WPS SDK Reference Manual

Version 4.9

Contents

1 Description **1**

1.1 Installation . . . 1

1.2 API Summary . . . 1

1.2.1 Registration . . . 1

1.2.2 API modes . . . 1

1.2.2.1 Network-centric mode . . . 2

1.2.2.2 Device-centric mode . . . 2

1.2.2.3 Tiling mode . . . 2

1.2.3 Geofencing . . . 2

1.2.4 Offline location . . . 3

2 Change Log **4**

2.1 Version 4.9.3 . . . 5

2.2 Version 4.9.2 . . . 5

2.3 Version 4.9.1 . . . 5

2.4 Version 4.9 . . . 5

2.5 Version 4.8 . . . 5

2.6 Version 4.7.6 . . . 5

2.7 Version 4.7.5 . . . 6

2.8 Version 4.7 . . . 6

2.9 Version 4.6 . . . 6

2.10 Version 4.5 . . . 6

2.11 Version 4.4 . . . 6

2.12 Version 4.3 . . . 7

2.13 Version 4.2 . . . 7

2.14 Version 4.1 . . . 7

2.15 Version 4.0 . . . 7

2.16 Version 3.4 . . . 7

2.17 Version 3.3 7
2.18 Version 3.2 7
2.19 Version 3.1 8
2.20 Version 3.0 8
2.21 Version 2.7 8
2.22 Version 2.6.1 8
2.23 Version 2.6 8
2.24 Version 2.5 8
2.25 Version 2.4 8
2.26 Version 2.3.1 8
2.27 Version 2.3 9

3 Installation 10

3.1 Requirements 11
3.1.1 wpsapi.so 11
3.2 Files 11
3.3 Logging 11
3.3.1 WPS log properties file 11
3.4 Troubleshooting 12

4 Patents Issued to Skyhook 13

5 Limited Use License 19

6 Location Settings 21

6.1 Windows 8 or later (Desktop Mode) 22
6.2 OS X 10.7 or later 22

7 Deprecated List 24

8 Data Structure Documentation 26

8.1 WPS_GeoFence Struct Reference 26
8.1.1 Detailed Description 26
8.1.2 Field Documentation 26
8.1.2.1 latitude 26
8.1.2.2 longitude 26
8.1.2.3 period 27
8.1.2.4 radius 27
8.1.2.5 size 27

8.1.2.6 type . . . 27
8.2 WPS_IPLocation Struct Reference . . . 28
8.2.1 Detailed Description . . . 28
8.2.2 Field Documentation . . . 28
8.2.2.1 ip . . . 28
8.2.2.2 latitude . . . 28
8.2.2.3 longitude . . . 28
8.2.2.4 street_address . . . 29
8.3 WPS_Location Struct Reference . . . 30
8.3.1 Detailed Description . . . 30
8.3.2 Field Documentation . . . 30
8.3.2.1 age . . . 30
8.3.2.2 altitude . . . 31
8.3.2.3 bearing . . . 31
8.3.2.4 hash . . . 31
8.3.2.5 hashLength . . . 31
8.3.2.6 hasScore . . . 32
8.3.2.7 historicalLocationCount . . . 32
8.3.2.8 hpe . . . 32
8.3.2.9 id . . . 32
8.3.2.10 ip . . . 33
8.3.2.11 latitude . . . 33
8.3.2.12 longitude . . . 33
8.3.2.13 nap . . . 33
8.3.2.14 ncell . . . 33
8.3.2.15 nlac . . . 34
8.3.2.16 nsat . . . 34
8.3.2.17 score . . . 34
8.3.2.18 signature . . . 34
8.3.2.19 signatureLength . . . 36
8.3.2.20 speed . . . 36
8.3.2.21 street_address . . . 36
8.3.2.22 timestamp . . . 36
8.3.2.23 type . . . 37
8.3.2.24 withIP . . . 37
8.4 WPS_NameCode Struct Reference . . . 38

8.4.1 Detailed Description 38
8.4.2 Field Documentation 38
8.4.2.1 code 38
8.4.2.2 name 38
8.5 WPS_SimpleAuthentication Struct Reference 39
8.5.1 Detailed Description 39
8.5.2 Field Documentation 39
8.5.2.1 realm 39
8.5.2.2 username 39
8.6 WPS_StreetAddress Struct Reference 40
8.6.1 Detailed Description 40
8.6.2 Field Documentation 40
8.6.2.1 address_line 40
8.6.2.2 city 40
8.6.2.3 country 40
8.6.2.4 county 41
8.6.2.5 postal_code 41
8.6.2.6 province 41
8.6.2.7 region 41
8.6.2.8 state 41
8.6.2.9 street_number 41

9 File Documentation 42
9.1 doc/sdk/locationsettings.dox File Reference 42
9.2 include/api/wpsapi.h File Reference 43
9.2.1 Typedef Documentation 44
9.2.1.1 WPS_CertifiedLocationCallback 44
9.2.1.2 WPS_GeoFenceCallback 45
9.2.1.3 WPS_GeoFenceHandle 46
9.2.1.4 WPS_LocationCallback 46
9.2.1.5 WPS_TilingCallback 46
9.2.2 Enumeration Type Documentation 47
9.2.2.1 WPS_Continuation 47
9.2.2.2 WPS_GeoFenceType 47
9.2.2.3 WPS_LocationType 48
9.2.2.4 WPS_ReturnCode 48
9.2.2.5 WPS_StreetAddressLookup 49

9.2.3 Function Documentation 50
9.2.3.1 WPS_certified_location 50
9.2.3.2 WPS_free_ip_location 50
9.2.3.3 WPS_free_location 50
9.2.3.4 WPS_free_offline_token 51
9.2.3.5 WPS_geofence_cancel 51
9.2.3.6 WPS_geofence_cancel_all 51
9.2.3.7 WPS_geofence_set 51
9.2.3.8 WPS_ip_location 53
9.2.3.9 WPS_load 53
9.2.3.10 WPS_location 54
9.2.3.11 WPS_offline_location 54
9.2.3.12 WPS_offline_token 56
9.2.3.13 WPS_periodic_location 57
9.2.3.14 WPS_register_user 58
9.2.3.15 WPS_set_key 60
9.2.3.16 WPS_set_local_files_path 60
9.2.3.17 WPS_set_proxy 61
9.2.3.18 WPS_set_registration_user 61
9.2.3.19 WPS_set_server_url 61
9.2.3.20 WPS_set_tier2_area 62
9.2.3.21 WPS_set_tiling 62
9.2.3.22 WPS_set_user_agent 63
9.2.3.23 WPS_signed_certified_location 64
9.2.3.24 WPS_tiling 64
9.2.3.25 WPS_tune_location 65
9.2.3.26 WPS_unload 65
9.2.3.27 WPS_version 65

10 Example Documentation **67**
10.1 CertifiedSignatureTestJava 67
10.2 CertifiedSignatureTestPhp 68
10.3 wpsapitest.cpp 69

Chapter 1

Description

Author:

Skyhook

1.1 Installation

For installation instructions please read Installation.

1.2 API Summary

The WPS API can be summarized to the following calls:

- `WPS_load()`
- `WPS_set_key()`
- `WPS_location()`
- `WPS_periodic_location()`
- `WPS_unload()`

1.2.1 Registration

Prior to using any API calls the application should set the API key. This is accomplished by calling `WPS_set_key()`.

See `WPS_set_key()` for more details.

1.2.2 API modes

The API works in 2 major modes: *network centric* and *device centric*. In addition the *tiling* mode is a mix of the two.

1.2.2.1 Network-centric mode

In the network-centric model the API issues calls to a remote server to determine a location. This is the default mode.

If a path to a local file has been setup (See `WPS_set_local_files_path()`), the API first tries to determine the location locally, without calling the remote server. This is called the *mixed-mode* model.

1.2.2.2 Device-centric mode

In the device-centric model the API determines location locally. This mode is used if the device cannot connect to the remote server.

This mode requires setting the path to at least one local file. See `WPS_set_local_files_path()`.

1.2.2.3 Tiling mode

The tiling mode is a mix of the network and device-centric mode. It downloads, from the server, a small portion of the database so the device can automonously determine its location, without further need to contact the server.

This mode is activated by calling `WPS_set_tiling()`.

1.2.3 Geofencing

The API supports *geofencing* that allows the application to be notifed when the user enters, or leaves, a defined zone.

This feature is activated by calling the `WPS_geofence_set()` API. Once defined the application runs `WPS_periodic_location()` to monitor the device's location. When `WPS_periodic_location()` determines the device is moving inside a geofence -- ie. a first location is determined to be outside, followed by a location determined to be inside the geofence -- it triggers a `WPS_GEOFENCE_ENTER` callback. Similarly when `WPS_periodic_location()` determines the device is moving outside a geofence -- ie. a first location is determined to be inside, followed by a location determined to be outside the geofence -- it triggers a `WPS_GEOGENCE_LEAVE` callback.

When an application no longer desires to monitor a geofence it can call `WPS_geofence_cancel()` or `WPS_geofence_cancel_all()`.

Note:

Geofences can be added or removed while `WPS_periodic_location()` is running from the `WPS_periodic_location()` callback.

The `WPS_periodic_location()` callback itself will still be invoked whether the location calculated corresponds to any defined geofence.

See also:

WPS_geofence_set()
WPS_geofence_cancel()
WPS_geofence_cancel_all()

1.2.4 Offline location

The API supports *offline* location that allows the application to determine the location of the device even offline and outside of tile coverage by collecting a *token* that can be replayed when the device is once again online. Offline tokens are only valid for 90 days after they are generated. Attempting to redeem a token more than 90 days old will result in an error.

See also:

`WPS_offline_token()`
`WPS_offline_location()`

Chapter 2

Change Log

2.1 Version 4.9.3

- Optimized network load and power consumption in tiling mode
- Improved geofencing in UMTS and LTE networks
- Introduced `WPS_load()` and `WPS_unload()` calls
- Fixed a buffer overflow issue on Linux happening on very large Wi-Fi scans
- Switched to Secure Transport API on OS X and removed the pre-built OpenSSL libraries from the SDK bundle

2.2 Version 4.9.2

- Added a check to respect the system-wide location permission settings. See Location Settings for more information

2.3 Version 4.9.1

- Added signed certified location
- Improved power consumption
- Improved registration for devices without Wi-Fi or Cell adapters. This feature enhances the usability of WPS_ip_location
- Added support for Raspbian linux

2.4 Version 4.9

- Introducing key-based authentication

2.5 Version 4.8

- Added certified location
- Improved power consumption during geofencing
- Two new geofence types "INSIDE" and "OUTSIDE" for cases where immediate triggering is desired
- Fixed an edge case that could result in the client downloading extra tile data
- Improved accuracy of location on slow scanning devices

2.6 Version 4.7.6

- Improved memory management
- Fixed a bug that could negatively affect time to fix on some CDMA devices

2.7 Version 4.7.5

- Added support for location based on LTE cell towers
- Improved the accuracy of all cell locations
- Fixed a bug in tiling when venue and tuned locations fall on different sides of a tile boundary

2.8 Version 4.7

- Tuned locations are now available to users even when location is run with tiling. As a reminder, tuning always applies to actual current location only. It can not be used to tune an offline token. Tuning is not an effective strategy for trying to make small adjustments to a location. It is best used for adding to our coverage or fixing results that are off by hundreds of meters.
- Background location on Android now works even when device is asleep. When your device goes to sleep, and Skyhook is running within a service, we will continue to wake your device at the specified user period in order to provide location per your specifications. Please be aware that a short user period will reduce battery life if allowed to run continually. For this reason, we recommend running background locations with a user period of 120 seconds or greater.
- Skyhook now returns coarse location based on region codes (known as LACs) provided by the network. A new parameter has been introduced, nlac, to indicate that LAC information was used in the fix. These location fixes will typically have a higher error estimate (HPE).
- In-flight positioning fixes. Due to a third party change, older versions of our SDK may no longer be able to resolve in-flight location.

2.9 Version 4.6

- Improvements in power consumption and accuracy for geofences
- Enhanced indoor location for surveyed sites
- Added offline location

2.10 Version 4.5

Version 4.5 was never released publicly.

2.11 Version 4.4

- Added geofencing
- Improved long period support
- Improvements to positioning in remote mode

2.12 Version 4.3

- Improved memory usage when downloading tiles

2.13 Version 4.2

- Added auto-registration
- Improved off-line location coverage

2.14 Version 4.1

- Improved first fix accuracy
- Improved location when stationary
- Various improvements to hybrid algorithm, particularly in tracking mode

2.15 Version 4.0

- Optimized power management
- Better data and bandwidth utilization
- Improvements to hybrid positioning
- Inertial Navigation System

2.16 Version 3.4

- Added support for Windows 7, Windows Mobile 6, Mac OS X 10.6 (Snow Leopard)

2.17 Version 3.3

- Replaced `WPS_tiling()` with `WPS_set_tiling()`
- `WPS_set_*` calls can now be made while `WPS_periodic_location()` or `WPS_location()` is in progress

2.18 Version 3.2

- Added `ncell` and `nsat` to `WPS_Location`
- Added `altitude` to `WPS_Location`

2.19 Version 3.1

- Added `WPS_version()`

2.20 Version 3.0

- Added `WPS_tune_location()`
- Added `WPS_ERROR_TIMEOUT`

2.21 Version 2.7

- Added `WPS_set_user_agent()`

2.22 Version 2.6.1

- Changed return type from `int` to `WPS_Continuation` for `WPS_LocationCallback`

2.23 Version 2.6

- Added `WPS_tiling()`
- Added `WPS_set_tier2_area()`

2.24 Version 2.5

- Official release.

2.25 Version 2.4

- Added *mixed-mode* -- local location determination if possible
- Added `WPS_periodic_location()`
- Added `WPS_set_local_files_path()`
- Changed returned code from `WPS_set_proxy()` and `WPS_set_server_url()` to `void`
- Added `speed`, `bearing` and `nap` to `WPS_Location`
- Removed `hpe` from `WPS_IPLocation`

2.26 Version 2.3.1

- Added `WPS_register_user()`

2.27 Version 2.3

- Faster scanning
- Caching to allow for better response time
- `WPSScanner.dll` is no longer needed

Chapter 3

Installation

3.1 Requirements

- Linux 2.6 with wireless-extension
- libiw29 (Fedora) or libiw30 (Ubuntu and Raspbian) and libcurl3-gnutls
- The user must have read-write access to `/proc/net/wireless`.
  On most desktop linux write access to `/proc/net/wireless` is restricted to `root`.
- Wi-Fi network card for location based on Wi-Fi networks.
- GSM radio for location based on cell networks.
- Active Internet connection for the network-centric model.

3.1.1 wpsapi.so

- `libwpsapi.so` must be installed on the client's device.

3.2 Files

```
include/
    wpsapi.h                      header file for libwpsapi.so
lib/
    libwpsapi.so                  WPS client library
documentation/
    sdk.pdf                       documentation
    html/                         documentation
example/
    wpsapitest/
        libwpsapi.so              WPS client library
        wpsapitest                sample application
        wpsapitest.cpp            sample application (source code)
        wpslog.properties         sample wpslog.properties
    CertifiedSignatureTestJava/   sample Java code for verifying the certified location signature
    CertifiedSignatureTestPhp/    sample PHP code for verifying the certified location signature
```

3.3 Logging

On Linux define an environment variable named `WPS_LOG_CONFIGURATION` that contains the path to the `wpslog.properties` file.

3.3.1 WPS log properties file

Here's an example of a `wpslog.properties` file:

```
DEBUG,wpslog.txt
```

You can also redirect logging to `stdout` or `stderr`:

```
DEBUG,stdout
```

The logging level can be one of the following:

- FATAL
- ALERT
- CRITICAL
- ERROR
- WARN
- NOTICE
- INFO
- DEBUG

3.4 Troubleshooting

For any questions, please sign into your My Skyhook account at: `my.skyhookwireless.com`

Chapter 4

Patents Issued to Skyhook

- Dec 4, 2007: US Patent 7305245: Location-based services that choose location algorithms based on number of detected access points within range of user device
- April 30, 2008: Singapore Patent 134837: Continuous data optimization in positioning system
- July 22, 2008: US Patent 7403762: Method and system for building a location beacon database
- Aug 19, 2008: US Patent 7414988: Server for updating location beacon database
- Oct 7, 2008: US Patent 7433694: Location beacon database
- Dec 30, 2008: US Patent 7471954: Methods and systems for estimating a user position in a WLAN positioning system based on user assigned access point locations
- Jan 6, 2009: US Patent 7474897: Continuous data optimization by filtering and positioning systems
- Feb 17, 2009: US Patent 7493127: Continuous data optimization of new access points in positioning systems
- Mar 10, 2009: US Patent 7502620: Encoding and compression of a location beacon database
- Apr 7, 2009: US Patent 7515578: Estimation of positioning using WLAN access point radio propagation characteristics in a WLAN positioning system
- May 20, 2009: China Patent 101438270A: Location-based services that choose location algorithms based on number of detected wireless signal stations within range of user device
- July 23, 2009: US Patent 7551579: Calculation of quality of WLAN access point characterization for use in a WLAN positioning system
- July 23, 2009: US Patent 7551929: Estimation of speed and direction of travel in a WLAN positioning system using multiple position estimations
- Nov 30, 2009: Singapore Patent 132039: Location beacon database and server method used in building location beacon database, and location-based service
- Mar 31, 2010: Singapore Patent 157355: Location beacon database and server method used in building location beacon database, and location-based service
- Aug 3, 2010: US Patent 7768963: System and method of improving sample of WLAN packet information to improve Doppler frequency of a WLAN positioning device
- Aug 3, 2010: US Patent 7769396: Location-based services that choose location algorithms based on number of detected access points within range of a user device
- Oct 19, 2010: US Patent 7818017: Location-based services that choose location algorithms based on number of detected access points within range of a user device
- Nov 16, 2010: US Patent 7835754: Estimation of speed and direction of travel in a WLAN positioning system.
- Nov 17, 2010: China Patent 101438270B: Method for constructing location beacon database and location beacon server.
- Dec 21, 2010: US Patent 7856234: System and method for estimating positioning error within a WLAN-based positioning system.
- Mar 15, 2011: Australia Patent 2010/226912: Continuous Data Optimization in Positioning System
- Mar 29, 2011: US Patent 7916661: Estimation of position using WLAN access point radio propagation characteristics in a WLAN positioning system.

- July 23, 2011: Australia Patent 2006/335359: Continuous Data Optimization in Positioning System
- Aug 16, 2011: US Patent 7999742: System and method for using a satellite positioning system to filter WLAN access points in a hybrid positioning system
- Sept 6, 2011: US Patent 8014788: Estimation of speed of travel using the dynamic signal strength variation of multiple WLAN access points
- Sept 13, 2011: US Patent 8019357: System and method for estimating positioning error within a WLAN-based positioning system
- Sept 20, 2011: US Patent 8022877: Systems and methods for using a satellite positioning system to detect moved WLAN access points
- Oct 4, 2011: US Patent 8031657: Server for updating location beacon database
- Oct 5, 2011: European Patent 2012830: Estimation of speed and direction of travel in a WLAN positioning system
- Oct 27, 2011: Australia Patent 2005330513: Location beacon database and server, method of building location beacon database, and location-based service
- Nov 8, 2011: US Patent 8054219: Systems and methods for determining position using a WLAN-PS estimated position as an initial position in a hybrid positioning system
- Nov 15, 2011: Singapore Patent 152385: Systems and methods for estimating positioning error within a WLAN-based positioning system
- Nov 18, 2011: Japanese Patent 4866361: Method and system for building a location beacon database
- Nov 22, 2011: US Patent 8063820: Methods and systems for determining location using a hybrid satellite and WLAN positioning system by selecting the best SPS measurements
- Dec 1, 2011: Australia Patent 2007/317677: System and method for estimating positioning error within a WLAN based positioning system
- Dec 14, 2011: European Patent 2022281: Calculation of quality of a WLAN access point characterization for use in a WLAN positioning system
- Jan 3, 2012: US Patent 8089398: Methods and systems for stationary user detection in a hybrid positioning system
- Jan 3, 2012: US Patent 8089399: System and method for refining a WLAN-PS estimated location using satellite measurements in a hybrid positioning system
- Jan 3, 2012: US Patent 8090386: Estimation of speed and direction of travel in a WLAN positioning system
- Jan 24, 2012: US Patent 8103288: Estimation of speed and direction of travel in a WLAN positioning system using multiple position estimations
- March 6, 2012: US Patent 8130148: System and method for using a satellite positioning system to filter WLAN access points in a hybrid positioning system.
- March 20, 2012: US Patent 8140094: Data optimization of new access points in positioning systems.
- March 27, 2012: US Patent 8144673: Method and system for employing a dedicated device for position estimation by a WLAN positioning system.
- April 10, 2012: US Patent 8154454: Systems and methods for using a satellite positioning system to detect moved WLAN access points link.

- April 10, 2012: US Patent 8155673: Estimation of position using WLAN access point radiopropagation characteristics in a WLAN positioning system.
- April 10, 2012: US Patent 8155666: Methods and systems for determining location using a cellular and WLAN positioning system by selecting the best cellular positioning system solution.
- April 26, 2012: Australia Patent 2010/226917: Continuous Data Optimization in Positioning System
- April 27, 2012: Japanese Patent 4980247: Continuous Data Optimization in Positioning System
- May 22, 2012: US Patent 8185129: System and method of passive and active scanning of WLAN-enabled access points to estimate position of a WLAN positioning device.
- July 17, 2012: US Patent 8223074: Systems and methods for using a satellite positioning system to detect moved WLAN access points.
- July 24, 2012: US Patent 8229455: System and method of gathering and caching WLAN packet information to improve position estimates of a WLAN positioning device.
- Aug 14, 2012: US Patent 8242960: Systems and methods for using a satellite positioning system to detect moved WLAN access points.
- Aug 14, 2012: US Patent 8244272: Continuous data optimization of moved access points in positioning systems.
- Oct 2, 2012: US Patent 8279114: Method of determining position in a hybrid positioning system using a dilution of precision metric.
- Oct 9, 2012: US Patent 8284103: Systems and methods for using a satellite positioning system to detect moved WLAN access points.
- Nov 20, 2012: US Patent 8315233: System and method of gathering WLAN packet samples to improve position estimates of WLAN positioning device.
- Feb 5, 2013: US Patent 8369264: Method and system for selecting and providing a relevant subset of Wi-Fi location information to a mobile client device so the client device may estimate its position with efficient utilization of resources.
- Mar 26, 2013: US Patent 8406785: Method and system for estimating range of mobile device to wireless installation.
- June 11, 2013: US Patent 8462745: Methods and systems for determining location using a cellular and WLAN positioning system by selecting the best WLAN PS solution.
- July 2, 2013: US Patent 8478297: Continuous data optimization of moved access points in positioning systems.
- Sept 3, 2013: US Patent 8526967: Estimation of speed and direction of travel in a WLAN Positioning System.
- Sept 17, 2013: US Patent 8538457: Continuous data optimization of moved access points in positioning systems.
- Sept 18, 2013: Japanese Patent 5291618: Methods of filtering and determining cofidence factors for reference points for use in triangulation systems based on Wi-Fi access points.
- Oct 15, 2013: US Patent 8559974: Methods of and systems for measuring beacon stability of wireless access points.
- Oct 22, 2013: US Patent 8564481: Systems and methods for using a satellite positioning system to detect moved WLAN access points.

- Dec 10, 2013: US Patent 8606294: Method of and system for estimating temporal demographics of mobile users.
- Dec 12, 2013: Australian Patent 2008345574: Providing Wi-Fi location information to a mobile device in order to estimate its position.
- Dec 31, 2013: US Patent 8619643: System and method for estimating the probability of movement of access points in a WLAN-based positioning system.
- Jan 14, 2014: US Patent 8630664: Creation of an access point database.
- Jan 14, 2014: US Patent 8630657: Systems for and methods of determing likelihood of reference point identify duplication in a positioning system.
- Jan 28, 2014: US Patent 8638725: Methods and systems for determing location using a cellular and WLAN positioning system by selecting the best WLAN PS solution.
- Jan 28, 2014: US Patent 8644852: Improvement of the accuracy and performance of a hybrid positioning system.
- Feb 4, 2014: US Patent 8644852: Method and system for determining location using a hybrid satellite and WLAN positioning system by selecting the best WLAN-PS solution.
- Feb 19, 2014: Japanese Patent 5419891: Method and system for selecting and providing a relevant subset of Wi-Fi location information to a mobile client device so the client device may estimate its position with efficient utilization of resources.
- Mar 26, 2014: Japanese Patent 5450689: Continuous data optimization in positioning systems.
- Mar 26, 2014: Japanese Patent 5450529: Location beacon database and server, method of building location beacon database, and location based service using the same.
- April 15, 2014: US Patent 8700053: System for and methods of determing likelikood of relocation of reference points in a positioning system.
- April 22, 2014: US Patent 8706140: System and method of passive and active scanning of WLAN-enabled access points to estimate position of a WLAN positioning device.
- May 1, 2014: Australian Patent 2012200417: Method and system for determining location using a hybrid satellite and WLAN positioning system by selecting the best WLAN-PS solution.
- Sept 16, 2014: US Patent 8837363: Location beacon database and server method of building a location beacon database, and location based services.
- Oct 21, 2014: US Patent 2600861: Continuous data optimization method in a positioning system.
- Nov 18, 2014: US Patent 8890746: Methods and systems of hybrid positioning for increasing the reliability and accuracy of location estimation.
- Nov 21, 2014: European Patent 101466411: Calculation of quality of estimation of WLAN access point characteristics for use in a WLAN positioning system.
- Dec 9, 2014: US Patent 8909245: Method of estimating an expected error of a position estimate for use in a WLAN positioning system that estimates the position of a WLAN-enabled device.
- Jan 22, 2015: Australian Patent 2011202783: Method of utilizing Wi-Fi-enabled devices to monitor Wi-Fi access points in a target area to indicate whether a Wi-Fi access point has moved 5 relative to its previously recorded location.

- Feb 11, 2015: European Patent 101494252: Method of estimating an expected error of a position estimate for use in a WLAN positioning system that estimates the position of a WLAN-enabled device.
- Feb 24, 2015: US Patent 8965412: Method of determining device location using the stored locations for the identified Wi-Fi access points, the signal strengths of the received messages and the chosen location-determination algorithm.
- Mar 3, 2015: US Patent 8971915: Systems for and methods of determining likelihood of mobility of reference points in a positioning system are disclosed.
- Mar 3, 2015: US Patent 8971923: Methods of and systems for measuring beacon stability of wireless access points are provided.
- Mar 17, 2015: US Patent 8983493: Method and system for selecting and providing a relevant subset of Wi-Fi location information to a mobile client device so the client device may estimate its position with efficient utilization of resources.
- Mar 17, 2015: US Patent 8983504: Systems for and methods of determining likelihood of relocation of reference points in a positioning system are disclosed.
- Mar 31, 2015: US Patent 8996032: Systems for and methods of determining likelihood of reference point identity duplication in a positioning system.
- April 7, 2015: US Patent 9001743: method and system for increasing the accuracy of a WLAN based position estimate using cellular positioning information.
- April 14, 2015: US Patent 9008690: Method for estimating the speed and bearing of a Wi-Fi enabled device using WLAN radio signals in a WLAN based location service.
- April 21, 2015: US Patent 9013350: Systems and methods for using a satellite positioning system to detect moved WLAN access points.
- April 21, 2015: US Patent 9014715: Systems for and methods of determining likelihood of atypical transmission characteristics of reference points in a positioning system.
- May 12, 2015: US Patent 9031580: Method of estimating demographic information associated with a selected tile based on tracks of a plurality of mobile device users.
- May 19, 2015: US Patent 9037162: Methods and systems of continuously optimizing data in Wi-Fi positioning systems.

See also:

`http://www.skyhookwireless.com/about-us/patents`

Chapter 5

Limited Use License

Chapter 6

Location Settings

Starting from version 4.9.2 WPS API will respect the system-wide location permission setting (applies only to Windows 8 and OS X 10.7 onwards). If location services are disabled, WPS API will return WPS_ERROR_LOCATION_SETTING_DISABLED in WPS_location, WPS_periodic_location, WPS_-certified_location and WPS_tune_location calls.

See more information on your platform below.

6.1 Windows 8 or later (Desktop Mode)

The system-wide location setting on Windows can be found in *Control Panel - Location Settings*. It is enabled by default. Check out **Administrator settings** section here.

If you receive WPS_ERROR_LOCATION_SETTING_DISABLED from WPS API in your application, that means the option *Turn on the Windows Location platform* is unchecked by the user.

You might want to handle this by notifying the user that the location platform is currently disabled in the system and hence the application will not be able to determine location. You can also navigate the user to the *Location Settings* page directly so the user may re-enable the location setting back.
Below is a sample code snippet:

```
if (MessageBox(NULL,
               L"Location is disabled in system. Do you want to open Control Pane
      l to enable it?",
               L"My Application",
               MB_YESNO | MB_ICONQUESTION) == IDYES)
{
    ShellExecute(NULL, NULL, L"control", L"-name Microsoft.LocationSettings", NUL
      L, SW_SHOW);
}
```

You can check here on more ways to pop up Control Panel items programmatically.

Note:

Windows 8 may show a prompt to enable location services the very first time when location is requested via Location API (check the **Dialogues for enabling location** section here). WPS API will never cause the system to show any kind of prompts.

6.2 OS X 10.7 or later

OS X stores location settings under *System Preferences - Security & Privacy - Privacy - Location Services*. In addition to the global setting (*Enable Location Services*) it also lists applications permitted to determine location. WPS API will not require the application to be listed and will grant permission whenever just the global setting is enabled.

If you receive WPS_ERROR_LOCATION_SETTING_DISABLED from WPS API, that means that the global setting is disabled by the user. You might want to handle this by notifying the user that the location services are currently disabled in the system and hence the application will not be able to determine location. You can also navigate the user to the location settings page directly so the user may re-enable the location services back.
Below is a sample Apple script:

```
tell application "System Preferences"
    set securityPane to pane id "com.apple.preference.security"
    tell securityPane to reveal anchor "Privacy_LocationServices"
    activate
end tell
```

You can save it to a file (e.g. `locationsettings.scpt`) and run:

```
osascript locationsettings.scpt
```

If you want to implement it in Objective C, refer to the `Scripting Bridge documentation`.

Note:

OS X will show a prompt to grant location permission to the application the first time it attempts to determine location (see `CLLocationManager API documentation`). WPS API will never cause the system to show any kind of prompts.

Chapter 7

Deprecated List

Global WPS_ERROR_SCANNER_NOT_FOUND This error code is no longer relevant.

Global WPS_register_user(const WPS_SimpleAuthentication ∗authentication, const WPS_SimpleAuthentication ∗new_
Replaced by `WPS_set_key()`

Global WPS_set_registration_user(const WPS_SimpleAuthentication ∗authentication) Replaced by `WPS_set_key()`

Global WPS_set_tier2_area(const char ∗dirpath, unsigned size) This call is no longer relevant.

Global WPS_tiling(const char ∗dirpath, unsigned maxDataSizePerSession, unsigned maxDataSizeTotal, WPS_TilingCa
Replaced by `WPS_set_tiling()`

Chapter 8

Data Structure Documentation

8.1 WPS_GeoFence Struct Reference

```
#include <wpsapi.h>
```

Data Fields

- unsigned size
- unsigned radius
- WPS_GeoFenceType type
- unsigned long period

  - double latitude
  - double longitude

8.1.1 Detailed Description

Geofence.

Since:

4.4

8.1.2 Field Documentation

8.1.2.1 double WPS_GeoFence::latitude

The center of the geofence zone

8.1.2.2 double WPS_GeoFence::longitude

The center of the geofence zone

8.1.2.3 unsigned long WPS_GeoFence::period

The desired Wi-Fi period when the device is outside a `WPS_GEOFENCE_ENTER` geofence, or is inside a `WPS_GEOFENCE_LEAVE` geofence.

A value of `0` can be used if no preferred period is desired (for this geofence).

8.1.2.4 unsigned WPS_GeoFence::radius

The radius of the geofence zone, in meters

8.1.2.5 unsigned WPS_GeoFence::size

Store the size of `WPS_GeoFence`, and must be initialized to `sizeof(WPS_GeoFence)`.

Used as a versioning mechanism.

Sample Code:

```
WPS_GeoFence geofence;
geofence.size = sizeof(WPS_GeoFence);
```

8.1.2.6 WPS_GeoFenceType WPS_GeoFence::type

The type of the geofence

The documentation for this struct was generated from the following file:

- include/api/wpsapi.h

8.2 WPS_IPLocation Struct Reference

`#include <wpsapi.h>`

Data Fields

- char ∗ ip
- WPS_StreetAddress ∗ street_address

  - double latitude
  - double longitude

8.2.1 Detailed Description

Geographic location based on the IP address of the client making the request.

Note:

This information is likely not accurate, but may give some indication as to the general location of the request and may provide some hints for the client software to act and react appropriately.

Examples:

wpsapitest.cpp.

8.2.2 Field Documentation

8.2.2.1 char∗ WPS_IPLocation::ip

the IP address of the client as received by the server.

Examples:

wpsapitest.cpp.

8.2.2.2 double WPS_IPLocation::latitude

the estimated physical geographic location.

Examples:

wpsapitest.cpp.

8.2.2.3 double WPS_IPLocation::longitude

the estimated physical geographic location.

Examples:

wpsapitest.cpp.

8.2.2.4 WPS_StreetAddress* WPS_IPLocation::street_address

physical street address, only returned when the `street_address_lookup` parameter is set to `WPS_-LIMITED_STREET_ADDRESS_LOOKUP` or `WPS_FULL_STREET_ADDRESS_LOOKUP`.

Examples:

wpsapitest.cpp.

The documentation for this struct was generated from the following file:

- include/api/wpsapi.h

8.3 WPS_Location Struct Reference

```
#include <wpsapi.h>
```

Data Fields

- double hpe
- double score
- unsigned short nap
- double speed
- double bearing
- WPS_StreetAddress ∗ street_address
- unsigned short ncell
- unsigned short nlac
- unsigned short nsat
- unsigned short historicalLocationCount
- unsigned char withIP
- unsigned char hasScore
- unsigned id
- char ∗ ip
- unsigned char ∗ hash
- unsigned hashLength
- unsigned char ∗ signature
- unsigned signatureLength
- unsigned long timestamp
- double altitude
- WPS_LocationType type
- unsigned long age

  - double latitude
  - double longitude

8.3.1 Detailed Description

Geographic location

Examples:

wpsapitest.cpp.

8.3.2 Field Documentation

8.3.2.1 unsigned long WPS_Location::age

Number of milliseconds elapsed since the time the location was calculated

Since:

3.5

Examples:

wpsapitest.cpp.

8.3.2.2 double WPS_Location::altitude

A calculated altitude above mean sea level in meters.

Since:

3.2

8.3.2.3 double WPS_Location::bearing

A calculated estimate of bearing as degree from north clockwise (+90 is East).

A negative value is used to indicate an unknown bearing.

Since:

2.4

Examples:

wpsapitest.cpp.

8.3.2.4 unsigned char* WPS_Location::hash

Hash of the measurement data used in this certified location or NULL otherwise.

Note:

only applies to signed certified locations.

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.5 unsigned WPS_Location::hashLength

Length of `hash`.

Note:

only applies to signed certified locations.

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.6 unsigned char WPS_Location::hasScore

Nonzero if the score was calculated and zero otherwise.

Since:

4.8

Examples:

wpsapitest.cpp.

8.3.2.7 unsigned short WPS_Location::historicalLocationCount

The number of previous locations used to corroborate this location.

Since:

4.8

Examples:

wpsapitest.cpp.

8.3.2.8 double WPS_Location::hpe

horizontal positioning error -- A calculated error estimate of the location result in meters.

Examples:

wpsapitest.cpp.

8.3.2.9 unsigned WPS_Location::id

Identifies this location within a list of returned signed certified locations.

Note:

only applies to signed certified locations.

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.10 char∗ WPS_Location::ip

IP Address as received by Skyhook's servers or NULL otherwise. The presence of an IP address does not imply that `withIP` is `true`.

Note:

only applies to certified locations, ie. `hasScore` is true.

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.11 double WPS_Location::latitude

the calculated physical geographic location.

Examples:

wpsapitest.cpp.

8.3.2.12 double WPS_Location::longitude

the calculated physical geographic location.

Examples:

wpsapitest.cpp.

8.3.2.13 unsigned short WPS_Location::nap

The number of access-point used to calculate this location.

Since:

2.4

Examples:

wpsapitest.cpp.

8.3.2.14 unsigned short WPS_Location::ncell

The number of cell tower used to calculate this location.

Since:

3.2

Examples:

wpsapitest.cpp.

8.3.2.15 unsigned short WPS_Location::nlac

The number of unique location area codes used to calculate this location.

Since:

4.7

Examples:

wpsapitest.cpp.

8.3.2.16 unsigned short WPS_Location::nsat

The number of satellite used to calculate this location.

Since:

3.2

Examples:

wpsapitest.cpp.

8.3.2.17 double WPS_Location::score

The location score -- a number between 0 (very low score, highly suspect location) and 10 (very high score, highly trusted location).

Note:

only applies to certified locations, ie. `hasScore` is `true`.

Since:

4.8

Examples:

wpsapitest.cpp.

8.3.2.18 unsigned char∗ WPS_Location::signature

Signature computed by Skyhook's remote servers over the contents of this location's fields or `NULL` otherwise.

Signature of the WPS_Location serialized as follows:

```
void
WPS_CertifiedLocationCallback(void* arg,
                              WPS_ReturnCode code,
                              const WPS_Location** locations,
                              unsigned nlocations,
                              const void* reserved)
```

```
{
    for (int i = 0; i < nlocations; i++)
    {
        char buf[256];
        int nBytes =
            snprintf(buf,
                     sizeof(buf),
                     "%d/%d,%.6f,%.6f,%d,%.1f,%d,%d,%d,%d,%d,%d,%d,%ul,%s",
                     location[i]->id,
                     nlocations,
                     location[i]->latitude,
                     location[i]->longitude,
                     location[i]->hpe,
                     location[i]->score,
                     location[i]->nap,
                     location[i]->ncell,
                     location[i]->nlac,
                     location[i]->nsat,
                     location[i]->historicalLocationCount,
                     location[i]->withIP ? 1 : 0,
                     location[i]->timestamp,
                     location[i]->ip);
        memcpy(bug + nBytes, location[i]->hash, location[i]->hashLength);
        memcpy(buf + nBytes + saltLen, salt, saltLen);
        ...
    }
}
```

Skyhook public key:

```
-----BEGIN CERTIFICATE-----
MIIDfDCCAmQCCQDs5/f7i+4vIjANBgkqhkiG9w0BAQUFADB/MQswCQYDVQQGEwJV
UzELMAkGA1UECAwCTUExDzANBgNVBAcMBkJvc3RvbjEeMBwGA1UECgwVU2t5aG9v
ayBXaXJlbGVzcyBJbmMuMRAwDgYDVQQLDAdTa3lob29rMSAwHgYDVQQDDBdhcGku
c2t5aG9va3dpcmVsZXNzLmNvbTAgFw0xNDAyMDMxOTA2NDBaGA8zMDEzMDYwNjE5
MDY0MFowfzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAk1BMQ8wDQYDVQQHDAZCb3N0
b24xHjAcBgNVBAoMFVNreWhvb2sgV2lyZWxlc3MgSW5jLjEQMA4GA1UECwwHU2t5
aG9vazEgMB4GA1UEAwwXYXBpLnNreWhvb2t3aXJlbGVzcy5jb20wggEiMA0GCSqG
SIb3DQEBAQUAA4IBDwAwggEKAoIBAQC/N4Y/JXq5x1T3F5Q1pViI/b6sHPhrU3bd
KNLSMkUb4ZPmwyde5ayPLSCAnS891Cmdn+MFolPNVlnu4/odDlIJNkBZxwqLyJjj
UagIoU6AyRY0gWZh24abMjm8BeKC+8CFFW9DEL/3rZXSLAm+ViVVmEV0+lA2M9+D
6i/L5Y47DkjsuU5B911uNzhOXS5qaW6WQOFBNtIMK8dEbFhfwMApxQOv4Pw998N7
v+ylrDhUbeUOlW/mSXi5UEEVn1Q4cmiyiIlDejljIfCgQecPttEAmotNThwh8nCo
KA6T0hdu6FL2bJmTrVbnz3ftOpuXLhCRTXMrsIICVizVijfxrhsBAgMBAAEwDQYJ
KoZIhvcNAQEFBQADggEBABfwvRIyVm9UrBiT18Xj2WMJwC3hSvoGvxOVAfYFb0gO
ciCwkMhhs+88F3CgQbXIILr7iaHoNZcH9zDmm8rZ2a8BMTVSkaF9b43xILTVwGmu
zi36eWMcvCS1jE9cRhSYhpX6I+scRLC6IRshvuleTX9viE8LwASLvekKEFi8yg07
8cFkJDRHXrqvYsG1hcDoKsW9ocr4ugj0annTUh38OMgAzVO0L2IFqKkb6+CuDPX7
jJh+KwZau/0K6hqr+4aB6zwLinQ32nNJnI1UI3ufFbPU9g4V9nzXyHBDF93zaPWF
R7IQB1FfB8DBgYeQUL9VJKSYUhogQUpSeqmQrwXbrgI=
-----END CERTIFICATE-----
```

See also:

https://api.skyhookwireless.com/wps2/pubkey/skyhook-rsa-x509.cert

Note:

only applies to signed certified locations.

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.19 unsigned WPS_Location::signatureLength

Length of `signature`.

Note:

only applies to signed certified locations.

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.20 double WPS_Location::speed

A calculated estimate of speed in km/hr.

A negative value is used to indicate an unknown speed.

Since:

2.4

Examples:

wpsapitest.cpp.

8.3.2.21 WPS_StreetAddress* WPS_Location::street_address

physical street address, only returned in the network-centric model when the `street_address_-lookup` parameter is set to `WPS_LIMITED_STREET_ADDRESS_LOOKUP` or `WPS_FULL_STREET_-ADDRESS_LOOKUP`.

Examples:

wpsapitest.cpp.

8.3.2.22 unsigned long WPS_Location::timestamp

Timestamp (number of seconds since unix epoch) of this location. This timestamp is calculated on Skyhook's servers at the time the request is received, and does not account for the network round-trip or the local time of the device.

Note:

only applies to certified locations, ie. `hasScore` is true

Since:

4.9.1

Examples:

wpsapitest.cpp.

8.3.2.23 WPS_LocationType WPS_Location::type

Type of calculated location.

Since:

3.2

8.3.2.24 unsigned char WPS_Location::withIP

Nonzero if the IP address of the client making the request was used to confirm the location, or zero otherwise.

Since:

4.8

Examples:

wpsapitest.cpp.

The documentation for this struct was generated from the following file:

- include/api/wpsapi.h

8.4 WPS_NameCode Struct Reference

```
#include <wpsapi.h>
```

Data Fields

- char ∗ name
- char code [3]

8.4.1 Detailed Description

Name and code

8.4.2 Field Documentation

8.4.2.1 char WPS_NameCode::code[3]

2-letter code

Examples:

wpsapitest.cpp.

8.4.2.2 char∗ WPS_NameCode::name

name

The documentation for this struct was generated from the following file:

- include/api/wpsapi.h

8.5 WPS_SimpleAuthentication Struct Reference

```
#include <wpsapi.h>
```

Data Fields

- const char ∗ username
- const char ∗ realm

8.5.1 Detailed Description

`WPS_SimpleAuthentication` is used to identify the user with the server.

Examples:

wpsapitest.cpp.

8.5.2 Field Documentation

8.5.2.1 const char∗ WPS_SimpleAuthentication::realm

the authentication realm.

Examples:

wpsapitest.cpp.

8.5.2.2 const char∗ WPS_SimpleAuthentication::username

the user's name, or unique identifier.

Examples:

wpsapitest.cpp.

The documentation for this struct was generated from the following file:

- include/api/wpsapi.h

8.6 WPS_StreetAddress Struct Reference

`#include <wpsapi.h>`

Data Fields

- char ∗ street_number
- char ∗∗ address_line
- char ∗ city
- char ∗ postal_code
- char ∗ county
- char ∗ province
- WPS_NameCode state
- char ∗ region
- WPS_NameCode country

8.6.1 Detailed Description

Street Address

Note:

Some map data ©`OpenStreetMap` contributors, `CC-BY-SA`.

Examples:

wpsapitest.cpp.

8.6.2 Field Documentation

8.6.2.1 char∗∗ WPS_StreetAddress::address_line

A `NULL` terminated array of address line

Examples:

wpsapitest.cpp.

8.6.2.2 char∗ WPS_StreetAddress::city

city

Examples:

wpsapitest.cpp.

8.6.2.3 WPS_NameCode WPS_StreetAddress::country

country, includes country name and 2-letter code.

8.6.2.4 char∗ WPS_StreetAddress::county

county

8.6.2.5 char∗ WPS_StreetAddress::postal_code

postal code

Examples:

wpsapitest.cpp.

8.6.2.6 char∗ WPS_StreetAddress::province

province

8.6.2.7 char∗ WPS_StreetAddress::region

region

8.6.2.8 WPS_NameCode WPS_StreetAddress::state

state, includes state name and 2-letter code.

Examples:

wpsapitest.cpp.

8.6.2.9 char∗ WPS_StreetAddress::street_number

street number

Examples:

wpsapitest.cpp.

The documentation for this struct was generated from the following file:

- include/api/wpsapi.h

Chapter 9

File Documentation

9.1 doc/sdk/locationsettings.dox File Reference

9.2 include/api/wpsapi.h File Reference

Data Structures

- struct WPS_SimpleAuthentication
- struct WPS_NameCode
- struct WPS_StreetAddress
- struct WPS_Location
- struct WPS_IPLocation
- struct WPS_GeoFence

- enum WPS_ReturnCode {

  WPS_OK = 0, WPS_ERROR_SCANNER_NOT_FOUND = 1, WPS_ERROR_WIFI_NOT_-AVAILABLE = 2, WPS_ERROR_NO_WIFI_IN_RANGE = 3,

  WPS_ERROR_UNAUTHORIZED = 4, WPS_ERROR_SERVER_UNAVAILABLE = 5, WPS_ERROR_LOCATION_CANNOT_BE_DETERMINED = 6, WPS_ERROR_PROXY_-UNAUTHORIZED = 7,

  WPS_ERROR_FILE_IO = 8, WPS_ERROR_INVALID_FILE_FORMAT = 9, WPS_ERROR_-TIMEOUT = 10, WPS_NOT_APPLICABLE = 11,

  WPS_GEOFENCE_ERROR = 12, WPS_ERROR_NOT_TUNED = 13, WPS_ERROR_NOT_-SIGNED = 14, WPS_ERROR_LOCATION_SETTING_DISABLED = 15,

  WPS_NOMEM = 98, WPS_ERROR = 99 }
- enum WPS_Continuation { WPS_STOP = 0, WPS_CONTINUE = 1 }
- enum WPS_StreetAddressLookup { WPS_NO_STREET_ADDRESS_LOOKUP, WPS_-LIMITED_STREET_ADDRESS_LOOKUP, WPS_FULL_STREET_ADDRESS_LOOKUP }
- enum WPS_LocationType { WPS_LOCATION_TYPE_2D, WPS_LOCATION_TYPE_3D }
- enum WPS_GeoFenceType { WPS_GEOFENCE_ENTER, WPS_GEOFENCE_LEAVE, WPS_-GEOFENCE_INSIDE, WPS_GEOFENCE_OUTSIDE }
- typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_LocationCallback)(void ∗arg, WPS_-ReturnCode code, const WPS_Location ∗location, const void ∗reserved)
- typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_CertifiedLocationCallback)(void ∗arg, WPS_ReturnCode code, const WPS_Location ∗∗locations, unsigned nlocations, const void ∗reserved)
- typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_TilingCallback)(void ∗arg, unsigned tileNum-ber, unsigned tileTotal)
- typedef void ∗ WPS_GeoFenceHandle
- typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_GeoFenceCallback)(void ∗arg, const WPS_-GeoFence ∗geofence, const WPS_Location ∗location, const void ∗reserved)
- const char ∗WPSAPI_CALL WPS_version ()
- WPS_ReturnCode WPSAPI_CALL WPS_load ()
- void WPSAPI_CALL WPS_unload ()
- WPS_ReturnCode WPSAPI_CALL WPS_location (const WPS_SimpleAuthentication ∗authentication, WPS_StreetAddressLookup street_address_lookup, WPS_Location ∗∗location)
- WPS_ReturnCode WPSAPI_CALL WPS_periodic_location (const WPS_SimpleAuthentication ∗authentication, WPS_StreetAddressLookup street_address_lookup, unsigned long period, unsigned iterations, WPS_LocationCallback callback, void ∗arg)
- WPS_ReturnCode WPSAPI_CALL WPS_certified_location (const WPS_SimpleAuthentication ∗authentication, WPS_StreetAddressLookup street_address_lookup, WPS_-CertifiedLocationCallback callback, void ∗arg)

- WPS_ReturnCode WPSAPI_CALL WPS_signed_certified_location (const WPS_-SimpleAuthentication ∗authentication, WPS_StreetAddressLookup street_address_lookup, const unsigned char ∗salt, unsigned saltLen, WPS_CertifiedLocationCallback callback, void ∗arg)
- WPS_ReturnCode WPSAPI_CALL WPS_ip_location (const WPS_SimpleAuthentication ∗authentication, WPS_StreetAddressLookup street_address_lookup, WPS_IPLocation ∗∗location)
- WPS_ReturnCode WPSAPI_CALL WPS_offline_token (const WPS_SimpleAuthentication ∗authentication, const unsigned char ∗key, unsigned key_length, unsigned char ∗∗token, unsigned ∗token_size)
- WPS_ReturnCode WPSAPI_CALL WPS_offline_location (const WPS_SimpleAuthentication ∗authentication, const unsigned char ∗key, unsigned key_length, const unsigned char ∗token, unsigned token_size, WPS_Location ∗∗location)
- void WPSAPI_CALL WPS_free_offline_token (unsigned char ∗token)
- void WPSAPI_CALL WPS_free_location (WPS_Location ∗)
- void WPSAPI_CALL WPS_free_ip_location (WPS_IPLocation ∗)
- void WPSAPI_CALL WPS_set_proxy (const char ∗address, int port, const char ∗user, const char ∗password)
- void WPSAPI_CALL WPS_set_server_url (const char ∗url)
- void WPSAPI_CALL WPS_set_user_agent (const char ∗ua)
- WPS_ReturnCode WPSAPI_CALL WPS_set_local_files_path (const char ∗∗paths)
- WPS_ReturnCode WPSAPI_CALL WPS_set_tier2_area (const char ∗dirpath, unsigned size)
- WPS_ReturnCode WPSAPI_CALL WPS_set_tiling (const WPS_SimpleAuthentication ∗authentication, const char ∗dirpath, unsigned maxDataSizePerSession, unsigned maxDataSizeTotal, WPS_TilingCallback callback, void ∗arg)
- WPS_ReturnCode WPSAPI_CALL WPS_tiling (const char ∗dirpath, unsigned maxDataSizePerSession, unsigned maxDataSizeTotal, WPS_TilingCallback callback, void ∗arg)
- WPS_ReturnCode WPSAPI_CALL WPS_set_key (const char ∗key)
- WPS_ReturnCode WPSAPI_CALL WPS_set_registration_user (const WPS_SimpleAuthentication ∗authentication)
- WPS_ReturnCode WPSAPI_CALL WPS_register_user (const WPS_SimpleAuthentication ∗authentication, const WPS_SimpleAuthentication ∗new_authentication)
- WPS_ReturnCode WPSAPI_CALL WPS_tune_location (const WPS_SimpleAuthentication ∗authentication, const WPS_Location ∗location)
- WPS_ReturnCode WPSAPI_CALL WPS_geofence_set (const WPS_GeoFence ∗geofence, WPS_-GeoFenceCallback callback, void ∗arg, WPS_GeoFenceHandle ∗handle)
- WPS_ReturnCode WPSAPI_CALL WPS_geofence_cancel (WPS_GeoFenceHandle handle)
- WPS_ReturnCode WPSAPI_CALL WPS_geofence_cancel_all ()

9.2.1 Typedef Documentation

9.2.1.1 typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_CertifiedLocationCallback)(void ∗arg, WPS_ReturnCode code, const WPS_Location ∗∗locations, unsigned nlocations, const void ∗reserved)

Callback routine for `WPS_certified_location()`.

Parameters:

arg the `arg` passed to `WPS_certified_location()`.

code the `WPS_ReturnCode` of the last request.

locations If `code` is `WPS_OK` points to an array of pointers to `WPS_Location`
This pointer does *not* need to be freed.

locationsSize size of locations array

Returns:

`WPS_CONTINUE` if `WPS_certified_location()` should continue,
`WPS_STOP` if `WPS_certified_location()` should stop.

Since:

4.8

9.2.1.2 typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_GeoFenceCallback)(void ∗arg, const WPS_GeoFence ∗geofence, const WPS_Location ∗location, const void ∗reserved)

Callback routine for `WPS_geofence_set()`.

For a `WPS_GEOFENCE_ENTER` geofence, this callback will be called whenever the user enters the geofence (but only once for as long as the user remains inside the geofence). For a `WPS_GEOFENCE_LEAVE` geofence, this callback will be called whenever the user leaves the geofence (but only once for as long as the user remains outside the geofence).

Parameters:

arg the `arg` passed to `WPS_geofence_set()`.

geofence the geofence that triggered this callback.

location the actual calculated location.

Returns:

`WPS_CONTINUE` if `WPS_periodic_location()` should continue,
`WPS_STOP` if `WPS_periodic_location()` should stop (and no more callback will be invoked).

Since:

4.4

Sample Code:

```
WPS_Continuation
geofence_callback(void*,
                  const WPS_GeoFence* geofence,
                  const WPS_Location* location,
                  const void*)
{
    if (geofence->type == WPS_GEOFENCE_ENTER)
    {
        printf("entering");
    }
    else // geofence->type == WPS_GEOFENCE_LEAVE
    {
        printf("leaving");
    }
    printf(" geofence (%.6f, %.6f +/-%um) at %.6f, %.6f +/-%.0fm\n",
           geofence->latitude,
           geofence->longitude,
           geofence->radius,
           location->latitude,
           location->longitude,
           location->hpe);

    return WPS_CONTINUE;
}
```

9.2.1.3 typedef void∗ WPS_GeoFenceHandle

Geofence handle.

Since:

4.4

9.2.1.4 typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_LocationCallback)(void ∗arg, WPS_ReturnCode code, const WPS_Location ∗location, const void ∗reserved)

Callback routine for `WPS_periodic_location()`.

Parameters:

arg the `arg` passed to `WPS_periodic_location()`.

code the `WPS_ReturnCode` of the last request.

location If `code` is `WPS_OK` points to a `WPS_Location`
This pointer does *not* need to be freed.

Returns:

`WPS_CONTINUE` if `WPS_periodic_location()` should continue,
`WPS_STOP` if `WPS_periodic_location()` should stop.

Since:

2.4

Sample Code:

```
WPS_Continuation
periodic_callback(void*,
                  WPS_ReturnCode code,
                  const WPS_Location* location,
                  const void*)
{
    if (code != WPS_OK)
    {
        fprintf(stderr, "*** failure (%d)!\n", code);
    }
    else
    {
        print_location(location);
    }

    return WPS_CONTINUE;
}
```

9.2.1.5 typedef WPS_Continuation(WPSAPI_CALL ∗ WPS_TilingCallback)(void ∗arg, unsigned tileNumber, unsigned tileTotal)

Callback routine for `WPS_set_tiling()`.

Parameters:

arg the `arg` passed to `WPS_set_tiling()`.

tileNumber the number of the tile to be downloaded next.

tileTotal the total number of tiles to be downlaoded (in this session).

Note:

`tileNumber` ranges from 0 to `tileTotal` (not inclusive), but if a tile has already been downloaded it will be skipped and the callback will not be called for that tile.

Returns:

`WPS_CONTINUE` if `WPS` should continue downloading tiles,
`WPS_STOP` if `WPS` should stop downloading tiles.

Since:

2.6

Sample Code:

```
WPS_Continuation
tiling_callback(void*,
                unsigned tileNumber,
                unsigned tileTotal)
{
    printf("downloading tile %d/%d...\n", tileNumber, tileTotal);
    return WPS_CONTINUE;
}
```

9.2.2 Enumeration Type Documentation

9.2.2.1 enum WPS_Continuation

`WPS` continuation.

Since:

2.6

Enumerator:

WPS_STOP

WPS_CONTINUE

9.2.2.2 enum WPS_GeoFenceType

Geofence type.

Since:

4.4

Enumerator:

WPS_GEOFENCE_ENTER identify a geofence that triggers when the user enters the defined zone

WPS_GEOFENCE_LEAVE identify a geofence that triggers when the user leaves the defined zone

WPS_GEOFENCE_INSIDE identify a geofence that triggers when the user's location starts within or enters the defined zone

Since:

4.8

WPS_GEOFENCE_OUTSIDE identify a geofence that triggers when the user's location starts outside of or leaves the defined zone

Since:

4.8

9.2.2.3 enum WPS_LocationType

Calculated location type.

Since:

3.2

Enumerator:

WPS_LOCATION_TYPE_2D

WPS_LOCATION_TYPE_3D

9.2.2.4 enum WPS_ReturnCode

WPS return codes.

Enumerator:

WPS_OK The call was successful.

WPS_ERROR_SCANNER_NOT_FOUND The `WPSScanner.dll` was not found.

Deprecated

This error code is no longer relevant.

WPS_ERROR_WIFI_NOT_AVAILABLE No Wi-Fi adapter was detected.

WPS_ERROR_NO_WIFI_IN_RANGE No Wi-Fi reference points in range.

WPS_ERROR_UNAUTHORIZED User authentication failed.

WPS_ERROR_SERVER_UNAVAILABLE The server is unavailable.

WPS_ERROR_LOCATION_CANNOT_BE_DETERMINED A location couldn't be determined.

WPS_ERROR_PROXY_UNAUTHORIZED Proxy authentication failed.

WPS_ERROR_FILE_IO A file IO error occurred while reading the local file.

Since:

2.4

WPS_ERROR_INVALID_FILE_FORMAT The local file has an invalid format.

Since:

2.4

WPS_ERROR_TIMEOUT Network operation timed out.

Since:

3.0

WPS_NOT_APPLICABLE Call cannot be completed at this time.

Since:

3.3

WPS_GEOFENCE_ERROR An error occurred when processing a geofence.

See also:

`WPS_geofence_set()`
`WPS_geofence_cancel()`

Since:

4.4

WPS_ERROR_NOT_TUNED Could not tune the location.

Since:

4.7

WPS_ERROR_NOT_SIGNED Could not sign the certified location.

Since:

4.9.1

WPS_ERROR_LOCATION_SETTING_DISABLED Location services are disabled in the system.

Since:

4.9.2

WPS_NOMEM Cannot allocate memory.

Since:

2.6.1

WPS_ERROR Some other error occurred.

9.2.2.5 enum WPS_StreetAddressLookup

Street address lookup.

Note:

The server returns as much information as requested, but is not required to fill in all the requested fields.
Only the fields the server could reverse geocode are returned.

Enumerator:

WPS_NO_STREET_ADDRESS_LOOKUP no street address lookup is performed.

WPS_LIMITED_STREET_ADDRESS_LOOKUP a limited address lookup is performed to return, at most, city information.

WPS_FULL_STREET_ADDRESS_LOOKUP a full street address lookup is performed returning the most specific street address.

9.2.3 Function Documentation

9.2.3.1 WPS_ReturnCode WPSAPI_CALL WPS_certified_location (const WPS_SimpleAuthentication ∗ *authentication*, WPS_StreetAddressLookup *street_address_lookup*, WPS_CertifiedLocationCallback *callback*, void ∗ *arg*)

Requests certified periodic geographic location based on observed Wi-Fi access points, cell towers, and GPS signals.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_-registration_user()`, otherwise set to the user's authentication information.

street_address_lookup request street address lookup in addition to latitude/longitude lookup
Note that street address lookup is only performed when the location is determined by the remote server (network-centric model), not when the location is determined locally.

callback the callback routine to report locations to.

arg an opaque parameter passed to the callback routine.

Returns:

a `WPS_ReturnCode`

Note:

This call is blocking -- it will not return until a- an error occurs while setting up, or b- the number of iterations have been processed, or c- the `callback` returns `WPS_STOP`.

Since:

4.8

Examples:

wpsapitest.cpp.

9.2.3.2 void WPSAPI_CALL WPS_free_ip_location (WPS_IPLocation ∗)

Free a `WPS_IPLocation` struct returned by `WPS_ip_location()`.

Examples:

wpsapitest.cpp.

9.2.3.3 void WPSAPI_CALL WPS_free_location (WPS_Location ∗)

Free a `WPS_Location` struct returned by `WPS_location()`.

Examples:

wpsapitest.cpp.

9.2.3.4 void WPSAPI_CALL WPS_free_offline_token (unsigned char $*$ *token*)

Free a token returned by `WPS_offline_token()`.

Since:

4.5

9.2.3.5 WPS_ReturnCode WPSAPI_CALL WPS_geofence_cancel (WPS_GeoFenceHandle *handle*)

Cancel a geofence.

Parameters:

handle the handle of the geofence to cancel.

Returns:

a `WPS_ReturnCode`. `WPS_GEOFENCE_ERROR` indicates the geofence for this handle isn't defined.

See also:

`WPS_geofence_set()`

Since:

4.4

9.2.3.6 WPS_ReturnCode WPSAPI_CALL WPS_geofence_cancel_all ()

Cancel all geofences.

Returns:

a `WPS_ReturnCode`

See also:

`WPS_geofence_set()`

Since:

4.4

9.2.3.7 WPS_ReturnCode WPSAPI_CALL WPS_geofence_set (const WPS_GeoFence $*$ *geofence*, WPS_GeoFenceCallback *callback*, void $*$ *arg*, WPS_GeoFenceHandle $*$ *handle*)

Define a new geofence.

Parameters:

geofence the geofence to monitor.

callback the `WPS_GeoFenceCallback` to invoke when this geofence is triggered.

arg an opaque parameter passed to the `WPS_GeoFenceCallback`.

handle receives `WPS_GeoFenceHandle` which can be used to cancel this geofence.

Returns:

a `WPS_ReturnCode`.

See also:

WPS_geofence_cancel()

Since:

4.4

Sample Code:

```
WPS_GeoFence geofence;
WPS_GeoFenceHandle handle;
WPS_ReturnCode rc;

geofence.size = sizeof(WPS_GeoFence);
geofence.latitude = 42.1234;
geofence.longitude = -71.5678;
geofence.radius = 100;
geofence.type = WPS_GEOFENCE_ENTER;
geofence.period = 0;

rc = WPS_geofence_set(&geofence,
                      geofence_callback,
                      NULL,
                      &handle);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_geofence_set failed (%d)!\n", rc);
}

rc = WPS_periodic_location(NULL,
                           WPS_NO_STREET_ADDRESS_LOOKUP,
                           60 * 60 * 1000,
                           0,
                           NULL,
                           NULL);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_periodic_location failed (%d)!\n", rc);
}

rc = WPS_geofence_cancel(handle);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_geofence_cancel failed (%d)!\n", rc);
}

rc = WPS_geofence_cancel_all();
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_geofence_cancel_all failed (%d)!\n", rc);
}
```

9.2.3.8 WPS_ReturnCode WPSAPI_CALL WPS_ip_location (const WPS_SimpleAuthentication ∗ *authentication*, WPS_StreetAddressLookup *street_address_lookup*, WPS_IPLocation ∗∗ *location*)

Request geographic location information based on the IP address of the client making the request.

Note:

This information is likely not accurate, but may give some indication as to the general location of the request and may provide some hints for the client software to act and react appropriately.

`WPS_ip_location()` requires network connectivity.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_registration_user()`, otherwise set to the user's authentication information.

street_address_lookup request street address lookup in addition to lat/long lookup.

location pointer to return a `WPS_IPLocation` struct.
This pointer must be freed by calling `WPS_free_ip_location()`

Returns:

a `WPS_ReturnCode`

Sample Code:

```
WPS_IPLocation* location;
WPS_ReturnCode rc;

rc = WPS_ip_location(NULL,
                     WPS_NO_STREET_ADDRESS_LOOKUP,
                     &location);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_ip_location failed (%d)!\n", rc);
}
else
{
    print_ip_location(location);
    WPS_free_ip_location(location);
}
```

Examples:

wpsapitest.cpp.

9.2.3.9 WPS_ReturnCode WPSAPI_CALL WPS_load ()

Initializes WPS API. Should be called before making other WPS API calls.

Since:

4.4

Note:

This call is not mandatory. WPS will be initialized automatically upon the first API call.

Examples:

wpsapitest.cpp.

9.2.3.10 WPS_ReturnCode WPSAPI_CALL WPS_location (const WPS_SimpleAuthentication ∗ *authentication*, WPS_StreetAddressLookup *street_address_lookup*, WPS_Location ∗∗ *location*)

Requests geographic location based on observed Wi-Fi access points, cell towers, and GPS signals.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_registration_user()`, otherwise set to the user's authentication information.

street_address_lookup request street address lookup in addition to latitude/longitude lookup.

location pointer to return a `WPS_Location` struct.
This pointer must be freed by calling `WPS_free_location()`.

Returns:

a `WPS_ReturnCode`

Sample Code:

```
WPS_Location* location;
WPS_ReturnCode rc;

rc = WPS_location(NULL,
                  WPS_NO_STREET_ADDRESS_LOOKUP,
                  &location);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_location failed (%d)!\n", rc);
}
else
{
    print_location(location);
    WPS_free_location(location);
}
```

Examples:

wpsapitest.cpp.

9.2.3.11 WPS_ReturnCode WPSAPI_CALL WPS_offline_location (const WPS_SimpleAuthentication ∗ *authentication*, const unsigned char ∗ *key*, unsigned *key_length*, const unsigned char ∗ *token*, unsigned *token_size*, WPS_Location ∗∗ *location*)

Request a geographic location from a token collected at a different time.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_registration_user()`, otherwise set to the user's authentication information.

key a key used to encrypt the `token`.

key_length the length of the key used to encrypt the `token`

token the token to be converted to a location

token_size the size of the `token`

location pointer to return a `WPS_Location` struct.
This pointer should be freed by calling `WPS_free_location()`

Returns:

a `WPS_ReturnCode`

Note:

A different `username` can be used to replay a token than the `username` used to collect the token, however both users must belong to the same `realm`.

Offline tokens are only valid for 90 days after they are generated. Attempting to redeem a token more than 90 days old will result in an error.

The same key used to encrypt the token in `WPS_offline_token()` must be provided to `WPS_offline_location()`.

Although a token can be collected when a location has been calculated `WPS_offline_location()` is not guaranteed to return a location with that same token.

See also:

`WPS_offline_token()`

Since:

4.5

Sample Code:

```
WPS_Location* location;
WPS_ReturnCode rc;
unsigned char key[16];
unsigned char* token;
unsigned token_size;

// restore the key and token stored earlier
// ...

rc = WPS_offline_location(NULL,
                          key,
                          sizeof(key),
                          token,
                          token_size,
                          &location);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_offline_location failed (%d)!\n", rc);
}
else
{
    print_location(location);
    WPS_free_location(location);
}
```

9.2.3.12 WPS_ReturnCode WPSAPI_CALL WPS_offline_token (const WPS_-SimpleAuthentication ∗ *authentication*, const unsigned char ∗ *key*, unsigned *key_length*, unsigned char ∗∗ *token*, unsigned ∗ *token_size*)

Retrieve a secure offline token based on the latest observed Wi-Fi access points and cell towers. This token can be saved and converted to a location later by calling `WPS_offline_location()`.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_-registration_user()`, otherwise set to the user's authentication information.

key a key used to encrypt the token.
The longer the key the more effective the encryption. It is recommended to have a key of at least 16 bytes.

key_length the length of the key.

token a pointer to receive the token.
This pointer must be freed by calling `WPS_free_offline_token()`.

token_size a pointer to receive the size of `token`.

Returns:

a `WPS_ReturnCode`

Note:

A sensible time to call `WPS_offline_token()` is after either `WPS_location()` failed with `WPS_ERROR_SERVER_UNAVAILABLE` or `WPS_ERROR_LOCATION_CANNOT_BE_-DETERMINED`, or from the `WPS_periodic_location()` callback again with a `WPS_ERROR_-SERVER_UNAVAILABLE` or `WPS_ERROR_LOCATION_CANNOT_BE_DETERMINED` error code.

Offline tokens are only valid for 90 days after they are generated. Attempting to redeem a token more than 90 days old will result in an error.

See also:

`WPS_offline_location()`
`WPS_free_offline_token()`

Since:

4.5

Sample Code:

```
WPS_ReturnCode rc;
unsigned char key[16];
unsigned char* token;
unsigned token_size;

// generate the key
// ...

rc = WPS_offline_token(NULL,
                       key,
                       sizeof(key),
                       &token,
                       &token_size);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_offline_token failed (%d)!\n", rc);
}
else
{
    // save the key and token in persistent storage
    // ...

    WPS_free_offline_token(token);
}
```

9.2.3.13 WPS_ReturnCode WPSAPI_CALL WPS_periodic_location (const WPS_SimpleAuthentication ∗ *authentication*, WPS_StreetAddressLookup *street_address_lookup*, unsigned long *period*, unsigned *iterations*, WPS_LocationCallback *callback*, void ∗ *arg*)

Requests periodic geographic location based on observed Wi-Fi access points, cell towers, and GPS signals.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_-registration_user()`, otherwise set to the user's authentication information.

street_address_lookup request street address lookup in addition to latitude/longitude lookup
Note that street address lookup is only performed when the location is determined by the remote server (network-centric model), not when the location is determined locally.

period maximum time in milliseconds between location reports.
WPS will invoke the callback at least every period, either with a location update or with an error if no location can be determined. WPS may report location updates more often than the user period if a new or better location becomes available within a given period.
This parameter is also used to specify the preferred Wi-Fi scan period, which WPS may override for optimal performance and power savings.

iterations number of times a location is to be reported.
A value of zero indicates an unlimited number of iterations.

callback the callback routine to report locations to.

arg an opaque parameter passed to the callback routine.

Returns:

a `WPS_ReturnCode`

Precondition:

`period` must be strictly greater than 0.

Note:

This call is blocking -- it will not return until a- an error occurs while setting up, or b- the number of iterations have been processed, or c- the `callback` returns `WPS_STOP`.

Since:

2.4

Sample Code:

```
WPS_ReturnCode rc;

rc = WPS_periodic_location(NULL,
                           WPS_NO_STREET_ADDRESS_LOOKUP,
                           1000,
                           0,
                           periodic_callback,
                           NULL);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_periodic_location failed (%d)!\n", rc);
}
```

Examples:

wpsapitest.cpp.

9.2.3.14 WPS_ReturnCode WPSAPI_CALL WPS_register_user (const WPS_-SimpleAuthentication ∗ *authentication*, const WPS_SimpleAuthentication ∗ *new_authentication*)

Register a new user.

Deprecated

Replaced by `WPS_set_key()`

Parameters:

authentication an existing user's authentication information.

new_authentication the new user's authentication information or `NULL` to trigger auto-registration.

Returns:

a `WPS_ReturnCode`

Since:

2.3.1

Sample Code:

New applications are encouraged to simply use the auto-registration feature by calling `WPS_-register_user()`, maybe during initialization of the application. For example:

```
// info from Skyhook
#define WPS_USERNAME "myusername"
#define WPS_REALM "myrealm"

WPS_SimpleAuthentication wpsUser;
WPS_ReturnCode rc;

void wpsInit()
{
    wpsUser.username = WPS_USERNAME;
    wpsUser.realm = WPS_REALM;

    while ((rc = WPS_register_user(&wpsUser, NULL)) != WPS_OK)
```

```
        {
             // registration failed... retry

            sleep(5 + rand() * 5); // some randomness
        }
    }
```

Applications that still want to control how usernames are generated can continue to do so as they did before since version 4.2 is backward compatible with 4.1. For example:

```
// info from Skyhook
#define WPS_USERNAME "myusername"
#define WPS_REALM "myrealm"

WPS_SimpleAuthentication wpsUser;

void saveUsername(const char*); // save registered username
const char* readUsername(); // read registered username, or NULL if not found

const char* generateUsername(); // your own generated username

void wpsInit()
{
    WPS_SimpleAuthentication wpsReg;
    WPS_ReturnCode rc;

    const char* savedUsername = readUsername(); // retrieve previously regist
  ered username
    const char* generatedUsername = generateUsername();

    wpsUser.username = generatedUsername;
    wpsUser.realm = WPS_REALM;

    if (savedUsername != NULL && strcmp(savedUsername, generatedUsername) ==
  0)
    {
        // user already registered (and match generated username)
        return;
    }

    wpsReg.username = WPS_USERNAME;
    wpsReg.realm = WPS_REALM;

    while ((rc = WPS_register_user(&wpsReg, &wpsUser) != WPS_OK)
    {
         // registration failed... retry

        sleep(5 + rand() * 5); // some randomness
    }

    saveUsername(generatedusername); // save registered username
}
```

Note that the generated username should be stable -- ie. remain constant over time, every time it is generated -- to avoid unneccesary re-registrations. Care should be taken when generating a username to protect the privacy of the user, and should not contain any personally identifiable information. This can be achieved, for example, by hashing a unique identifier of the device, like the IMEI, or the MAC, or some other hardware ID.

Once registration is complete `wpsUser` can be used in any `WPS_*` calls. For example:

```
WPS_Location* location;
WPS_ReturnCode rc = WPS_location(&wpsUser,
                                 WPS_NO_STREET_ADDRESS_LOOKUP,
                                 &location);
```

9.2.3.15 WPS_ReturnCode WPSAPI_CALL WPS_set_key (const char ∗ *key*)

Set the user's API key.

Parameters:

key an existing user's API key.

This is the preferred method to authenticate for new applications.

Make sure to call this method before making any other calls, such as during initialization of the application.

After this call you should pass `NULL` as `authentication` to other calls like `WPS_location`.

Note:

Must not be used in conjunction with `WPS_set_registration_user()`

Since:

4.9

Examples:

wpsapitest.cpp.

9.2.3.16 WPS_ReturnCode WPSAPI_CALL WPS_set_local_files_path (const char ∗∗ *paths*)

Set the path to local files so location determination can be performed locally.

Parameters:

paths an array (terminated by `NULL`) of complete path to local files.
This array replaces all previous values.
Use `NULL` to clear and release local files.

Returns:

a `WPS_ReturnCode`

Since:

2.4

Sample Code:

```
const char* paths[] = { "myfile1", "myfile2", NULL };
WPS_ReturnCode rc = WPS_set_local_files_path(paths);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_set_local_files_path failed (%d)!\n", rc);
}
```

Examples:

wpsapitest.cpp.

9.2.3.17 void WPSAPI_CALL WPS_set_proxy (const char * *address*, int *port*, const char * *user*, const char * *password*)

Setup a proxy server.

Parameters:

address	the internet address of the proxy server.
port	the TCP port number of the proxy server.
user	the username to authenticate with the proxy server.
password	the password to authentication with the proxy server.

Note:

`user` and `password` are optional and can be set to `NULL` for un-authenticated proxy servers.

Returns:

a `WPS_ReturnCode`

9.2.3.18 WPS_ReturnCode WPSAPI_CALL WPS_set_registration_user (const WPS_SimpleAuthentication * *authentication*)

Set the user's authentication information.

Deprecated

Replaced by `WPS_set_key()`

Parameters:

authentication	an existing user's authentication information.

Make sure to call this method before making any other calls, such as during initialization of the application.

After this call you should pass `NULL` as `authentication` to other calls like `WPS_location`.

Note:

Must not be used in conjunction with `WPS_set_key()`

Since:

4.8

Examples:

wpsapitest.cpp.

9.2.3.19 void WPSAPI_CALL WPS_set_server_url (const char * *url*)

Overwrite the WPS server's url from its default value.

Parameters:

url the url to the server.

A value of `NULL` turns off remote determination of location.

Examples:

wpsapitest.cpp.

9.2.3.20 WPS_ReturnCode WPSAPI_CALL WPS_set_tier2_area (const char * *dirpath*, unsigned *size*)

Setup the tier2 area.

Parameters:

dirpath the path to a directory to store the tier2 files.

size the approximate max total size of the tier2 files, in kilobytes.

Returns:

a `WPS_ReturnCode`

Deprecated

This call is no longer relevant.

Since:

2.6

9.2.3.21 WPS_ReturnCode WPSAPI_CALL WPS_set_tiling (const WPS_SimpleAuthentication * *authentication*, const char * *dirpath*, unsigned *maxDataSizePerSession*, unsigned *maxDataSizeTotal*, WPS_TilingCallback *callback*, void * *arg*)

Setup tiling so location determination can be performed locally.

Tiles are managed automatically (download of new tiles when required and remove of old tiles) by `WPS_location` and `WPS_periodic_location`.

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_registration_user()`, otherwise set to the user's authentication information.

dirpath the path to a directory to store tiles.

`dirpath` must end with a directory separator character. (i.e. '\' on Windows platforms and '/' on Unix platforms)

maxDataSizePerSession the maximum size of data downloaded per session, in bytes.

A value of 0 disables any further tile downloads.

maxDataSizeTotal the maximum size of all stored tiles, in bytes.

A value of 0 causes all stored tiles to be deleted.

callback the callback function to control the tile download. By default, all tiles are downloaded.

arg the opaque parameter to pass to the callback function.

Note:

Tiles are typically less then 50KB in size, so to download an area of 3x3 tiles for each session you would set `maxDataSizePerSession` to 450KB, i.e. 460,800.

It is recommended that `maxDataSizePerSession` be a factor of 2 - 10 smaller than `maxDataSizeTotal`, so that tiles from several areas can be cached.

Applications should not attempt to access tiles directly, in particular to delete them.

Returns:

a `WPS_ReturnCode`

Since:

3.3

Sample Code:

```
WPS_ReturnCode rc;

rc = WPS_set_tiling(NULL,
                    "/tiles/",
                    450 * 1024, // 450KB
                    2 * 1024 * 1024, // 2MB
                    NULL,
                    NULL);
if (rc != WPS_OK)
{
    fprintf(stderr, "*** WPS_set_tiling failed (%d)!\n", rc);
}
```

Examples:

wpsapitest.cpp.

9.2.3.22 void WPSAPI_CALL WPS_set_user_agent (const char ∗ *ua*)

Overwrite the default user agent of the requests going to the server.

Parameters:

ua the user agent string.

Since:

2.7

9.2.3.23 WPS_ReturnCode WPSAPI_CALL WPS_signed_certified_location (const WPS_SimpleAuthentication ∗ *authentication*, WPS_StreetAddressLookup *street_address_lookup*, const unsigned char ∗ *salt*, unsigned *saltLen*, WPS_CertifiedLocationCallback *callback*, void ∗ *arg*)

Requests certified geographic location updates that are signed by Skyhook's servers. The user may verify the signature of the returned location using Skyhook's public certificate.

See also:

`signature` in `WPS_Location` for details on the format of the signature.
sample code for verifying the signature in the `example/CertifiedSignatureTestJava` and `example/CertifiedSignatureTestPhp` directories

Parameters:

authentication must be `NULL` for new applications that use `WPS_set_key()` or `WPS_set_-registration_user()`, otherwise set to the user's authentication information.

street_address_lookup request street address lookup in addition to latitude/longitude lookup
Note that street address lookup is only performed when the location is determined by the remote server (network-centric model), not when the location is determined locally.

salt random data that is used as an additional input to the hash of the returned location.

saltLen length of the salt.

callback the callback routine to report locations to.

arg an opaque parameter passed to the callback routine.

Returns:

a `WPS_ReturnCode`

Note:

This call is blocking -- it will not return until a- an error occurs while setting up, or b- the number of iterations have been processed, or c- the `callback` returns `WPS_STOP`.
The `salt` is a mandatory parameter and will return `WPS_ERROR` if `salt` is `NULL` or `saltLen` is not greater than 0.
This call requires the device's local time to be correct. `WPS_ERROR_NOT_SIGNED` is returned to `WPS_CertifiedLocationCallback` if the local time is outside of the acceptable range.

Since:

4.9.1

Examples:

wpsapitest.cpp.

9.2.3.24 WPS_ReturnCode WPSAPI_CALL WPS_tiling (const char ∗ *dirpath*, unsigned *maxDataSizePerSession*, unsigned *maxDataSizeTotal*, WPS_TilingCallback *callback*, void ∗ *arg*)

Deprecated

Replaced by `WPS_set_tiling()`

Since:

2.6

9.2.3.25 WPS_ReturnCode WPSAPI_CALL WPS_tune_location (const WPS_SimpleAuthentication ∗ *authentication*, const WPS_Location ∗ *location*)

Tune the last location returned.

Parameters:

authentication an existing user's authentication information.

location the tuned location.

Returns:

a `WPS_ReturnCode`

Note:

The device must be at the location specified in `location` during the entire duration of this call. Note that pure GPS locations will not reflect any tuning.

Tuning is not an effective strategy for trying to make small adjustments to a location. It is best used for adding to our coverage or fixing results that are off by hundreds of meters.

Since:

3.0

9.2.3.26 void WPSAPI_CALL WPS_unload ()

Releases all resources consumed by WPS API and stops any background activity. Must be called before the calling application or module is unloaded.

Since:

4.4

Examples:

wpsapitest.cpp.

9.2.3.27 const char∗ WPSAPI_CALL WPS_version ()

Return a string containing the version information as `<major>.<minor>.<revision>.<build>`

Returns:

the version information

Since:

3.0

Examples:

wpsapitest.cpp.

Chapter 10

Example Documentation

10.1 CertifiedSignatureTestJava

This sample Java application, located in the `example/CertifiedSignatureTestJava` directory, is to demonstrate how to verify the signature of location returned by the `WPS_signed_certified_location()` API call.

Please refer to the `example/CertifiedSignatureTestJava/README` file for more details.

10.2 CertifiedSignatureTestPhp

This sample PHP application, located in the `example/CertifiedSignatureTestPhp` directory, is to demonstrate how to verify the signature of location returned by the `WPS_signed_certified_-location()` API call.

Please refer to the `example/CertifiedSignatureTestPhp/README` file for more details.

10.3 wpsapitest.cpp

This sample application, located in the `example/wpsapitest` directory, is a simple console based application.

When run, it first issues an ip location (`WPS_ip_location()`) request to locate itself based on the ip address of the device. It prints the latitude and longitude returned from the server.

Second it requests a geographic location (`WPS_location()`), with street address reverse lookup. It prints the latitude, longitude and address returned.

Finally, it requests a series of locations (`WPS_periodic_location()`).

Here's a sample output:

```
66.228.70.195: 42.342500, -71.067700

42.350950, -71.049709
328 Congress St
Boston, MA 02210
```

Note:

The sample application needs a direct connection to the internet in order for all functions to operate successfully.

```
#include <cassert>
#include <list>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <string.h>
#include <signal.h>

#ifdef _WIN32
#  define sleep(s) _sleep((s)*1000L)
#else
#  include <unistd.h>
#endif

#include "wpsapi.h"

#if HAVE_GETOPT_H
#  include <getopt.h>
#else
#  include "getopt.h"
#endif

#define TILES_TOTAL_DATA_SIZE (2 * 1024 * 1024) // 2 MB

static volatile bool interrupted = false;
static unsigned certified_iteration = 0;
static unsigned certified_iterations = 0;
static bool certified_output_json = false;
std::vector<unsigned char> salt;

static void
register_interrupt_handler(void (*handler)(int))
{
    signal(SIGTERM, handler);
    signal(SIGINT, handler);
#ifdef SIGHUP
```

```
    signal(SIGHUP, handler);
#endif
}

static void
interrupt_handler(int)
{
    if (! interrupted)
    {
        fprintf(stderr, "*** interrupted\n");
        interrupted = true;
    }

    register_interrupt_handler(interrupt_handler);
}

static void
print_street_address(const WPS_StreetAddress* address)
{
    if (! address)
        return;

    if (strlen(address->street_number) > 0)
        printf("%s ", address->street_number);

    char** line;
    for (line = address->address_line; *line; ++line)
        printf("%s\n", *line);

    printf("%s, %s %s\n",
           address->city,
           address->state.code,
           address->postal_code);
}

static void
print_location(const WPS_Location* location)
{
    printf("%f, %f\t+/-%.0fm\t%d+%d+%d  %lums\n",
           location->latitude,
           location->longitude,
           location->hpe,
           location->nap,
           location->ncell,
           location->nsat,
           location->age);

    if (location->speed >= 0)
    {
        printf("\t%.1fkm/h", location->speed);

        if (location->bearing >= 0)
            printf("\t%.0f", location->bearing);

        printf("\n");
    }

    if (location->hasScore != 0)
    {
        if (location->signature != NULL)
            printf("\tid: %d\n", location->id);

        printf("\tscore: %.1f\n", location->score);
        printf("\twith-ip: %s\n", location->withIP ? "true" : "false");

        if (location->ip != NULL)
            printf("\tip: %s\n", location->ip);
        printf("\ttimestamp: %lu\n", location->timestamp);
        printf("\thasSignature: %s\n", location->signature != NULL ? "true" : "fa
      lse");
    }

    print_street_address(location->street_address);
}

static void
print_ip_location(const WPS_IPLocation* ip_location)
{
    printf("%s: %f, %f\n\n",
           ip_location->ip,
           ip_location->latitude,
           ip_location->longitude);
    print_street_address(ip_location->street_address);
}

static void
print_hex(const unsigned char* data,
          unsigned length)
{
    for (unsigned i = 0; i < length; ++i)
        printf("%02x", data[i]);
}

static void
print_json_location(const WPS_Location* location)
{
    printf("\t{\n");
    printf("\t\t\"id\":\"%u\",\n", location->id);
    printf("\t\t\"latitude\":\"%f\",\n", location->latitude);
    printf("\t\t\"longitude\":\"%f\",\n", location->longitude);
    printf("\t\t\"hpe\":\"%d\",\n", (int) location->hpe);
    printf("\t\t\"nap\":\"%u\",\n", location->nap);
    printf("\t\t\"ncell\":\"%u\",\n", location->ncell);
    printf("\t\t\"nlac\":\"%u\",\n", location->nlac);
    printf("\t\t\"nsat\":\"%u\",\n", location->nsat);
    printf("\t\t\"nloc\":\"%u\",\n", location->historicalLocationCount);
    printf("\t\t\"age\":\"%lu\",\n", location->age);
    printf("\t\t\"score\":\"%f\",\n", location->score);
    printf("\t\t\"with-ip\":\"%s\",\n", location->withIP ? "true" : "false");
    printf("\t\t\"timestamp\":\"%lu\",\n", location->timestamp);
    printf("\t\t\"ip\":\"%s\",\n", location->ip);
    printf("\t\t\"hash\":\"");
    print_hex(location->hash, location->hashLength);
    printf("\",\n");
    printf("\t\t\"signature\":\"");
    print_hex(location->signature, location->signatureLength);
    printf("\"\n\t}");
}

static void
print_json_locations(const WPS_Location** locations,
                     unsigned nlocations)
{
    printf("\n{\n");
    printf("\t\"salt\":\"");
    print_hex(&salt[0], salt.size());
    printf("\",\n");
    printf("\t\"locations\":[\n");

    for (unsigned i = 0; i < nlocations; ++i)
    {
        if (i > 0)
            printf(",\n");
        print_json_location(locations[i]);
    }

    printf("]\n}\n");
}

static WPS_Continuation
certified_callback(void* arg,
                   WPS_ReturnCode code,
                   const WPS_Location** locations,
                   unsigned nlocations,
                   const void* reserved)
{
    if (interrupted)
        return WPS_STOP;

    if (code != WPS_OK)
    {
        fprintf(stderr, "*** failure (%d)!\n", code);

        if (code == WPS_ERROR_WIFI_NOT_AVAILABLE)
            return WPS_STOP;
    }
    else
    {
        for (unsigned i = 0; i < nlocations; ++i)
            print_location(locations[i]);

        if (certified_output_json && ! salt.empty())
            print_json_locations(locations, nlocations);
    }

    if (++certified_iteration >= certified_iterations)
        return WPS_STOP;

    return WPS_CONTINUE;
}

static WPS_Continuation
periodic_callback(void*,
                  WPS_ReturnCode code,
                  const WPS_Location* location,
                  const void* measurements)
{
    if (interrupted)
        return WPS_STOP;

    if (code != WPS_OK)
    {
        fprintf(stderr, "*** failure (%d)!\n", code);

        if (code == WPS_ERROR_WIFI_NOT_AVAILABLE)
            return WPS_STOP;
    }
    else
    {
        print_location(location);
    }

    return WPS_CONTINUE;
}

static WPS_Continuation
tiling_callback(void* arg,
                unsigned tileNumber,
                unsigned tileTotal)
```

```
{
    unsigned maxTiles = arg ? *static_cast<unsigned*>(arg) : 0;

    if (maxTiles > 0 && tileNumber >= maxTiles)
        return WPS_STOP;
    fprintf(stderr, "downloading tile %d/%d...\n", tileNumber+1, tileTotal);
    return WPS_CONTINUE;
}

static struct option longopts[] =
{
    { "help",        no_argument,        NULL, 'h' },
    { "key",         required_argument,  NULL, 'k' },
    { "user",        required_argument,  NULL, 'u' },
    { "realm",       required_argument,  NULL, 'r' },
    { "server",      optional_argument,  NULL, 's' },
    { "local",       optional_argument,  NULL, 'l' },
    { "period",      required_argument,  NULL, 'p' },
    { "iterations",  required_argument,  NULL, 'i' },
    { "Iterations",  required_argument,  NULL, 'I' },
    { "certified",   required_argument,  NULL, 'C' },
    { "salt",        required_argument,  NULL, 'S' },
    { "output-json", optional_argument,  NULL, 'J' },
    { "tiling",      required_argument,  NULL, 't' },
    { "max-tiles",   required_argument,  NULL, 'm' },
    { "max-total",   optional_argument,  NULL, 'c' },
    { "max-session", optional_argument,  NULL, 'd' },
    { "version",     no_argument,        NULL, 'v' },
    { NULL,          0,                  NULL, 0   }
};

static int
usage(char* const argv[])
{
    fprintf(stderr,
            "Usage: %s [arguments]\n\n"
            "\t--key=<key> | -k <key>\n"
            "\t--user=<username> | -u <username>\n"
            "\t--realm=<realm> | -r <realm>\n"
            "\t--server[=<server url>] | -s [<server url>]\n"
            "\t--local[=<compressed file path>] | -l [<compressed file path>]\n"
            "\t--period=<period> | -p <period> (in seconds)\n"
            "\t--iterations=<iterations> | -i <iterations>\n"
            "\t--certified=<iterations> | -C <iterations>\n"
            "\t--salt=<salt value> | -S <salt value>\n"
            "\t--output-json | -J\n"
            "\t--tiling=<tiling directory> | -t <tiling directory>\n"
            "\t--max-tiles=<max # of tiles to download> | -m <max tiles>\n"
            "\t--version | -v\n",
            argv[0]);
    return -1;
}

/**********************************************************************/
/*                                                                    */
/* main                                                               */
/*                                                                    */
/**********************************************************************/

int
main(int argc, char* argv[])
{
    // parse arguments

    WPS_SimpleAuthentication authentication;
    const char* key = "";
    authentication.username = "";
    authentication.realm = "";
    unsigned long period = 0;
    unsigned int iterations = 0, Iterations = 1;
    std::list<std::string> localfiles;
    const char* tiledir = NULL;
    unsigned maxtiles = 0;
    unsigned maxtotal = TILES_TOTAL_DATA_SIZE;
    unsigned maxsession = maxtotal / 5;
    WPS_ReturnCode rc;

    while (1)
    {
        int c = getopt_long(argc,
                            argv,
                            "hu:r:s:l:p:i:I:C:S:t:m:v",
                            longopts,
                            NULL);
        if (c == -1)
        {
            if (optind != argc)
            {
                fprintf(stderr, "%s: unknown option\n", argv[optind]);
                return usage(argv);
            }
            break;
        }
        switch (c)
        {
        case 'k':
            key = optarg;
            break;
        case 'u':
            authentication.username = optarg;
            break;
        case 'r':
            authentication.realm = optarg;
            break;
        case 's':
            WPS_set_server_url(optarg);
            break;
        case 'l':
            localfiles.push_back(optarg);
            break;
        case 'p':
            period = atol(optarg);
            break;
        case 'i':
            iterations = atol(optarg);
            break;
        case 'I':
            Iterations = atol(optarg);
            break;
        case 'C':
            certified_iterations = atol(optarg);
            break;
        case 'S':
            {
                size_t len = strlen(optarg);
                const unsigned char* buf =
                    reinterpret_cast<const unsigned char*>(optarg);
                salt.assign(buf, buf + len);
                break;
            }
        case 'J':
            certified_output_json = true;
            break;
        case 't':
            tiledir = optarg;
            break;
        case 'm':
            maxtiles = atoi(optarg);
            break;
        case 'c':
            maxtotal = atoi(optarg);
            break;
        case 'd':
            maxsession = atoi(optarg);
            break;
        case 'v':
            printf("WPS version %s\n", WPS_version());
            return 0;
        case 'h':
            return usage(argv);
        default:
            fprintf(stderr, "%c: unknown option\n", c);
        }
    }

    if (argc == 1)
        return usage(argv);

    register_interrupt_handler(interrupt_handler);

    /*****************************************************************/
    /* WPS_load                                                      */
    /*****************************************************************/

    rc = WPS_load();
    if (rc != WPS_OK)
    {
        fprintf(stderr, "*** WPS_load failed (%d)!\n\n", rc);
        return rc;
    }

    /*****************************************************************/
    /* WPS_set_key / WPS_set_registration_user                       */
    /*****************************************************************/

    if (strlen(key) > 0)
    {
        rc = WPS_set_key(key);
        if (rc != WPS_OK)
            fprintf(stderr, "*** WPS_set_key failed (%d)!\n\n", rc);
    }
    else
    {
        rc = WPS_set_registration_user(&authentication);
        if (rc != WPS_OK)
            fprintf(stderr, "*** WPS_set_registration_user failed (%d)!\n\n", rc)
      ;
    }

    /*****************************************************************/
    /* WPS_ip_location                                               */
    /*****************************************************************/

    WPS_IPLocation* ip_location;
    rc = WPS_ip_location(NULL,
                         WPS_NO_STREET_ADDRESS_LOOKUP,
                         &ip_location);
    if (rc != WPS_OK)
    {
        fprintf(stderr, "*** WPS_ip_location failed (%d)!\n\n", rc);
    }
    else
    {
        print_ip_location(ip_location);
        WPS_free_ip_location(ip_location);
    }

    /************************************************************/
    /* WPS_set_local_files_path                                 */
    /************************************************************/

    if (localfiles.size() > 0 && ! interrupted)
    {
        const char** paths = new const char*[localfiles.size() + 1];
        paths[localfiles.size()] = NULL;
        unsigned int path_index = 0;
        for (std::list<std::string>::const_iterator i = localfiles.begin();
             i != localfiles.end();
             ++i)
        {
            paths[path_index++] = i->c_str();
        }

        rc = WPS_set_local_files_path(paths);
        delete[] paths;

        if (rc != WPS_OK)
            fprintf(stderr, "*** WPS_set_local_files_path failed (%d)!\n\n", rc);

    }

    /************************************************************/
    /* WPS_set_tiling                                           */
    /************************************************************/

    // NOTE: WPS_set_tiling() must be called
    //       before WPS_periodic_location() or WPS_location()

    if (tiledir)
    {
        if (maxtiles > 0)
        {
            // limit the number of tiles downloaded,
            // around the current location,
            // to maxtiles via tiling_callback()

            rc = WPS_set_tiling(NULL,
                                tiledir,
                                maxtotal,
                                maxtotal,
                                tiling_callback,
                                &maxtiles);
        }
        else
        {
            // limit the amount of tiles downloaded,
            // around the current location,
            // to (maxtotal / 5).

            rc = WPS_set_tiling(NULL,
                                tiledir,
                                maxsession,
                                maxtotal,
                                tiling_callback,
                                NULL);
        }

        if (rc != WPS_OK)
```

```
            fprintf(stderr, "*** WPS_set_tiling failed (%d)!\n\n", rc);
    }

    /*****************************************************************/
    /* WPS_location                                                  */
    /*****************************************************************/

    if (period > 0 && Iterations > 0 && ! interrupted)
    {
        for (unsigned i = 0; i < Iterations; ++i)
        {
            WPS_Location* location;
            rc = WPS_location(NULL,
                              WPS_NO_STREET_ADDRESS_LOOKUP,
                              &location);
            if (rc != WPS_OK)
            {
                fprintf(stderr, "*** WPS_location failed (%d)!\n\n", rc);
            }
            else
            {
                print_location(location);
                WPS_free_location(location);
            }

            sleep(period);
        }
    }
    else if (period == 0 && ! interrupted)
    {
        WPS_Location* location;
        rc = WPS_location(NULL,
                          WPS_FULL_STREET_ADDRESS_LOOKUP,
                          &location);
        if (rc != WPS_OK)
        {
            fprintf(stderr, "*** WPS_location failed (%d)!\n\n", rc);
        }
        else
        {
            print_location(location);
            WPS_free_location(location);
        }
    }

    /*****************************************************************/
    /* WPS_certified_location                                        */
    /*****************************************************************/

    if (certified_iterations > 0 && ! interrupted)
    {
        if (! salt.empty())
        {
            rc = WPS_signed_certified_location(NULL,
                                               WPS_NO_STREET_ADDRESS_LOOKUP,
                                               &salt[0],
                                               salt.size(),
                                               certified_callback,
                                               NULL);
            if (rc != WPS_OK)
                fprintf(stderr, "*** WPS_signed_certified_location failed (%d)!\n
      \n", rc);
        }
        else
        {
            rc = WPS_certified_location(NULL,
                                        WPS_NO_STREET_ADDRESS_LOOKUP,
                                        certified_callback,
                                        NULL);
            if (rc != WPS_OK)
                fprintf(stderr, "*** WPS_certified_location failed (%d)!\n\n", rc
      );
        }
    }

    /************************************************************/
    /* WPS_periodic_location                                    */
    /************************************************************/

    if (period > 0 && ! interrupted)
    {
        rc = WPS_periodic_location(NULL,
                                   WPS_NO_STREET_ADDRESS_LOOKUP,
                                   period * 1000,
                                   iterations,
                                   periodic_callback,
                                   NULL);
        if (rc != WPS_OK)
            fprintf(stderr, "*** WPS_periodic_location failed (%d)!\n\n", rc);
    }

    /************************************************************/
    /* WPS_unload                                               */
    /************************************************************/

    WPS_unload();

    return 0;
}
```

Index

address_line
    WPS_StreetAddress, 40
age
    WPS_Location, 30
altitude
    WPS_Location, 30

bearing
    WPS_Location, 31

city
    WPS_StreetAddress, 40
code
    WPS_NameCode, 38
country
    WPS_StreetAddress, 40
county
    WPS_StreetAddress, 40

doc/sdk/locationsettings.dox, 42

hash
    WPS_Location, 31
hashLength
    WPS_Location, 31
hasScore
    WPS_Location, 31
historicalLocationCount
    WPS_Location, 32
hpe
    WPS_Location, 32

id
    WPS_Location, 32
include/api/wpsapi.h, 43
ip
    WPS_IPLocation, 28
    WPS_Location, 32

latitude
    WPS_GeoFence, 26
    WPS_IPLocation, 28
    WPS_Location, 33
longitude
    WPS_GeoFence, 26
    WPS_IPLocation, 28
    WPS_Location, 33

name
    WPS_NameCode, 38
nap
    WPS_Location, 33
ncell
    WPS_Location, 33
nlac
    WPS_Location, 33
nsat
    WPS_Location, 34

period
    WPS_GeoFence, 26
postal_code
    WPS_StreetAddress, 41
province
    WPS_StreetAddress, 41

radius
    WPS_GeoFence, 27
realm
    WPS_SimpleAuthentication, 39
region
    WPS_StreetAddress, 41

score
    WPS_Location, 34
signature
    WPS_Location, 34
signatureLength
    WPS_Location, 36
size
    WPS_GeoFence, 27
speed
    WPS_Location, 36
state
    WPS_StreetAddress, 41
street_address
    WPS_IPLocation, 28
    WPS_Location, 36
street_number
    WPS_StreetAddress, 41

timestamp

WPS_Location, 36
type
WPS_GeoFence, 27
WPS_Location, 36

username
WPS_SimpleAuthentication, 39

withIP
WPS_Location, 37
WPS_CONTINUE
wpsapi.h, 47
WPS_ERROR
wpsapi.h, 49
WPS_ERROR_FILE_IO
wpsapi.h, 48
WPS_ERROR_INVALID_FILE_FORMAT
wpsapi.h, 48
WPS_ERROR_LOCATION_CANNOT_BE_-DETERMINED
wpsapi.h, 48
WPS_ERROR_LOCATION_SETTING_-DISABLED
wpsapi.h, 49
WPS_ERROR_NO_WIFI_IN_RANGE
wpsapi.h, 48
WPS_ERROR_NOT_SIGNED
wpsapi.h, 49
WPS_ERROR_NOT_TUNED
wpsapi.h, 49
WPS_ERROR_PROXY_UNAUTHORIZED
wpsapi.h, 48
WPS_ERROR_SCANNER_NOT_FOUND
wpsapi.h, 48
WPS_ERROR_SERVER_UNAVAILABLE
wpsapi.h, 48
WPS_ERROR_TIMEOUT
wpsapi.h, 49
WPS_ERROR_UNAUTHORIZED
wpsapi.h, 48
WPS_ERROR_WIFI_NOT_AVAILABLE
wpsapi.h, 48
WPS_FULL_STREET_ADDRESS_LOOKUP
wpsapi.h, 49
WPS_GEOFENCE_ENTER
wpsapi.h, 47
WPS_GEOFENCE_ERROR
wpsapi.h, 49
WPS_GEOFENCE_INSIDE
wpsapi.h, 48
WPS_GEOFENCE_LEAVE
wpsapi.h, 47
WPS_GEOFENCE_OUTSIDE
wpsapi.h, 48
WPS_LIMITED_STREET_ADDRESS_LOOKUP
wpsapi.h, 49
WPS_LOCATION_TYPE_2D
wpsapi.h, 48
WPS_LOCATION_TYPE_3D
wpsapi.h, 48
WPS_NO_STREET_ADDRESS_LOOKUP
wpsapi.h, 49
WPS_NOMEM
wpsapi.h, 49
WPS_NOT_APPLICABLE
wpsapi.h, 49
WPS_OK
wpsapi.h, 48
WPS_STOP
wpsapi.h, 47
WPS_certified_location
wpsapi.h, 50
WPS_CertifiedLocationCallback
wpsapi.h, 44
WPS_Continuation
wpsapi.h, 47
WPS_free_ip_location
wpsapi.h, 50
WPS_free_location
wpsapi.h, 50
WPS_free_offline_token
wpsapi.h, 50
WPS_GeoFence, 26
latitude, 26
longitude, 26
period, 26
radius, 27
size, 27
type, 27
WPS_geofence_cancel
wpsapi.h, 51
WPS_geofence_cancel_all
wpsapi.h, 51
WPS_geofence_set
wpsapi.h, 51
WPS_GeoFenceCallback
wpsapi.h, 45
WPS_GeoFenceHandle
wpsapi.h, 45
WPS_GeoFenceType
wpsapi.h, 47
WPS_ip_location
wpsapi.h, 52
WPS_IPLocation, 28
ip, 28
latitude, 28
longitude, 28
street_address, 28

WPS_load
    wpsapi.h, 53
WPS_Location, 30
    age, 30
    altitude, 30
    bearing, 31
    hash, 31
    hashLength, 31
    hasScore, 31
    historicalLocationCount, 32
    hpe, 32
    id, 32
    ip, 32
    latitude, 33
    longitude, 33
    nap, 33
    ncell, 33
    nlac, 33
    nsat, 34
    score, 34
    signature, 34
    signatureLength, 36
    speed, 36
    street_address, 36
    timestamp, 36
    type, 36
    withIP, 37
WPS_location
    wpsapi.h, 54
WPS_LocationCallback
    wpsapi.h, 46
WPS_LocationType
    wpsapi.h, 48
WPS_NameCode, 38
    code, 38
    name, 38
WPS_offline_location
    wpsapi.h, 54
WPS_offline_token
    wpsapi.h, 55
WPS_periodic_location
    wpsapi.h, 57
WPS_register_user
    wpsapi.h, 58
WPS_ReturnCode
    wpsapi.h, 48
WPS_set_key
    wpsapi.h, 59
WPS_set_local_files_path
    wpsapi.h, 60
WPS_set_proxy
    wpsapi.h, 60
WPS_set_registration_user
    wpsapi.h, 61
WPS_set_server_url
    wpsapi.h, 61
WPS_set_tier2_area
    wpsapi.h, 62
WPS_set_tiling
    wpsapi.h, 62
WPS_set_user_agent
    wpsapi.h, 63
WPS_signed_certified_location
    wpsapi.h, 63
WPS_SimpleAuthentication, 39
    realm, 39
    username, 39
WPS_StreetAddress, 40
    address_line, 40
    city, 40
    country, 40
    county, 40
    postal_code, 41
    province, 41
    region, 41
    state, 41
    street_number, 41
WPS_StreetAddressLookup
    wpsapi.h, 49
WPS_tiling
    wpsapi.h, 64
WPS_TilingCallback
    wpsapi.h, 46
WPS_tune_location
    wpsapi.h, 65
WPS_unload
    wpsapi.h, 65
WPS_version
    wpsapi.h, 65
wpsapi.h
    WPS_CONTINUE, 47
    WPS_ERROR, 49
    WPS_ERROR_FILE_IO, 48
    WPS_ERROR_INVALID_FILE_FORMAT, 48
    WPS_ERROR_LOCATION_CANNOT_BE_-DETERMINED, 48
    WPS_ERROR_LOCATION_SETTING_-DISABLED, 49
    WPS_ERROR_NO_WIFI_IN_RANGE, 48
    WPS_ERROR_NOT_SIGNED, 49
    WPS_ERROR_NOT_TUNED, 49
    WPS_ERROR_PROXY_UNAUTHORIZED, 48
    WPS_ERROR_SCANNER_NOT_FOUND, 48
    WPS_ERROR_SERVER_UNAVAILABLE, 48

WPS_ERROR_TIMEOUT, 49
WPS_ERROR_UNAUTHORIZED, 48
WPS_ERROR_WIFI_NOT_AVAILABLE, 48
WPS_FULL_STREET_ADDRESS_-LOOKUP, 49
WPS_GEOFENCE_ENTER, 47
WPS_GEOFENCE_ERROR, 49
WPS_GEOFENCE_INSIDE, 48
WPS_GEOFENCE_LEAVE, 47
WPS_GEOFENCE_OUTSIDE, 48
WPS_LIMITED_STREET_ADDRESS_-LOOKUP, 49
WPS_LOCATION_TYPE_2D, 48
WPS_LOCATION_TYPE_3D, 48
WPS_NO_STREET_ADDRESS_LOOKUP, 49
WPS_NOMEM, 49
WPS_NOT_APPLICABLE, 49
WPS_OK, 48
WPS_STOP, 47
WPS_certified_location, 50
WPS_CertifiedLocationCallback, 44
WPS_Continuation, 47
WPS_free_ip_location, 50
WPS_free_location, 50
WPS_free_offline_token, 50
WPS_geofence_cancel, 51
WPS_geofence_cancel_all, 51
WPS_geofence_set, 51
WPS_GeoFenceCallback, 45
WPS_GeoFenceHandle, 45
WPS_GeoFenceType, 47
WPS_ip_location, 52
WPS_load, 53
WPS_location, 54
WPS_LocationCallback, 46
WPS_LocationType, 48
WPS_offline_location, 54
WPS_offline_token, 55
WPS_periodic_location, 57
WPS_register_user, 58
WPS_ReturnCode, 48
WPS_set_key, 59
WPS_set_local_files_path, 60
WPS_set_proxy, 60
WPS_set_registration_user, 61
WPS_set_server_url, 61
WPS_set_tier2_area, 62
WPS_set_tiling, 62
WPS_set_user_agent, 63
WPS_signed_certified_location, 63
WPS_StreetAddressLookup, 49
WPS_tiling, 64
WPS_TilingCallback, 46
WPS_tune_location, 65
WPS_unload, 65
WPS_version, 65