# Table of Contents

## FingKit Desktop/Embedded library
- Overview
- Available platforms
- Integration within a Linux app
- Integration within a Windows app
- Integration within a Mac OSx app

## Package
- Structure
- Installation

## API Specification
- Asynchronous design
- Error Handling
- License Key validation
- Get Network Info
- Configure FingKit
- Get FingKit library version
- Discovery lifecycle - Start
- Discovery lifecycle - Force Refresh
- Discovery lifecycle - Stop

## Discovery data structure
- Progress dataset of the discovery
- Discovery dataset of the network
- Network dataset
- Internet Service Provider dataset
- Network Node base dataset
  - Network node detail dataset for NetBIOS
  - Network node detail dataset for Bonjour
  - Network node detail dataset for UPnP
  - Network node detail dataset for Dhcp
  - Network node detail dataset for Dhcpv6
  - Network node detail dataset for Http
  - Network node detail dataset for Snmp
  - Network node detail dataset for NetBIOS
  - Network node detail dataset for Bonjour
Network node detail dataset for UPnP 16
Network node detail dataset for Http User Agent 16
Network node detail dataset for SNMP 16
Network node detail dataset for DHCP 17
Network node detail dataset for DHCP6 17
Full Samples 17

Appendix 1 - Fing Categorization - Groups and Types 21
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.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Architecture</th>
<th>Operating System</th>
<th>Package Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>Intel i686, x86_64, arm, arm64, armhf, arm64hf</td>
<td>Other Linux flavors</td>
<td>.tar.gz</td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>x86 (compatible with 64-bit processors)</td>
<td>Microsoft Windows</td>
<td>.zip</td>
</tr>
<tr>
<td>Darwin</td>
<td>x86_64</td>
<td>Apple macOS</td>
<td>.zip</td>
</tr>
</tbody>
</table>

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.

```c
#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:
## 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.

```c
#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.

```c
#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:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>GCC version</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86_64</td>
<td>Apple LLVM version 9.1.0 (clang-902.0.39.2)</td>
</tr>
<tr>
<td></td>
<td>Target: x86_64-apple-darwin17.7.0</td>
</tr>
<tr>
<td></td>
<td>Thread model: posix</td>
</tr>
</tbody>
</table>

---

FingKit – Find and Recognize devices, on any network.
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
│   │   └── 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.
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

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

The callback block accepts the following list of parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>const char*</td>
<td>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.</td>
</tr>
<tr>
<td>statusCode</td>
<td>int</td>
<td>Status code, in case the operation may not be completed correctly.</td>
</tr>
</tbody>
</table>

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**

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

**failed**

```json
{
    "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.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-101</td>
<td>The service replied, but could not validate the key</td>
</tr>
<tr>
<td>-103</td>
<td>Configuration operation failed</td>
</tr>
<tr>
<td>-104</td>
<td>Background Process an error occurred</td>
</tr>
<tr>
<td>-105</td>
<td>Network Interfaces retrieval failed</td>
</tr>
<tr>
<td>-106</td>
<td>Discovery operation failed or an error occurred</td>
</tr>
</tbody>
</table>

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

The method accepts the following list of parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>licenseKey</td>
<td>const char *</td>
<td>Required. The unique license key that enable the usage of FingKit. 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.</td>
</tr>
<tr>
<td>usageToken</td>
<td>const char *</td>
<td>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.</td>
</tr>
<tr>
<td>callback</td>
<td>HandleFingEvent</td>
<td>Required. A callback block that is invoked when the validation terminates.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
</table>

FingKit – Find and Recognize devices, on any network.
<table>
<thead>
<tr>
<th><strong>kitLicenseId</strong></th>
<th>Your license key</th>
<th>Will be the same value passed as parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>kitCustomerId</strong></td>
<td>Your unique customer identifier, assigned on sign up. Usually, it’s your company or App name</td>
<td>ACME</td>
</tr>
<tr>
<td><strong>expiryDate</strong></td>
<td>The time at which the provided key expires and a new key or new validation shall be performed</td>
<td>2016/11/23 02:00:07</td>
</tr>
<tr>
<td><strong>state</strong></td>
<td>The state of the license. It may be one of:</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>● Ok</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Suspended</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Revoked</td>
<td></td>
</tr>
<tr>
<td><strong>grantDiscovery</strong></td>
<td>A Boolean value indicating if the network discovery feature is granted by your license</td>
<td>true</td>
</tr>
<tr>
<td><strong>grantEnrichment</strong></td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td><strong>usageToken</strong></td>
<td>A token assigned to the running device for the present month</td>
<td>ABC123</td>
</tr>
<tr>
<td><strong>usageCounted</strong></td>
<td>A Boolean value indicating if this validation was the first validation of the licensing period</td>
<td>true</td>
</tr>
</tbody>
</table>

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

```c
void getNetworkInterfaces();
```

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The Interface name</td>
<td>en0</td>
</tr>
<tr>
<td>description</td>
<td>The Interface description if available</td>
<td></td>
</tr>
<tr>
<td>address</td>
<td>The base IP address of the network</td>
<td>192.168.0.0</td>
</tr>
<tr>
<td>netmask</td>
<td>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</td>
<td>24</td>
</tr>
<tr>
<td>type</td>
<td>The Interface type</td>
<td>Ethernet</td>
</tr>
<tr>
<td>defaultGateway</td>
<td>The default gateway of the network, true/false if available</td>
<td>True</td>
</tr>
<tr>
<td>gatewayAddress</td>
<td>The IP Address of the network gateway, if available</td>
<td>192.168.0.1</td>
</tr>
<tr>
<td>dnsAddress</td>
<td>The IP Address of the network DNS, if available</td>
<td>192.168.0.1</td>
</tr>
</tbody>
</table>

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",
```

FingKit – Find and Recognize devices, on any network.
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);
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>networkinterface</td>
<td>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.</td>
</tr>
<tr>
<td>discoveryinterval</td>
<td>Discovery round interval in milliseconds</td>
</tr>
<tr>
<td>discoveryround</td>
<td>Discovery rounds number, 0 to discover forever until stopped.</td>
</tr>
<tr>
<td>discoverydatachunksize</td>
<td>Discovery result may be chunked in pages with list of devices.</td>
</tr>
<tr>
<td>discoveryhttpuseragent</td>
<td>Flag to enable http user agent discovery</td>
</tr>
<tr>
<td>fullprotocolinfo</td>
<td>Discovery result can contain the complete network protocols details or only device recognition summary.</td>
</tr>
</tbody>
</table>

An example of the configuration JSON is reported below.

```json
{
    "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",
}
{
    "name": "lo0",
    "type": "BSD loopback encapsulation",
    "address": "127.0.0.1",
    "netmask": "127.0.0.0/8"
}
"dns": [8.8.8.8]
}
```
"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:

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

Get FingKit library version

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

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

Discovery lifecycle - Start

FingKit discovery must to be started by the API:

```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

```json
{
  "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

```c
void refreshFingKitDiscovery()
```

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

JSON: Engine refreshed

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

Discovery lifecycle - Stop

To terminate discovery engine the API:

ANSI C

```c
void stopFingKit()
```

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

JSON: Engine terminated

```json
{
    "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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>round</td>
<td>The round number of the discovery</td>
<td>“1”</td>
</tr>
<tr>
<td>state</td>
<td>The current state discovery, may assume the value:</td>
<td>“discovering”</td>
</tr>
<tr>
<td></td>
<td>started</td>
<td></td>
</tr>
<tr>
<td></td>
<td>discovering</td>
<td></td>
</tr>
<tr>
<td></td>
<td>completed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>failed</td>
<td></td>
</tr>
<tr>
<td>progress</td>
<td>The progress of the scan, in percentage from 0 to 100</td>
<td>80</td>
</tr>
<tr>
<td>discoverydata</td>
<td>Discovery dataset of the network, if the state value is completed.</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>result_state</td>
<td>Flag discriminating if this scan has been enriched by Fing Device Recognition service</td>
<td>“enriched”</td>
</tr>
<tr>
<td>last_scan_timestamp</td>
<td>The time of the last scan</td>
<td>“2016/11/23 02:00:07”</td>
</tr>
<tr>
<td>time_zone</td>
<td>The time zone of the scanning device</td>
<td>“CEST”</td>
</tr>
<tr>
<td>nodes_count</td>
<td>The amount of nodes found in the network</td>
<td>“12”</td>
</tr>
<tr>
<td>nodes_up_count</td>
<td>The amount of nodes found online in the network.</td>
<td>“10”</td>
</tr>
<tr>
<td>network</td>
<td>Network dataset</td>
<td></td>
</tr>
<tr>
<td>isp</td>
<td>Service Provider dataset</td>
<td></td>
</tr>
<tr>
<td>page_current</td>
<td>The current page of the complete result</td>
<td>“1”</td>
</tr>
<tr>
<td>page_total</td>
<td>The total page of the complete result</td>
<td>“3”</td>
</tr>
<tr>
<td>page_is_first</td>
<td>Flag discriminating if this is first page</td>
<td>“true”</td>
</tr>
<tr>
<td>page_is_last</td>
<td>Flag discriminating if this I last page</td>
<td>“false”</td>
</tr>
<tr>
<td>nodes</td>
<td>List of Network node base dataset</td>
<td></td>
</tr>
</tbody>
</table>
Network dataset
This is the set of details returned to network interface monitored.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>address_type</td>
<td>IPv4 or IPv6</td>
<td>IPv4</td>
</tr>
<tr>
<td>name</td>
<td>The network name from the interface</td>
<td>eth0</td>
</tr>
<tr>
<td>address</td>
<td>The network address</td>
<td>192.16.0.0</td>
</tr>
<tr>
<td>mask_prefix_length</td>
<td>The netmask length applied by the scan engine, in bits</td>
<td>24</td>
</tr>
<tr>
<td>gateway_ip_address</td>
<td>The IP address of the gateway</td>
<td>192.168.0.1</td>
</tr>
<tr>
<td>dns_address</td>
<td>The IP address of the DNS</td>
<td>192.168.0.1</td>
</tr>
</tbody>
</table>

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.

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

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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_address</td>
<td>The MAC Address of the device that is currently using to connect to the network</td>
<td>“06:5c:89:c9:e7:d1”</td>
</tr>
<tr>
<td>addresses_list</td>
<td>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 temporarily being assigned multiple addresses</td>
<td>“172.28.0.14”</td>
</tr>
<tr>
<td>state</td>
<td>Discriminates if the device is connected to the network or not. Can be “UP” or “DOWN”</td>
<td>“UP”</td>
</tr>
<tr>
<td>best_name</td>
<td>The best name of the device, evaluated from the names returned from the various protocols it replies to</td>
<td>“HP 2832”, “Marco’s iPhone”</td>
</tr>
</tbody>
</table>
**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 advising 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

**FingKit exports for NetBIOS**

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The NetBIOS name is used to uniquely identify the NetBIOS services listening on the first IP address that is bound to an adapter. The NetBIOS name is also known as a NetBIOS computer name.</td>
<td>&quot;MACBOOKPRO&quot;</td>
</tr>
<tr>
<td>domain</td>
<td>A type of Fully-qualified Domain Name.</td>
<td>&quot;mypc.locallan&quot;</td>
</tr>
<tr>
<td>user</td>
<td>An optional user name. Due to security concerns, this is rarely available in the standard implementation</td>
<td>&quot;MARCO&quot;</td>
</tr>
<tr>
<td>is_file_server</td>
<td>An optional flag to detect if available file server is running.</td>
<td>&quot;1&quot; or &quot;0&quot;</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_domain_controller</td>
<td>An optional flag to detect if available domain controller is enabled.</td>
<td>“1” or “0”</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The Bonjour name the device publishes</td>
<td>&quot;name&quot;: &quot;Giuseppes-MacBook-Pro&quot;</td>
</tr>
<tr>
<td>model</td>
<td>The Bonjour model the device publishes</td>
<td>&quot;model&quot;: &quot;MacBookPro11,4&quot;</td>
</tr>
<tr>
<td>os</td>
<td>The Bonjour Operating System name the device publishes</td>
<td>&quot;os&quot;: &quot;OSX:17&quot;</td>
</tr>
</tbody>
</table>
| 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The UPnP name the device publishes</td>
<td>&quot;My Macbook&quot;</td>
</tr>
<tr>
<td>make</td>
<td>The UPnP Make name the device publishes</td>
<td>&quot;Samsung&quot;</td>
</tr>
<tr>
<td>model</td>
<td>The UPnP Model the device publishes</td>
<td>&quot;SCD8291221&quot;</td>
</tr>
<tr>
<td>type_list</td>
<td>A list of UPnP device types published by the device</td>
<td>&quot;urn:Belkin:device:controllee:1&quot;</td>
</tr>
<tr>
<td>service_list</td>
<td>A list of UPnP services published by the device</td>
<td>&quot;urn:Belkin:service:manufacture:1&quot; &quot;urn:Belkin:service:smartsetup:1&quot;</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>useragent</td>
<td>The Http user agent list</td>
<td>&quot;Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET&quot;</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysoid</td>
<td>The unique identifier of the device type</td>
<td>&quot;1.3.6.1.4.1.9.1.516&quot;</td>
</tr>
<tr>
<td>name</td>
<td>The SNMP name the device publishes</td>
<td>HP</td>
</tr>
<tr>
<td>services</td>
<td>The SNMP list services the device publishes</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>The SNMP description of the device</td>
<td>&quot;Cisco IOS Software, C3750 Software (C3750-IPSERVICESK9-M), Version 12.2(46)SE&quot;</td>
</tr>
<tr>
<td>contact</td>
<td>The SNMP contact point</td>
<td>&quot;<a href="mailto:admin@cisco.com">admin@cisco.com</a>&quot;</td>
</tr>
<tr>
<td>location</td>
<td>The SNMP location of device</td>
<td>&quot;North Corridor&quot;</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The DHCP name the device publishes</td>
<td>&quot;My Macbook&quot;</td>
</tr>
<tr>
<td>vendor</td>
<td>The DHCP vendor</td>
<td>&quot;Samsung&quot;</td>
</tr>
<tr>
<td>params</td>
<td>The DHCP params</td>
<td>&quot;1,33,6,15,28,51,58,59&quot;</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The DHCPv6 name the device publishes</td>
<td>&quot;DESKTOP-TR18HAM&quot;</td>
</tr>
<tr>
<td>vendor</td>
<td>The DHCPv6 vendor</td>
<td>&quot;Samsung&quot;</td>
</tr>
<tr>
<td>options</td>
<td>The DHCPv6 options</td>
<td>&quot;1:8,1,3,39,16,6&quot;</td>
</tr>
<tr>
<td>params</td>
<td>The DHCPv6 option params</td>
<td>&quot;17,23,24,39&quot;</td>
</tr>
<tr>
<td>enterpriseid</td>
<td>The DHCPv6 enterprise id</td>
<td>311</td>
</tr>
</tbody>
</table>

### Full Samples

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

**Discovery started**

```json
{
   "type": "discovery",
   "statusCode": 0,
}
```

FingKit – Find and Recognize devices, on any network.
"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",
"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"
}]
}
}
null};

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;

null;
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.

<table>
<thead>
<tr>
<th>Group</th>
<th>Device types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mobile</strong></td>
<td>Generic, Mobile, Tablet, MP3 Player, eBook Reader, Smart Watch, Wearable, Car</td>
</tr>
<tr>
<td><strong>Audio &amp; Video</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Home &amp; Office</strong></td>
<td>Computer, Laptop, Desktop, Printer, Fax, IP Phone, Scanner, Point of Sale, Clock, Barcode Scanner</td>
</tr>
<tr>
<td><strong>Home Automation</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>Router, Wi-Fi, Wi-Fi Extender, NAS, Modem, Switch, Gateway, Firewall, VPN, PoE Switch, USB, Small Cell, Cloud, UPS, Network Appliance</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Virtual Machine, Server, Terminal, Mail Server, File Server, Proxy Server, Web Server, Domain Server, Communication, Database</td>
</tr>
<tr>
<td><strong>Engineering</strong></td>
<td>Raspberry, Arduino, Processing, Circuit Board, RFID Tag</td>
</tr>
</tbody>
</table>