private void updateSpeedByLocation(Location location) {

int tempSpeed = (int) (location.getSpeed() \* 3.6); // m/s --> Km/h

adasSpeed = tempSpeed;

recordSpeed = tempSpeed;

nowLatitude = location.getLatitude();

nowLongitude = location.getLongitude();

MyLog.i("GPS", "Speed:" + tempSpeed);

if (recorderFront != null) {

if (recordSpeed > 0) {

recorderFront.setSpeed(recordSpeed);

recordSpeed = 0; // 清除速度

}

recorderFront.setLat(new DecimalFormat("#.00000")

.format(nowLatitude) + "");

recorderFront.setLong(new DecimalFormat("#.00000")

.format(nowLongitude) + "");

}

}

————————————————

版权声明：本文为CSDN博主「周木水」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/zhoumushui/article/details/54782411>

https://gitlab.com/friendlyelec/rk3399-nougat/blob/nanopc-t4-nougat/hardware/libhardware/include/hardware/gps.h

/\*

\* Copyright (C) 2010 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

#ifndef ANDROID\_INCLUDE\_HARDWARE\_GPS\_H

#define ANDROID\_INCLUDE\_HARDWARE\_GPS\_H

#include <stdint.h>

#include <sys/cdefs.h>

#include <sys/types.h>

#include <pthread.h>

#include <sys/socket.h>

#include <stdbool.h>

#include <hardware/hardware.h>

\_\_BEGIN\_DECLS

/\*\*

\* The id of this module

\*/

#define GPS\_HARDWARE\_MODULE\_ID "gps"

/\*\* Milliseconds since January 1, 1970 \*/

typedef int64\_t GpsUtcTime;

/\*\* Maximum number of SVs for gps\_sv\_status\_callback(). \*/

#define GPS\_MAX\_SVS 32

/\*\* Maximum number of SVs for gps\_sv\_status\_callback(). \*/

#define GNSS\_MAX\_SVS 64

/\*\* Maximum number of Measurements in gps\_measurement\_callback(). \*/

#define GPS\_MAX\_MEASUREMENT 32

/\*\* Maximum number of Measurements in gnss\_measurement\_callback(). \*/

#define GNSS\_MAX\_MEASUREMENT 64

/\*\* Requested operational mode for GPS operation. \*/

typedef uint32\_t GpsPositionMode;

/\* IMPORTANT: Note that the following values must match

\* constants in GpsLocationProvider.java. \*/

/\*\* Mode for running GPS standalone (no assistance). \*/

#define GPS\_POSITION\_MODE\_STANDALONE 0

/\*\* AGPS MS-Based mode. \*/

#define GPS\_POSITION\_MODE\_MS\_BASED 1

/\*\*

\* AGPS MS-Assisted mode. This mode is not maintained by the platform anymore.

\* It is strongly recommended to use GPS\_POSITION\_MODE\_MS\_BASED instead.

\*/

#define GPS\_POSITION\_MODE\_MS\_ASSISTED 2

/\*\* Requested recurrence mode for GPS operation. \*/

typedef uint32\_t GpsPositionRecurrence;

/\* IMPORTANT: Note that the following values must match

\* constants in GpsLocationProvider.java. \*/

/\*\* Receive GPS fixes on a recurring basis at a specified period. \*/

#define GPS\_POSITION\_RECURRENCE\_PERIODIC 0

/\*\* Request a single shot GPS fix. \*/

#define GPS\_POSITION\_RECURRENCE\_SINGLE 1

/\*\* GPS status event values. \*/

typedef uint16\_t GpsStatusValue;

/\* IMPORTANT: Note that the following values must match

\* constants in GpsLocationProvider.java. \*/

/\*\* GPS status unknown. \*/

#define GPS\_STATUS\_NONE 0

/\*\* GPS has begun navigating. \*/

#define GPS\_STATUS\_SESSION\_BEGIN 1

/\*\* GPS has stopped navigating. \*/

#define GPS\_STATUS\_SESSION\_END 2

/\*\* GPS has powered on but is not navigating. \*/

#define GPS\_STATUS\_ENGINE\_ON 3

/\*\* GPS is powered off. \*/

#define GPS\_STATUS\_ENGINE\_OFF 4

/\*\* Flags to indicate which values are valid in a GpsLocation. \*/

typedef uint16\_t GpsLocationFlags;

/\* IMPORTANT: Note that the following values must match

\* constants in GpsLocationProvider.java. \*/

/\*\* GpsLocation has valid latitude and longitude. \*/

#define GPS\_LOCATION\_HAS\_LAT\_LONG 0x0001

/\*\* GpsLocation has valid altitude. \*/

#define GPS\_LOCATION\_HAS\_ALTITUDE 0x0002

/\*\* GpsLocation has valid speed. \*/

#define GPS\_LOCATION\_HAS\_SPEED 0x0004

/\*\* GpsLocation has valid bearing. \*/

#define GPS\_LOCATION\_HAS\_BEARING 0x0008

/\*\* GpsLocation has valid accuracy. \*/

#define GPS\_LOCATION\_HAS\_ACCURACY 0x0010

/\*\* Flags for the gps\_set\_capabilities callback. \*/

/\*\*

\* GPS HAL schedules fixes for GPS\_POSITION\_RECURRENCE\_PERIODIC mode. If this is

\* not set, then the framework will use 1000ms for min\_interval and will start

\* and call start() and stop() to schedule the GPS.

\*/

#define GPS\_CAPABILITY\_SCHEDULING (1 << 0)

/\*\* GPS supports MS-Based AGPS mode \*/

#define GPS\_CAPABILITY\_MSB (1 << 1)

/\*\* GPS supports MS-Assisted AGPS mode \*/

#define GPS\_CAPABILITY\_MSA (1 << 2)

/\*\* GPS supports single-shot fixes \*/

#define GPS\_CAPABILITY\_SINGLE\_SHOT (1 << 3)

/\*\* GPS supports on demand time injection \*/

#define GPS\_CAPABILITY\_ON\_DEMAND\_TIME (1 << 4)

/\*\* GPS supports Geofencing \*/

#define GPS\_CAPABILITY\_GEOFENCING (1 << 5)

/\*\* GPS supports Measurements. \*/

#define GPS\_CAPABILITY\_MEASUREMENTS (1 << 6)

/\*\* GPS supports Navigation Messages \*/

#define GPS\_CAPABILITY\_NAV\_MESSAGES (1 << 7)

/\*\*

\* Flags used to specify which aiding data to delete when calling

\* delete\_aiding\_data().

\*/

typedef uint16\_t GpsAidingData;

/\* IMPORTANT: Note that the following values must match

\* constants in GpsLocationProvider.java. \*/

#define GPS\_DELETE\_EPHEMERIS 0x0001

#define GPS\_DELETE\_ALMANAC 0x0002

#define GPS\_DELETE\_POSITION 0x0004

#define GPS\_DELETE\_TIME 0x0008

#define GPS\_DELETE\_IONO 0x0010

#define GPS\_DELETE\_UTC 0x0020

#define GPS\_DELETE\_HEALTH 0x0040

#define GPS\_DELETE\_SVDIR 0x0080

#define GPS\_DELETE\_SVSTEER 0x0100

#define GPS\_DELETE\_SADATA 0x0200

#define GPS\_DELETE\_RTI 0x0400

#define GPS\_DELETE\_CELLDB\_INFO 0x8000

#define GPS\_DELETE\_ALL 0xFFFF

/\*\* AGPS type \*/

typedef uint16\_t AGpsType;

#define AGPS\_TYPE\_SUPL 1

#define AGPS\_TYPE\_C2K 2

typedef uint16\_t AGpsSetIDType;

#define AGPS\_SETID\_TYPE\_NONE 0

#define AGPS\_SETID\_TYPE\_IMSI 1

#define AGPS\_SETID\_TYPE\_MSISDN 2

typedef uint16\_t ApnIpType;

#define APN\_IP\_INVALID 0

#define APN\_IP\_IPV4 1

#define APN\_IP\_IPV6 2

#define APN\_IP\_IPV4V6 3

/\*\*

\* String length constants

\*/

#define GPS\_NI\_SHORT\_STRING\_MAXLEN 256

#define GPS\_NI\_LONG\_STRING\_MAXLEN 2048

/\*\*

\* GpsNiType constants

\*/

typedef uint32\_t GpsNiType;

#define GPS\_NI\_TYPE\_VOICE 1

#define GPS\_NI\_TYPE\_UMTS\_SUPL 2

#define GPS\_NI\_TYPE\_UMTS\_CTRL\_PLANE 3

/\*\*

\* GpsNiNotifyFlags constants

\*/

typedef uint32\_t GpsNiNotifyFlags;

/\*\* NI requires notification \*/

#define GPS\_NI\_NEED\_NOTIFY 0x0001

/\*\* NI requires verification \*/

#define GPS\_NI\_NEED\_VERIFY 0x0002

/\*\* NI requires privacy override, no notification/minimal trace \*/

#define GPS\_NI\_PRIVACY\_OVERRIDE 0x0004

/\*\*

\* GPS NI responses, used to define the response in

\* NI structures

\*/

typedef int GpsUserResponseType;

#define GPS\_NI\_RESPONSE\_ACCEPT 1

#define GPS\_NI\_RESPONSE\_DENY 2

#define GPS\_NI\_RESPONSE\_NORESP 3

/\*\*

\* NI data encoding scheme

\*/

typedef int GpsNiEncodingType;

#define GPS\_ENC\_NONE 0

#define GPS\_ENC\_SUPL\_GSM\_DEFAULT 1

#define GPS\_ENC\_SUPL\_UTF8 2

#define GPS\_ENC\_SUPL\_UCS2 3

#define GPS\_ENC\_UNKNOWN -1

/\*\* AGPS status event values. \*/

typedef uint16\_t AGpsStatusValue;

/\*\* GPS requests data connection for AGPS. \*/

#define GPS\_REQUEST\_AGPS\_DATA\_CONN 1

/\*\* GPS releases the AGPS data connection. \*/

#define GPS\_RELEASE\_AGPS\_DATA\_CONN 2

/\*\* AGPS data connection initiated \*/

#define GPS\_AGPS\_DATA\_CONNECTED 3

/\*\* AGPS data connection completed \*/

#define GPS\_AGPS\_DATA\_CONN\_DONE 4

/\*\* AGPS data connection failed \*/

#define GPS\_AGPS\_DATA\_CONN\_FAILED 5

typedef uint16\_t AGpsRefLocationType;

#define AGPS\_REF\_LOCATION\_TYPE\_GSM\_CELLID 1

#define AGPS\_REF\_LOCATION\_TYPE\_UMTS\_CELLID 2

#define AGPS\_REF\_LOCATION\_TYPE\_MAC 3

#define AGPS\_REF\_LOCATION\_TYPE\_LTE\_CELLID 4

/\* Deprecated, to be removed in the next Android release. \*/

#define AGPS\_REG\_LOCATION\_TYPE\_MAC 3

/\*\* Network types for update\_network\_state "type" parameter \*/

#define AGPS\_RIL\_NETWORK\_TYPE\_MOBILE 0

#define AGPS\_RIL\_NETWORK\_TYPE\_WIFI 1

#define AGPS\_RIL\_NETWORK\_TYPE\_MOBILE\_MMS 2

#define AGPS\_RIL\_NETWORK\_TYPE\_MOBILE\_SUPL 3

#define AGPS\_RIL\_NETWORK\_TTYPE\_MOBILE\_DUN 4

#define AGPS\_RIL\_NETWORK\_TTYPE\_MOBILE\_HIPRI 5

#define AGPS\_RIL\_NETWORK\_TTYPE\_WIMAX 6

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint16\_t GpsClockFlags;

#define GPS\_CLOCK\_HAS\_LEAP\_SECOND (1<<0)

#define GPS\_CLOCK\_HAS\_TIME\_UNCERTAINTY (1<<1)

#define GPS\_CLOCK\_HAS\_FULL\_BIAS (1<<2)

#define GPS\_CLOCK\_HAS\_BIAS (1<<3)

#define GPS\_CLOCK\_HAS\_BIAS\_UNCERTAINTY (1<<4)

#define GPS\_CLOCK\_HAS\_DRIFT (1<<5)

#define GPS\_CLOCK\_HAS\_DRIFT\_UNCERTAINTY (1<<6)

/\*\*

\* Flags to indicate what fields in GnssClock are valid.

\*/

typedef uint16\_t GnssClockFlags;

/\*\* A valid 'leap second' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_LEAP\_SECOND (1<<0)

/\*\* A valid 'time uncertainty' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_TIME\_UNCERTAINTY (1<<1)

/\*\* A valid 'full bias' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_FULL\_BIAS (1<<2)

/\*\* A valid 'bias' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_BIAS (1<<3)

/\*\* A valid 'bias uncertainty' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_BIAS\_UNCERTAINTY (1<<4)

/\*\* A valid 'drift' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_DRIFT (1<<5)

/\*\* A valid 'drift uncertainty' is stored in the data structure. \*/

#define GNSS\_CLOCK\_HAS\_DRIFT\_UNCERTAINTY (1<<6)

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint8\_t GpsClockType;

#define GPS\_CLOCK\_TYPE\_UNKNOWN 0

#define GPS\_CLOCK\_TYPE\_LOCAL\_HW\_TIME 1

#define GPS\_CLOCK\_TYPE\_GPS\_TIME 2

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint32\_t GpsMeasurementFlags;

#define GPS\_MEASUREMENT\_HAS\_SNR (1<<0)

#define GPS\_MEASUREMENT\_HAS\_ELEVATION (1<<1)

#define GPS\_MEASUREMENT\_HAS\_ELEVATION\_UNCERTAINTY (1<<2)

#define GPS\_MEASUREMENT\_HAS\_AZIMUTH (1<<3)

#define GPS\_MEASUREMENT\_HAS\_AZIMUTH\_UNCERTAINTY (1<<4)

#define GPS\_MEASUREMENT\_HAS\_PSEUDORANGE (1<<5)

#define GPS\_MEASUREMENT\_HAS\_PSEUDORANGE\_UNCERTAINTY (1<<6)

#define GPS\_MEASUREMENT\_HAS\_CODE\_PHASE (1<<7)

#define GPS\_MEASUREMENT\_HAS\_CODE\_PHASE\_UNCERTAINTY (1<<8)

#define GPS\_MEASUREMENT\_HAS\_CARRIER\_FREQUENCY (1<<9)

#define GPS\_MEASUREMENT\_HAS\_CARRIER\_CYCLES (1<<10)

#define GPS\_MEASUREMENT\_HAS\_CARRIER\_PHASE (1<<11)

#define GPS\_MEASUREMENT\_HAS\_CARRIER\_PHASE\_UNCERTAINTY (1<<12)

#define GPS\_MEASUREMENT\_HAS\_BIT\_NUMBER (1<<13)

#define GPS\_MEASUREMENT\_HAS\_TIME\_FROM\_LAST\_BIT (1<<14)

#define GPS\_MEASUREMENT\_HAS\_DOPPLER\_SHIFT (1<<15)

#define GPS\_MEASUREMENT\_HAS\_DOPPLER\_SHIFT\_UNCERTAINTY (1<<16)

#define GPS\_MEASUREMENT\_HAS\_USED\_IN\_FIX (1<<17)

#define GPS\_MEASUREMENT\_HAS\_UNCORRECTED\_PSEUDORANGE\_RATE (1<<18)

/\*\*

\* Flags to indicate what fields in GnssMeasurement are valid.

\*/

typedef uint32\_t GnssMeasurementFlags;

/\*\* A valid 'snr' is stored in the data structure. \*/

#define GNSS\_MEASUREMENT\_HAS\_SNR (1<<0)

/\*\* A valid 'carrier frequency' is stored in the data structure. \*/

#define GNSS\_MEASUREMENT\_HAS\_CARRIER\_FREQUENCY (1<<9)

/\*\* A valid 'carrier cycles' is stored in the data structure. \*/

#define GNSS\_MEASUREMENT\_HAS\_CARRIER\_CYCLES (1<<10)

/\*\* A valid 'carrier phase' is stored in the data structure. \*/

#define GNSS\_MEASUREMENT\_HAS\_CARRIER\_PHASE (1<<11)

/\*\* A valid 'carrier phase uncertainty' is stored in the data structure. \*/

#define GNSS\_MEASUREMENT\_HAS\_CARRIER\_PHASE\_UNCERTAINTY (1<<12)

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint8\_t GpsLossOfLock;

#define GPS\_LOSS\_OF\_LOCK\_UNKNOWN 0

#define GPS\_LOSS\_OF\_LOCK\_OK 1

#define GPS\_LOSS\_OF\_LOCK\_CYCLE\_SLIP 2

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. Use GnssMultipathIndicator instead.

\*/

typedef uint8\_t GpsMultipathIndicator;

#define GPS\_MULTIPATH\_INDICATOR\_UNKNOWN 0

#define GPS\_MULTIPATH\_INDICATOR\_DETECTED 1

#define GPS\_MULTIPATH\_INDICATOR\_NOT\_USED 2

/\*\*

\* Enumeration of available values for the GNSS Measurement's multipath

\* indicator.

\*/

typedef uint8\_t GnssMultipathIndicator;

/\*\* The indicator is not available or unknown. \*/

#define GNSS\_MULTIPATH\_INDICATOR\_UNKNOWN 0

/\*\* The measurement is indicated to be affected by multipath. \*/

#define GNSS\_MULTIPATH\_INDICATOR\_PRESENT 1

/\*\* The measurement is indicated to be not affected by multipath. \*/

#define GNSS\_MULTIPATH\_INDICATOR\_NOT\_PRESENT 2

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint16\_t GpsMeasurementState;

#define GPS\_MEASUREMENT\_STATE\_UNKNOWN 0

#define GPS\_MEASUREMENT\_STATE\_CODE\_LOCK (1<<0)

#define GPS\_MEASUREMENT\_STATE\_BIT\_SYNC (1<<1)

#define GPS\_MEASUREMENT\_STATE\_SUBFRAME\_SYNC (1<<2)

#define GPS\_MEASUREMENT\_STATE\_TOW\_DECODED (1<<3)

#define GPS\_MEASUREMENT\_STATE\_MSEC\_AMBIGUOUS (1<<4)

/\*\*

\* Flags indicating the GNSS measurement state.

\*

\* The expected behavior here is for GPS HAL to set all the flags that applies.

\* For example, if the state for a satellite is only C/A code locked and bit

\* synchronized, and there is still millisecond ambiguity, the state should be

\* set as:

\*

\* GNSS\_MEASUREMENT\_STATE\_CODE\_LOCK | GNSS\_MEASUREMENT\_STATE\_BIT\_SYNC |

\* GNSS\_MEASUREMENT\_STATE\_MSEC\_AMBIGUOUS

\*

\* If GNSS is still searching for a satellite, the corresponding state should be

\* set to GNSS\_MEASUREMENT\_STATE\_UNKNOWN(0).

\*/

typedef uint32\_t GnssMeasurementState;

#define GNSS\_MEASUREMENT\_STATE\_UNKNOWN 0

#define GNSS\_MEASUREMENT\_STATE\_CODE\_LOCK (1<<0)

#define GNSS\_MEASUREMENT\_STATE\_BIT\_SYNC (1<<1)

#define GNSS\_MEASUREMENT\_STATE\_SUBFRAME\_SYNC (1<<2)

#define GNSS\_MEASUREMENT\_STATE\_TOW\_DECODED (1<<3)

#define GNSS\_MEASUREMENT\_STATE\_MSEC\_AMBIGUOUS (1<<4)

#define GNSS\_MEASUREMENT\_STATE\_SYMBOL\_SYNC (1<<5)

#define GNSS\_MEASUREMENT\_STATE\_GLO\_STRING\_SYNC (1<<6)

#define GNSS\_MEASUREMENT\_STATE\_GLO\_TOD\_DECODED (1<<7)

#define GNSS\_MEASUREMENT\_STATE\_BDS\_D2\_BIT\_SYNC (1<<8)

#define GNSS\_MEASUREMENT\_STATE\_BDS\_D2\_SUBFRAME\_SYNC (1<<9)

#define GNSS\_MEASUREMENT\_STATE\_GAL\_E1BC\_CODE\_LOCK (1<<10)

#define GNSS\_MEASUREMENT\_STATE\_GAL\_E1C\_2ND\_CODE\_LOCK (1<<11)

#define GNSS\_MEASUREMENT\_STATE\_GAL\_E1B\_PAGE\_SYNC (1<<12)

#define GNSS\_MEASUREMENT\_STATE\_SBAS\_SYNC (1<<13)

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint16\_t GpsAccumulatedDeltaRangeState;

#define GPS\_ADR\_STATE\_UNKNOWN 0

#define GPS\_ADR\_STATE\_VALID (1<<0)

#define GPS\_ADR\_STATE\_RESET (1<<1)

#define GPS\_ADR\_STATE\_CYCLE\_SLIP (1<<2)

/\*\*

\* Flags indicating the Accumulated Delta Range's states.

\*/

typedef uint16\_t GnssAccumulatedDeltaRangeState;

#define GNSS\_ADR\_STATE\_UNKNOWN 0

#define GNSS\_ADR\_STATE\_VALID (1<<0)

#define GNSS\_ADR\_STATE\_RESET (1<<1)

#define GNSS\_ADR\_STATE\_CYCLE\_SLIP (1<<2)

/\* The following typedef together with its constants below are deprecated, and

\* will be removed in the next release. \*/

typedef uint8\_t GpsNavigationMessageType;

#define GPS\_NAVIGATION\_MESSAGE\_TYPE\_UNKNOWN 0

#define GPS\_NAVIGATION\_MESSAGE\_TYPE\_L1CA 1

#define GPS\_NAVIGATION\_MESSAGE\_TYPE\_L2CNAV 2

#define GPS\_NAVIGATION\_MESSAGE\_TYPE\_L5CNAV 3

#define GPS\_NAVIGATION\_MESSAGE\_TYPE\_CNAV2 4

/\*\*

\* Enumeration of available values to indicate the GNSS Navigation message

\* types.

\*

\* For convenience, first byte is the GnssConstellationType on which that signal

\* is typically transmitted

\*/

typedef int16\_t GnssNavigationMessageType;

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_UNKNOWN 0

/\*\* GPS L1 C/A message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GPS\_L1CA 0x0101

/\*\* GPS L2-CNAV message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GPS\_L2CNAV 0x0102

/\*\* GPS L5-CNAV message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GPS\_L5CNAV 0x0103

/\*\* GPS CNAV-2 message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GPS\_CNAV2 0x0104

/\*\* Glonass L1 CA message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GLO\_L1CA 0x0301

/\*\* Beidou D1 message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_BDS\_D1 0x0501

/\*\* Beidou D2 message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_BDS\_D2 0x0502

/\*\* Galileo I/NAV message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GAL\_I 0x0601

/\*\* Galileo F/NAV message contained in the structure. \*/

#define GNSS\_NAVIGATION\_MESSAGE\_TYPE\_GAL\_F 0x0602

/\*\*

\* Status of Navigation Message

\* When a message is received properly without any parity error in its navigation words, the

\* status should be set to NAV\_MESSAGE\_STATUS\_PARITY\_PASSED. But if a message is received

\* with words that failed parity check, but GPS is able to correct those words, the status

\* should be set to NAV\_MESSAGE\_STATUS\_PARITY\_REBUILT.

\* No need to send any navigation message that contains words with parity error and cannot be

\* corrected.

\*/

typedef uint16\_t NavigationMessageStatus;

#define NAV\_MESSAGE\_STATUS\_UNKNOWN 0

#define NAV\_MESSAGE\_STATUS\_PARITY\_PASSED (1<<0)

#define NAV\_MESSAGE\_STATUS\_PARITY\_REBUILT (1<<1)

/\* This constant is deprecated, and will be removed in the next release. \*/

#define NAV\_MESSAGE\_STATUS\_UNKONW 0

/\*\*

\* Flags that indicate information about the satellite

\*/

typedef uint8\_t GnssSvFlags;

#define GNSS\_SV\_FLAGS\_NONE 0

#define GNSS\_SV\_FLAGS\_HAS\_EPHEMERIS\_DATA (1 << 0)

#define GNSS\_SV\_FLAGS\_HAS\_ALMANAC\_DATA (1 << 1)

#define GNSS\_SV\_FLAGS\_USED\_IN\_FIX (1 << 2)

/\*\*

\* Constellation type of GnssSvInfo

\*/

typedef uint8\_t GnssConstellationType;

#define GNSS\_CONSTELLATION\_UNKNOWN 0

#define GNSS\_CONSTELLATION\_GPS 1

#define GNSS\_CONSTELLATION\_SBAS 2

#define GNSS\_CONSTELLATION\_GLONASS 3

#define GNSS\_CONSTELLATION\_QZSS 4

#define GNSS\_CONSTELLATION\_BEIDOU 5

#define GNSS\_CONSTELLATION\_GALILEO 6

/\*\*

\* Name for the GPS XTRA interface.

\*/

#define GPS\_XTRA\_INTERFACE "gps-xtra"

/\*\*

\* Name for the GPS DEBUG interface.

\*/

#define GPS\_DEBUG\_INTERFACE "gps-debug"

/\*\*

\* Name for the AGPS interface.

\*/

#define AGPS\_INTERFACE "agps"

/\*\*

\* Name of the Supl Certificate interface.

\*/

#define SUPL\_CERTIFICATE\_INTERFACE "supl-certificate"

/\*\*

\* Name for NI interface

\*/

#define GPS\_NI\_INTERFACE "gps-ni"

/\*\*

\* Name for the AGPS-RIL interface.

\*/

#define AGPS\_RIL\_INTERFACE "agps\_ril"

/\*\*

\* Name for the GPS\_Geofencing interface.

\*/

#define GPS\_GEOFENCING\_INTERFACE "gps\_geofencing"

/\*\*

\* Name of the GPS Measurements interface.

\*/

#define GPS\_MEASUREMENT\_INTERFACE "gps\_measurement"

/\*\*

\* Name of the GPS navigation message interface.

\*/

#define GPS\_NAVIGATION\_MESSAGE\_INTERFACE "gps\_navigation\_message"

/\*\*

\* Name of the GNSS/GPS configuration interface.

\*/

#define GNSS\_CONFIGURATION\_INTERFACE "gnss\_configuration"

/\*\* Represents a location. \*/

typedef struct {

/\*\* set to sizeof(GpsLocation) \*/

size\_t size;

/\*\* Contains GpsLocationFlags bits. \*/

uint16\_t flags;

/\*\* Represents latitude in degrees. \*/

double latitude;

/\*\* Represents longitude in degrees. \*/

double longitude;

/\*\*

\* Represents altitude in meters above the WGS 84 reference ellipsoid.

\*/

double altitude;

/\*\* Represents speed in meters per second. \*/

float speed;

/\*\* Represents heading in degrees. \*/

float bearing;

/\*\* Represents expected accuracy in meters. \*/

float accuracy;

/\*\* Timestamp for the location fix. \*/

GpsUtcTime timestamp;

} GpsLocation;

/\*\* Represents the status. \*/

typedef struct {

/\*\* set to sizeof(GpsStatus) \*/

size\_t size;

GpsStatusValue status;

} GpsStatus;

/\*\*

\* Legacy struct to represents SV information.

\* Deprecated, to be removed in the next Android release.

\* Use GnssSvInfo instead.

\*/

typedef struct {

/\*\* set to sizeof(GpsSvInfo) \*/

size\_t size;

/\*\* Pseudo-random number for the SV. \*/

int prn;

/\*\* Signal to noise ratio. \*/

float snr;

/\*\* Elevation of SV in degrees. \*/

float elevation;

/\*\* Azimuth of SV in degrees. \*/

float azimuth;

} GpsSvInfo;

typedef struct {

/\*\* set to sizeof(GnssSvInfo) \*/

size\_t size;

/\*\*

\* Pseudo-random number for the SV, or FCN/OSN number for Glonass. The

\* distinction is made by looking at constellation field. Values should be

\* in the range of:

\*

\* - GPS: 1-32

\* - SBAS: 120-151, 183-192

\* - GLONASS: 1-24, the orbital slot number (OSN), if known. Or, if not:

\* 93-106, the frequency channel number (FCN) (-7 to +6) offset by + 100

\* i.e. report an FCN of -7 as 93, FCN of 0 as 100, and FCN of +6 as 106.

\* - QZSS: 193-200

\* - Galileo: 1-36

\* - Beidou: 1-37

\*/

int16\_t svid;

/\*\*

\* Defines the constellation of the given SV. Value should be one of those

\* GNSS\_CONSTELLATION\_\* constants

\*/

GnssConstellationType constellation;

/\*\*

\* Carrier-to-noise density in dB-Hz, typically in the range [0, 63].

\* It contains the measured C/N0 value for the signal at the antenna port.

\*

\* This is a mandatory value.

\*/

float c\_n0\_dbhz;

/\*\* Elevation of SV in degrees. \*/

float elevation;

/\*\* Azimuth of SV in degrees. \*/

float azimuth;

/\*\*

\* Contains additional data about the given SV. Value should be one of those

\* GNSS\_SV\_FLAGS\_\* constants

\*/

GnssSvFlags flags;

} GnssSvInfo;

/\*\*

\* Legacy struct to represents SV status.

\* Deprecated, to be removed in the next Android release.

\* Use GnssSvStatus instead.

\*/

typedef struct {

/\*\* set to sizeof(GpsSvStatus) \*/

size\_t size;

int num\_svs;

GpsSvInfo sv\_list[GPS\_MAX\_SVS];

uint32\_t ephemeris\_mask;

uint32\_t almanac\_mask;

uint32\_t used\_in\_fix\_mask;

} GpsSvStatus;

/\*\*

\* Represents SV status.

\*/

typedef struct {

/\*\* set to sizeof(GnssSvStatus) \*/

size\_t size;

/\*\* Number of GPS SVs currently visible, refers to the SVs stored in sv\_list \*/

int num\_svs;

/\*\*

\* Pointer to an array of SVs information for all GNSS constellations,

\* except GPS, which is reported using sv\_list

\*/

GnssSvInfo gnss\_sv\_list[GNSS\_MAX\_SVS];

} GnssSvStatus;

/\* CellID for 2G, 3G and LTE, used in AGPS. \*/

typedef struct {

AGpsRefLocationType type;

/\*\* Mobile Country Code. \*/

uint16\_t mcc;

/\*\* Mobile Network Code .\*/

uint16\_t mnc;

/\*\* Location Area Code in 2G, 3G and LTE. In 3G lac is discarded. In LTE,

\* lac is populated with tac, to ensure that we don't break old clients that

\* might rely in the old (wrong) behavior.

\*/

uint16\_t lac;

/\*\* Cell id in 2G. Utran Cell id in 3G. Cell Global Id EUTRA in LTE. \*/

uint32\_t cid;

/\*\* Tracking Area Code in LTE. \*/

uint16\_t tac;

/\*\* Physical Cell id in LTE (not used in 2G and 3G) \*/

uint16\_t pcid;

} AGpsRefLocationCellID;

typedef struct {

uint8\_t mac[6];

} AGpsRefLocationMac;

/\*\* Represents ref locations \*/

typedef struct {

AGpsRefLocationType type;

union {

AGpsRefLocationCellID cellID;

AGpsRefLocationMac mac;

} u;

} AGpsRefLocation;

/\*\*

\* Callback with location information. Can only be called from a thread created

\* by create\_thread\_cb.

\*/

typedef void (\* gps\_location\_callback)(GpsLocation\* location);

/\*\*

\* Callback with status information. Can only be called from a thread created by

\* create\_thread\_cb.

\*/

typedef void (\* gps\_status\_callback)(GpsStatus\* status);

/\*\*

\* Legacy callback with SV status information.

\* Can only be called from a thread created by create\_thread\_cb.

\*

\* This callback is deprecated, and will be removed in the next release. Use

\* gnss\_sv\_status\_callback() instead.

\*/

typedef void (\* gps\_sv\_status\_callback)(GpsSvStatus\* sv\_info);

/\*\*

\* Callback with SV status information.

\* Can only be called from a thread created by create\_thread\_cb.

\*/

typedef void (\* gnss\_sv\_status\_callback)(GnssSvStatus\* sv\_info);

/\*\*

\* Callback for reporting NMEA sentences. Can only be called from a thread

\* created by create\_thread\_cb.

\*/

typedef void (\* gps\_nmea\_callback)(GpsUtcTime timestamp, const char\* nmea, int length);

/\*\*

\* Callback to inform framework of the GPS engine's capabilities. Capability

\* parameter is a bit field of GPS\_CAPABILITY\_\* flags.

\*/

typedef void (\* gps\_set\_capabilities)(uint32\_t capabilities);

/\*\*

\* Callback utility for acquiring the GPS wakelock. This can be used to prevent

\* the CPU from suspending while handling GPS events.

\*/

typedef void (\* gps\_acquire\_wakelock)();

/\*\* Callback utility for releasing the GPS wakelock. \*/

typedef void (\* gps\_release\_wakelock)();

/\*\* Callback for requesting NTP time \*/

typedef void (\* gps\_request\_utc\_time)();

/\*\*

\* Callback for creating a thread that can call into the Java framework code.

\* This must be used to create any threads that report events up to the

\* framework.

\*/

typedef pthread\_t (\* gps\_create\_thread)(const char\* name, void (\*start)(void \*), void\* arg);

/\*\*

\* Provides information about how new the underlying GPS/GNSS hardware and

\* software is.

\*

\* This information will be available for Android Test Applications. If a GPS

\* HAL does not provide this information, it will be considered "2015 or

\* earlier".

\*

\* If a GPS HAL does provide this information, then newer years will need to

\* meet newer CTS standards. E.g. if the date are 2016 or above, then N+ level

\* GpsMeasurement support will be verified.

\*/

typedef struct {

/\*\* Set to sizeof(GnssSystemInfo) \*/

size\_t size;

/\* year in which the last update was made to the underlying hardware/firmware

\* used to capture GNSS signals, e.g. 2016 \*/

uint16\_t year\_of\_hw;

} GnssSystemInfo;

/\*\*

\* Callback to inform framework of the engine's hardware version information.

\*/

typedef void (\*gnss\_set\_system\_info)(const GnssSystemInfo\* info);

/\*\* New GPS callback structure. \*/

typedef struct {

/\*\* set to sizeof(GpsCallbacks) \*/

size\_t size;

gps\_location\_callback location\_cb;

gps\_status\_callback status\_cb;

gps\_sv\_status\_callback sv\_status\_cb;

gps\_nmea\_callback nmea\_cb;

gps\_set\_capabilities set\_capabilities\_cb;

gps\_acquire\_wakelock acquire\_wakelock\_cb;

gps\_release\_wakelock release\_wakelock\_cb;

gps\_create\_thread create\_thread\_cb;

gps\_request\_utc\_time request\_utc\_time\_cb;

gnss\_set\_system\_info set\_system\_info\_cb;

gnss\_sv\_status\_callback gnss\_sv\_status\_cb;

} GpsCallbacks;

/\*\* Represents the standard GPS interface. \*/

typedef struct {

/\*\* set to sizeof(GpsInterface) \*/

size\_t size;

/\*\*

\* Opens the interface and provides the callback routines

\* to the implementation of this interface.

\*/

int (\*init)( GpsCallbacks\* callbacks );

/\*\* Starts navigating. \*/

int (\*start)( void );

/\*\* Stops navigating. \*/

int (\*stop)( void );

/\*\* Closes the interface. \*/

void (\*cleanup)( void );

/\*\* Injects the current time. \*/

int (\*inject\_time)(GpsUtcTime time, int64\_t timeReference,

int uncertainty);

/\*\*

\* Injects current location from another location provider (typically cell

\* ID). Latitude and longitude are measured in degrees expected accuracy is

\* measured in meters

\*/

int (\*inject\_location)(double latitude, double longitude, float accuracy);

/\*\*

\* Specifies that the next call to start will not use the

\* information defined in the flags. GPS\_DELETE\_ALL is passed for

\* a cold start.

\*/

void (\*delete\_aiding\_data)(GpsAidingData flags);

/\*\*

\* min\_interval represents the time between fixes in milliseconds.

\* preferred\_accuracy represents the requested fix accuracy in meters.

\* preferred\_time represents the requested time to first fix in milliseconds.

\*

\* 'mode' parameter should be one of GPS\_POSITION\_MODE\_MS\_BASED

\* or GPS\_POSITION\_MODE\_STANDALONE.

\* It is allowed by the platform (and it is recommended) to fallback to

\* GPS\_POSITION\_MODE\_MS\_BASED if GPS\_POSITION\_MODE\_MS\_ASSISTED is passed in, and

\* GPS\_POSITION\_MODE\_MS\_BASED is supported.

\*/

int (\*set\_position\_mode)(GpsPositionMode mode, GpsPositionRecurrence recurrence,

uint32\_t min\_interval, uint32\_t preferred\_accuracy, uint32\_t preferred\_time);

/\*\* Get a pointer to extension information. \*/

const void\* (\*get\_extension)(const char\* name);

} GpsInterface;

/\*\*

\* Callback to request the client to download XTRA data. The client should

\* download XTRA data and inject it by calling inject\_xtra\_data(). Can only be

\* called from a thread created by create\_thread\_cb.

\*/

typedef void (\* gps\_xtra\_download\_request)();

/\*\* Callback structure for the XTRA interface. \*/

typedef struct {

gps\_xtra\_download\_request download\_request\_cb;

gps\_create\_thread create\_thread\_cb;

} GpsXtraCallbacks;

/\*\* Extended interface for XTRA support. \*/

typedef struct {

/\*\* set to sizeof(GpsXtraInterface) \*/

size\_t size;

/\*\*

\* Opens the XTRA interface and provides the callback routines

\* to the implementation of this interface.

\*/

int (\*init)( GpsXtraCallbacks\* callbacks );

/\*\* Injects XTRA data into the GPS. \*/

int (\*inject\_xtra\_data)( char\* data, int length );

} GpsXtraInterface;

/\*\* Extended interface for DEBUG support. \*/

typedef struct {

/\*\* set to sizeof(GpsDebugInterface) \*/

size\_t size;

/\*\*

\* This function should return any information that the native

\* implementation wishes to include in a bugreport.

\*/

size\_t (\*get\_internal\_state)(char\* buffer, size\_t bufferSize);

} GpsDebugInterface;

/\*

\* Represents the status of AGPS augmented to support IPv4 and IPv6.

\*/

typedef struct {

/\*\* set to sizeof(AGpsStatus) \*/

size\_t size;

AGpsType type;

AGpsStatusValue status;

/\*\*

\* Must be set to a valid IPv4 address if the field 'addr' contains an IPv4

\* address, or set to INADDR\_NONE otherwise.

\*/

uint32\_t ipaddr;

/\*\*

\* Must contain the IPv4 (AF\_INET) or IPv6 (AF\_INET6) address to report.

\* Any other value of addr.ss\_family will be rejected.

\*/

struct sockaddr\_storage addr;

} AGpsStatus;

/\*\*

\* Callback with AGPS status information. Can only be called from a thread

\* created by create\_thread\_cb.

\*/

typedef void (\* agps\_status\_callback)(AGpsStatus\* status);

/\*\* Callback structure for the AGPS interface. \*/

typedef struct {

agps\_status\_callback status\_cb;

gps\_create\_thread create\_thread\_cb;

} AGpsCallbacks;

/\*\*

\* Extended interface for AGPS support, it is augmented to enable to pass

\* extra APN data.

\*/

typedef struct {

/\*\* set to sizeof(AGpsInterface) \*/

size\_t size;

/\*\*

\* Opens the AGPS interface and provides the callback routines to the

\* implementation of this interface.

\*/

void (\*init)(AGpsCallbacks\* callbacks);

/\*\*

\* Deprecated.

\* If the HAL supports AGpsInterface\_v2 this API will not be used, see

\* data\_conn\_open\_with\_apn\_ip\_type for more information.

\*/

int (\*data\_conn\_open)(const char\* apn);

/\*\*

\* Notifies that the AGPS data connection has been closed.

\*/

int (\*data\_conn\_closed)();

/\*\*

\* Notifies that a data connection is not available for AGPS.

\*/

int (\*data\_conn\_failed)();

/\*\*

\* Sets the hostname and port for the AGPS server.

\*/

int (\*set\_server)(AGpsType type, const char\* hostname, int port);

/\*\*

\* Notifies that a data connection is available and sets the name of the

\* APN, and its IP type, to be used for SUPL connections.

\*/

int (\*data\_conn\_open\_with\_apn\_ip\_type)(

const char\* apn,

ApnIpType apnIpType);

} AGpsInterface;

/\*\* Error codes associated with certificate operations \*/

#define AGPS\_CERTIFICATE\_OPERATION\_SUCCESS 0

#define AGPS\_CERTIFICATE\_ERROR\_GENERIC -100

#define AGPS\_CERTIFICATE\_ERROR\_TOO\_MANY\_CERTIFICATES -101

/\*\* A data structure that represents an X.509 certificate using DER encoding \*/

typedef struct {

size\_t length;

u\_char\* data;

} DerEncodedCertificate;

/\*\*

\* A type definition for SHA1 Fingerprints used to identify X.509 Certificates

\* The Fingerprint is a digest of the DER Certificate that uniquely identifies it.

\*/

typedef struct {

u\_char data[20];

} Sha1CertificateFingerprint;

/\*\* AGPS Interface to handle SUPL certificate operations \*/

typedef struct {

/\*\* set to sizeof(SuplCertificateInterface) \*/

size\_t size;

/\*\*

\* Installs a set of Certificates used for SUPL connections to the AGPS server.

\* If needed the HAL should find out internally any certificates that need to be removed to

\* accommodate the certificates to install.

\* The certificates installed represent a full set of valid certificates needed to connect to

\* AGPS SUPL servers.

\* The list of certificates is required, and all must be available at the same time, when trying

\* to establish a connection with the AGPS Server.

\*

\* Parameters:

\* certificates - A pointer to an array of DER encoded certificates that are need to be

\* installed in the HAL.

\* length - The number of certificates to install.

\* Returns:

\* AGPS\_CERTIFICATE\_OPERATION\_SUCCESS if the operation is completed successfully

\* AGPS\_CERTIFICATE\_ERROR\_TOO\_MANY\_CERTIFICATES if the HAL cannot store the number of

\* certificates attempted to be installed, the state of the certificates stored should

\* remain the same as before on this error case.

\*

\* IMPORTANT:

\* If needed the HAL should find out internally the set of certificates that need to be

\* removed to accommodate the certificates to install.

\*/

int (\*install\_certificates) ( const DerEncodedCertificate\* certificates, size\_t length );

/\*\*

\* Notifies the HAL that a list of certificates used for SUPL connections are revoked. It is

\* expected that the given set of certificates is removed from the internal store of the HAL.

\*

\* Parameters:

\* fingerprints - A pointer to an array of SHA1 Fingerprints to identify the set of

\* certificates to revoke.

\* length - The number of fingerprints provided.

\* Returns:

\* AGPS\_CERTIFICATE\_OPERATION\_SUCCESS if the operation is completed successfully.

\*

\* IMPORTANT:

\* If any of the certificates provided (through its fingerprint) is not known by the HAL,

\* it should be ignored and continue revoking/deleting the rest of them.

\*/

int (\*revoke\_certificates) ( const Sha1CertificateFingerprint\* fingerprints, size\_t length );

} SuplCertificateInterface;

/\*\* Represents an NI request \*/

typedef struct {

/\*\* set to sizeof(GpsNiNotification) \*/

size\_t size;

/\*\*

\* An ID generated by HAL to associate NI notifications and UI

\* responses

\*/

int notification\_id;

/\*\*

\* An NI type used to distinguish different categories of NI

\* events, such as GPS\_NI\_TYPE\_VOICE, GPS\_NI\_TYPE\_UMTS\_SUPL, ...

\*/

GpsNiType ni\_type;

/\*\*

\* Notification/verification options, combinations of GpsNiNotifyFlags constants

\*/

GpsNiNotifyFlags notify\_flags;

/\*\*

\* Timeout period to wait for user response.

\* Set to 0 for no time out limit.

\*/

int timeout;

/\*\*

\* Default response when time out.

\*/

GpsUserResponseType default\_response;

/\*\*

\* Requestor ID

\*/

char requestor\_id[GPS\_NI\_SHORT\_STRING\_MAXLEN];

/\*\*

\* Notification message. It can also be used to store client\_id in some cases

\*/

char text[GPS\_NI\_LONG\_STRING\_MAXLEN];

/\*\*

\* Client name decoding scheme

\*/

GpsNiEncodingType requestor\_id\_encoding;

/\*\*

\* Client name decoding scheme

\*/

GpsNiEncodingType text\_encoding;

/\*\*

\* A pointer to extra data. Format:

\* key\_1 = value\_1

\* key\_2 = value\_2

\*/

char extras[GPS\_NI\_LONG\_STRING\_MAXLEN];

} GpsNiNotification;

/\*\*

\* Callback with NI notification. Can only be called from a thread created by

\* create\_thread\_cb.

\*/

typedef void (\*gps\_ni\_notify\_callback)(GpsNiNotification \*notification);

/\*\* GPS NI callback structure. \*/

typedef struct

{

/\*\*

\* Sends the notification request from HAL to GPSLocationProvider.

\*/

gps\_ni\_notify\_callback notify\_cb;

gps\_create\_thread create\_thread\_cb;

} GpsNiCallbacks;

/\*\*

\* Extended interface for Network-initiated (NI) support.

\*/

typedef struct

{

/\*\* set to sizeof(GpsNiInterface) \*/

size\_t size;

/\*\* Registers the callbacks for HAL to use. \*/

void (\*init) (GpsNiCallbacks \*callbacks);

/\*\* Sends a response to HAL. \*/

void (\*respond) (int notif\_id, GpsUserResponseType user\_response);

} GpsNiInterface;

struct gps\_device\_t {

struct hw\_device\_t common;

/\*\*

\* Set the provided lights to the provided values.

\*

\* Returns: 0 on succes, error code on failure.

\*/

const GpsInterface\* (\*get\_gps\_interface)(struct gps\_device\_t\* dev);

};

#define AGPS\_RIL\_REQUEST\_SETID\_IMSI (1<<0L)

#define AGPS\_RIL\_REQUEST\_SETID\_MSISDN (1<<1L)

#define AGPS\_RIL\_REQUEST\_REFLOC\_CELLID (1<<0L)

#define AGPS\_RIL\_REQUEST\_REFLOC\_MAC (1<<1L)

typedef void (\*agps\_ril\_request\_set\_id)(uint32\_t flags);

typedef void (\*agps\_ril\_request\_ref\_loc)(uint32\_t flags);

typedef struct {

agps\_ril\_request\_set\_id request\_setid;

agps\_ril\_request\_ref\_loc request\_refloc;

gps\_create\_thread create\_thread\_cb;

} AGpsRilCallbacks;

/\*\* Extended interface for AGPS\_RIL support. \*/

typedef struct {

/\*\* set to sizeof(AGpsRilInterface) \*/

size\_t size;

/\*\*

\* Opens the AGPS interface and provides the callback routines

\* to the implementation of this interface.

\*/

void (\*init)( AGpsRilCallbacks\* callbacks );

/\*\*

\* Sets the reference location.

\*/

void (\*set\_ref\_location) (const AGpsRefLocation \*agps\_reflocation, size\_t sz\_struct);

/\*\*

\* Sets the set ID.

\*/

void (\*set\_set\_id) (AGpsSetIDType type, const char\* setid);

/\*\*

\* Send network initiated message.

\*/

void (\*ni\_message) (uint8\_t \*msg, size\_t len);

/\*\*

\* Notify GPS of network status changes.

\* These parameters match values in the android.net.NetworkInfo class.

\*/

void (\*update\_network\_state) (int connected, int type, int roaming, const char\* extra\_info);

/\*\*

\* Notify GPS of network status changes.

\* These parameters match values in the android.net.NetworkInfo class.

\*/

void (\*update\_network\_availability) (int avaiable, const char\* apn);

} AGpsRilInterface;

/\*\*

\* GPS Geofence.

\* There are 3 states associated with a Geofence: Inside, Outside, Unknown.

\* There are 3 transitions: ENTERED, EXITED, UNCERTAIN.

\*

\* An example state diagram with confidence level: 95% and Unknown time limit

\* set as 30 secs is shown below. (confidence level and Unknown time limit are

\* explained latter)

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* | Unknown (30 secs) |

\* """"""""""""""""""""""""""""

\* ^ | | ^

\* UNCERTAIN| |ENTERED EXITED| |UNCERTAIN

\* | v v |

\* \_\_\_\_\_\_\_\_ EXITED \_\_\_\_\_\_\_\_\_

\* | Inside | -----------> | Outside |

\* | | <----------- | |

\* """""""" ENTERED """""""""

\*

\* Inside state: We are 95% confident that the user is inside the geofence.

\* Outside state: We are 95% confident that the user is outside the geofence

\* Unknown state: Rest of the time.

\*

\* The Unknown state is better explained with an example:

\*

\* \_\_\_\_\_\_\_\_\_\_

\* | c|

\* | \_\_\_ | \_\_\_\_\_\_\_

\* | |a| | | b |

\* | """ | """""""

\* | |

\* """"""""""

\* In the diagram above, "a" and "b" are 2 geofences and "c" is the accuracy

\* circle reported by the GPS subsystem. Now with regard to "b", the system is

\* confident that the user is outside. But with regard to "a" is not confident

\* whether it is inside or outside the geofence. If the accuracy remains the

\* same for a sufficient period of time, the UNCERTAIN transition would be

\* triggered with the state set to Unknown. If the accuracy improves later, an

\* appropriate transition should be triggered. This "sufficient period of time"

\* is defined by the parameter in the add\_geofence\_area API.

\* In other words, Unknown state can be interpreted as a state in which the

\* GPS subsystem isn't confident enough that the user is either inside or

\* outside the Geofence. It moves to Unknown state only after the expiry of the

\* timeout.

\*

\* The geofence callback needs to be triggered for the ENTERED and EXITED

\* transitions, when the GPS system is confident that the user has entered

\* (Inside state) or exited (Outside state) the Geofence. An implementation

\* which uses a value of 95% as the confidence is recommended. The callback

\* should be triggered only for the transitions requested by the

\* add\_geofence\_area call.

\*

\* Even though the diagram and explanation talks about states and transitions,

\* the callee is only interested in the transistions. The states are mentioned

\* here for illustrative purposes.

\*

\* Startup Scenario: When the device boots up, if an application adds geofences,

\* and then we get an accurate GPS location fix, it needs to trigger the

\* appropriate (ENTERED or EXITED) transition for every Geofence it knows about.

\* By default, all the Geofences will be in the Unknown state.

\*

\* When the GPS system is unavailable, gps\_geofence\_status\_callback should be

\* called to inform the upper layers of the same. Similarly, when it becomes

\* available the callback should be called. This is a global state while the

\* UNKNOWN transition described above is per geofence.

\*

\* An important aspect to note is that users of this API (framework), will use

\* other subsystems like wifi, sensors, cell to handle Unknown case and

\* hopefully provide a definitive state transition to the third party

\* application. GPS Geofence will just be a signal indicating what the GPS

\* subsystem knows about the Geofence.

\*

\*/

#define GPS\_GEOFENCE\_ENTERED (1<<0L)

#define GPS\_GEOFENCE\_EXITED (1<<1L)

#define GPS\_GEOFENCE\_UNCERTAIN (1<<2L)

#define GPS\_GEOFENCE\_UNAVAILABLE (1<<0L)

#define GPS\_GEOFENCE\_AVAILABLE (1<<1L)

#define GPS\_GEOFENCE\_OPERATION\_SUCCESS 0

#define GPS\_GEOFENCE\_ERROR\_TOO\_MANY\_GEOFENCES -100

#define GPS\_GEOFENCE\_ERROR\_ID\_EXISTS -101

#define GPS\_GEOFENCE\_ERROR\_ID\_UNKNOWN -102

#define GPS\_GEOFENCE\_ERROR\_INVALID\_TRANSITION -103

#define GPS\_GEOFENCE\_ERROR\_GENERIC -149

/\*\*

\* The callback associated with the geofence.

\* Parameters:

\* geofence\_id - The id associated with the add\_geofence\_area.

\* location - The current GPS location.

\* transition - Can be one of GPS\_GEOFENCE\_ENTERED, GPS\_GEOFENCE\_EXITED,

\* GPS\_GEOFENCE\_UNCERTAIN.

\* timestamp - Timestamp when the transition was detected.

\*

\* The callback should only be called when the caller is interested in that

\* particular transition. For instance, if the caller is interested only in

\* ENTERED transition, then the callback should NOT be called with the EXITED

\* transition.

\*

\* IMPORTANT: If a transition is triggered resulting in this callback, the GPS

\* subsystem will wake up the application processor, if its in suspend state.

\*/

typedef void (\*gps\_geofence\_transition\_callback) (int32\_t geofence\_id, GpsLocation\* location,

int32\_t transition, GpsUtcTime timestamp);

/\*\*

\* The callback associated with the availability of the GPS system for geofencing

\* monitoring. If the GPS system determines that it cannot monitor geofences

\* because of lack of reliability or unavailability of the GPS signals, it will

\* call this callback with GPS\_GEOFENCE\_UNAVAILABLE parameter.

\*

\* Parameters:

\* status - GPS\_GEOFENCE\_UNAVAILABLE or GPS\_GEOFENCE\_AVAILABLE.

\* last\_location - Last known location.

\*/

typedef void (\*gps\_geofence\_status\_callback) (int32\_t status, GpsLocation\* last\_location);

/\*\*

\* The callback associated with the add\_geofence call.

\*

\* Parameter:

\* geofence\_id - Id of the geofence.

\* status - GPS\_GEOFENCE\_OPERATION\_SUCCESS

\* GPS\_GEOFENCE\_ERROR\_TOO\_MANY\_GEOFENCES - geofence limit has been reached.

\* GPS\_GEOFENCE\_ERROR\_ID\_EXISTS - geofence with id already exists

\* GPS\_GEOFENCE\_ERROR\_INVALID\_TRANSITION - the monitorTransition contains an

\* invalid transition

\* GPS\_GEOFENCE\_ERROR\_GENERIC - for other errors.

\*/

typedef void (\*gps\_geofence\_add\_callback) (int32\_t geofence\_id, int32\_t status);

/\*\*

\* The callback associated with the remove\_geofence call.

\*

\* Parameter:

\* geofence\_id - Id of the geofence.

\* status - GPS\_GEOFENCE\_OPERATION\_SUCCESS

\* GPS\_GEOFENCE\_ERROR\_ID\_UNKNOWN - for invalid id

\* GPS\_GEOFENCE\_ERROR\_GENERIC for others.

\*/

typedef void (\*gps\_geofence\_remove\_callback) (int32\_t geofence\_id, int32\_t status);

/\*\*

\* The callback associated with the pause\_geofence call.

\*

\* Parameter:

\* geofence\_id - Id of the geofence.

\* status - GPS\_GEOFENCE\_OPERATION\_SUCCESS

\* GPS\_GEOFENCE\_ERROR\_ID\_UNKNOWN - for invalid id

\* GPS\_GEOFENCE\_ERROR\_INVALID\_TRANSITION -

\* when monitor\_transitions is invalid

\* GPS\_GEOFENCE\_ERROR\_GENERIC for others.

\*/

typedef void (\*gps\_geofence\_pause\_callback) (int32\_t geofence\_id, int32\_t status);

/\*\*

\* The callback associated with the resume\_geofence call.

\*

\* Parameter:

\* geofence\_id - Id of the geofence.

\* status - GPS\_GEOFENCE\_OPERATION\_SUCCESS

\* GPS\_GEOFENCE\_ERROR\_ID\_UNKNOWN - for invalid id

\* GPS\_GEOFENCE\_ERROR\_GENERIC for others.

\*/

typedef void (\*gps\_geofence\_resume\_callback) (int32\_t geofence\_id, int32\_t status);

typedef struct {

gps\_geofence\_transition\_callback geofence\_transition\_callback;

gps\_geofence\_status\_callback geofence\_status\_callback;

gps\_geofence\_add\_callback geofence\_add\_callback;

gps\_geofence\_remove\_callback geofence\_remove\_callback;

gps\_geofence\_pause\_callback geofence\_pause\_callback;

gps\_geofence\_resume\_callback geofence\_resume\_callback;

gps\_create\_thread create\_thread\_cb;

} GpsGeofenceCallbacks;

/\*\* Extended interface for GPS\_Geofencing support \*/

typedef struct {

/\*\* set to sizeof(GpsGeofencingInterface) \*/

size\_t size;

/\*\*

\* Opens the geofence interface and provides the callback routines

\* to the implementation of this interface.

\*/

void (\*init)( GpsGeofenceCallbacks\* callbacks );

/\*\*

\* Add a geofence area. This api currently supports circular geofences.

\* Parameters:

\* geofence\_id - The id for the geofence. If a geofence with this id

\* already exists, an error value (GPS\_GEOFENCE\_ERROR\_ID\_EXISTS)

\* should be returned.

\* latitude, longtitude, radius\_meters - The lat, long and radius

\* (in meters) for the geofence

\* last\_transition - The current state of the geofence. For example, if

\* the system already knows that the user is inside the geofence,

\* this will be set to GPS\_GEOFENCE\_ENTERED. In most cases, it

\* will be GPS\_GEOFENCE\_UNCERTAIN.

\* monitor\_transition - Which transitions to monitor. Bitwise OR of

\* GPS\_GEOFENCE\_ENTERED, GPS\_GEOFENCE\_EXITED and

\* GPS\_GEOFENCE\_UNCERTAIN.

\* notification\_responsiveness\_ms - Defines the best-effort description

\* of how soon should the callback be called when the transition

\* associated with the Geofence is triggered. For instance, if set

\* to 1000 millseconds with GPS\_GEOFENCE\_ENTERED, the callback

\* should be called 1000 milliseconds within entering the geofence.

\* This parameter is defined in milliseconds.

\* NOTE: This is not to be confused with the rate that the GPS is

\* polled at. It is acceptable to dynamically vary the rate of

\* sampling the GPS for power-saving reasons; thus the rate of

\* sampling may be faster or slower than this.

\* unknown\_timer\_ms - The time limit after which the UNCERTAIN transition

\* should be triggered. This parameter is defined in milliseconds.

\* See above for a detailed explanation.

\*/

void (\*add\_geofence\_area) (int32\_t geofence\_id, double latitude, double longitude,

double radius\_meters, int last\_transition, int monitor\_transitions,

int notification\_responsiveness\_ms, int unknown\_timer\_ms);

/\*\*

\* Pause monitoring a particular geofence.

\* Parameters:

\* geofence\_id - The id for the geofence.

\*/

void (\*pause\_geofence) (int32\_t geofence\_id);

/\*\*

\* Resume monitoring a particular geofence.

\* Parameters:

\* geofence\_id - The id for the geofence.

\* monitor\_transitions - Which transitions to monitor. Bitwise OR of

\* GPS\_GEOFENCE\_ENTERED, GPS\_GEOFENCE\_EXITED and

\* GPS\_GEOFENCE\_UNCERTAIN.

\* This supersedes the value associated provided in the

\* add\_geofence\_area call.

\*/

void (\*resume\_geofence) (int32\_t geofence\_id, int monitor\_transitions);

/\*\*

\* Remove a geofence area. After the function returns, no notifications

\* should be sent.

\* Parameter:

\* geofence\_id - The id for the geofence.

\*/

void (\*remove\_geofence\_area) (int32\_t geofence\_id);

} GpsGeofencingInterface;

/\*\*

\* Legacy struct to represent an estimate of the GPS clock time.

\* Deprecated, to be removed in the next Android release.

\* Use GnssClock instead.

\*/

typedef struct {

/\*\* set to sizeof(GpsClock) \*/

size\_t size;

GpsClockFlags flags;

int16\_t leap\_second;

GpsClockType type;

int64\_t time\_ns;

double time\_uncertainty\_ns;

int64\_t full\_bias\_ns;

double bias\_ns;

double bias\_uncertainty\_ns;

double drift\_nsps;

double drift\_uncertainty\_nsps;

} GpsClock;

/\*\*

\* Represents an estimate of the GPS clock time.

\*/

typedef struct {

/\*\* set to sizeof(GnssClock) \*/

size\_t size;

/\*\*

\* A set of flags indicating the validity of the fields in this data

\* structure.

\*/

GnssClockFlags flags;

/\*\*

\* Leap second data.

\* The sign of the value is defined by the following equation:

\* utc\_time\_ns = time\_ns - (full\_bias\_ns + bias\_ns) - leap\_second \*

\* 1,000,000,000

\*

\* If the data is available 'flags' must contain GNSS\_CLOCK\_HAS\_LEAP\_SECOND.

\*/

int16\_t leap\_second;

/\*\*

\* The GNSS receiver internal clock value. This is the local hardware clock

\* value.

\*

\* For local hardware clock, this value is expected to be monotonically

\* increasing while the hardware clock remains power on. (For the case of a

\* HW clock that is not continuously on, see the

\* hw\_clock\_discontinuity\_count field). The receiver's estimate of GPS time

\* can be derived by substracting the sum of full\_bias\_ns and bias\_ns (when

\* available) from this value.

\*

\* This GPS time is expected to be the best estimate of current GPS time

\* that GNSS receiver can achieve.

\*

\* Sub-nanosecond accuracy can be provided by means of the 'bias\_ns' field.

\* The value contains the 'time uncertainty' in it.

\*

\* This field is mandatory.

\*/

int64\_t time\_ns;

/\*\*

\* 1-Sigma uncertainty associated with the clock's time in nanoseconds.

\* The uncertainty is represented as an absolute (single sided) value.

\*

\* If the data is available, 'flags' must contain

\* GNSS\_CLOCK\_HAS\_TIME\_UNCERTAINTY. This value is effectively zero (it is

\* the reference local clock, by which all other times and time

\* uncertainties are measured.) (And thus this field can be not provided,

\* per GNSS\_CLOCK\_HAS\_TIME\_UNCERTAINTY flag, or provided & set to 0.)

\*/

double time\_uncertainty\_ns;

/\*\*

\* The difference between hardware clock ('time' field) inside GPS receiver

\* and the true GPS time since 0000Z, January 6, 1980, in nanoseconds.

\*

\* The sign of the value is defined by the following equation:

\* local estimate of GPS time = time\_ns - (full\_bias\_ns + bias\_ns)

\*

\* This value is mandatory if the receiver has estimated GPS time. If the

\* computed time is for a non-GPS constellation, the time offset of that

\* constellation to GPS has to be applied to fill this value. The error

\* estimate for the sum of this and the bias\_ns is the bias\_uncertainty\_ns,

\* and the caller is responsible for using this uncertainty (it can be very

\* large before the GPS time has been solved for.) If the data is available

\* 'flags' must contain GNSS\_CLOCK\_HAS\_FULL\_BIAS.

\*/

int64\_t full\_bias\_ns;

/\*\*

\* Sub-nanosecond bias.

\* The error estimate for the sum of this and the full\_bias\_ns is the

\* bias\_uncertainty\_ns

\*

\* If the data is available 'flags' must contain GNSS\_CLOCK\_HAS\_BIAS. If GPS

\* has computed a position fix. This value is mandatory if the receiver has

\* estimated GPS time.

\*/

double bias\_ns;

/\*\*

\* 1-Sigma uncertainty associated with the local estimate of GPS time (clock

\* bias) in nanoseconds. The uncertainty is represented as an absolute

\* (single sided) value.

\*

\* If the data is available 'flags' must contain

\* GNSS\_CLOCK\_HAS\_BIAS\_UNCERTAINTY. This value is mandatory if the receiver

\* has estimated GPS time.

\*/

double bias\_uncertainty\_ns;

/\*\*

\* The clock's drift in nanoseconds (per second).

\*

\* A positive value means that the frequency is higher than the nominal

\* frequency, and that the (full\_bias\_ns + bias\_ns) is growing more positive

\* over time.

\*

\* The value contains the 'drift uncertainty' in it.

\* If the data is available 'flags' must contain GNSS\_CLOCK\_HAS\_DRIFT.

\*

\* This value is mandatory if the receiver has estimated GNSS time

\*/

double drift\_nsps;

/\*\*

\* 1-Sigma uncertainty associated with the clock's drift in nanoseconds (per second).

\* The uncertainty is represented as an absolute (single sided) value.

\*

\* If the data is available 'flags' must contain

\* GNSS\_CLOCK\_HAS\_DRIFT\_UNCERTAINTY. If GPS has computed a position fix this

\* field is mandatory and must be populated.

\*/

double drift\_uncertainty\_nsps;

/\*\*

\* When there are any discontinuities in the HW clock, this field is

\* mandatory.

\*

\* A "discontinuity" is meant to cover the case of a switch from one source

\* of clock to another. A single free-running crystal oscillator (XO)

\* should generally not have any discontinuities, and this can be set and

\* left at 0.

\*

\* If, however, the time\_ns value (HW clock) is derived from a composite of

\* sources, that is not as smooth as a typical XO, or is otherwise stopped &

\* restarted, then this value shall be incremented each time a discontinuity

\* occurs. (E.g. this value may start at zero at device boot-up and

\* increment each time there is a change in clock continuity. In the

\* unlikely event that this value reaches full scale, rollover (not

\* clamping) is required, such that this value continues to change, during

\* subsequent discontinuity events.)

\*

\* While this number stays the same, between GnssClock reports, it can be

\* safely assumed that the time\_ns value has been running continuously, e.g.

\* derived from a single, high quality clock (XO like, or better, that's

\* typically used during continuous GNSS signal sampling.)

\*

\* It is expected, esp. during periods where there are few GNSS signals

\* available, that the HW clock be discontinuity-free as long as possible,

\* as this avoids the need to use (waste) a GNSS measurement to fully

\* re-solve for the GPS clock bias and drift, when using the accompanying

\* measurements, from consecutive GnssData reports.

\*/

uint32\_t hw\_clock\_discontinuity\_count;

} GnssClock;

/\*\*

\* Legacy struct to represent a GPS Measurement, it contains raw and computed

\* information.

\* Deprecated, to be removed in the next Android release.

\* Use GnssMeasurement instead.

\*/

typedef struct {

/\*\* set to sizeof(GpsMeasurement) \*/

size\_t size;

GpsMeasurementFlags flags;

int8\_t prn;

double time\_offset\_ns;

GpsMeasurementState state;

int64\_t received\_gps\_tow\_ns;

int64\_t received\_gps\_tow\_uncertainty\_ns;

double c\_n0\_dbhz;

double pseudorange\_rate\_mps;

double pseudorange\_rate\_uncertainty\_mps;

GpsAccumulatedDeltaRangeState accumulated\_delta\_range\_state;

double accumulated\_delta\_range\_m;

double accumulated\_delta\_range\_uncertainty\_m;

double pseudorange\_m;

double pseudorange\_uncertainty\_m;

double code\_phase\_chips;

double code\_phase\_uncertainty\_chips;

float carrier\_frequency\_hz;

int64\_t carrier\_cycles;

double carrier\_phase;

double carrier\_phase\_uncertainty;

GpsLossOfLock loss\_of\_lock;

int32\_t bit\_number;

int16\_t time\_from\_last\_bit\_ms;

double doppler\_shift\_hz;

double doppler\_shift\_uncertainty\_hz;

GpsMultipathIndicator multipath\_indicator;

double snr\_db;

double elevation\_deg;

double elevation\_uncertainty\_deg;

double azimuth\_deg;

double azimuth\_uncertainty\_deg;

bool used\_in\_fix;

} GpsMeasurement;

/\*\*

\* Represents a GNSS Measurement, it contains raw and computed information.

\*

\* Independence - All signal measurement information (e.g. sv\_time,

\* pseudorange\_rate, multipath\_indicator) reported in this struct should be

\* based on GNSS signal measurements only. You may not synthesize measurements

\* by calculating or reporting expected measurements based on known or estimated

\* position, velocity, or time.

\*/

typedef struct {

/\*\* set to sizeof(GpsMeasurement) \*/

size\_t size;

/\*\* A set of flags indicating the validity of the fields in this data structure. \*/

GnssMeasurementFlags flags;

/\*\*

\* Satellite vehicle ID number, as defined in GnssSvInfo::svid

\* This is a mandatory value.

\*/

int16\_t svid;

/\*\*

\* Defines the constellation of the given SV. Value should be one of those

\* GNSS\_CONSTELLATION\_\* constants

\*/

GnssConstellationType constellation;

/\*\*

\* Time offset at which the measurement was taken in nanoseconds.

\* The reference receiver's time is specified by GpsData::clock::time\_ns and should be

\* interpreted in the same way as indicated by GpsClock::type.

\*

\* The sign of time\_offset\_ns is given by the following equation:

\* measurement time = GpsClock::time\_ns + time\_offset\_ns

\*

\* It provides an individual time-stamp for the measurement, and allows sub-nanosecond accuracy.

\* This is a mandatory value.

\*/

double time\_offset\_ns;

/\*\*

\* Per satellite sync state. It represents the current sync state for the associated satellite.

\* Based on the sync state, the 'received GPS tow' field should be interpreted accordingly.

\*

\* This is a mandatory value.

\*/

GnssMeasurementState state;

/\*\*

\* The received GNSS Time-of-Week at the measurement time, in nanoseconds.

\* Ensure that this field is independent (see comment at top of

\* GnssMeasurement struct.)

\*

\* For GPS & QZSS, this is:

\* Received GPS Time-of-Week at the measurement time, in nanoseconds.

\* The value is relative to the beginning of the current GPS week.

\*

\* Given the highest sync state that can be achieved, per each satellite, valid range

\* for this field can be:

\* Searching : [ 0 ] : GNSS\_MEASUREMENT\_STATE\_UNKNOWN

\* C/A code lock : [ 0 1ms ] : GNSS\_MEASUREMENT\_STATE\_CODE\_LOCK is set

\* Bit sync : [ 0 20ms ] : GNSS\_MEASUREMENT\_STATE\_BIT\_SYNC is set

\* Subframe sync : [ 0 6s ] : GNSS\_MEASUREMENT\_STATE\_SUBFRAME\_SYNC is set

\* TOW decoded : [ 0 1week ] : GNSS\_MEASUREMENT\_STATE\_TOW\_DECODED is set

\*

\* Note well: if there is any ambiguity in integer millisecond,

\* GNSS\_MEASUREMENT\_STATE\_MSEC\_AMBIGUOUS should be set accordingly, in the 'state' field.

\*

\* This value must be populated if 'state' != GNSS\_MEASUREMENT\_STATE\_UNKNOWN.

\*

\* For Glonass, this is:

\* Received Glonass time of day, at the measurement time in nanoseconds.

\*

\* Given the highest sync state that can be achieved, per each satellite, valid range for

\* this field can be:

\* Searching : [ 0 ] : GNSS\_MEASUREMENT\_STATE\_UNKNOWN

\* C/A code lock : [ 0 1ms ] : GNSS\_MEASUREMENT\_STATE\_CODE\_LOCK is set

\* Symbol sync : [ 0 10ms ] : GNSS\_MEASUREMENT\_STATE\_SYMBOL\_SYNC is set

\* Bit sync : [ 0 20ms ] : GNSS\_MEASUREMENT\_STATE\_BIT\_SYNC is set

\* String sync : [ 0 2s ] : GNSS\_MEASUREMENT\_STATE\_GLO\_STRING\_SYNC is set

\* Time of day : [ 0 1day ] : GNSS\_MEASUREMENT\_STATE\_GLO\_TOD\_DECODED is set

\*

\* For Beidou, this is:

\* Received Beidou time of week, at the measurement time in nanoseconds.

\*

\* Given the highest sync state that can be achieved, per each satellite, valid range for

\* this field can be:

\* Searching : [ 0 ] : GNSS\_MEASUREMENT\_STATE\_UNKNOWN

\* C/A code lock: [ 0 1ms ] : GNSS\_MEASUREMENT\_STATE\_CODE\_LOCK is set

\* Bit sync (D2): [ 0 2ms ] : GNSS\_MEASUREMENT\_STATE\_BDS\_D2\_BIT\_SYNC is set

\* Bit sync (D1): [ 0 20ms ] : GNSS\_MEASUREMENT\_STATE\_BIT\_SYNC is set

\* Subframe (D2): [ 0 0.6s ] : GNSS\_MEASUREMENT\_STATE\_BDS\_D2\_SUBFRAME\_SYNC is set

\* Subframe (D1): [ 0 6s ] : GNSS\_MEASUREMENT\_STATE\_SUBFRAME\_SYNC is set

\* Time of week : [ 0 1week ] : GNSS\_MEASUREMENT\_STATE\_TOW\_DECODED is set

\*

\* For Galileo, this is:

\* Received Galileo time of week, at the measurement time in nanoseconds.

\*

\* E1BC code lock : [ 0 4ms ] : GNSS\_MEASUREMENT\_STATE\_GAL\_E1BC\_CODE\_LOCK is set

\* E1C 2nd code lock: [ 0 100ms ] :

\* GNSS\_MEASUREMENT\_STATE\_GAL\_E1C\_2ND\_CODE\_LOCK is set

\*

\* E1B page : [ 0 2s ] : GNSS\_MEASUREMENT\_STATE\_GAL\_E1B\_PAGE\_SYNC is set

\* Time of week: [ 0 1week ] : GNSS\_MEASUREMENT\_STATE\_TOW\_DECODED is set

\*

\* For SBAS, this is:

\* Received SBAS time, at the measurement time in nanoseconds.

\*

\* Given the highest sync state that can be achieved, per each satellite,

\* valid range for this field can be:

\* Searching : [ 0 ] : GNSS\_MEASUREMENT\_STATE\_UNKNOWN

\* C/A code lock: [ 0 1ms ] : GNSS\_MEASUREMENT\_STATE\_CODE\_LOCK is set

\* Symbol sync : [ 0 2ms ] : GNSS\_MEASUREMENT\_STATE\_SYMBOL\_SYNC is set

\* Message : [ 0 1s ] : GNSS\_MEASUREMENT\_STATE\_SBAS\_SYNC is set

\*/

int64\_t received\_sv\_time\_in\_ns;

/\*\*

\* 1-Sigma uncertainty of the Received GPS Time-of-Week in nanoseconds.

\*

\* This value must be populated if 'state' != GPS\_MEASUREMENT\_STATE\_UNKNOWN.

\*/

int64\_t received\_sv\_time\_uncertainty\_in\_ns;

/\*\*

\* Carrier-to-noise density in dB-Hz, typically in the range [0, 63].

\* It contains the measured C/N0 value for the signal at the antenna port.

\*

\* This is a mandatory value.

\*/

double c\_n0\_dbhz;

/\*\*

\* Pseudorange rate at the timestamp in m/s. The correction of a given

\* Pseudorange Rate value includes corrections for receiver and satellite

\* clock frequency errors. Ensure that this field is independent (see

\* comment at top of GnssMeasurement struct.)

\*

\* It is mandatory to provide the 'uncorrected' 'pseudorange rate', and provide GpsClock's

\* 'drift' field as well (When providing the uncorrected pseudorange rate, do not apply the

\* corrections described above.)

\*

\* The value includes the 'pseudorange rate uncertainty' in it.

\* A positive 'uncorrected' value indicates that the SV is moving away from the receiver.

\*

\* The sign of the 'uncorrected' 'pseudorange rate' and its relation to the sign of 'doppler

\* shift' is given by the equation:

\* pseudorange rate = -k \* doppler shift (where k is a constant)

\*

\* This should be the most accurate pseudorange rate available, based on

\* fresh signal measurements from this channel.

\*

\* It is mandatory that this value be provided at typical carrier phase PRR

\* quality (few cm/sec per second of uncertainty, or better) - when signals

\* are sufficiently strong & stable, e.g. signals from a GPS simulator at >=

\* 35 dB-Hz.

\*/

double pseudorange\_rate\_mps;

/\*\*

\* 1-Sigma uncertainty of the pseudorange\_rate\_mps.

\* The uncertainty is represented as an absolute (single sided) value.

\*

\* This is a mandatory value.

\*/

double pseudorange\_rate\_uncertainty\_mps;

/\*\*

\* Accumulated delta range's state. It indicates whether ADR is reset or there is a cycle slip

\* (indicating loss of lock).

\*

\* This is a mandatory value.

\*/

GnssAccumulatedDeltaRangeState accumulated\_delta\_range\_state;

/\*\*

\* Accumulated delta range since the last channel reset in meters.

\* A positive value indicates that the SV is moving away from the receiver.

\*

\* The sign of the 'accumulated delta range' and its relation to the sign of 'carrier phase'

\* is given by the equation:

\* accumulated delta range = -k \* carrier phase (where k is a constant)

\*

\* This value must be populated if 'accumulated delta range state' != GPS\_ADR\_STATE\_UNKNOWN.

\* However, it is expected that the data is only accurate when:

\* 'accumulated delta range state' == GPS\_ADR\_STATE\_VALID.

\*/

double accumulated\_delta\_range\_m;

/\*\*

\* 1-Sigma uncertainty of the accumulated delta range in meters.

\* This value must be populated if 'accumulated delta range state' != GPS\_ADR\_STATE\_UNKNOWN.

\*/

double accumulated\_delta\_range\_uncertainty\_m;

/\*\*

\* Carrier frequency at which codes and messages are modulated, it can be L1 or L2.

\* If the field is not set, the carrier frequency is assumed to be L1.

\*

\* If the data is available, 'flags' must contain

\* GNSS\_MEASUREMENT\_HAS\_CARRIER\_FREQUENCY.

\*/

float carrier\_frequency\_hz;

/\*\*

\* The number of full carrier cycles between the satellite and the receiver.

\* The reference frequency is given by the field 'carrier\_frequency\_hz'.

\* Indications of possible cycle slips and resets in the accumulation of

\* this value can be inferred from the accumulated\_delta\_range\_state flags.

\*

\* If the data is available, 'flags' must contain

\* GNSS\_MEASUREMENT\_HAS\_CARRIER\_CYCLES.

\*/

int64\_t carrier\_cycles;

/\*\*

\* The RF phase detected by the receiver, in the range [0.0, 1.0].

\* This is usually the fractional part of the complete carrier phase measurement.

\*

\* The reference frequency is given by the field 'carrier\_frequency\_hz'.

\* The value contains the 'carrier-phase uncertainty' in it.

\*

\* If the data is available, 'flags' must contain

\* GNSS\_MEASUREMENT\_HAS\_CARRIER\_PHASE.

\*/

double carrier\_phase;

/\*\*

\* 1-Sigma uncertainty of the carrier-phase.

\* If the data is available, 'flags' must contain

\* GNSS\_MEASUREMENT\_HAS\_CARRIER\_PHASE\_UNCERTAINTY.

\*/

double carrier\_phase\_uncertainty;

/\*\*

\* An enumeration that indicates the 'multipath' state of the event.

\*

\* The multipath Indicator is intended to report the presence of overlapping

\* signals that manifest as distorted correlation peaks.

\*

\* - if there is a distorted correlation peak shape, report that multipath

\* is GNSS\_MULTIPATH\_INDICATOR\_PRESENT.

\* - if there is not a distorted correlation peak shape, report

\* GNSS\_MULTIPATH\_INDICATOR\_NOT\_PRESENT

\* - if signals are too weak to discern this information, report

\* GNSS\_MULTIPATH\_INDICATOR\_UNKNOWN

\*

\* Example: when doing the standardized overlapping Multipath Performance

\* test (3GPP TS 34.171) the Multipath indicator should report

\* GNSS\_MULTIPATH\_INDICATOR\_PRESENT for those signals that are tracked, and

\* contain multipath, and GNSS\_MULTIPATH\_INDICATOR\_NOT\_PRESENT for those

\* signals that are tracked and do not contain multipath.

\*/

GnssMultipathIndicator multipath\_indicator;

/\*\*

\* Signal-to-noise ratio at correlator output in dB.

\* If the data is available, 'flags' must contain GNSS\_MEASUREMENT\_HAS\_SNR.

\* This is the power ratio of the "correlation peak height above the

\* observed noise floor" to "the noise RMS".

\*/

double snr\_db;

} GnssMeasurement;

/\*\*

\* Legacy struct to represents a reading of GPS measurements.

\* Deprecated, to be removed in the next Android release.

\* Use GnssData instead.

\*/

typedef struct {

/\*\* set to sizeof(GpsData) \*/

size\_t size;

size\_t measurement\_count;

GpsMeasurement measurements[GPS\_MAX\_MEASUREMENT];

/\*\* The GPS clock time reading. \*/

GpsClock clock;

} GpsData;

/\*\*

\* Represents a reading of GNSS measurements. For devices where GnssSystemInfo's

\* year\_of\_hw is set to 2016+, it is mandatory that these be provided, on

\* request, when the GNSS receiver is searching/tracking signals.

\*

\* - Reporting of GPS constellation measurements is mandatory.

\* - Reporting of all tracked constellations are encouraged.

\*/

typedef struct {

/\*\* set to sizeof(GnssData) \*/

size\_t size;

/\*\* Number of measurements. \*/

size\_t measurement\_count;

/\*\* The array of measurements. \*/

GnssMeasurement measurements[GNSS\_MAX\_MEASUREMENT];

/\*\* The GPS clock time reading. \*/

GnssClock clock;

} GnssData;

/\*\*

\* The legacy callback for to report measurements from the HAL.

\*

\* This callback is deprecated, and will be removed in the next release. Use

\* gnss\_measurement\_callback() instead.

\*

\* Parameters:

\* data - A data structure containing the measurements.

\*/

typedef void (\*gps\_measurement\_callback) (GpsData\* data);

/\*\*

\* The callback for to report measurements from the HAL.

\*

\* Parameters:

\* data - A data structure containing the measurements.

\*/

typedef void (\*gnss\_measurement\_callback) (GnssData\* data);

typedef struct {

/\*\* set to sizeof(GpsMeasurementCallbacks) \*/

size\_t size;

gps\_measurement\_callback measurement\_callback;

gnss\_measurement\_callback gnss\_measurement\_callback;

} GpsMeasurementCallbacks;

#define GPS\_MEASUREMENT\_OPERATION\_SUCCESS 0

#define GPS\_MEASUREMENT\_ERROR\_ALREADY\_INIT -100

#define GPS\_MEASUREMENT\_ERROR\_GENERIC -101

/\*\*

\* Extended interface for GPS Measurements support.

\*/

typedef struct {

/\*\* Set to sizeof(GpsMeasurementInterface) \*/

size\_t size;

/\*\*

\* Initializes the interface and registers the callback routines with the HAL.

\* After a successful call to 'init' the HAL must begin to provide updates at its own phase.

\*

\* Status:

\* GPS\_MEASUREMENT\_OPERATION\_SUCCESS

\* GPS\_MEASUREMENT\_ERROR\_ALREADY\_INIT - if a callback has already been registered without a

\* corresponding call to 'close'

\* GPS\_MEASUREMENT\_ERROR\_GENERIC - if any other error occurred, it is expected that the HAL

\* will not generate any updates upon returning this error code.

\*/

int (\*init) (GpsMeasurementCallbacks\* callbacks);

/\*\*

\* Stops updates from the HAL, and unregisters the callback routines.

\* After a call to stop, the previously registered callbacks must be considered invalid by the

\* HAL.

\* If stop is invoked without a previous 'init', this function should perform no work.

\*/

void (\*close) ();

} GpsMeasurementInterface;

/\*\*

\* Legacy struct to represents a GPS navigation message (or a fragment of it).

\* Deprecated, to be removed in the next Android release.

\* Use GnssNavigationMessage instead.

\*/

typedef struct {

/\*\* set to sizeof(GpsNavigationMessage) \*/

size\_t size;

int8\_t prn;

GpsNavigationMessageType type;

NavigationMessageStatus status;

int16\_t message\_id;

int16\_t submessage\_id;

size\_t data\_length;

uint8\_t\* data;

} GpsNavigationMessage;

/\*\* Represents a GPS navigation message (or a fragment of it). \*/

typedef struct {

/\*\* set to sizeof(GnssNavigationMessage) \*/

size\_t size;

/\*\*

\* Satellite vehicle ID number, as defined in GnssSvInfo::svid

\* This is a mandatory value.

\*/

int16\_t svid;

/\*\*

\* The type of message contained in the structure.

\* This is a mandatory value.

\*/

GnssNavigationMessageType type;

/\*\*

\* The status of the received navigation message.

\* No need to send any navigation message that contains words with parity error and cannot be

\* corrected.

\*/

NavigationMessageStatus status;

/\*\*

\* Message identifier. It provides an index so the complete Navigation

\* Message can be assembled.

\*

\* - For GPS L1 C/A subframe 4 and 5, this value corresponds to the 'frame

\* id' of the navigation message, in the range of 1-25 (Subframe 1, 2, 3

\* does not contain a 'frame id' and this value can be set to -1.)

\*

\* - For Glonass L1 C/A, this refers to the frame ID, in the range of 1-5.

\*

\* - For BeiDou D1, this refers to the frame number in the range of 1-24

\*

\* - For Beidou D2, this refers to the frame number, in the range of 1-120

\*

\* - For Galileo F/NAV nominal frame structure, this refers to the subframe

\* number, in the range of 1-12

\*

\* - For Galileo I/NAV nominal frame structure, this refers to the subframe

\* number in the range of 1-24

\*/

int16\_t message\_id;

/\*\*

\* Sub-message identifier. If required by the message 'type', this value

\* contains a sub-index within the current message (or frame) that is being

\* transmitted.

\*

\* - For GPS L1 C/A, BeiDou D1 & BeiDou D2, the submessage id corresponds to

\* the subframe number of the navigation message, in the range of 1-5.

\*

\* - For Glonass L1 C/A, this refers to the String number, in the range from

\* 1-15

\*

\* - For Galileo F/NAV, this refers to the page type in the range 1-6

\*

\* - For Galileo I/NAV, this refers to the word type in the range 1-10+

\*/

int16\_t submessage\_id;

/\*\*

\* The length of the data (in bytes) contained in the current message.

\* If this value is different from zero, 'data' must point to an array of the same size.

\* e.g. for L1 C/A the size of the sub-frame will be 40 bytes (10 words, 30 bits/word).

\*

\* This is a mandatory value.

\*/

size\_t data\_length;

/\*\*

\* The data of the reported GPS message. The bytes (or words) specified

\* using big endian format (MSB first).

\*

\* - For GPS L1 C/A, Beidou D1 & Beidou D2, each subframe contains 10 30-bit

\* words. Each word (30 bits) should be fit into the last 30 bits in a

\* 4-byte word (skip B31 and B32), with MSB first, for a total of 40

\* bytes, covering a time period of 6, 6, and 0.6 seconds, respectively.

\*

\* - For Glonass L1 C/A, each string contains 85 data bits, including the

\* checksum. These bits should be fit into 11 bytes, with MSB first (skip

\* B86-B88), covering a time period of 2 seconds.

\*

\* - For Galileo F/NAV, each word consists of 238-bit (sync & tail symbols

\* excluded). Each word should be fit into 30-bytes, with MSB first (skip

\* B239, B240), covering a time period of 10 seconds.

\*

\* - For Galileo I/NAV, each page contains 2 page parts, even and odd, with

\* a total of 2x114 = 228 bits, (sync & tail excluded) that should be fit

\* into 29 bytes, with MSB first (skip B229-B232).

\*/

uint8\_t\* data;

} GnssNavigationMessage;

/\*\*

\* The legacy callback to report an available fragment of a GPS navigation

\* messages from the HAL.

\*

\* This callback is deprecated, and will be removed in the next release. Use

\* gnss\_navigation\_message\_callback() instead.

\*

\* Parameters:

\* message - The GPS navigation submessage/subframe representation.

\*/

typedef void (\*gps\_navigation\_message\_callback) (GpsNavigationMessage\* message);

/\*\*

\* The callback to report an available fragment of a GPS navigation messages from the HAL.

\*

\* Parameters:

\* message - The GPS navigation submessage/subframe representation.

\*/

typedef void (\*gnss\_navigation\_message\_callback) (GnssNavigationMessage\* message);

typedef struct {

/\*\* set to sizeof(GpsNavigationMessageCallbacks) \*/

size\_t size;

gps\_navigation\_message\_callback navigation\_message\_callback;

gnss\_navigation\_message\_callback gnss\_navigation\_message\_callback;

} GpsNavigationMessageCallbacks;

#define GPS\_NAVIGATION\_MESSAGE\_OPERATION\_SUCCESS 0

#define GPS\_NAVIGATION\_MESSAGE\_ERROR\_ALREADY\_INIT -100

#define GPS\_NAVIGATION\_MESSAGE\_ERROR\_GENERIC -101

/\*\*

\* Extended interface for GPS navigation message reporting support.

\*/

typedef struct {

/\*\* Set to sizeof(GpsNavigationMessageInterface) \*/

size\_t size;

/\*\*

\* Initializes the interface and registers the callback routines with the HAL.

\* After a successful call to 'init' the HAL must begin to provide updates as they become

\* available.

\*

\* Status:

\* GPS\_NAVIGATION\_MESSAGE\_OPERATION\_SUCCESS

\* GPS\_NAVIGATION\_MESSAGE\_ERROR\_ALREADY\_INIT - if a callback has already been registered

\* without a corresponding call to 'close'.

\* GPS\_NAVIGATION\_MESSAGE\_ERROR\_GENERIC - if any other error occurred, it is expected that

\* the HAL will not generate any updates upon returning this error code.

\*/

int (\*init) (GpsNavigationMessageCallbacks\* callbacks);

/\*\*

\* Stops updates from the HAL, and unregisters the callback routines.

\* After a call to stop, the previously registered callbacks must be considered invalid by the

\* HAL.

\* If stop is invoked without a previous 'init', this function should perform no work.

\*/

void (\*close) ();

} GpsNavigationMessageInterface;

/\*\*

\* Interface for passing GNSS configuration contents from platform to HAL.

\*/

typedef struct {

/\*\* Set to sizeof(GnssConfigurationInterface) \*/

size\_t size;

/\*\*

\* Deliver GNSS configuration contents to HAL.

\* Parameters:

\* config\_data - a pointer to a char array which holds what usually is expected from

file(/etc/gps.conf), i.e., a sequence of UTF8 strings separated by '\n'.

\* length - total number of UTF8 characters in configuraiton data.

\*

\* IMPORTANT:

\* GPS HAL should expect this function can be called multiple times. And it may be

\* called even when GpsLocationProvider is already constructed and enabled. GPS HAL

\* should maintain the existing requests for various callback regardless the change

\* in configuration data.

\*/

void (\*configuration\_update) (const char\* config\_data, int32\_t length);

} GnssConfigurationInterface;

\_\_END\_DECLS

#endif /\* ANDROID\_INCLUDE\_HARDWARE\_GPS\_H \*/