceINTEGER         NOT NULL
 ,qrng            INTEGER NOT NULL
);
```

We created the **nblock** table.

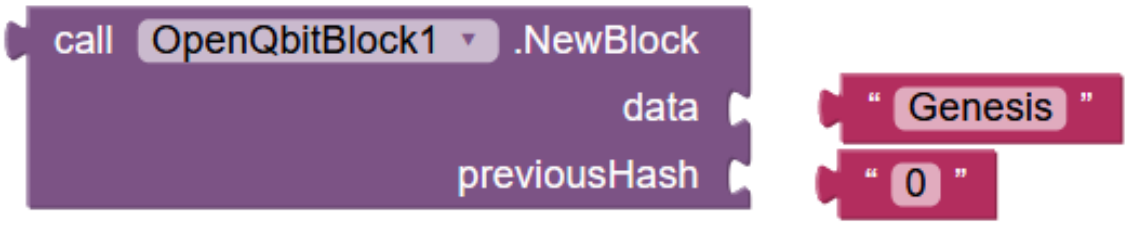
```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .open minibc.db
sqlite> CREATE TABLE nblock ( id integer primary key AUTOINCREMENT , prevhash VARCHAR NOT NULL ,    newhash VARCHAR NOT NULL, ntrans INTEGER NOT NULL ,nonce INTEGER NOT NULL ,qrng INTEGER NOT NULL );
```

We'll see something similar to:


SQL statement for the creation of nblock table.

Next we'll create the initial hash of the system called "**Genesis**", which is based on creating a new hash of the first block of the block chain we'll use the block (**NewBlock**). Then we'll create a second hash that we'll call "one" based on the "Genesis" hash because it must be consistent with the first hash because the hash validation test on the initial block chain must always comply with: prevhash = newhash.

**NewBlock**. To create the "**Genesis**" hash we will use the input parameters:




Input parameter: **data** <String>, **previousHash <String>**

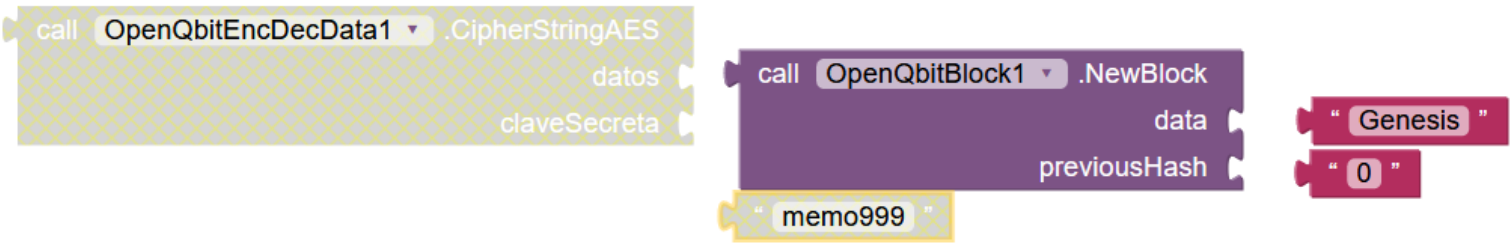
Output parameter: One SHA256 Hash.

In this case we will use as "previousHash" initial equal to "0" and in the case of input data "data" will be the word "Genensis".

This will give us a calculated hash of the combination of both data:

**SHA256 (Genesis0) = eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642**

The above string will be inserted into the nblock table, this string must be encrypted for this we use the extension (**OpenQbitEncDecData**).




We will use the OpenQbitSSHClient extension to insert the data in the minibc.db database, it would be the INSERT statement to the minibc.db database of the nblock table.

sqlite3 keystore.db "insert into nblock (prevhash, newhash, ntrans, nonce, qrng) values ('0', ' eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642', '1', '0', '0');"

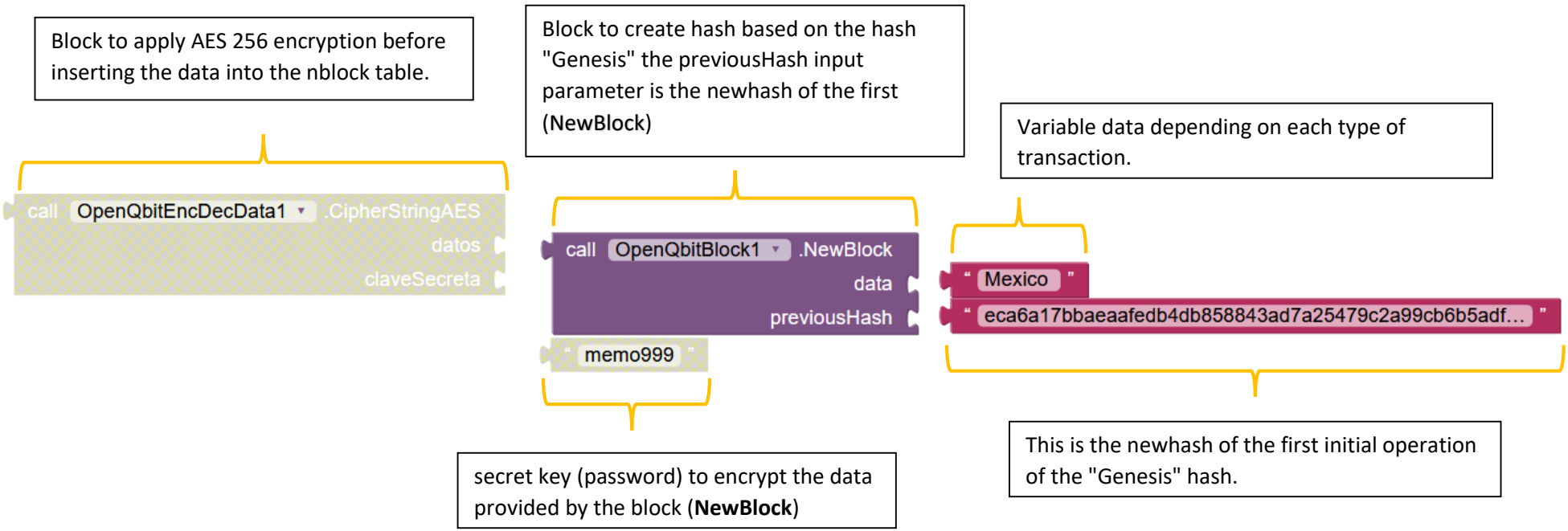
We created the "one" hash with the SQL statement based on the "Genensis" hash:

sqlite3 keystore.db "insert into nblock (prevhash, newhash, ntrans, nonce, qrng) values ('eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642','f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68','1', '0', '0');"

The hash "one" newhash is calculated as:

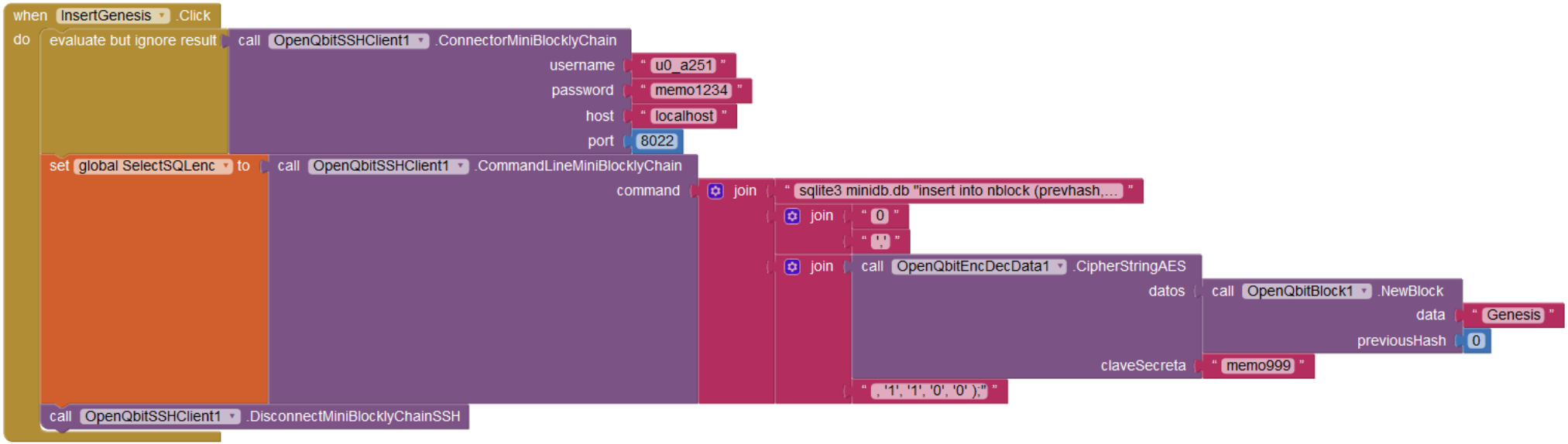
SHA256 (eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642Mexico) =
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68

Second hash based on the first initialization hash "Genesis", we must do two INSERTS at the beginning because any initial validation of the block chain must comply with the equality of fields: prevhash (penultimate data) = newhash (last data).



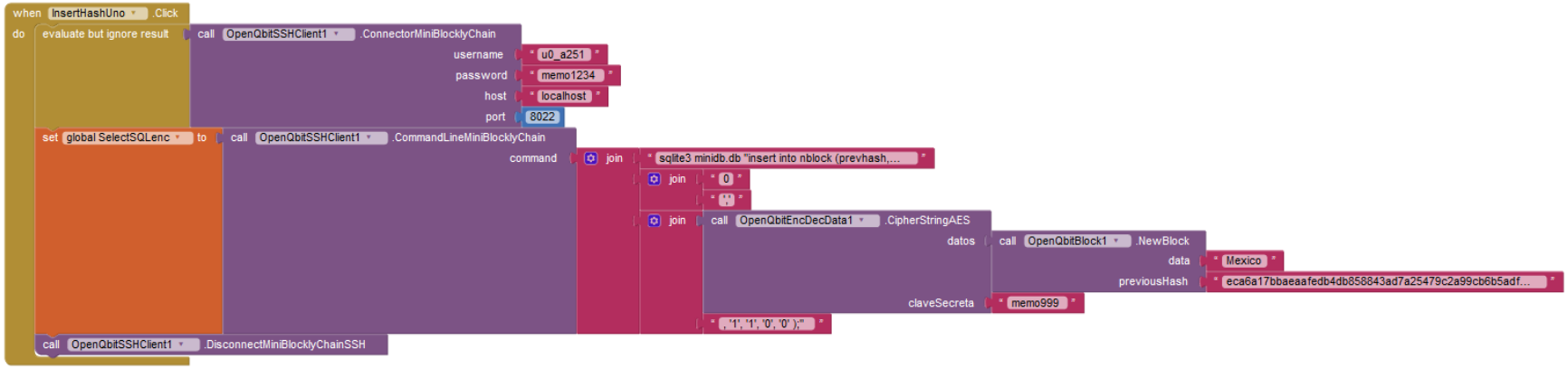

We continue with the creation of the execution within App Inventor of the hashes above; "Genesis" hash and "start" hash.

Now we show how the INSERT of the "**Genesis**" hash would be integrated into the initial structure of the block chain within the nblock table in the App Inventor system:




Since we have inserted the first data in the nblock table based on the "Genesis" hash, we proceed to make the second INSERT referenced to the "one" hash.

It would integrate the INSERT of the "**One**" hash into the initial structure of the block chain within the nblock table in the App Inventor system:




The two previous programming structures (hash "Genesis" and hash "one") should be integrated by the initial node of the Mini BlocklyChain system, an important point is that the minibc.db database with all its internal structure (tables) will be replicated through the Mini BlocklyChain network based on a "Peer to Peer" architecture.

So far we have completed the basic and fundamental structure of the block chain storage and it is ready to receive new blocks from future transactions originating or requested in the system.

We have already inserted the first hash data "Genesis" and the hash "one" to our **minibc.db** database, now we will use the following SQL statements to query the **prevhash** and **newhash** fields in the **nblock** table.

This point will be fundamental since based on the comparison of these two fields we will know if the block chain is being consistent and if it maintains the integrity and security of the transactions. This is just a starting mechanism to review the block chain as in the future we will review the use of the block to calculate the merkle tree which will be another essential tool to ensure the integrity and security of the block chain in our system as well.

For now we will only review the structure or SQL sentences that we must use as we have previously with the extension (**OpenQbitSSHClient),** with these we will be able to consult the prevhash and newhash fields. Later we can do the simple comparison of equal data for that check every time we are going to add a new block.

**For prevhash:**

```
sqlite3 minibc.db "SELECT prevhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

**For newhash:**

```
sqlite3 minibc.db "SELECT newhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

We will now focus on how we will make a request to submit a new trade and/or transaction to be inserted in the transaction queue.

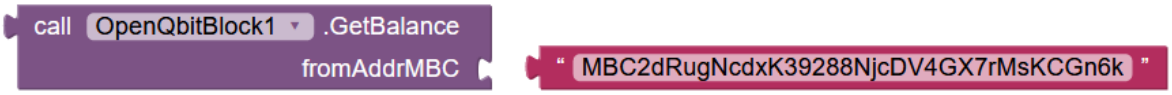
There are two steps as requirements to send a transaction to the transaction queue.

The first requirement is that our account balance has sufficient "assets" available to cover the amount to be sent. Remember that the "assets" can not only be a number or amount, but we can create "assets" of any kind depending on each system to be created.

Examples of assets:

- ✓ Intrinsic amount of a "virtual or crypto-currency" amount of new origin.
- ✓ Data referenced to digital data (all types of documents)
- ✓ Hash authentication signatures for strings or all types of files.
- ✓ Any transactions or transfer of digital information or representation thereof.

The balance of "assets" of each node is obtained using the block (**GetBalance**).




The second requirement is to provide the minimum parameters when sending a transaction, which are

- ✓ Source address. - is the address from which the assets will be shipped.
- ✓ Destination address. - is the address of the user who will receive the sent assets.
- ✓ Active. - It is the data with intrinsic value given in each system to be sent in each transaction.

The above addresses (origin-destination) correspond to the public keys of each user when the accounts of each user were created. Remember that when creating a user account, two types of public and private keys are created.

The public key is the address that will be shared throughout the system and that any user of the system can see and use to send or receive transactions.

The private key is the address that is used locally by each node and as its name indicates it is for private use that helps to create digital signatures to give security to the sent transactions, this key should never be shared and it is for local use and unique to the source node.

In order to use the previous parameters we will rely on two repositories where the "private keys" addresses that are in the keystore.db database are stored and will be of local use of each node without sharing in the system and the other repository where all the "public keys" addresses that are in the publickeys.db database are stored will be shared through the "Peer to Peer" protocol in all the nodes of the system.

The databases "keystore.db" and "publickeys.db" were already created in the Annex "KeyStore & PublicKeys database creation".

The database and system for the transaction queue has already been created in the section "Installation and configuration of the RESTful Mini Sentinel network". Where we already created a database for the transactions called op.sqlite3 in this we have two tables one is "trans" that takes care of the transactions **(transaction queue)** and other called "sign" where the digital signatures of each operation are stored.

The SQLite RESTful system is the backup network in this occasion we will use it for ease and to test the backup system, however, to change and use the same database op.sqlite3 in a "Peer to Peer" model we will only have to use the database in a shared model again in the syncthing system, this we will see later.

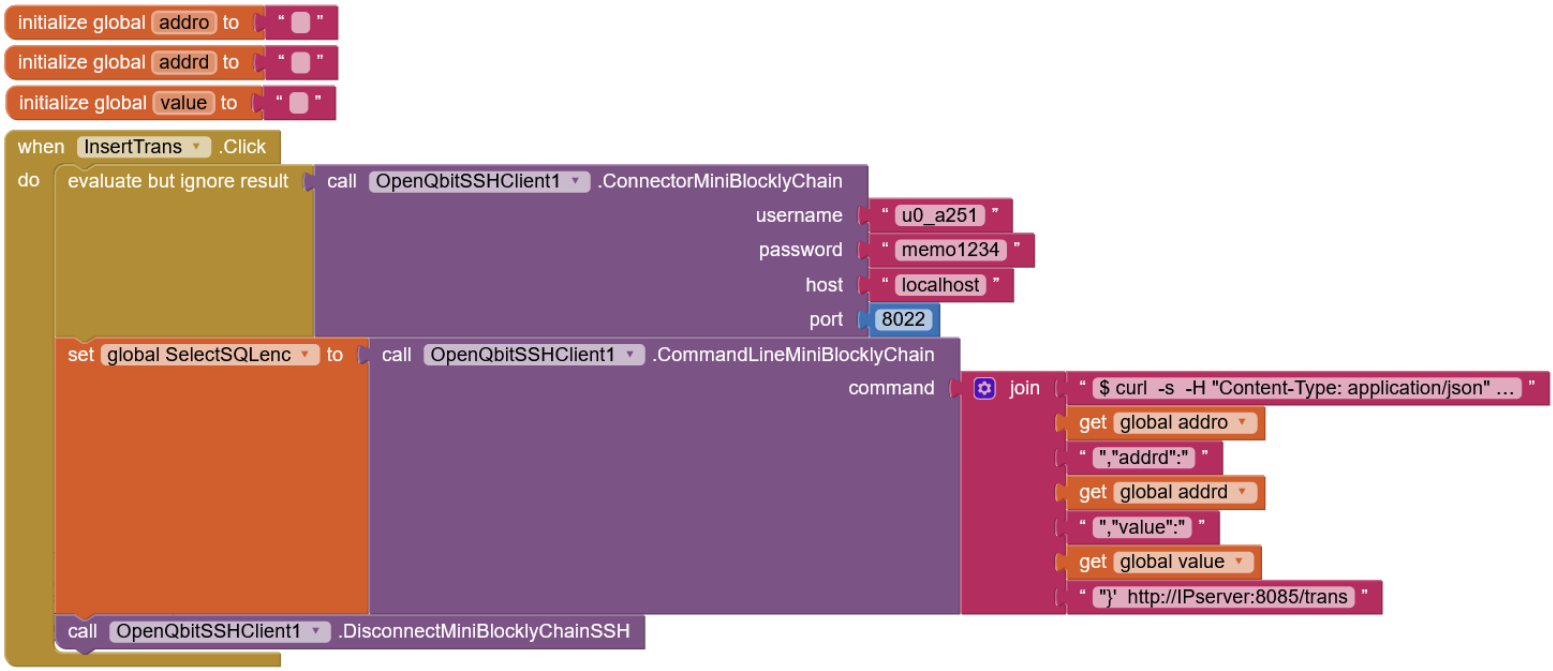
We will start sending operations to the transaction queue using RESTful applied to the op.sqlite database.

We created a new transaction in the "**trans**" table

$ curl -s -H "Content-Type: application/json" –d '{"addro":"MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9", "addrd":"MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh", "value": "999"}' http://IPserver:8085/trans

```
# outputs
{
 "id": 1,
 "addro": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
 "addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh",
 "value": "999"
}
```

Implementation in App Inventor:



The input parameters: addro, addrd, value are variables that can be entered into the algorithm and are extracted from the keystore.db database

See Appendix "KeyStore database creation".

We now insert the digital signatures from the previous transaction. In the table "**sign**"

$ curl -s -H "Content-Type: application/json" -d '{
"trans_id":1
"sign": "59ahGVn8pZ4oduxtTWCcvRiqjWEEy1c62",
"hash": " f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68"}'
http://IPserver:8085/sign

```
# outputs
{
"id": 1,
"trans_id": 1,
"sign": "59ahGVn8pZ4oduxtTWCcvRiqjWEEy1c62",
"hash": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78ab59a68"
}
```

NOTE: The digital signature (sign) must be obtained before sending the curl command statement, it is generated with the block (GenerateSignature).

The input parameters: Trans_id, sign, hash are variables that can be entered in the algorithm and the digital signature of the transaction will be generated with the block (GenerateSignature).

We will now see the procedure for generating a digital signature to be applied to each transaction that is made.

Generation of digital signature for transaction. When we use the block (GenerateSignature) we'll have as a result a file type .sig this file we'll use it to be saved in the sign field in the table "sign, this signature will be used when the transaction queue is processed and it's another parameter to give security between the origin and destination, as well as the amount or type of transaction between them.

To handle binary data like the file.sig we will have to convert it to base64 and then store it in the sign variable of the "sign" table. To do this we will use the block (EncoderFileBase64).

How the hash field value is calculated for the "sign" table of each transaction:

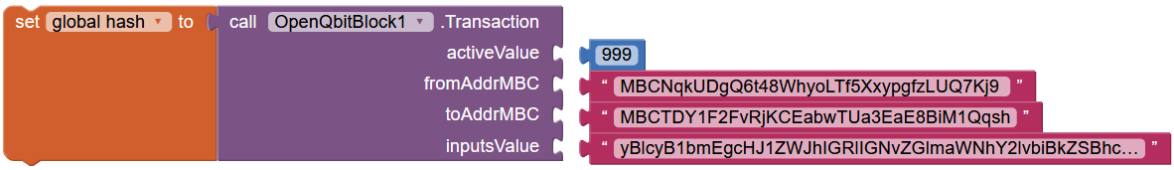
Transaction Hash = SHA256(Source Address + Destination Address + Asset + fileBase64.sig)

Example of hash type SHA256:

SHA256(MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9 +
MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh + 999 + ).

**Output**: 9d45198faaef624f2e7d1897dd9b3cede6ecca7fbac516ed1756b350fe1d56b4

For the calculation of the hash we will have to lean on the Block(**Transaction)**.




The output parameter is the hash that will be stored for each transaction that is sent from any node in the system. This is stored in the sign field of the "sign" table in the base op.sqlite

With the previous operation we made sure that only a unique and unrepeatable hash will represent each transaction sent to the queue and this representative hash is the totality of the information transmitted.

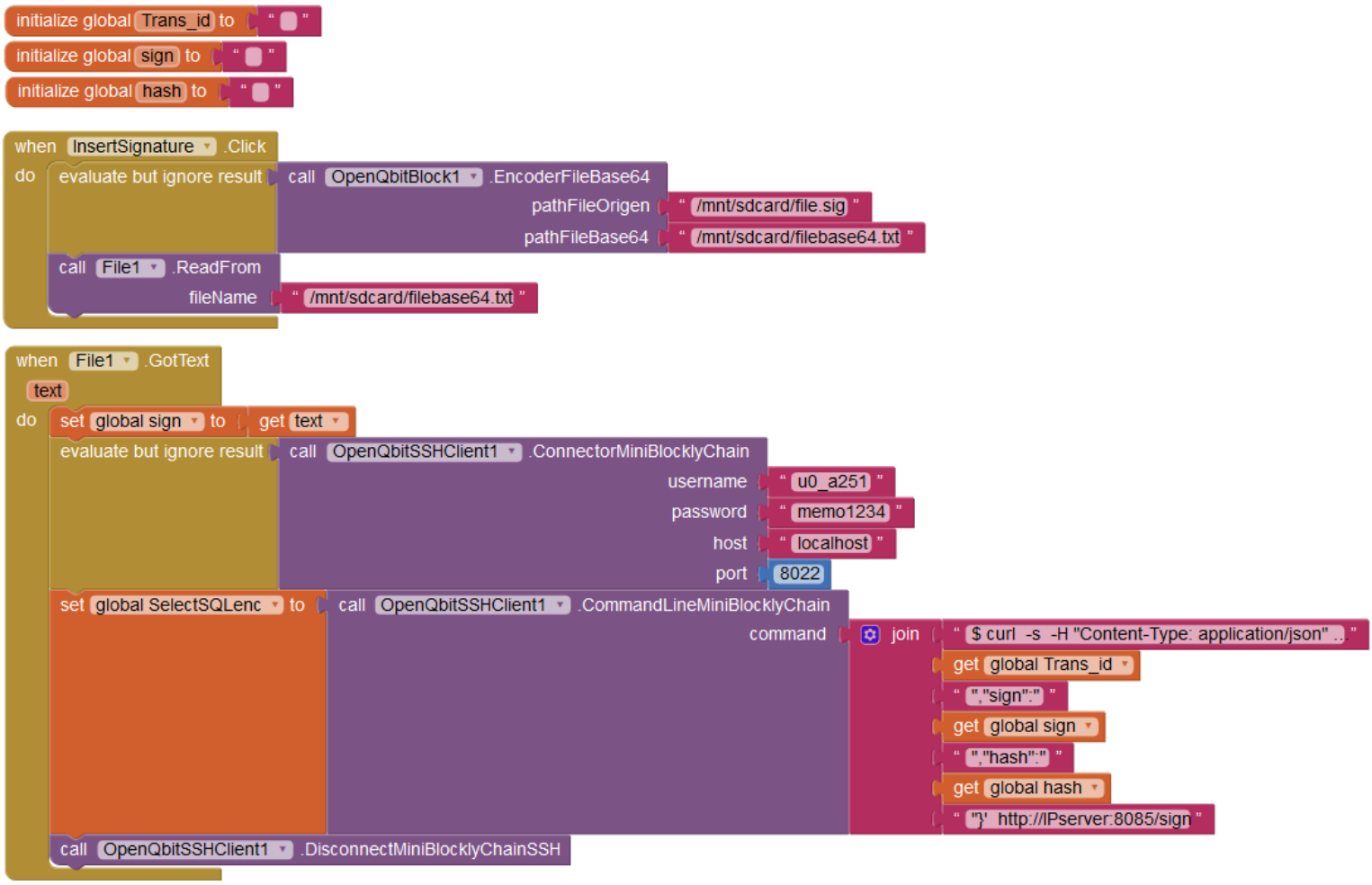
The only thing missing is to include the Trans_id variable, which can be an identifier to differentiate which node is sent for each transaction. In our example we will put the Trans_id variable with a fixed value of "1". Because it is the first node that is being configured in our network. This value can vary the syntax according to each particular design, so for simplicity we have chosen a simple identifier.

Since we have all the variables defined we will create the structure and sequence of block execution within App Inventor, to perform the INSERT in the table "sign".

NOTE: It is important to take into account that each sending of a transaction is composed of two INSERTS in the op.sqlite3 database, one must be done in the "trans" table and their respective values in the "sign" table.

In the design of the op.sqlite3 database, two separate tables were created due to the fact that in the future it is possible to encrypt the "sign" table only since it contains information that should not be sent in the network in a flat way, this option is left to the consideration of each future design.

We will create the execution structure of blocks and methods inside the App Inventor of the INSERT in the table "sign".




Up to this moment we have already finished the INSERT to the "**sign" table** and to the "**trans**" table. These are inside the **op.sqlite3** database and the data inserted in these two tables will be used to create the transaction queue that will be sent to all the nodes for its processing.

At this time we will use the Java SQLite-Redis Connector. This connector will perform the conversion of the data from the op.sqlite3 database to the Redis database. See Appendix "Java SQLite-Redis Code Connector".

After having implemented the SQLite-Redis Connector the transaction queue will be delivered to all the nodes of the system through the Redis system.

We will start the process of how we can receive the transaction queue in a proper way to be able to process it.

The transaction queue will be delivered through the Redis service. This is a real-time database that has its respective control blocks in the App Inventor environment.

Configuration of the Redis environment within the Blockly system of App Inventor.

An important point to note is that the blocks for controlling a Redis server up to the time of writing this manual are only available in the App Inventor system. In case of using a Blockly system different from the App Inventor you will have to use the Redis CLI (Command-Line) execution this through the sending of commands to the Termux terminal via the extension (OpenQbitSSHClient).

Example of uses in Redis CLI (Command-Line):

To obtain all the labels or keys at Redis.

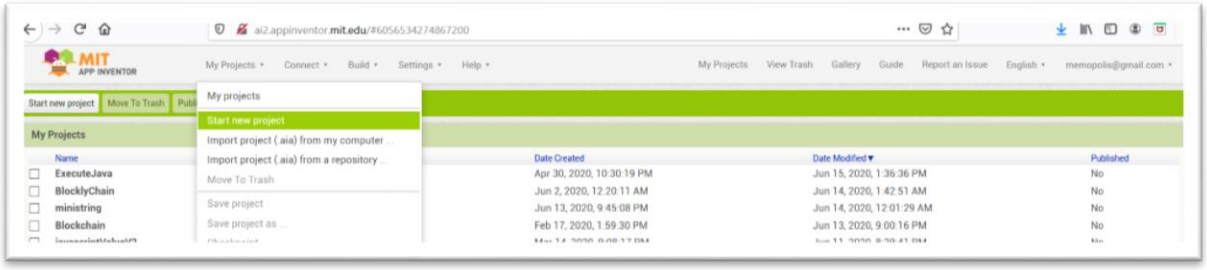
$ redis-cli KEYS '*'

To get value from a key.

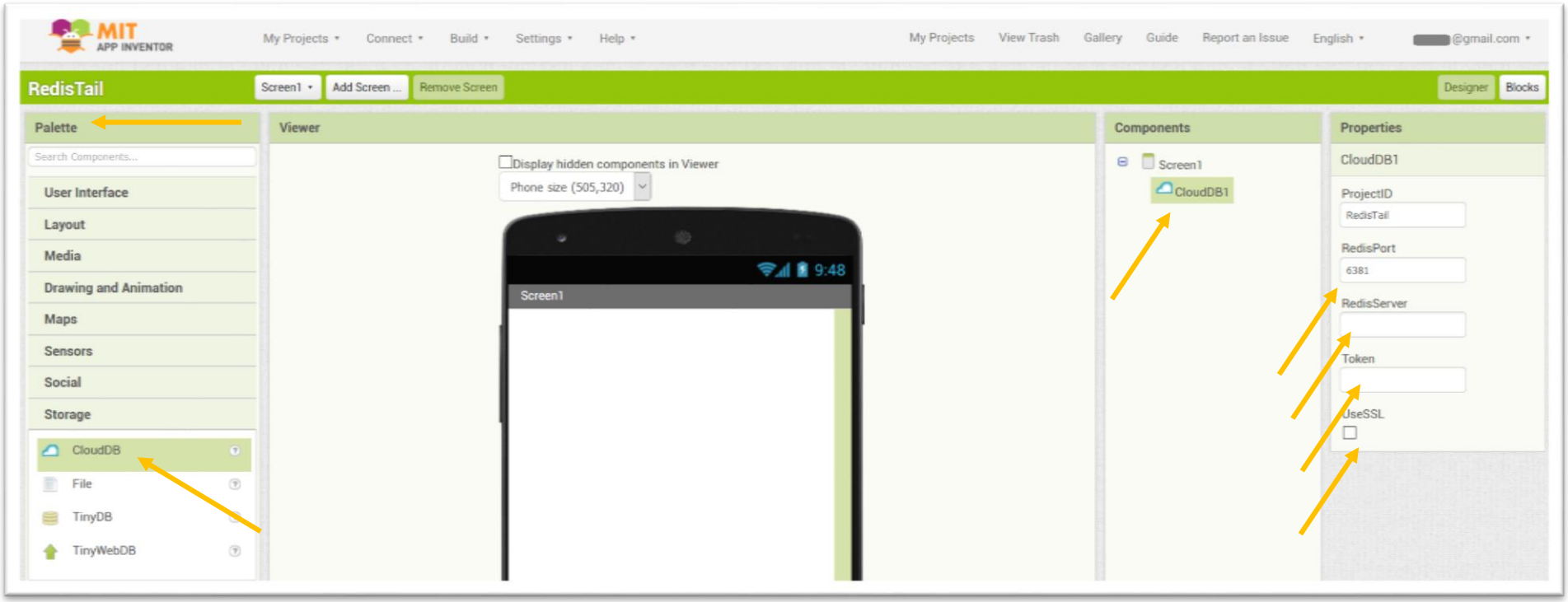
$ redis-cli GET <key>

In our case because we are using App inventor we will review how to use the blocks to work with Redis.

Inside App Inventor we have several blocks to use with Redis, we started to create a new project we will: My projects > Start new project



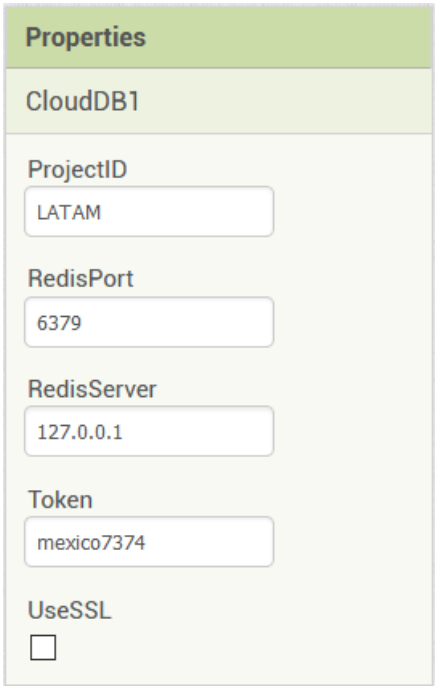

After creating the project we go to the top left and in the section "Palette" click on "Storage" and drag the object "CloudDB" this will integrate all the features to configure a connection to a Redis server.




The configuration is very simple we only have to give the following parameters to be able to connect to our Redis server (Local) that is running in our node (Mobile phone).

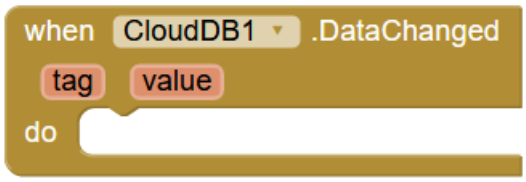
ProjectID. - This is the ID that will start the label that identifies the transaction queue in Redis.
RedisPort. - This is the port of the Redis server where we will connect by default is 6379
RedisServer. - This is the domain or Local IP of the node in our case and that of all the nodes will be 127.0.0.1
Token. - This is the password of the Redis server enabled to connect.
UseSSL. - This option gives us the opportunity to use SSL certificates in our case we leave it unenabled.

In our model and system design we will have the following parameters in all network nodes.




It is time to review the blocks that provide us with control and data processing over a Redis database environment.

We start with the method block (**DataChanged**)




This method will give us two values every time there is a change in the Redis server we have configured:

tag. - This value is the tag that identifies the transaction queue.

value. - This value contains the list of all transactions sent for processing. This value is a string list of type <String>.

In the case of the "value" is where we will begin to perform our information processing as follows.

As you provide us with a chain of all hash value transactions, we will start by checking if this chain is valid and verify if it has not been modified since its origin. We do this by using a very useful algorithm for the verification of large amounts of information.

We'll use merkle's tree algorithm. Applying this algorithm gives us a security in the integrity of the information we receive (transaction queue).

Let's see the following example of a transaction queue in Redis, we execute the Redis CLI (Command-Line) from a Termux terminal to check the "LATAM:HASH" key, we must have already executed the SQLite-Redis Connector to be able to consult this key.

```
C:memo>redis-cli
127.0.0.1:6379> get LATAM:HASH
"9d45198faaef624f2e7d1897dd9b3cde6ecca7fbac516ed1756b350fe1d56b4,"
f71c801a5fd25fc303ebc8c616204b4877ffb93006ec6a88bc30acf43ec250f5,"\60f8a3bcac1
ea7d38e86efbc3e3e00480807d23f980391000766e804ed14ecb2 ,
8a6dfe1d38c22e0f9212052efa6136da3edf1fb1b2a3e25224ac3d689124b754]
127.0.0.1:6379>
```

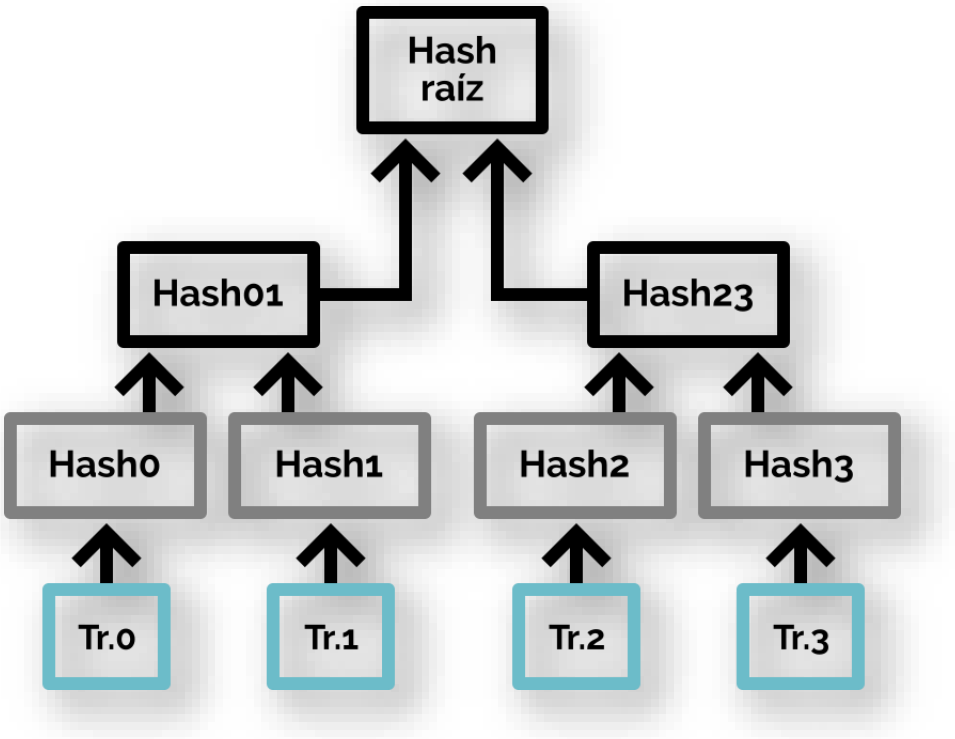
The previous transaction queue only has three elements since we see four hashes representing an individual transaction that make it up and they are the hashes of each transaction that was initially injected into the op.sqlite3 database within the "trans" and "sign" tables.

Now it's time to understand how the merkle tree works.

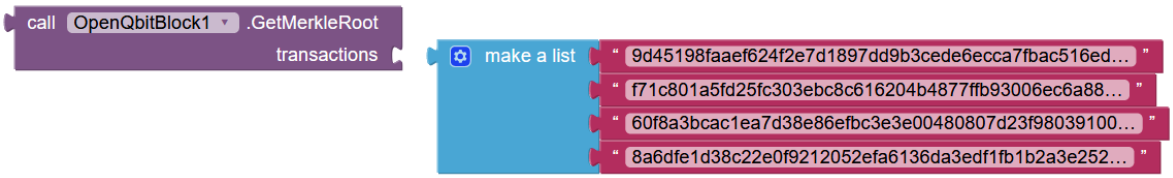
A hash merkle tree is a binary or non-binary data tree structure in which each node that is not a sheet is tagged with the hash of the concatenation of the labels or values of its child nodes. They are a generalization of hash lists and hash strings.

In our case we will make the following calculation to calculate the merkle tree for four elements this is done by calculating the result of taking pairs of data concatenated and get their respective hashes, the results of the first level will be applied to the results of the second level until there is only one final element, this will be called the hash merkleroot (root hash).

Let's look at the following diagram that describes this process.




The hash-based transaction queue will be pulled out via the Block (GetMerkleRoot)




Hash root: 51431822de7c94b90dc06d47b8f6275f315a4976c8479d30c32747fa90325432

The result is then compared with the LATAM:merkleroot key in the local Redis system and both keys must match to verify data integrity.

Another security point provided by the merkle tree is to confirm that a specific transaction is included in the transaction queue from its origin and that it has not been introduced by any external or internal communication means in a fraudulent way.

In the case that the chain is of odd elements in its summation, the last element duplicates to have an arrangement for and start the execution of the algorithm.

Another no less important point is the time to validate that each transaction corresponds to its origin-destination and that the asset is the one sent by the origin address.

This is where the Block (VerifySignature) is.


Before executing the previous block, you must first download the file (file.sig) in binary format from the "sign" table:

sqlite3 op.sqlite3 “select sign from sign where=id_addro;”

**id_addro:** This is the id of the source address of the "trans" table.

The previous search request will give us a Base64 format data of the digital signature of the current transaction, this to convert it into its original format (binary) we will need the Block(**DecoderFileBase64**).

Since we have our binary file (file.sig) we will have to load into the system the public key of the origin and the public key of the recipient also in binary format, having the four data in their respective formats will be the time to run the verification of the digital signature in the system.

- ✓ Public key home address
- ✓ Public key address of the recipient
- ✓ Active sent.
- ✓ Digital signature (file.sig)

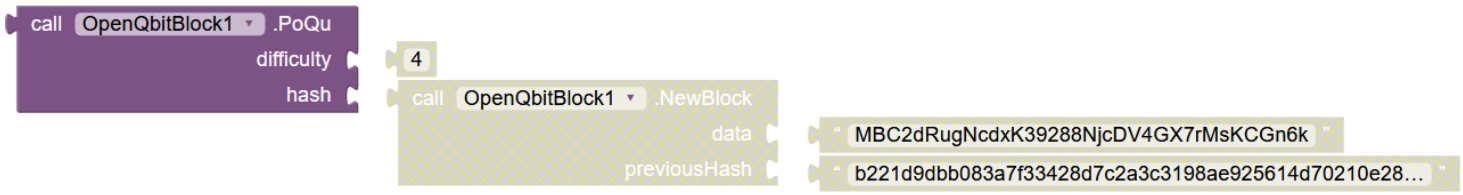
Public keys in binary format can be downloaded in the appropriate format (binary) from the shared database publickeys.db

This process must be executed for each individual operation in the transaction queue.

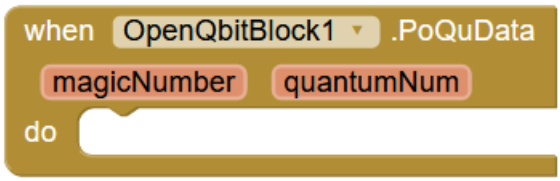
Finally we will review how the system chooses a node in a consensual way to be able to be chosen to add the next block to the block chain and be the one that processes the transaction queue.

This way of choosing the winning node to process the transaction queue is based on our algorithm developed for a Mini BlocklyChain system.

How we implemented the PoQu "Proof of Quantum" consensus. This consensus process is based on the generation of quantum random numbers and is applied using the (**PoQu**) block.




First we get two parameters that the block (**PoQu**) gives us delivered in the method (**PoQuData)** the parameters are the magicNumber and quantumNum.




The **magicNumber** is the number "nonce", is an integer that is obtained making an internal PoW with difficulty not bigger than 5, this number is in charge of giving a first requirement to the node with the magicNumber the node will be able to make a requirement to obtain a number of the system of generation of quantum random numbers that is in the range of the 0 to 1, this number will give a probability established randomly for the node in execution that will be stored in the table called "vote".

The "vote" table will store the quantumNum calculated by each node, this storage will be done with a number of nodes until a certain time established in each system design it is recommended to have of entries ((N/2) +1) where "N" is the amount of nodes available in the system and it can be established or controlled by an action in the "cron" task management tool of each node.

An important point is that by this point you should have already established a local time synchronization of each node through the automatic system of the mobile phone in case of using the **"Peer to Peer"** network in the transmission of the transaction queue.

The configuration of the cron agent in the nodes. See section "Time synchronization in the system nodes (Mobile phone) in minutes and seconds".

In this example as we are using the backup communications network we don't need the synchronization of minutes and seconds for the nodes because our example occupies a

"Client-Server" scheme this type of communication is only in the transmission process of the transaction queue. All other processes between nodes are through a "Peer to Peer" communication.

Now we'll look at the structure, design and creation of the "vote" table that will be located within the database called "quorum.db" that we'll also be creating in this example.

```
$ sqlite3

SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (id integer primary key AUTOINCREMENT NOT NULL, node_imei VARCHAR NOT NULL, quantumNum INTEGER NOT NULL, nonce INTEGER NOT NULL);
sqlite> .quit
```

The creation of the same vote table is shown below, but it is by introducing the SQL statement in a segmented form:

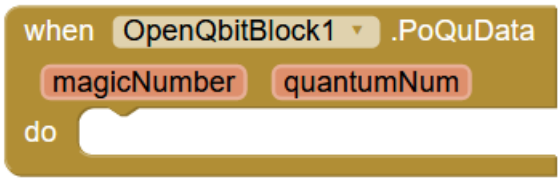
```
$ sqlite3

SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use " .open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
   ...> id    integer primary key AUTOINCREMENT
   ...> node_imei    VARCHAR NOT NULL,
   ...> quantumNum    INTEGER NOT NULL,
   ...> nonce    INTEGER NOT NULL
 ...> );
sqlite> .tables
vote
sqlite> .quit
```

We will see something similar in the creation of the base "quorum.db" and table vote.


Now let's see how we get the values from the "vote" table.

node_imei. - We **get** this value using the block (**GetDeviceID**).

quantumNum - This value is one of the results of the (**PoQuData)** method which is obtained using the (**PoQu**) block.




nonce. - This value is obtained from having already executed the block (**PoQu) in an** integral way and the same as the magicNumber of the method (**PoQuData**).

After completing the INSERTS in the "vote" table, it is necessary to make a copy of the database.

After a certain time and based on the design of each system, the "cron" service will be executed on each network node and will have the next SELECT to be processed in the "vote" table:

SELECT node_imei FROM vote WHERE magicNumber= (SELECT max(magicNumber) FROM vote);

The previous SELECT returns the IMEI result with higher probability, now each node that runs SELECT will make a comparison of its IMEI with the result IMEI and only the node that matches will create a file with the highest probability number that will be replicated in the network "Peer to Peer" by a file with the format IMEI.mbc that will contain the IMEI of the winning node.

The winning node will be able to start processing the transaction queue. Based on all the blocks above.

Two important points is that depending on each system creation three processes will have to be reviewed and customized by each designer.

1.- When the winning node starts processing the transaction queue, a method or process must be implemented where it must be checked that the winning node is online and communication with the network has not been lost.

2.- When the transaction queue processing starts, the winning node must start two control flags indicating the beginning of the processing and another one to confirm that the transaction queue processing has been finished. These two flags must be shared in the network to all the nodes, this will help to locate connection failures or processing failures or too much processing time.

3.- In case of communication failure or other event where the winning node has not been able to process the transaction queue, the next node in the immediate probability range must be chosen.

The previous point can be controlled with a service that verifies that the winning node is online and can use the "cron" service, the script must be developed for each designed case, however, a generic example of a shell script is shown below so that it can be modified according to the needs of each Mini Blocklychain system.

```
#!/bin/bash
dir="/data/data/com.termux/files/home/Sync/imei";
if [ !"$(ls $directory)" ]
 then
  sqlite3 quorum.db "UPDATE vote SET magicNumber=0 WHERE magicNumber= (SELECT
max(magicNumber) FROM vote);"
else

MAX_NUM=$(sqlite3 quorum.db "SELECT max(magicNumber) FROM vote);")
IMEI_quorum=$(sqlite3 quorum.db "SELECT node_id FROM vote WHERE=MAX_NUM")
IMEI_local=$(cat device_imei) // Use the block (GetDevice)
 if [ IMEI_quorum -eq IMEI_local ]
     then
        touch $MAX_NUM > IMEI.mbc
 fi
fi
exit
```

## 31. Annex "Integration with Ethereum & Bitcoin environments".

Now we will see how we can integrate the two blockchain systems best known worldwide specialized in cryptomonies such as Ethereum and Bitcoin.

Let's start with the installation of the software that will help us perform all possible transactions in the Ethereum environment.

What is Ethereum?

Ethereum is an open source platform, decentralized unlike other block chains, Ethereum can do much more. It is programmable, which means that developers can use it to create new types of applications.

These decentralized applications (or "dapps") get the benefits of cryptomontage and blockchain technology. They are reliable and predictable, which means that once they are "loaded" into the Ethereum, they will always run on schedule. They can control digital assets to create new types of financial applications. They can be decentralized, which means no one entity or person controls them.

Right now, thousands of developers around the world are creating applications on Ethereum and inventing new types of applications, many of which you can use today:

- Crypto-currency portfolios that allow you to make cheap, instant payments with ETH or other assets
- Financial applications that allow you to borrow, lend or invest your digital assets
- Decentralized markets, allowing you to exchange digital assets, or even exchange "predictions" about real-world events.
- Games where you have assets in the game and can even win real money.
- It has intelligent contracts that are programs with agreements to be executed when the premises with which it was elaborated or created are fulfilled.

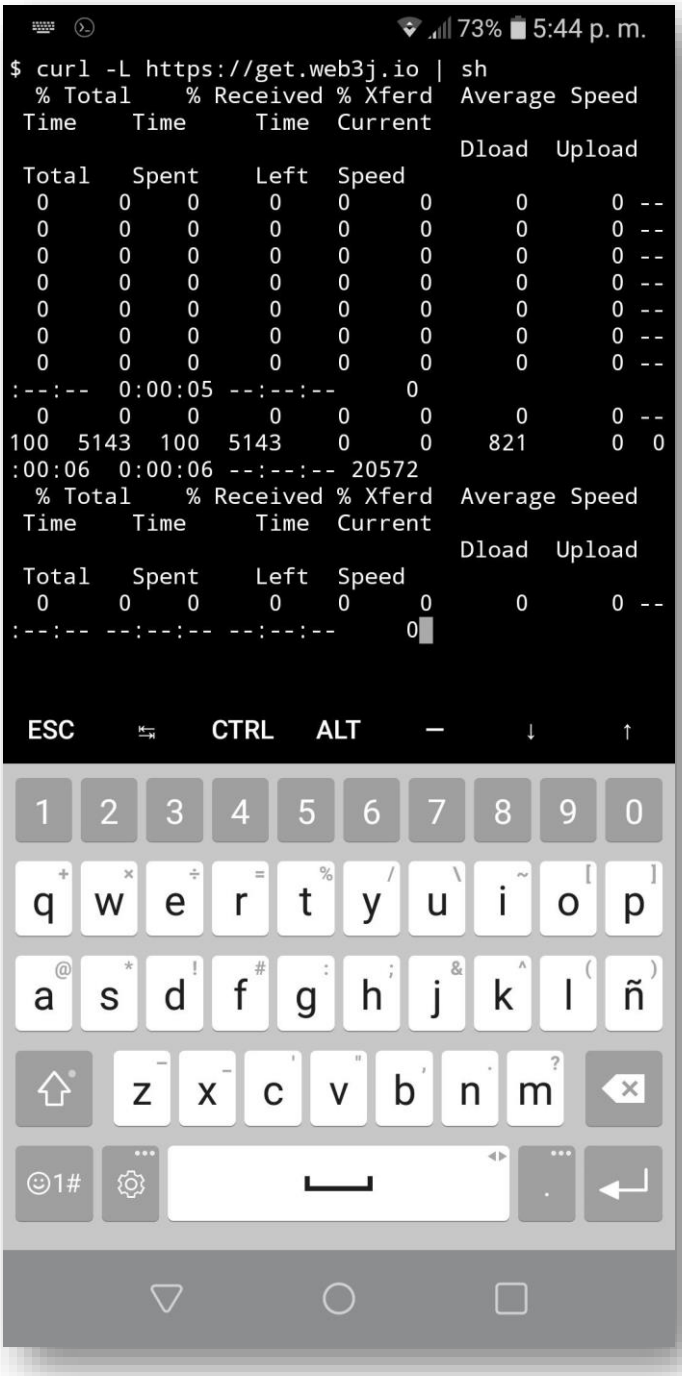
Intelligent contracts share similarities with DApps (decentralized applications), but also separate them from some important differences.

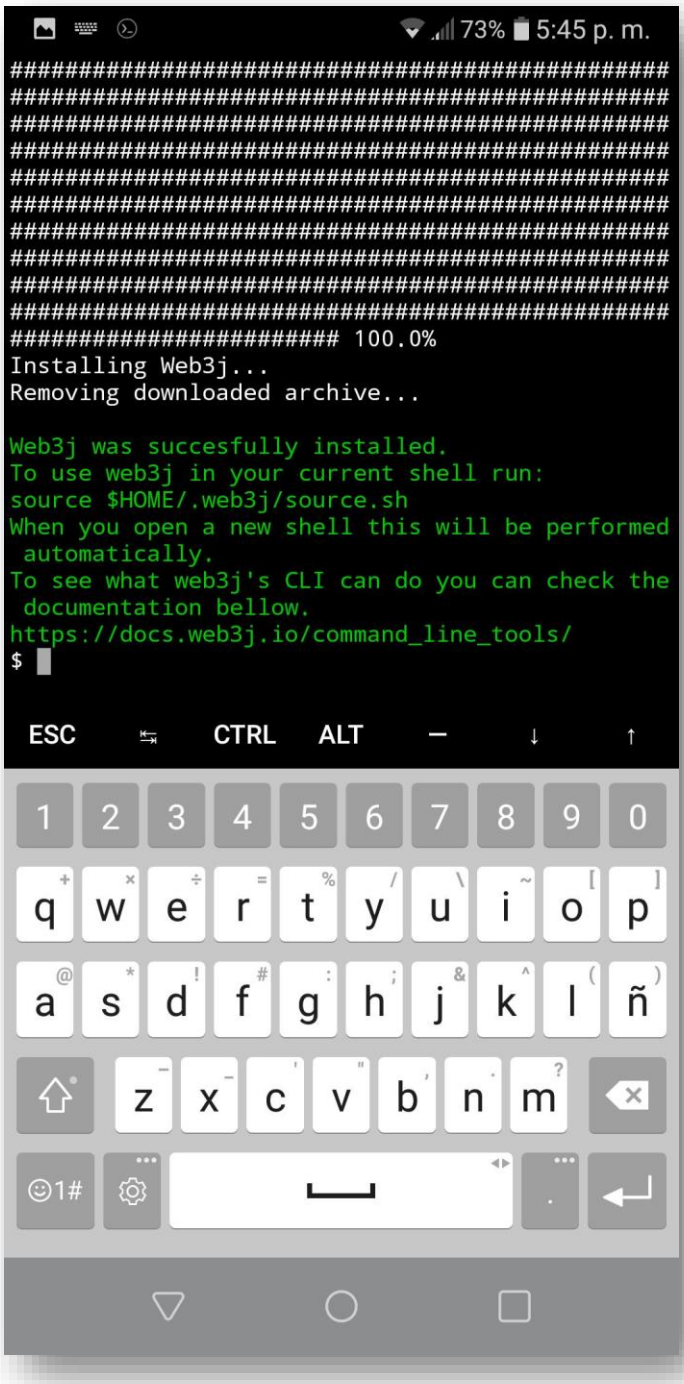
Like smart contracts, a DApp is an interface that connects a user to a service from a provider through a decentralized peer network. But, while smart contracts need a fixed number of participants to be created, DApps have no limit to the number of users. Moreover, they are not just limited to financial applications such as smart contracts: a DApp can serve any purpose you can think of.

In our case we will use the library called "Web3j" that is developed in Java, and that allows to interact with the Ethereum blockchain in a simple and intuitive way.

Execute the following command to install "Web3j":

```
$ curl -L get.web3j.io | sh
```



Then we must create the environment variable $JAVA_HOME with the path where the executable "java" is located since the library will search this variable to be able to be executed successfully.

$ JAVA_HOME= /data/data/com.termux/files/usr/bin

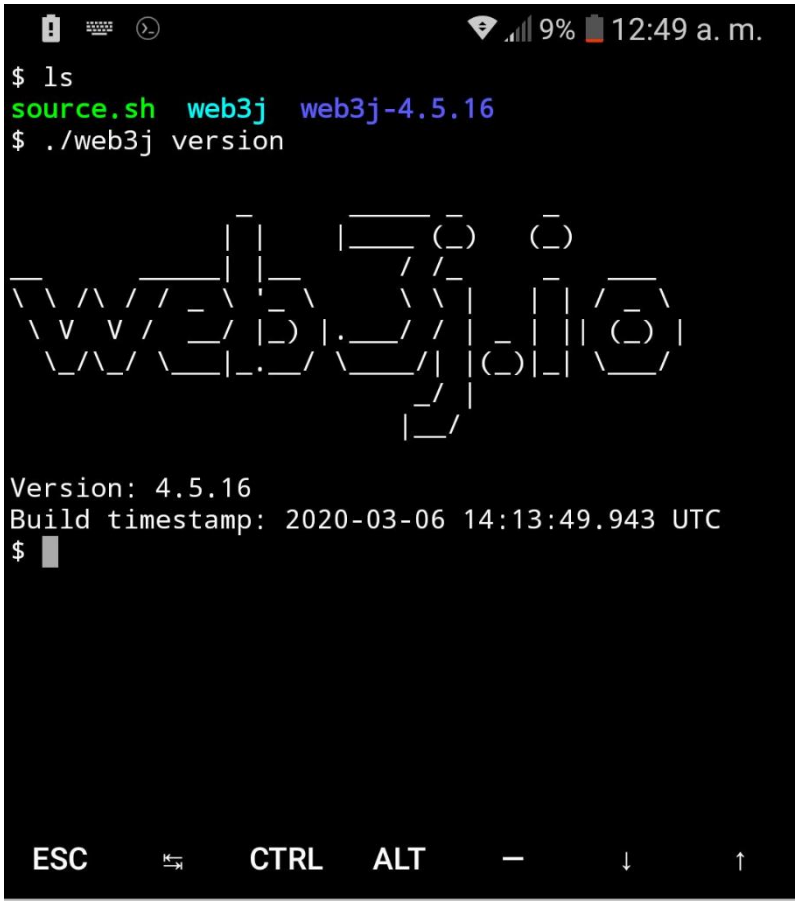
Once the variable is created we have to go to the directory where the library "Web3j" was installed by executing the following command, an important point is that the directory is hidden after dr the command of "cd" we put a point "." And then the directory without spaces, as follows:

$ cd .web3j

Once inside we test if the library is running properly with the following command:

./web3j version

It results in something very similar to:

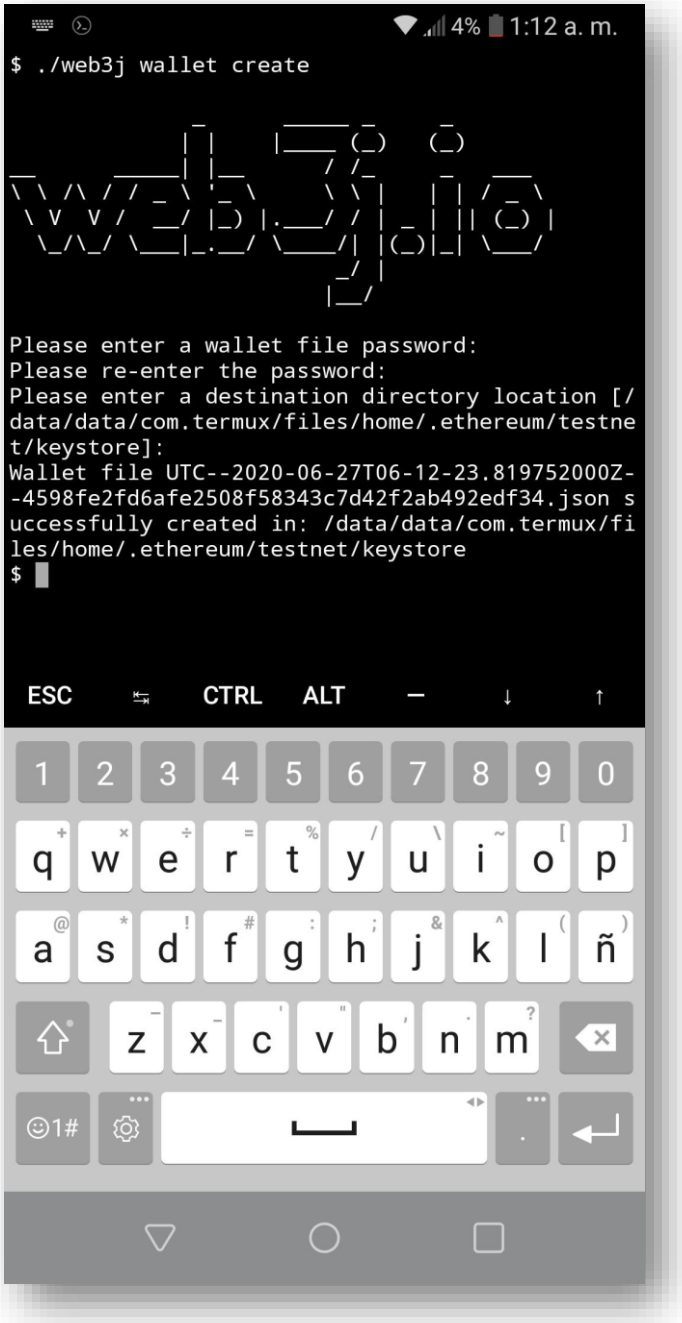



Later we will create a Wallet to be used in the blockchain environment of Ethereum in the following way:

## $ ./web3j wallet create

The previous command gives us the address of ethereum:

4598fe2fd6afe2508f58343c7d42f2ab492edf34




It results in a file in JSON format containing the address and encrypted data to be used later to generate the private key for the account that has just been created.

This JSON-type file will be created in a default directory called keystore.

From here on we can already perform operations using and/or executing the Web3j command.

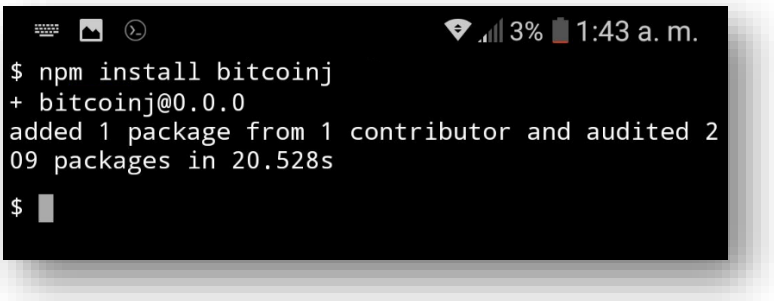
We can do this by using the extension (ConnectorSSHClient).

To know how to use the different parameters and operations of the "Web3j" library we can help us by consulting the documentation on its official site.

https://docs.web3j.io/

For the Bitcoin environment, we have two options: using the block () that generates the Bitcoin account and its respective public and private keys. We can use these keys to integrate into the Bitcoin environment by installing the "Bitcoij" java library.

$ npm install bitcoinj




In order to use this library we will rely on its official site.

https://bitcoinj.github.io/

A second option to integrate into the Bitcoin blockchain environment is to install the Bitcoind package for Termux as shown below.

$ apt install bitcoin


Now when we run the bitcoind agent on the Termux terminal it will automatically create an address in the directory:

/data/data/com.termux/files/hme/.bitcoin

In this case of Bitcoin we will rely on the following documentation from the "Bitcoind" agent.

In this case we can also use the extension (ConnectorSSHClient) to execute the "bitcoind" agent depending on each need.

Description of parameters for use with bitcoind.

## NAME

bitcoind - launch Bitcoin Core Daemon

## SYNOPSIS

**bitcoind** [*options*] Start *Bitcoin Core Daemon*

## DESCRIPTION

Start Bitcoin Core Daemon

## OPTIONS

**-?**

Print this help message and exit

**-alertnotify=<cmd>**

Execute the command when a relevant alert is received or we see a very long fork (%s in cmd is replaced by the message)

**-assumevalid=<hex>**

If this block is in the string assume that he and his ancestors are valid and potentially skip your write verification (0 for verify all, default: 000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee, testnet: 00000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7cfc2b4c75)

**-blocknotify=<cmd>**

Execute the command when the best block changes (%s in cmd is replaced by the block hash)

**-blockreconstructionextratxn=<n>**

Extra transactions to keep in memory for compact block reconstructions (default: 100)

**-blocksdir=<dir>**

Specify the block directory (default: <datadir>/blocks)

**-only in block**

If you refuse transactions from network partners. Portfolio or RPC transactions are not affected. (default: 0)

**-conf=<archive>**

Specify the configuration file. The relative paths will be prefixed by the location of the datadir. (default: bitcoin.conf)

**-daemon**

It runs in the background like a demon and accepts the commands

**-datadir=<dir>**

Specify the data directory

**-dbcache=<n>**

Maximum database cache size <n> MiB (4 to 16384, default: 450). In addition, unused mempool memory is shared for this cache (see **-maxmempool**).

**-debuglogfile=<file>**

Specify the location of the debug log file. Relative paths will be prefixed with a network-specific datadir location. (**-nodebuglogfile** to disable; default: debug.log)

**-includeconf=<file>**

Specify an additional configuration file, relative to the **-datadir** path (can only be used from the configuration file, not from the command line)

**-loadblock=<file>**

Import blocks from the external file blk000??.dat at start

**-maxmempool=<n>**

Keep the transaction memory pool below <n> megabytes (default: 300)

**-maxorphantx=<n>**

Keep maximum <n> disconnectable transactions in memory (default: 100)

**-mempoolexpiry=<n>**

Do not keep transactions in mempool longer than <n> hours (default: 336)

**-par=<n>**

Sets the number of verification threads in the dash (**-6** to 16, 0 = auto, <0 = leaves all cores free, default: 0)

**-persistmempool**

If you save the mempool when you shut down and load it when you restart (default: 1)

**-pid=<file>**

Specify the PID file. Relative paths will be prefixed with a network-specific datadir location. (default: bitcoind.pid)

**-punza=<n>**

Reduce storage needs by allowing for the pruning (removal) of old blocks This allows the RPC pruning chain to be called to remove specific blocks, and allows automatic pruning of old blocks if a target size in MIB is provided. This mode is incompatible with **-txindex** and **-rescan**. Warning: To reverse this setting it is necessary to re-download the entire block chain (default: 0 = disable block pruning, 1 = allow manual pruning via RPC, >=550 = automatically prune block files to stay below the target size specified in MIB)

**-reindex**

Rebuilds the string state and block index of blk*.dat files on disk

**-reindex-chainstate**

Reconstruct the state of the chain from the currently indexed blocks. When in pruning mode or if the blocks on the disk may be corrupted, use the full **re-index** instead.

**-sigh**

Create new files with default system permissions, instead of umask 077 (only effective with portfolio functionality disabled)

**-txindex**

Maintain a complete transaction index, used by the getrawtransaction rpc call (default: 0)

**-version**

Print & Go Version

Connection options:

**-addnode=<ip>**

Add a node to connect to and try to keep the connection open (see the RPC command help of the "addendum" for more information). This option can be specified several times to add several nodes.

**-banscore=<n>**

Threshold for disconnecting misbehaving colleagues (default: 100)

**-Meanwhile...**

Number of seconds to prevent misbehaving colleagues from reconnecting (default: 86400)

**-bind=<addr>**

Tie yourself to the address given and always listen to it. Use the [host]:port notation for IPv6

**-connection=<ip>**

Connect only to the specified node; **-noconnect disables** automatic connections (the rules for this pair are the same as for **-addnode**). This option can be specified several times to connect to several nodes.

**-discover**

Discover your own IP addresses (default: 1 when listening and not **-externalip** or **-proxy**)

**-dns**

Allow DNS searches for **-addnode**, **-seednode** and **-connect** (default: 1)

**-dnsseed**

Pair address query through DNS lookup, if low on addresses (default: 1 unless **-connection is** used)

**-enablebip61**

Send rejection messages by BIP61 (default: 1)

**-externalip=<ip>**

Specify your own public address

**-forcednsseed**

Always query the addresses of colleagues through DNS lookup (default: 0)

**-listen to**

Accept connections from the outside (default: 1 if no **-proxy** or **-connection**)

**-listenonionion**

Automatically create the Tor hidden service (default: 1)

**-maxconnections=<n>**

Maintain at most <n> connections with colleagues (default: 125)

**-maxreceivebuffer=<n>**

Maximum receive buffer per connection, <n>*1000 bytes (default: 5000)

**-maxsendbuffer=<n>**

Maximum send buffer per connection, <n>*1000 bytes (default: 1000)

**-maximum time setting**

The maximum allowed for the adjustment of the average time compensation of the pairs. The local time perspective can be influenced by the forward or backward pairs by this amount. (default: 4200 seconds)

**-maxuploadtarget=<n>**

Try to keep the outgoing traffic under the given target (in MiB for 24h), 0 = no limit (default: 0)

**-onion=<<ip:port>**

Use a separate SOCKS5 proxy to reach peers through Tor's hidden services, set **-noonion** to disable (default: **-proxy**)

**-onlynet=<net>**

Make outgoing connections only through the <net> network (ipv4, ipv6 or onion). Incoming connections are not affected by this option. This option can be specified several times to allow multiple networks.

**-pair filters**

Supports blocking filtering and transaction with flowering filters (default: 1)

**-permitbaremultisig**

Relay not P2SH multisig (default: 1)

**-port=<port>**

Listen to the connections in 'port' (default: 8333, testnet: 18333, regtest: 18444)

**-proxy=<ip:**port>

Connect through SOCKS5 proxy, set **-noproxy** to off (default: off)

**-proxyrandomize**

Randomize the credentials for each proxy connection This enables Tor flow isolation (default: 1)

**-seednode=<ip>**

Connect to a node to retrieve the pair addresses, and disconnect. This option can be specified several times to connect to several nodes.

**-timeout=<n>**

Specify the connection timeout in milliseconds (minimum: 1, default: 5000)

**-torcontrol=<ip>:<port>**

Tor control port to use if onion listening is enabled (default: 127.0.0.1:9051)

**-password=<pass>**

Tor control port password (default: blank)

**-upnp**

Use UPnP to assign the listening port (default: 0)

**-whitebind=<addr>**

Link to a certain address and the whitelisted partners who connect to it. Use the [host]:port notation for IPv6

**-whitelist=<IP address or** network>

The whitelisted pairs are connected from the given IP address (e.g. 1.2.3.4) or the annotated network of the CIDR (e.g. 1.2.3.0/24). It can be specified several times. Whitelisted pairs cannot be prohibited by the DoS

Portfolio options:

**-address type**

What type of addresses to use ("legacy", "p2sh-segwit", or "bech32", default: "p2sh-segwit")

**-avoid partial costs**

Group the outputs by direction, selecting all or none, rather than selecting by each output. Privacy is enhanced as an address is only used once (unless someone sends it after it has been spent), but may result in slightly higher rates as the selection of currencies may be suboptimal due to the added limitation (default: 0)

**-type change**

Which exchange rate to use ("legacy", "p2sh-segwit", or "bech32"). The default is the same as **-addresstype,** except that **-addresstype=p2sh-segwit** uses native segwit output when sending to a native segwit address)

**-from the wallet**

Do not load the wallet and deactivate RPC calls from the wallet

**-discardfee=<amt>**

The rate of the tariff (in BTC/kB) which indicates its tolerance to discard the change by adding it to the tariff (default: 0.0001). Note: An output is discarded if it is dust at this rate,

but we will always discard up to the dust retransmission rate and a discard rate above that is limited by the rate estimate for the longer target

**-fallbackfee=<amt>**

A fee rate (in BTC/kB) to be used when the fee estimate has insufficient data (default: 0.0002)

**-keypool=<n>**

Set key pool size to <n> (default: 1000)

**-mintxfee=<amt>**

Rates (in BTC/kB) lower than this are considered as a zero rate for the creation of the transaction (default: 0.00001)

**-paytxfee=<amt>**

Rate (in BTC/kB) to add to the transactions you send (default: 0.00)

**-rescan**

Rescan the block chain to look for missing portfolio transactions at the beginning

**-salvagewallet**

Attempting to retrieve the private keys to a corrupt wallet in the boot

**-wasting the exchange of information**

Spend the unconfirmed change when sending the transactions (default: 1)

**-txconfirmtarget=<n>**

If no payment fee is set, include a sufficient fee for transactions to begin confirmation on average within n blocks (default: 6)

**-portfolio maintenance**

Update the portfolio to the latest format at the beginning

**-wallet=<path>**

Specify the path of the portfolio database. You can specify it several times to load several portfolios. The path is interpreted in relation to <walletdir> if it is not absolute, and will be created if it does not exist (such as a directory containing a wallet.dat file and log files). For backward compatibility, it will also accept the names of existing data files in <walletdir>).

**-walletbroadcast**

Make the portfolio diffusion transactions (default: 1)

**-walletdir=<dir>**

Specify the directory to save the portfolios (default: <datadir>/portfolios if exists, otherwise <datadir>)

**-walletnotify=<cmd>**

Execute command when a portfolio transaction changes (%s in cmd is replaced by TxID)

**-walletrbf**

Send the transactions with the option to include the whole RBF activated (RPC only, default: 0)

**-zapwallettxes=<mode>**

Delete all transactions from the portfolio and only retrieve the parts of the blocking chain through **-rescan** at the beginning (1 = keep tx-metadata, e.g. payment request information, 2 = drop tx-metadata)

ZeroMQ notification options:

**-zmqpubhashblock=<address>**

Enable the publishing block in 'address

**-zmqpubhashblockhwm=<n>**

Set outgoing message publication block with a high watermark (default: 1000)

**-zmqpubhashtx=<address>**

Enable publishing the hashish transaction in 'address

**-zmqpubhashtxhwm=<n>**

Set publish the output message of the high watermark hash transaction (default: 1000)

**-zmqpubrawblock=<address>**

Enable the raw publishing block in 'address

**-zmqpubrawblockhwm=<n>**

Set Raw Block Outgoing Message High Watermark (default: 1000)

**-zmqpubrawtx=<address>**

Enable publication of the raw transaction in 'address

**-zmqpubrawtxhwm=<n>**

Set publish gross transaction output message high watermark (default: 1000)

Debugging/Testing options:

**-debug=<category>**

Output debugging information (default: **-nodebug**, providing 'category' is optional). If <category> is not supplied, or if <category> = 1, all debug information is output. <category> can be: net, tor, mempool, http, bench, zmq, db, rpc, estimatefee, addrman, selectcoins, reindex, cmpctblock, rand, prune, proxy, mempoolrej, libevent, coindb, qt, leveldb.

**-debugexclude=<category>**

Exclude debugging information from a category. Can be used in conjunction with **-debug=1** to generate deb**ug** records for all categories except one or more specified categories.

**-help-debug**

Print help message with debugging options and exit

**-logips**

Include the IP addresses in the debug output (default: 0)

**-logtimestamps**

Prepare the debugging output with the time stamp (default: 1)

**-maxtxfee=<amt>**

Maximum total fees (in BTC) for use on a single portfolio transaction or on a gross transaction; if set too low, it can abort large transactions (default: 0.10)

**-printing for the console**

Send trace/debugging information to the console (default: 1 when there is no **-daemon**. To disable logging to file, set **-nodebuglogfile**)

**-shrinkdebugfile**

Reduce debug.log file at client startup (default: 1 when no **-debug**)

**-uacomment=<cmt>**

Add a comment to the user agent string

Chain selection options:

**-testnet**

Use the test chain...

Node retransmission options:

**-bytespersigop**

Byte equivalents per sigop in broadcast and mining transactions (default: 20)

**-datacarrier**

Relay and mine data carrier transactions (default: 1)

**-datacarrierize**

Maximum size of data in the transactions of the data carriers we retransmit and extract (default: 83)

**-mempool-replacement**

Enable replacement of transactions in the memory pool (default: 1)

**-minrelaytxfee=<amt>**

Fees (in BTC/kB) lower than this are considered a zero fee for retransmission, extraction and creation of transactions (default: 0.00001)

**-whitelistforcerelay**

Forcing the retransmission of whitelisted partners' transactions, even if the transactions were already in mempool or violate the local retransmission policy (default: 0)

**-whitelistrelay**

Accept transmitted transactions received from whitelisted pairs even when no transactions are transmitted (default: 1)

Block the creation options:

**-blockmaxweight=<n>**

Set the maximum weight of the BIP141 block (default: 3996000)

**-blockmintxfee=<amt>**

Set the lowest commission rate (in BTC/kB) for transactions that are included in block creation. (default: 0.00001)

RPC server options:

**-Restaurant**

Accept public REST requests (default: 0)

**-rpcallowip=<ip>**

Allow JSON-RPC connections from the specified source. They are valid for <ip> a single IP (e.g., 1.2.3.4), a network/network mask (e.g., 1.2.3.4/255.255.255.0), or a network/CIDR (e.g., 1.2.3.4/24). This option can be specified several times

**-rpcauth=<userpw>**

User name and password HMAC-SHA-256 for JSON-RPC connections. The 'userpw' field comes in the format: 'username': 'SALT': $ 'HASH'. A canonical python script is included in share/rpcauth. The client then connects normally using the rpcuser=<USERNAME>/rpcpassword=<PASSWORD> argument pair. This option can be specified several times

**-rpcbind=<addr>[:**port]

Link to a certain address to listen to JSON-RPC connections. Do not expose the RPC server to unreliable networks such as the public Internet! This option is ignored unless **-rpcallowip** is also passed. The port is optional and overrides **-rpcport**. Use the [host]:port notation for IPv6. This option can be specified several times (default: 127.0.0.1 and ::1 i.e. localhost)

**-rpccookiefile=<loc>**

Location of the authorization cookie. Relative paths will be prefixed by a network-specific datadir location. (default: data dir)

**-rpcpassword=<pw>**

Password for JSON-RPC connections

**-rpcport=<port>**

Listen to JSON-RPC connections in 'port' (default: 8332, testnet: 18332, regtest: 18443)

**-rpcserialversion**

Sets the raw transaction serialization or the block hex returned in non-verbal mode, not segwit(0) or segwit(1) (default: 1)

**-rpcthreads=<n>**

Set the number of threads to handle RPC calls (default: 4)

**-rpcuser=<user>**

User name for JSON-RPC connections

**-server**

Accept command line commands and JSON-RPC

## 32. Licensing and use of software.

Android
https://source.android.com/setup/start/licenses
Termux
https://github.com/termux/termux-app/blob/master/LICENSE.md
Node
https://raw.githubusercontent.com/nodejs/node/master/LICENSE
SQLite
https://www.sqlite.org/copyright.html
Git
https://git-scm.com/about/free-and-open-source
sqlite-to-rest
https://github.com/olsonpm/sqlite-to-rest/blob/dev/license.txt
Redis DB
https://redis.io/topics/license
WorldTimeAPI NTP
http://worldtimeapi.org/pages/faqs#commercial-apps
Tor Network
https://github.com/torproject/tor/blob/master/LICENSE
Syncthing Network
https://forum.syncthing.net/t/syncthing-is-now-mplv2-licensed/2133
OpenSSH
https://www.openssh.com/features.html
Putty SSH
https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html
MIT App Inventor 2 Companion and App Inventor Blockly
https://appinventor.mit.edu/about/termsofservice
SQLite Expert Personal -freeware
http://www.sqliteexpert.com/download.html
Apache Ant
https://ant.apache.org/license.html
WGET
https://www.gnu.org/software/wget/
OpenJDK
https://openjdk.java.net/legal/

External extensions:
JSONTOOLs
https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html

Licensing opensource and commercial versions of Mini BlocklyChain system consult the official website http://www.openqbit.com

Mini BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly son marcas registradas por OpenQbit.

Mini BlocklyChain is in the public domain.

All code and documentation in Mini BlocklyChain has been dedicated to the public domain by the authors. All code authors and representatives of the companies they work for have signed affidavits dedicating their contributions to the public domain and the originals of those affidavits are stored in a safe at OpenQbit Mexico's main offices. Anyone is free to publish, use or distribute the original Mini BlocklyChain (OpenQbit) extensions, either as source code or as compiled binaries, for any purpose, commercial or non-commercial, and by any means.

The previous paragraph applies to the code and documentation deliverable in Mini BlocklyChain, those parts of the Mini BlocklyChain library that actually group and ship with a larger application. Some scripts used as part of the compilation process (for example, "configuration" scripts generated by autoconf) may be included in other open source licenses. However, none of these compilation scripts make it into the final Mini BlocklyChain deliverable library, so the licenses associated with those scripts should not be a factor in evaluating your rights to copy and use the Mini BlocklyChain library.

All the deliverable code in Mini BlocklyChain has been written from scratch. No code has been taken from other projects or from the open internet. Each line of code can be traced back to its original author, and all those authors have public domain dedications on file. Therefore, the Mini BlocklyChain code base is clean and not contaminated with code licensed from other open source projects, not open contribution

Mini BlocklyChain is open source, which means you can make as many copies as you want and do what you want with those copies, without limitation. But Mini BlocklyChain is not open source. To keep Mini BlocklyChain in the public domain and to ensure that the code is not contaminated with proprietary or licensed content, the project does not accept patches from unknown people. All code in Mini BlocklyChain is original, as it has been written specifically for use by Mini BlocklyChain. No code has been copied from unknown sources on the Internet.

Mini BlocklyChain is in the public domain and does not require a license. Even so, some organizations want legal proof of their right to use Mini BlocklyChain. The circumstances where this occurs include the following:

- Your company wants compensation for claims of copyright infringement.
- You are using Mini BlocklyChain in a jurisdiction that does not recognize the public domain.
- You are using Mini BlocklyChain in a jurisdiction that does not recognize an author's right to place his or her work in the public domain.
- You want to have a tangible legal document as evidence that you have the legal right to use and distribute Mini BlocklyChain.
- Your legal department tells you that you must buy a license.

If any of the above circumstances apply to you, OpenQbit, the company that employs all Mini BlocklyChain developers, will sell you a Mini BlocklyChain Title Guarantee. A Title Warranty is a legal document that states that the claimed authors of Mini BlocklyChain are the true authors, and that the authors have the legal right to dedicate the Mini BlocklyChain to the public domain, and that OpenQbit will vigorously defend itself against the licensing claims. All proceeds from the sale of Mini BlocklyChain's title warranties are used to fund the continuous improvement and support of Mini BlocklyChain.

Contributed Code

To keep Mini BlocklyChain completely free and royalty free, the project does not accept patches. If you want to make a suggested change and include a patch as a proof of concept, that would be great. However, don't be offended if we rewrite your patch from scratch. The type of non-commercial or opensource license who uses it in this modality and some similar without purchase of support either individual or corporate use no matter the size of the company will be governed by the following legal premises.

Warranty disclaimer. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) "AS IS", WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the correct use or redistribution of the Work and for assuming any risks associated with your exercise of permissions under this License.

Any financial or other losses incurred by the use of this software will be borne by the affected party. All legal disputes the parties will submit to courts only in the jurisdiction of Mexico City, country Mexico.

For commercial support, use and licensing an agreement or contract must be established between OpenQbit or its corporate and the interested party.

The terms and conditions of distribution marketing may change without notice, please go to the official website www.openqbit.com to see any changes to support and licensing clauses non-commercial and commercial.

Any person, user, private or public entity of any legal nature or from any part of the world who simply uses the software accepts without conditions the clauses established in this document and those that can be modified at any time in the portal of www.openqbit.co without previous notice and can be applied at the discretion of OpenQbit in non-commercial or commercial use.

Any questions and information about Mini BlocklyChain should be directed to the App Inventor community or to the various Blockly system communities as they are: AppBuilder, Trunkable, etc. and/or to the mail opensource@openqbit.com for the demand of questions can take the answer from 3 to 5 working days.

Support with commercial use.
support@openqbit.com

Sales for commercial use.
sales@openqbit.com

Legal information and licensing questions or concerns
legal@openqbit.com