Fing

Fing Limited
1st Floor Minerva House
Simmonscourt Road
Dublin 4, Ireland

fing.com/business

Fing Kit for Device Recognition

Fing Desktop & Embedded SDK - Developers Guide

Fing Kit for Device Recognition
Last Update: 21 March 2021
Document Version: 1.4.2

Table of Contents

1. FingKit Desktop/Embedded library

Overview

Fing has developed over the last few years a number of AI-driven algorithms to recognise connected devices by brand, make model and OS based on analysing network protocols along with top level tools for network scanning. Now the technology is available to be integrated to 3rd parties' application.

This document provides the guidelines for the Fing Software Development Kit (the **Fing Kit** from this point forward).

FingKit is available as ANSI C library and support the most common Operating System in the Desktop and Embedded space. The software library requires just an Internet connection and a Fing License Key.

To ensure decoupling of runtime and dynamic dependencies, FingKit Embedded runs as a separate command (or service, or daemon) in the embedding environment, thus isolating the execution of the command from the caller's environment. The process is configured through a configuration API, and generates output to a destination callback.

Available platforms

FingKit library is a cross-platform solution, supporting many of the most common platforms used in embedded and desktop devices. As the table groups the supported platforms in two categories: Kernel and Operating System. Every combination of these option is supported.

Kernel	Architecture	Operating System	Package Format
Linux	Intel i686, x86_64, arm, arm64, armhf, arm64hf	Other Linux flavors	.tar.gz
Microsoft Windows	x86 (compatible with 64-bit processors)	Microsoft Windows	.zip
Darwin	x86_64	Apple macOS	.zip

Integration within a Linux app

The Kit is available as an ANSI C library standard, suitable to be used with the Gnu Compiler Collection and the native Linux Operating System.

The FingKit library itself shall be included your application project as well. To import and use the functionalities of the FingKit modules, you shall simply import the module main header and the library.

```
#include <fingkit.h>
```

The FingKit functionalities are accessed via shared library (so) provide, list of architecture available and GCC version used to build FingKit Library as follows:

Architecture	GCC version
x86_64	5.4.0
armhf	5.4.0

Integration within a Windows app

The Kit is available as an ANSI C library standard, suitable to be used with the Microsoft .NET Framework and the native Microsoft Windows.

The FingKit library itself shall be included your application project as well. To import and use the functionalities of the FingKit modules, you shall simply import the module main header and the library.

```
#include <fingkit.h>
```

The FingKit functionalities are accessed via dynamic library (DLL) provide.

Integration within a Mac OSx app

The Kit is available as an ANSI C library standard, suitable to be used with the Gnu Compiler Collection and the native Darwin Operating System.

The FingKit library itself shall be included your application project as well. To import and use the functionalities of the FingKit modules, you shall simply import the module main header and the library.

```
#include <fingkit.h>
```

The FingKit functionalities are accessed via shared library (dylib) provide, list of architecture available and GCC version used to build FingKit Library as follows:

Architecture	GCC version
x86_64	Apple LLVM version 9.1.0 (clang-902.0.39.2) Target: x86_64-apple-darwin17.7.0 Thread model: posix

2. Package

The Fing SDK is a lightweight development kit containing a portable C Header, the libraries and some working example to simplify users' job. The examples are available both in C as in C++.

Structure

```
.
├── example/
│   ├── bin
│   │   └── FingKitDemo.bin
│   ├── fingkitdemo
│   └── src
│       ├── FingKitDemo.c
│       └── FingKitDemoCpp.cpp
├── include/
│   └── fingkit.h
└── lib/
    ├── ...
    └── ...
```

The `lib/` folder contains all the dependencies in different format:

- `.dll` for Windows
- `.so` for Linux / Unix / OpenWRT
- `.dylib` for MAC OSx

The `include/` folder hold the library interface that should be included by the integrator.

The `example/` folder contains the demo programs with the source code. On Windows the executable `FingKitDemo.bin` is called `FingKitDemo` and the script `fingkitdemo` is called `runFingKitDemo.bat`.

Installation

Installing the Fing Kit on your application is straightforward: it's sufficient to copy the content of `include/` and `lib/` folders in the corresponding directories of the application.

The example section might not be included in the source project.

Software Licenses

The Fing SDK has some linked or embedded dependencies to some libraries for cross-platform development, networking, encoding/compression and security:

- Boost – Cross-platform C++ library to provide base framework for applications.
- libPCAP / winPCAP – Low level networking.

- Protocol Buffer – Google's open-source technology open-source standard for binary data format/protocol.
- ZLIB – ZLib Compression.
- LZ4 – LZ4 Compression (*)
- ZSTD – ZStandard Compression (*)
- openSSL – Open-source standard toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols.
- NetSNMP – SNMP library.

Below the full list of software licenses:

Library	License
Boost Library	Boost Software License 1.0
OpenSSL	OpenSSL License / SSLEay License
NetSNMP	BSD License
TCP Dump - LibPCAP	BSD 3-Clause License
Protobuf	BSD License
ZLIB	ZLIB / LibPNG License
LZ4	BSD 3-Clause License
ZSTD	BSD 3-Clause License

(*) Embedded in the source code

3. API Specification

Asynchronous design

FingKit library operates asynchronously, to ensure your App is never blocked during each operation. A callback block is used to deliver the result of an operation, or the error object in case the operation could not be completed.

All callback methods are invoked in the main thread.

ANSI C

```
typedef void(* HandleFingEvent)(const char *result, int statusCode);
```

The callback block accepts the following list of parameters:

Parameter	Type	Description
result	`const char*`	The result coming from the FingKit. The result is usually in JSON format, but in general it depends on the type of operation or if an error occurred.
statusCode	`int`	Status code, in case the operation may not be completed correctly.

If successful, the callback result string contains a JSON-formatted result and return always 0 statusCode.

Each result contains a header for message type and status code and a body with content according to type, though an error description if it has be detected.

An example of successful and failed JSON structure:

successful

```
{
  "type": "<typeResult>",
  "statusCode": 0,
  "status":"OK",
  "result": { ..."<body>"... }
}
```

failed

```
{
  "type":"<typeResult>",
  "statusCode":<ErrorCode>,
  "status":"NOK",
  "result": { "error": "<error description>" }
}
```

Error Handling

The callback may return one of the following error codes in the statusCode if the attempt to validate the key failed.

Error Code	Description
-101	The service replied, but could not validate the key
-103	Configuration operation failed
-104	Background Process an error occurred
-105	Network Interfaces retrieval failed
-106	Discovery operation failed or an error occurred

API Suite

License Key validation

To enable the functionalities delivered by the FingKit, you must first obtain an API key and validate it. The validation requires access to the Internet, and it shall be executed at every application session in order to activate the features; a missing or failed validation disables the features of the FingKit.

ANSI C

```
void validateLicenseKey(const char *licenseKey, const char *userAgentToken,
                        HandleFingEvent *callback);
```

The method accepts the following list of parameters:

Parameter	Type	Description
licenseKey	`const char *`	**Required**. The unique license key that enable the usage of Fing Kit. The key is used to identify the Kit owner, assess the services that are enabled for a given license and to ensure the usage of the functionalities within the agreed terms
usageToken	`const char *`	**Optional**, max 512 characters. If available, the token provided by earlier calls. Make sure to keep and resend same token to have usage counts counted once per MAU.
callback	`HandleFingEvent`	**Required**. A callback block that is invoked when the validation terminates.

If successful (StatusCode equal to 0), the callback contains a JSON-formatted result as described in the following table.

Key	Value	Example
kitLicenseId	Your license key	Will be the same value passed as parameter
kitCustomerId	Your unique customer identifier, assigned on sign up. Usually, it's your company or App name	ACME
expiryDate	The time at which the provided key expires and a new key or new validation shall be performed	2016/11/23 02:00:07
state	The state of the license. It may be one of: • Ok • Suspended • Revoked	Ok
grantDiscovery	A Boolean value indicating if the network discovery feature is granted by your license	true
grantEnrichment	A Boolean value indicating if a Fing Service enrichment is enabled. Enrichment provides additional results on top the local scan, such as device type recognition.	true
usageToken	A token assigned to the running device for the present month	ABC123
usageCounted	A Boolean value indicating if this validation was the first validation of the licensing period	true

If the validation could not be performed or fails, a description of the error is reported in the result with corresponding status code. An example of the JSON result is reported below.

JSON

```
{
  "type":"license",
  "statusCode":0,
  "status":"OK",
  "result":{
      "kitLicenseId":"ABC123",
      "kitCustomerId":"ACME",
      "expiryDate":"2016/12/30 00:00:00",
      "state":"Ok",
      "grantDiscovery": "true",
      "grantEnrichment": "true",
      "usageToken": "ABC123",
      "usageCounted": "false"
  }
}
```

A failure to validate the key is reported via an JSON result. Every error in the validation process disables all functionalities.

Get Network Info

The FingKit allows to conveniently retrieve network details from the Interfaces the device is connected to. The network details may be retrieved through the following method.

ANSI C
```
void getNetworkInterfaces();
```

If successful (StatusCode equal to 0), the callback contains a JSON-formatted result as described in the following table.

Key	Value	Example
name	The Interface name	en0
description	The Interface description if available	
address	The base IP address of the network	192.168.0.0
netmask	The netmask expressed as CIDR notation. It represents the number of bits that make up the subnet part, and consequently the remaining bits identify the host part	24
type	The Interface type	Ethernet
defaultGateway	The default gateway of the network, true/false if available	True
gatewayAddress	The IP Address of the network gateway, if available	192.168.0.1
dnsAddress	The IP Address of the network DNS, if available	192.168.0.1

An example of the JSON result is reported below.

JSON
```
{
  "type": "networkinterfaces",
  "statusCode": 0,
  "status": "OK",
  "result": {
    "networkinfos": [
        {
            "name": "en0",
            "type": "Ethernet",
            "address": "192.168.1.245",
            "netmask": "192.168.1.0\/24",
            "gatewayAddress": "192.168.1.1",
            "defaultGateway": true
        },
```

```
        {
            "name": "awdl0",
            "type": "Ethernet",
            "address": "FE80:0000:0000:0000:24A7:46FF:FE54:E244",
            "netmask": "FE80:0000:0000:0000:24A7:46FF:FE54:E244\/128"
        },
        {
            "name": "utun0",
            "type": "BSD loopback encapsulation",
            "address": "FE80:0000:0000:0000:3396:4FF9:80E4:48D5",
            "netmask": "FE80:0000:0000:0000:3396:4FF9:80E4:48D5\/128"
        },
        {
            "name": "lo0",
            "type": "BSD loopback encapsulation",
            "address": "127.0.0.1",
            "netmask": "127.0.0.0\/8"
        }
    ],
    "dns": [
        "8.8.8.8"
    ]
  }
}
```


Configure FingKit

You may enable and tune the scan process through a set of options. The following scan options may be specified through the appropriate JSON configure passed on your API key:

ANSI C

```
void configureFingKit(const char *config);
```

Option	Description
networkinterface	The interface name. Discovery can run on default interface or on specific one. Providing 'default' makes engine automatically select and discover the current one to reach internet, at each round.
discoveryinterval	Discovery round interval in milliseconds
discoveryround	Discovery rounds number, 0 to discover forever until stopped.
discoverydatachunksize	Discovery result may be chunked in pages with list of devices.
discoveryhttpuseragent	Flag to enable http user agent discovery
fullprotocolinfo	Discovery result can contain the complete network protocols details or only device recognition summary.

An example of the configuration JSON is reported below.

```
{
  "networkinterface": "en0",
  "discoveryinterval": "60000",
  "discoveryround": "0",
  "discoverydatachunksize": "100",
  "fullprotocolinfo": "false"
}
```

If successful (StatusCode equal to 0), the callback contains a JSON-formatted result as described below

```
{
  "type":"configure",
  "statusCode":0,
  "status":"OK"
}
```

Get FingKit library version

To get the API version of the FingKit library is currently running

ANSI C

```
const char* getFingKitVersion()
```

Discovery lifecycle - Start

FingKit discovery must to be started by the API:

ANSI C

```
void startFingKit()
```

It will enable the discovery in according to configuration options. It can run continuously or for a configured number of rounds, on default interface or on specific one, returning output result in the configured callback, one for each scan. The output discovery format is JSON and follows the specs described in section Discovery data structure, result may be chunked at completion according to configured options.

JSON: Engine started

```
{
   "type":"engine",
   "statusCode":0,
```

```
   "status":"OK",
   "result": {
     "state": "started"
   }
}
```

Discovery lifecycle - Force Refresh

If you need to refresh a network discovery before the configured round interval, you can force a refresh through the API:

ANSI C

```
void refreshFingKitDiscovery()
```

It will start a new discovery round and will return output result in the configured callback.

JSON: Engine refreshed

```
{
  "type": "engine",
  "statusCode": 0,
  "status": "OK",
  "result": {
    "state": "refreshed"
  }
}
```

Discovery lifecycle - Stop

To terminate discovery engine the API:

ANSI C

```
void stopFingKit()
```

allows you to close it gracefully; upon correct engine stop a callback is called.
An example below:

JSON: Engine terminated

```
{
      "type":"engine",
      "statusCode":0,
      "status":"OK",
      "result":{
        "state":"terminated"
```

```
        }
}
```

3. Discovery data structure

The FingKit library returns the set of results format in according to configuration. At the moment, JSON format is supported, which allow an easy integration with any kind of hosting app or process. You can enable the result chunking at completion and/or the full protocol details.

Progress dataset of the discovery

For the current discovery, FingKit will provide a JSON data structure describing the progress status. This is the set of details returned.

Key	Value	Example
round	The round number of the discovery	"1"
state	The current state discovery, may assume the value: - started - discovering - completed - failed	"discovering"
progress	The progress of the scan, in percentage from 0 to 100	80
discoverydata	Discovery dataset of the network, if the state value is completed.	

Discovery dataset of the network

For the current network, FingKit will provide a JSON data structure describing the network details and analysed properties. This is the set of details returned.

Key	Value	Example
result_state	Flag discriminating if this scan has been enriched by Fing Device Recognition service	"enriched"
last_scan_timestamp	The time of the last scan	"2016/11/23 02:00:07"
time_zone	The time zone of the scanning device	"CEST"
nodes_count	The amount of nodes found in the network	"12"
nodes_up_count	The amount of nodes found online in the network.	"10"
network	Network dataset	
isp	Service Provider dataset	
page_current	The current page of the complete result	"1"
page_total	The total page of the complete result	"3"

page_is_first	Flag discriminating if this is first page	"true"
page_is_last	Flag discriminating if this I last page	"false"
nodes	List of Network node base dataset	

Network dataset

This is the set of details returned to network interface monitored.

Key	Value	Example
address_type	IPv4 or IPv6	IPv4
name	The network name from the interface	eth0
address	The network address	192.16.0.0
mask_prefix_length	The netmask length applied by the scan engine, in bits	24
gateway_ip_address	The IP address of the gateway	192.168.0.1
dns_address	The IP address of the DNS	192.168.0.1

Internet Service Provider dataset

If internet connection is available, the scan reports also additional details on the ISP connection and location. Some of these details may not be available, depending on the user's connection.

Key	Value	Example
country_city	The city name	Rome
address	The public IP address	62.23.136.134
host_name	The public host name	acces.134.136.23.62.rev.coltfrance.com
latitude	The latitude of the ISP point in decimal degrees	12.4833
longitude	The longitude of the ISP point in decimal degrees	41.8999
timezone	The time zone of the ISP	Europe
organization	The name of the organization providing Internet Access	COLT Technology Services Group Limited
country_code	The 2-letters country code	UK
country_region_code	The region code	LAZ
continent_code	The 2-letters country code	EU
country_postal_code	The postal code of the address	W10 5BN

Network Node base dataset

For each identified device, Fing provides a data structure describing the network details and recognition result and also analysed network protocols properties.

Key	Value	Description
mac_address	The MAC Address of the device that is currently using to connect to the network	"06:5c:89:c9:e7:d1"
addresses_list	The list of IP address assigned to the device in the current network. It may be multiple if the element is a network bridge or if it's	"172.28.0.14"

	temporarily being assigned multiple addresses	
state	Discriminates if the device is connected to the network or not. Can be "UP" or "DOWN"	"UP"
best_name	The best name of the device, evaluated from the names returned from the various protocols it replies to	"HP 2832", "Marco's iPhone"
best_type	A single type identifying its major role. It's intended to be as brandless as possible. **See Appendix 1 for further details.**	"Laptop", "Mobile", "Photo Camera", "Desktop".
best_make	The name of the makers/vendor of the device. It may overlap with the manufacturer, but it may be also different in case the network interface (ETH, WIFI) is different.	"Apple", "Huawei" (but not "Foxconn")
best_model	The human-readable name of the model	"iPhone 5S", "P9"
is_family	Flag advicing if the model is a generic family and not a specific model.	true
best_os	The name of the Operating system, when applicable	"iOS", "Android", "Windows", "macOS".
best_osver	The version of the Operating system, when applicable	"7 Ultimate", "10 Pro", "Mojave"
best_osbuild	The build number of the Operating system, when applicable	"19D88", "30.3454"
recog_rank	Rank value of the device recognition	95
host_name	The DNS name of the device	"mydevice.thissite.com"
mac_vendor	The name of the company that is officially manufacturing the network interface (ETH or WIFI). Names are reviewed and optimized to be consistent	"Samsung", "Apple", "Lenovo" for major brands, but also "Foxconn" for manufacturers that registered their components directly
netbios	Network node detail dataset for NetBIOS	
bonjour	Network node detail dataset for Bonjour	
upnp	Network node detail dataset for UPnP	
dhcp	Network node detail dataset for Dhcp	
dhcp6	Network node detail dataset for Dhcpv6	
http	Network node detail dataset for Http	
snmp	Network node detail dataset for Snmp	

Network node detail dataset for NetBIOS

FingKit exports for NetBIOS the following JSON structure, contained in the "netbios" JSON key, if the full protocol detail is configured.

Property	Description	Example
name	The NetBIOS name is used to uniquely identify the NetBIOS services listening on the first IP address that is bound to an adapter.	"MACBOOKPRO"

	The NetBIOS name is also known as a NetBIOS computer name.	
domain	A type of Fully-qualified Domain Name.	“mypc.locallan”
user	An optional user name. Due to security concerns, this is rarely available in the standard implementation	“MARCO”
is_file_server	An optional flag to detect if available file server is running.	“1” or “0”
is_domain_controller	An optional flag to detect if available domain controller is enabled.	“1” or “0”

Network node detail dataset for Bonjour

FingKit exports for Bonjour the following JSON structure, contained in the “bonjour” JSON key. If the full protocol detail is configured.

Property	Description	Example
name	The Bonjour name the device publishes	"name": "Giuseppes-MacBook-Pro"
model	The Bonjour model the device publishes	"model": "MacBookPro11,4"
os	The Bonjour Operating System name the device publishes	"os": "OSX:17"
serviceinfo_list	A list of bonjour additional services published by the device	{"name": "Giuseppe\u0019s MacBook Pro._device-info._tcp.local.", "addinfos": { "model":"MacBookPro11,4", “osxvers": "17" } }

Network node detail dataset for UPnP

FingKit exports for UPnP the following JSON structure, contained in the “upnp” JSON key. If the full protocol detail is configured.

Property	Description	Example
name	The UPnP name the device publishes	“My Macbook”
make	The UPnP Make name the device publishes	“Samsung”
model	The UPnP Model the device publishes	“SCD8291221”
type_list	A list of UPnP device types published by the device	“urn:Belkin:device:controllee:1”
service_list	A list of UPnP services published by the device	“urn:Belkin:service:manufacture:1 ” “urn:Belkin:service:smartsetup:1”

Network node detail dataset for Http User Agent

FingKit exports for Http User Agent the following JSON structure, contained in the "http" JSON key. If the full protocol detail is configured and http user agent if available and it is enabled as option. Please note HTTP user agent can be got only if FingKit is running on a gateway device, like e.g. the network router.

Property	Description	Example
useragent	The Http user agent list	"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; Media Center PC 6.0; InfoPath.3; BRI/2)"

Network node detail dataset for SNMP

FingKit exports for SNMP the following JSON structure, contained in the "snmp" JSON key. If the full protocol detail is configured.

Property	Description	Example
sysoid	The unique identifier of the device type	"1.3.6.1.4.1.9.1.516"
name	The SNMP name the device publishes	"HP
services	The SNMP list services the device publishes	
description	The SNMP description of the device	"Cisco IOS Software, C3750 Software (C3750-IPSERVICESK9-M), Version 12.2(46)SE"
contact	The SNMP contact point	"admin@cisco.com"
location	The SNMP location of device	"North Corridor"

Network node detail dataset for DHCP

FingKit exports for DHCP the following JSON structure, contained in the "dhcp" JSON key.
If the full protocol detail is configured.

Property	Description	Example
name	The DHCP name the device publishes	"My Macbook"
vendor	The DHCP vendor	"Samsung"
params	The DHCP params	"1,33,3,6,15,28,51,58,59"

Network node detail dataset for DHCP6

FingKit exports for DHCPv6 the following JSON structure, contained in the "dhcp6" JSON key.
If the full protocol detail is configured.

Property	Description	Example
name	The DHCPv6 name the device publishes	"DESKTOP-TR18HAM"
vendor	The DHCPv6 vendor	"Samsung"
options	The DHCPv6 options	"1:8,1,3,39,16,6"
params	The DHCPv6 option params	"17,23,24,39"

enterpriseid	The DHCPv6 enterprise id	311

Full Samples

An example of the JSON discovery life cycle and result is reported below.

Discovery started


```
{
  "type":"discovery",
  "statusCode":0,
  "status":"OK",
  "result":{
    "round":"1",
    "state":"started",
    "progress":"0"
   }
 }
```


Discovery running, at 10% progress


```
{
  "type":"discovery",
  "statusCode":0,
  "status":"OK",
  "result":{
    "round":"1",
    "state":"discovering",
    "progress":"10"
   }
 }
```


Discovery completed, with paged results
FIRST PAGE


```
{
   "type":"discovery",
   "statusCode":0,
   "status":"OK",
   "result":{
      "round":"1",
      "state":"completed",
      "progress":100,
      "discoverydata":{
         "result_state":"enriched",
         "last_scan_timestamp":"2018-10-24 10:11:48",
         "time_zone":"CEST",
         "nodes_count":"66",
         "nodes_up_count":"66",
         "network":{
            "address_type":"IPv4",
```

```
            "name":"en0",
            "address":"192.168.12.0\/22",
            "mask_prefix_length":"22",
            "gateway_ip_address":"192.168.12.1",
            "dns_address":"8.8.8.8"
         },
         "isp":{
            "country_city":"Rome",
            "address":"62.23.136.134",
            "host_name":"acces.134.136.23.62.rev.coltfrance.com",
            "longitude":"12.4833",
            "latitude":"41.899999999999999",
            "timezone":"Europe\/Rome",
            "organization":"COLT Technology Services Group Limited",
            "country_name":"Italy",
            "country_code":"IT",
            "continent_code":"EU"
         },
         "page_current":"1",
         "page_total":"3",
         "page_is_first":"true",
         "page_is_last":"false",
         "nodes":[ ... ]
      }
   }
}
```


NEXT PAGE


```
{
   "type":"discovery",
   "statusCode":0,
   "status":"OK",
   "result":{
      "round":"1",
      "state":"completed",
      "progress":100,
      "discoverydata":{
         "page_current":"2",
         "page_total":"3",
         "page_is_first":"false",
         "page_is_last":"false",
         "nodes":[
            {
               "mac_addresses":"70:5A:0F:90:F9:78",
               "address_list":[
                  "192.168.14.97"
               ],
               "state":"up",
```

```
                "best_name":"HP705A0F90F977",
                "best_type":"PRINTER",
                "best_make":"HP",
                "best_model":"Officejet Pro 6830",
                "is_family": false,
                "recog_rank":"45",
                "mac_vendor":"HP"
            },
            ......
            {
                "mac_addresses":"BC:83:85:DA:A1:C3",
                "address_list":[
                   "192.168.13.213"
                ],
                "state":"up",
                "best_type":"TABLET",
                "best_make":"Microsoft",
                "best_model":"Surface",
                "recog_rank":"40",
                "mac_vendor":"Microsoft"
            }
         ]
      }
   }
}
```


LAST PAGE


```
{
      "type":"discovery",
      "statusCode":0,
      "status":"OK",
      "result":{
         "round":"1",
         "state":"completed",
         "progress":100,
         "discoverydata":{
            "page_current":"3",
            "page_total":"3",
            "page_is_first":"false",
            "page_is_last":"true",
            "nodes":[
               {
                  "mac_addresses":"C8:14:51:58:40:58",
                  "address_list":[
                     "192.168.13.215"
                  ],
                  "state":"up",
                  "best_type":"MOBILE",
```

```
                "best_make":"Huawei",
                "best_model":"P10",
                "best_os":"Android",
                "recog_rank":"95",
                "mac_vendor":"Huawei"
            },
            .........
            {
                "mac_addresses":"DC:41:5F:E6:51:60",
                "address_list":[
                    "192.168.14.53"
                ],
                "state":"up",
                "best_type":"MOBILE",
                "best_make":"Apple",
                "best_model":"iPhone",
                "best_os":"iOS",
                "recog_rank":"90",
                "mac_vendor":"Apple"
            }
        ]
    }
  }
}
```

Appendix 1 - Fing Categorization - Groups and Types

For each device, Fing will analyze all the details and provide the best match among its supported types and categories. The list is reviewed and grows constantly as our Machine Learning system evolves.

Group	Device types
Mobile	Generic, Mobile, Tablet, MP3 Player, eBook Reader, Smart Watch, Wearable, Car
Audio & Video	Media Player, Television, Game Console, Streaming Dongle, Speaker/Amp, AV Receiver, Cable Box, Disc Player, Satellite, Audio Player, Remote Control, Radio, Photo Camera, Photo Display, Mic, Projector
Home & Office	Computer, Laptop, Desktop, Printer, Fax, IP Phone, Scanner, Point of Sale, Clock, Barcode Scanner
Home Automation	IP Camera, Smart Device, Smart Plug, Light, Voice Control, Thermostat, Power System, Solar Panel, Smart Meter, HVAC, Smart Appliance, Smart Washer, Smart Fridge, Smart Cleaner, Sleep Tech, Garage Door, Sprinkler, Electric, Doorbell, Smart Lock, Touch Panel, Controller, Scale, Toy, Robot, Weather Station, Health Monitor, Baby Monitor, Pet Monitor, Alarm, Motion Detector, Smoke Detector, Water Sensor, Sensor, Fingbox, Domotz Box
Network	Router, Wi-Fi, Wi-Fi Extender, NAS, Modem, Switch, Gateway, Firewall, VPN, PoE Switch, USB, Small Cell, Cloud, UPS, Network Appliance
Server	Virtual Machine, Server, Terminal, Mail Server, File Server, Proxy Server, Web Server, Domain Server, Communication, Database
Engineering	Raspberry, Arduino, Processing, Circuit Board, RFID Tag