Programmer's Reference for Garmin Mobile® XT®

SDK Release 1.60

Table of Contents

1

Overview

Purpose of This Document

Programmer's Reference for Garmin Mobile XT is a part of the Garmin Software Development Kit. This document details the information necessary for software development for Garmin Mobile XT.

Conventions Used in This Document

Throughout this document, a `fixed width font` is used to signify code elements such as files, functions, structures, fields, and bitfields.

Tools for Software Development

Pocket PC applications are typically developed with Microsoft Visual Studio 2005. They can also be developed using Microsoft eMbedded Visual C++. The latest version of this tool, as well as Pocket PC SDKs and other developer resources can be obtained freely from Microsoft (http://www.microsoft.com/downloads).

Garmin SDK

Components

Extract the SDK to a convenient folder. The \Garmin folder contains the Garmin specific include files and the library QueAPI.lib. \Samples folder contains two sample applications demonstrating the usage of GPS library and Que API library.

2

GPS Library

To begin learning more about GPS, visit http://www.garmin.com/aboutGPS.

This chapter describes the GPS Library declared in the header file `GPSLib.h`. It discusses the following topics:

- Introduction to the GPS Library
- GPS Library Data Structures
- GPS Library Constants
- GPS Library Functions

Introduction to the GPS Library

Using the GPS Library

The GPS Library provides access to the data from the internal GPS. To get access to the GPS Library, `#include GPSLib.h` in your application.

Before the GPS Library can be used, it must be opened by calling QueAPIOpen:

```
/*-----------------------------------------
Open the library.
-----------------------------------------*/
err = QueAPIOpen(QueCallback);

if ( err != gpsErrNone)
  {
  AfxMessageBox(L"Failed to open Que
  library\n",MB_OK|MB_ICONEXCLAMATION);
  }
```

The GPS Library normally computes new data once a second. When data is computed, the GPS Library calls the notification call back function provided in QueAPIOpen if your application has registered for this notification, your application can call the GPSGet* functions when this notification is received. The GPSGet functions can also be used strictly on a polling or as needed basis.

Once your application is done using the GPS Library (normally when the application stops), you should close the library:

```
/*-----------------------------------------
```

```
Close the library.
-----------------------------------------*/
err = QueAPIClose (QueCallback );
```

GPS Library Data Structures

GPSFixT8

`GPSFixT8` defines the quality of the position computation. Based on the number of satellites being received and the availability of differential correction (such as WAAS), the position may be known in two dimensions (latititude and longitude) or three dimensions (latitude, longitude, and altitude).

```
typedef uint8 GPSFixT8; enum
  {
  gpsFixUnusable  = 0,
  gpsFixInvalid   = 1,
  gpsFix2D        = 2,
  gpsFix3D        = 3,
  gpsFix2DDiff    = 4,
  gpsFix3DDiff    = 5
  };
```

Value Descriptions

`gpsFixUnusable`	GPS failed integrity check.
`gpsFixInvalid`	GPS is invalid or unavailable.
`gpsFix2D`	Two dimensional position.
`gpsFix3D`	Three dimensional position.
`gpsFix2DDiff`	Two dimensional differential position.
`gpsFix3DDiff`	Three dimensional differential position.

GPSModeT8

`GPSModeT8` defines the modes for the GPS.

```
typedef uint8 GPSModeT8; enum
  {
  gpsModeOff      = 0,
  gpsModeNormal   = 1,
  gpsModeBatSaver = 2,
  gpsModeSim      = 3,
  gpsModeExternal = 4
  };
```

Value Descriptions

`gpsModeOff`	GPS is off.
`gpsModeNormal`	Continuous satellite tracking or attempting to track satellites.
`gpsModeBatSaver`	Periodic satellite tracking to conserve battery power (only on iQue).
`gpsModeSim`	Simulated GPS information (may be same as gpsModeOff).
`gpsModeExternal`	External source of GPS information (only on iQue).

GPSPositionDataType

`GPSPositionDataType` defines the position data returned by the GPS. The `GPSPositionDataType` uses integers to indicate latitude and longitude in semicircles, where 2^{31} semicircles are equal to 180 degrees. North latitudes and East longitudes are indicated with positive numbers; South latitudes and West longitudes are indicated with
negative numbers. The following formulas show how to convert between degrees and semicircles:

degrees = semicircles * (180 / 2^{31})
semicircles = degrees * (2^{31} / 180)

```
typedef struct
  {
  sint32       lat;
  sint32       lon;
  float       altMSL;
  float       altWGS84;
  } GPSPositionDataType;
```

Field Descriptions

`lat`	Latitude component of the position in semicircles.
`lon`	Longitude component of the position in semicircles.
`altMSL`	Altitude above mean sea level component of the position in meters.
`altWGS84`	Altitude above WGS84 ellipsoid component of the position in meters.

GPSPVTDataType

GPSPVTDataType combines the GPS data types into one structure.

```
typedef struct
  {
  GPSStatusDataType       status;
  GPSPositionDataType     position;
  GPSVelocityDataType     velocity;
  GPSTimeDataType         time;
  } GPSPVTDataType;
```

Field Descriptions

`status`	GPS status.
`position`	GPS position.
`velocity`	GPS velocity.
`time`	GPS time.

GPSSatDataType

`GPSSatDataType` defines the data for one satellite.

```
typedef struct
  {
  uint8  svid;
  uint8  status;
  sint16 snr;
  float  azimuth;
  float  elevation;
  } GPSSatDataType;
```

Field Descriptions

`svid`	The space vehicle identifier for the satellite.
`status`	The status bitfield the for satellite (see constants later).
`snr`	The satellite signal to noise ratio * 100 (dB Hz).
`azimuth`	The satellite azimuth (radians).
`elevation`	The satellite elevation (radians).

GPSStatusDataType

`GPSStatusDataType` defines the status data reported by the GPS.

```
typedef struct
  {
  GPSModeT8   mode;
  GPSFixT8    fix;
```

```
    sint16      filler2;
    float       epe;
    float       eph;
    float       epv;
    } GPSStatusDataType;
```

Field Descriptions

`mode`	GPS mode.
`fix`	GPS fix.
`filler2`	Alignment padding.
`epe`	The one-sigma estimated position error in meters.
`eph`	The one-sigma horizontal only estimated position error in meters.
`epv`	The one-sigma vertical only estimated position error in meters.

GPSTimeDataType

`GPSTimeDataType` defines the time data returned by the GPS.

```
  typedef struct
    {
    UINT32      seconds;
    UINT32      fracSeconds;
    } GPSTimeDataType;
```

Field Descriptions

`seconds`	Seconds since midnight UTC.
`fracSeconds`	To determine the fractional seconds, divide the value in this field by 2^{32}.

GPSVelocityDataType

`GPSVelocityDataType` defines the velocity data returned by the GPS. The individual East, North, and up components completely describe the velocity. The track and speed fields are provided for convenient access to the most commonly used application of GPS velocity.

```
  typedef struct
    {
    float       east;
    float       north;
    float       up;
    float       track;
```

```
    float       speed;
    } GPSVelocityDataType;
```

Field Descriptions

`east`	The East component of the velocity in meters per second.
`north`	The North component of the velocity in meters per second.
`up`	The upwards component of the velocity in meters per second.
`track`	The horizontal vector of the velocity in radians.
`speed`	The horizontal speed in meters per second.

GPSCarrierPhaseOutputPositionDataType

`GPSCarrierPhaseOutputPositionDataType` defines the carrier phase output data such as position and velocity information returned by the GPS.

```
  typedef struct
    {
    double      lat;
    double      lon;
    double      tow;
    float       alt;
    float       epe;
    float       eph;
    float       epv;
    float       msl;
    float       east;
    float       north;
    float       up;
    uint32       grmn_days;
    uint16       fix;
    uint16       leap_scnds;
    } GPSCarrierPhaseOutputPositionDataType;
```

Field Descriptions

`lat`	Latitude (radians)
`lon`	Longitude (radians)
`tow`	GPS time of week (sec)
`alt`	Ellipsoid altitude (meters)
`epe`	Estimated position error (meters)

`eph`	Estimated position error, horizontal (meters)
`epv`	Estimated position error, verticle (meters)
`msl`	mean sea level height (meters)
`east`	The East component of the velocity in meters per second.
`north`	The North component of the velocity in meters per second.
`up`	The upwards component of the velocity in meters per second.
`grmn_days`	GARMIN days (day since December 31, 1989)
`fix`	0 = no fix; 1 = no fix; 2 = 2D; 3 = 3D; 4 = 2D differential; 5 = 3D differential; 6 and greater – not defined
`leap_scnds`	UTC leap seonds

GPSSatelliteInstRecordDataType

`GPSSatelliteInstRecordDataType` defines the satellite receiver measurement data returned by the GPS.

```
typedef struct
  {
  double          pr;
  uint32          cycles;
  uint16          phse;
  uint8           svid;
  uint8           snr_dbhz;
  boolean         slp_dtct;
  boolean         valid;
  } GPSSatelliteInstRecordDataType;
```

Field Descriptions

`pr`	Pseudorange (meters)
`cycles`	Number of accumulated cycles
`phse`	Carrier phase , 1/2048 cycle
`svid`	Satellite number (0 – 31)
`snr_dbhz`	Satellite strength, snr in dB*Hz

`slp_dtct`	cycle slip detected, 0 = no cycle slip detected, non-zero = cycle slip detected
`valid`	Pseudorange valid flag, 0 = information not valid, non-zero = information valid

GPS Library Constants

Error Codes

`queErrNone`	Success.
`queErrNotOpen`	Attempted to close the library without opening it first.
`queErrBadArg`	Invalid parameter passed.
`queErrMemory`	Out of memory.
`queErrNoData`	No data available.
`queErrAlreadyOpen`	The library is already open.
`queErrInvalidVersion`	The library is an incompatible version.
`queErrComm`	There was an error communicating with the API.
`queErrCmndUnavail`	The command is unavailable.
`queErrStillOpen`	Library is still open.
`queErrFail`	General failure.
`queErrCancel`	Action cancelled by user.

Extended Notification Information

The GPS Library broadcasts a `sysNotifyGPSDataEvent` when the GPS information changes. The `notifyDetailsP` of this notification is a `uint32` (not a pointer to a `uint32`) which contains one of the following extended notification information values indicating the reason for the notification.

`queLocationChange`	The GPS position has changed.
`queStatusChange`	The GPS status has changed.
`queLostFix`	The quality of the GPS position computation has become less than two dimensional.
`queSatDataChange`	The GPS satellite data has changed.

`queModeChange`	The GPS mode has changed.
`queEvent`	An generic event has occurred (i.e. sunrise/set, etc.)
`queCPOPositionChange`	The GPS CPO position data has been updated.
`queSatelliteInstChange`	The GPS CPO satellite data has been updated.
`queNavigationEvent`	The navigation status has changed. (Added in version 1.50)

Satellite Status Bitfield Values

These define the bits in the status field of `GPSSatDataType`.

`gpsSatEphMask`	Ephemeris: 0 = no ephemeris, 1 = has ephemeris.
`gpsSatDifMask`	Differential: 0 = no differential correction, 1 = differential correction.
`gpsSatUsedMask`	Used in solution: 0 = no, 1 = yes.
`gpsSatRisingMask`	Satellite rising: 0 = no, 1 = yes.

GPS Library Functions

QueAPIOpen

Purpose Opens the Que API Library.

Prototype

```
QueErrT16 QueAPIOpen
 (QueNotificationCallback callback )
```

Parameters

`-> callback`	GPS data notification call back function.

Result

`gpsErrNone`	No error.
`gpsErrMemory`	Not enough memory to open the library.
`queErrInvalidVersion`	The expected version of the library is not compatible with this library.

Comments Opens the Que API Library and prepares it for use. Called by any application or library that wants to use the services that the library provides.

`QueAPIOpen()` must be called before calling any other Que API Library functions, with the exception of QueGetAPIVersion. If the return value is anything other than `queErrNone` the library was not opened.

The application can register GPS data notification by supplying callback function.

QueAPIClose

Purpose Close the `Que API` Library.

Prototype
```
QueErrT16 QueAPIClose
 (QueNotificationCallback callback )
```

Parameters

`-> callback`	Notification callback function.

Result

`queErrNone`	No error.
`queErrNotOpen`	The library is not open.

Comments Closes the QueAPI Library and disposes of the global data memory if required. Called by any application or library that's been using the QueAPI Library and is now finished with it. Supply the callback function supplied in the corresponding QueAPIOpen call.

This should not be called if `GPSOpen` failed.

QueGetAPIVersion

Purpose Get the `Que` Library API version.

Prototype
```
uint16 QueGetAPIVersion()
```

Parameters `None`.

Result The API version of the library multiplied by 100. For example, version 1.10 will be returned as 110. If the version of the library is less than you expect, it is likely not safe to use, as some functions may not be available.

Comments Can be called without opening the QueAPI Library first.

QueLaunchApp

Purpose Launch the portion of the application specified.

Prototype
```
QueErrT16 QueLaunchApp( QueAppT8 app )
```

Parameters

`<- app`	Select a page to show by default.

Result

`gpsErrNone`	No error.
`queErrBadArg`	Invalid app value.
`queErrFail`	Unable to launch the application.

Comments Can be called without opening the QueAPI Library first.

GPSGetMaxSatellites

Purpose Get the maximum number of satellites.

Prototype
```
uint8 GPSGetMaxSatellites()
```

Parameters `None.`

Result Maximum number of satellites that are currently supported.

Comments The value returned by this routine should be used in the dynamic allocation of the array of satellites (`GPSSatDataType`).

GPSGetPosition

Purpose Get current position data.

Prototype
```
QueErrT16 GPSGetPosition
  (GPSPositionDataType *position )
```

Parameters

<- `position`	Contains the latest position from the GPS.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.

`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSGetPVT

Purpose Get current position, velocity, and time data.

Prototype
```
QueErrT16 GPSGetPVT(GPSPVTDataType *pvt )
```

Parameters

`<- pvt`	Contains the latest position, velocity, and time data from the GPS.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

If `pvt->status.fix` is equal to `gpsFixUnusable` or `gpsFixInvalid`, the rest of the data in the structure should be considered invalid.

GPSGetSatellites

Purpose Get current satellite data.

Prototype
```
QueErrT16 GPSGetSatellites
  (GPSSatDataType *sat )
```

Parameters `<- sat`
Contains latest satellite information
from the GPS.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.

`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

The `sat` parameter must point to enough memory to hold the maximum number of satellites worth of satellite data.

On an iQue M3/M5 expect SNR to be between 30dB and 50dB.
On an iQue M4 expect SNR to be between 15dB and 40dB.

GPSGetStatus

Purpose Get current status data.

Prototype
```
QueErrT16 GPSGetStatus
 (GPSStatusDataType *status )
```

Parameters

`<- status`	Contains the latest status from the GPS.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSGetTime

Purpose Get current time data.

Prototype
```
QueErrT16 GPSGetTime(GPSTimeDataType *time )
```

Parameters

`<- time`	Contains latest time data from the GPS.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.

`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSGetVelocity

Purpose Get current velocity data.

Prototype
```
QueErrT16 GPSGetVelocity(GPSVelocityDataType
*velocity )
```

Parameters

`<- velocity`	Contains the latest velocity data from the GPS.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSGetCPOPositionData

Purpose Get current PVT data as carrier phase output.

Prototype
```
QueErrT16 GPSGetCPOPositionData
(GPSCarrierPhaseOutputPositionDataType
*position)
```

Parameters

`<- position`	Contains the latest PVT data as carrier phase output.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.
`gpsErrCmndUnavail`	The function isn't supported on this platform.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSGetSatelliteInstRecordData

Purpose Get current satellite receiver measurement data from the GPS.

Prototype

```
QueErrT16 GPSGetSatelliteInstRecordData
(double *rcvr_tow, uint16 *rcvr_wn,
GPSSatelliteInstRecordDataType * sats_inst)
```

Parameters

<- `rcvr_tow`	Receiver time of week (seconds)
<- `rcvr_wn`	Receiver week number
<- `sats_inst`	The satellite record data.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.
`gpsErrCmndUnavail`	The function isn't supported on this platform.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSGetSatelliteEphInstData

Purpose Get current satellite receiver ephemeris data from the GPS.

Prototype

```
QueErrT16 GPSGetSatelliteEphInstData
(GPSSatelliteEphInstDataType eph_inst[24],
uint8* valid_sats)
```

Parameters

<- `eph_inst`	Ephemeris data array
<- `valid_sats`	Number of valid entries returned in the eph_inst array.

Result

`gpsErrNone`	No error.

`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.
`gpsErrCmndUnavail`	The function isn't supported on this platform.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

GPSSetMode

Purpose Set the current GPS mode.

Prototype

```
QueErrT16 GPSSetMode(GPSModeT8 mode)
```

Parameters

`-> mode`	The mode to put the GPS in.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrBadArg`	The mode isn't supported.
`gpsErrCmndUnavail`	The function isn't supported on this platform.

Comments If the return value is not `gpsErrNone`, the mode was not changed.

3

Que API Library

This chapter describes the Que API declared in the header file `QueAPI.h`. It discusses the following topics:

- Introduction to the Que API Library
- Que API Library Data Structures
- Que API Library Constants
- Que API Library Functions

Introduction to the Que API Library

Que API Library

The Que API Library provides access to Garmin map data stored in device's internal memory or stored on an external card. The Que API library allows applications to create points at a specified latitude and longitude, at the location of an address, at the location the user selects from a map, and at the location of an item the user selects through the QueFind menu. The Que API library also allows applications to get information about a point, display a form showing the details of a point including its location on a map, display the map application centered on the point, and create a route from the current location to a point.

Que API Library Concepts

The data returned when a point is created is a **handle** to the point, not the actual data for the point. The advantages to this approach include:

- Isolates applications from memory management issues.
- Isolates applications from the details of the point data structure and size, which helps ensure future compatibility.

Point handles can either be **open** or **closed**. A handle is **open** when it is associated with data for a point; handles are opened when a point is created. A handle is **closed** when it is not associated with data for a point. Calling `QueClosePoint()` closes open handles; closed handles have a value of `queInvalidPointHandle`.

The use of handles requires following a few simple rules:

- Before a handle is used it is considered closed; therefore handles must be initialized to `QueInvalidPointHandle`.
- A handle is opened when it is assigned a value from one of the following APIs:

  - `QueCreatePoint()`
  - `QueCreatePointFromEvent()`
  - `QueDeserializePoint()`
- Before your application exits, or when you are through using a point, the handle must be closed by calling `QueClosePoint()`. After calling `QueClosePoint()` your application must set the handle to `QueInvalidPointHandle`.
- To store a point between invocations of your application, you must store the serialized data using `QueSerializePoint()` before your application exits and re-create the point from the serialized data using `QueDeserializePoint()` when your application starts. You must never store a handle between invocations of your application.

Opening and Closing the Que API Library

To get access to the Que API Library, `#include QueAPI.h` in your application.

`Before the Que API Library can be used, it must be opened by calling` `QueAPIOpen()`. The `queAPIVersion` constant from `QueAPI.h` is supplied as a parameter to allow the library to determine if the version of the library expected by calling application is compatible with the version of the library that is loaded. `QueAPIOpen()` returns `queErrInvalidVersion` when the versions are not compatible.

Once your application is done using the Que API Library (normally when the application stops), you should close the library.

Que API Library Data Structures

Basic Data Types

`uint8`	Unsigned 8 bit integer.
`uint16`	Unsigned 16 bit integer.
`uint32`	Unsigned 32 bit integer.
`sint8`	Signed 8 bit integer.
`sint16`	Signed 16 bit integer.
`sint32`	Signed 32 bit integer.

`WCHAR`	`Char` type.

QuePositionDataType

`QuePositionDataType` specifies the 3 dimensional position of a point.

```
typedef struct
    {
    sint32      lat;
    sint32      lon;
    float       altMSL;
    } QuePositionDataType;
```

Field Descriptions

`lat`	The latitude of the point in semicircles. Semicircles are described in GPS data structure GPSPositionDataType.
`lon`	The longitude of the point in semicircles. Semicircles are described in GPS data structure GPSPositionDataType.
`altMSL`	The altitude above mean sea level of the point in meters. This field is not used.

QuePointType

`QuePointType` specifies the information about the position that is available to an application.

```
typedef struct
    {
    char                id[ quePointIdLen ];
    QueSymbolT16        smbl;
    QuePositionDataType posn;
    } QuePointType;
```

Field Descriptions

`id`	A NULL-terminated string containing the name of the point.
`smbl`	The symbol assocated with the point. This field is not used.
`posn`	The 3 dimensional position of the point.

QueSelectAddressType

`QueSelectAddressType` specifies the address fields that can be supplied when creating a point at an address.

```
typedef struct
    {
    const WCHAR *streetAddress;
    const WCHAR *city;
    const WCHAR *state;
    const WCHAR *country;
    const WCHAR *postalCode;
    } QueSelectAddressType;
```

Field Descriptions

`streetAddress`	A pointer to a NULL-terminated string containing the street number and street name of the address.
`city`	A pointer to a NULL-terminated string containing the city of the address.
`state`	A pointer to a NULL-terminated string containing the state of the address.
`country`	A pointer to a NULL-terminated string containing the country of the address.
`postalCode`	A pointer to a NULL-terminated string containing the postal code of the address.

QueRouteSortT8

`QueRouteSortT8` defines the options for sorting a list of destinations when generating a route to a list of points.

```
typedef uint8 QueRouteSortT8; enum
    {
    queRouteSortNone                = 0,
    queRouteSortAll                 = 1,
    queRouteSortIgnoreDest          = 3,
    queRouteSortIgnoreStart         = 5,
    queRouteSortIgnoreStartAndDest  = 7
    };
```

Value Descriptions

`queRouteSortNone`	Do not apply any sort to the points.
`queRouteSortAll`	Sort all of the points.
`queRouteSortIgnoreDest`	Sort all of the points except for the final destination.
`queRouteSortIgnoreStart`	Sort all of the points except for the start point.
`queRouteSortIgnoreStartAndDest`	Sort all of the points except for the start point and the final destination.

QueAppT8

`QueAppT8` defines the options for launching the Que application based on which page should be initially shown (if any).

```
typedef uint8 QueAppT8; enum
    {
    queAppMap,
    queAppWhereTo,
    queAppGps,
    queAppTurns,
    queAppTrip,
    queAppSettings,
    queAppGpsSettings,
    queAppMarkWaypoint,

    queAppMenu,

    queAppLaunchBackground,
    queAppCloseBackground,

    queAppCloseBackgroundDelay
    };
```

Value Descriptions

`queAppMap`	Default to main map page.
`queAppWhereTo`	Default to main search page.
`queAppGps`	Default to GPS status page.
`queAppTurns`	Default to turns list page.
`queAppTrip`	Default to trip computer page.

`queAppSettings`	Default to general settings page.
`queAppGpsSettings`	Default to GPS settings page.
`queAppMarkWaypoint`	Default to waypoint marking page.
`queAppMenu`	Default to main menu page (which contains Where To?, Main Map buttons, etc).
`queAppLaunchBackground`	Launch in the background without showing any user interface.
`queAppCloseBackground`	Close the application running in the background with no user interface. Must be called exactly once for every use of `queAppLaunchBackground`.
`queAppCloseBackgroundDelay`	Close the application running in the background with no user interface but delay first so that if the user immediately does a `queAppLaunchBackground`, Que will respond quickly. Must be called exactly once for every use of `queAppLaunchBackground`.

QueRiseSetType

`QueRiseSetType` defines Que sun/moon rise and set data.

```
typedef struct
  {
  QueGarminTimeT32    rise;
  QueGarminTimeT32    set;
  uint8               is_day;
  } QueRiseSetType;
```

Field Descriptions

`rise`	Rise time in Garmin time formart.
`set`	Set time in Garmin time formart.
`is_day`	Non-zero if day, zero if night.

QueRouteStatusT8

`QueRouteStatusT8` defines the current route state.

```
typedef uint8 QueRouteStatusT8; enum
    {
```

```
    queRouteStatusNone = 0,
    queRouteStatusActive,
    queRouteStatusOffRoute,
    queRouteStatusArrived,
    queRouteStatusCalculating,
    queRouteStatusCanceled,
    queRouteStatusInvalidStart,
    queRouteStatusInvalidEnd,
    queRouteStatusFailed,
    };
```

Value Descriptions

`queRouteStatusNone`	There is no route currently active.
`queRouteStatusActive`	There is a route being actively navigated.
`queRouteStatusOffRoute`	There is an active route without turn by turn guidance.
`queRouteStatusArrived`	The active route's destination has been reached.
`queRouteStatusCalculating`	A route is being calculated.
`queRouteStatusCanceled`	The route calculation has been canceled.
`queRouteStatusInvalidStart`	There are not any roads near the start point of the route
`queRouteStatusInvalidEnd`	There are not any roads near the end point of the route
`queRouteStatusFailed`	A general route failure occurred.

QueRouteInfoType

`QueSelectAddressType` specifies the address fields that can be supplied when creating a point at an address.

```
typedef struct
    {
    QueRouteStatusT8    routeStatus;
    float               distanceToTurn;
    float               distanceToDest;
    QueGarminTimeT32    timeOfTurn;
    QueGarminTimeT32    timeOfArrival;
    WCHAR               destName[41];
    } QueRouteInfoType;
```

Field Descriptions

`routeStatus`	Status of the route.

`distanceToTurn`	Distance to the next turn in meters.
`distanceToDest`	Distance to the destination in meters.
`timeOfTurn`	Estimated time of the next turn.
`timeOfArrival`	Estimated time of arrival.
`destName`	A pointer to a NULL-terminated string containing the name of the destination.

Que API Library Constants

Error Codes

`queErrNone`	Success.
`queErrNotOpen`	Attempted to close the library without opening it first.
`queErrBadArg`	Invalid parameter passed.
`queErrMemory`	Out of memory.
`queErrNoData`	No data available.
`queErrAlreadyOpen`	The library is already open.
`queErrInvalidVersion`	The library is an incompatible version.
`queErrCmndUnavail`	The command is unavailable.
`queErrStillOpen`	Library is still open.
`queErrFail`	General failure.
`queErrCancel`	Action cancelled by user.

Other values

`quePointIdLen`	Length of the point identifier string including the NULL-termination character.
`queInvalidSemicircles`	Invalid semicircle value.
`queInvalidAltitude`	Invalid altitude value.
`queInvalidPointHandle`	Invalid point handle.
`queInvalidSymbol`	Invalid symbol value.

Que API Library Functions

QueClosePoint

Purpose Closes the handle to a point.

Prototype
```
QueErrT16 QueClosePoint
( const QuePointHandle point )
```

Parameters

`-> point`	Point handle to be closed.

Result

`queErrNone`	No error.
`queErrBadArg`	The point handle was not open.

Comments Closes the handle to a point. This must be called for all open point handles before exiting your application. After calling this procedure, the caller should set the point handle to `queInvalidPointHandle` to indicate that it has been closed.

QueCreatePoint

Purpose Creates a point with the specified data.

Prototype
```
QueErrT16 QueCreatePoint
(const QuePointType *pointData,
QuePointHandle *point )
```

Parameters

`-> pointData`	Pointer to the data to use when creating the point.
`<- point`	Contains the point handle of the created point.

Result

`queErrNone`	No error.
`queErrMemory`	Unable to get memory for the point.
`queErrBadArg`	The point handle was already open.

Comments If an error occurs, the returned point handle may not be open.

QueCreatePointFromAddress

Purpose Creates a point from the specified address data.

Prototype

```
QueErrT16 QueCreatePointFromAddress
( const QueSelectAddressType *address,
QuePointHandle*          point )
```

Parameters

`-> address`	Pointer to the address data to use when creating the point.
`<- point`	Contains the point handle of the created point.

Result

`queErrNone`	No error.
`queErrMemory`	Unable to get the library's global data.

Comments Creates a point at the location of the specified address.

Not all fields of the input address data need to be supplied; a match will be attempted using the fields that contain data. Any unused fields should be set to NULL.

If a single address match cannot be found an invalid point handle will be returned.

QueSerializePoint

Purpose Returns the serialized data that represents the point.

Prototype

```
uint32 QueSerializePoint(
const QuePointHandle point,
void *pointData, const uint32 pointDataSize )
```

Parameters

`-> point`	Point handle to serialize.
`<- pointData`	Contains the serialized data.
`-> pointDataSize`	Size in bytes of the `pointData` buffer.

Result Returns the size in bytes of the serialized data.

Comments Returns the serialized data (i.e. series of bytes) that represents the point. This is used for long-term storage of the point. The point can be re-created by calling `QueDeserializePoint()`.

This always returns the size in bytes of the serialized data. If the supplied buffer is not large enough to hold all the serialized data, no data will be written into the buffer.

Typical usage is to call `QueSerializePoint()` once with `pointData` set to NULL and `pointDataSize` set to 0, then use the returned size to allocate a buffer to hold the serialized data. Then call `QueSerializePoint()` again with the address and size of the allocated buffer.

IMPORTANT: Never set `pointData` to NULL without setting `pointDataSize` equal to 0.

QueDeserializePoint

Purpose Creates a point from serialized point data.

Prototype

```
QueErrT16 QueDeserializePoint
( const void *pointData, const uint32
pointDataSize, QuePointHandle *point )
```

Parameters

`-> pointData`	Pointer to the serialized point data.
`-> pointDataSize`	Size in bytes of the serialized point data.
`<- point`	Contains the point handle of the created point.

Result

`queErrNone`	No error.
`queErrBadArg`	The point handle was already open, the pointer to the serialized data was NULL, the point data size was incorrect, or the format of the serialized point data was not recognized.

Comments Creates a point from the serialized point data created by `QueSerializePoint()`. See the description of `QueSerializePoint()` for more information.

If an error is returned the point handle will not be open.

QueGetPointInfo

Purpose Returns information about the point.

Prototype
```
QueErrT16 QueGetPointInfo(const QuePointHandle
point, QuePointType *pointInfo )
```

Parameters

`-> point`	Point handle from which to get information.
`<- pointInfo`	Contains the information about the point.

Result

`queErrNone`	No error.
`queErrBadArg`	The point handle was not open.

QueRouteDetour

Purpose Cause the current route to avoid the next distance meters of the route

Prototype
```
QueErrT16 QueRouteDetour( float distance )
```

Parameters

`-> distance`	Distance in meters

Result

`queErrNone`	No error.

QueGetRouteInformation

Purpose Returns information such as status, distance, and time about the current route.

Prototype
```
QueErrT16 QueRouteDetour( QueRouteInfoType
aRouteInformation )
```

Parameters

`-> aRouteInformation`	Information about the current route.

Result

`queErrNone`	No error.

QueGetNearestAddress

Purpose Get current address data in a parsed form

Prototype
```
QueErrT16 QueGetAddressString
( QueAddressType *aAddress )
```

Parameters

`<- aAddress`	Contains a structure to hold the nearest address information.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone,` the data should be considered invalid.

QueGetAddressString

Purpose Get current address data (nearest city).

Prototype
```
QueErrT16 QueGetAddressString
( WCHAR *aAddress, uint16 aStringLength )
```

Parameters

`<- aAddress`	Contains a buffer for the text string for the nearest city.
`<- aStringLength`	Size of the buffer in chracaters.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone,` the data should be considered invalid.

QueGetDrvRteStatusString

Purpose Get current driving/routing status.

Prototype
```
QueErrT16 QueGetDrvRteStatusString
( WCHAR *aStatus, uint16 aStringLentgh )
```

Parameters

`<- aStatus`	Contains the latest driving/routing status.
`<- aStringLength`	Size of the buffer in chracaters.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

QueGetStringFromLocation

Purpose Get text string for given location.

Prototype
```
QueErrT16 QueGetStringFromLocation
( const QuePositionDataType * aPosn,
WCHAR * aString, uint16 aStringLength )
```

Parameters

`-> aPosn`	Contains the position data from the GPS. If null, current position is used
`<- aString`	Contains the text string for given location.
`-> aStringLength`	Size of the buffer in characters.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone`, the data should be considered invalid.

If the position is NULL, the current position is used.

QueGetSunRiseSet

Purpose Get sunrise and sunset times for given location.

Prototype

```
QueErrT16 QueGetSunRiseSet
( const QuePositionDataType * aPosn,
const QueGarminTimeT32 *    aDate,
QueRiseSetType *  aRiseSet)
```

Parameters

`-> aPosn`	Contains the position data for which to calculate rise or set. If null, current time is used
`-> aDate`	Contains the date for which to calculate rise or set . If null, current time is used
`<- aRiseSet`	Contains sunrise/sunset info.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone,` the data should be considered invalid.

In extreme latitudes, the sun may not rise or set in a particular day. When this occurs, this function returns gpsErrNone and fills the invalid time memebers of aRiseSet with queTimeInvalid, which is defined in QueApiTypes.h.

QueGetMoonRiseSet

Purpose Get moonrise and moonset times for given location.

Prototype

```
QueErrT16 QueGetMoonRiseSet
( const QuePositionDataType * aPosn,
const QueGarminTimeT32 *    aDate,
QueRiseSetType *  aRiseSet)
```

Parameters

`-> aPosn`	Contains the position data for which to calculate rise or set. If null, current time is used

	`-> aDate`	Contains the date for which to calculate rise or set . If null, current time is used
	`<- aRiseSet`	Contains moonrise/moonset info.
Result	`gpsErrNone`	No error.
	`gpsErrNotOpen`	The GPS Library is not open.
	`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone,` the data should be considered invalid.

During certain days of its cycle, the moon may not rise or set in a particular day. When this occurs, this function returns gpsErrNone and fills the invalid time memebers of aRiseSet with queTimeInvalid, which is defined in QueApiTypes.h.

QueConvertGarminToSystemTime

Purpose Convert from Garmin time to system time.

Prototype

```
QueErrT16 QueConvertGarminToSystemTime
(QueGarminTimeT32 input,
SYSTEMTIME* output)
```

Parameters	`-> input`	Contains time value in Garmin format.
	`<- output`	System time expressed in UTC
Result	`gpsErrNone`	No error.
	`gpsErrNotOpen`	The GPS Library is not open.
	`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone,` the data should be considered invalid.

QueConvertSystemToGarminTime

Purpose Convert from system time to Garmin time.

Prototype
```
QueErrT16 QueConvertSystemToGarminTime
(SYSTEMTIME *input,
QueGarminTimeT32* output)
```

Parameters

`-> input`	Contains System time expressed in UTC.
`<- output`	Time value in Garmin format.

.

Result

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrNoData`	No data has been received for a period of time.

Comments If the return value is not `gpsErrNone,` the data should be considered invalid.

QueRouteIsActive

Purpose Sets active to TRUE if there is currently an active route.

Prototype
```
QueErrT16 QueRouteIsActive( boolean* active )
```

Parameters

`-> active`	Is there an active route

Result

`queErrNone`	No error.

QueRouteStop

Purpose Stops any currently active route.

Prototype
```
QueErrT16 QueRouteStop()
```

Parameters `(none)`

Result

`queErrNone`	No error.

QueRouteToPoint

Purpose Creates a route from the current location to the point.

Prototype

```
QueErrT16 QueRouteToPoint(
const QuePointHandle point
)
```

Parameters

`-> point`	Point handle to route to.

Result

`queErrNone`	No error.
`queErrBadArg`	The point handle is not open.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.

QueRouteToVias

Purpose Creates a route from the current location to a series of points.

Prototype

```
QueErrT16 QueRouteToPoint(
const QuePointHandle*  points,
uint32                 point_count,
QueRouteSortT8         sort_type
)
```

Parameters

`-> points`	List of point handles to route to.
`-> point_count`	Number of points in list of point handles.
`-> sort_type`	Type of sort to be applied to the points before routing.

Result

`queErrNone`	No error.
`queErrBadArg`	The list of points is empty.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.

QueSelectAddressFromFind

Purpose Allows the user to create a point by selecting an address from the find address form.

Prototype

```
QueErrT16 QueSelectAddressFromFind(
const HWND                      parent,
const QueSelectAddressType address,
QuePointHandle point,
)
```

Parameters

`-> parent`	Handle to parent window. NULL if no parent..
`-> address`	Pointer to the address data to use when selecting the point.
`<- point`	Contains the point handle of the created point..

Result

`queErrNone`	No error.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.

Comments

Displays the QueFind address form to allow the user to select an address from which to create a point.

The fields of the address form will be pre-filled with the supplied address data. Not all fields of the input address data need to be supplied; any unused fields should be set to NULL.

This call will first attempt to create a point at the location of the specified address exactly like `QueCreatePointFromAddress()`. If a single address match is found, it will be returned and the QueFind address form will not be displayed. If a single address match cannot be found, then the QueFind address form is displayed.

If the user cancels finding an address an invalid point handle will be returned.

QueSelectPointFromFind

Purpose

Allows the user to create a point by selecting an item using QueFind.

Prototype

```
QueErrT16 QueSelectPointFromFind(
const HWND  parent,
 QuePointHandle*  point)
```

Parameters

`-> parent`	Handle to parent window. NULL if no parent..
`<- point`	Contains the point handle of the created point..

Result

`queErrNone`	No error.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.

Comments Displays QueFind to allow the user to select an item from which to create a point. This is similar to `QueSelectAddressFromFind()` except this will display the top-level QueFind page.

If the user cancels finding an item an invalid point handle will be returned through the launch code.

QueSelectPointFromMap

Purpose Allows the user to create a point by selecting it from a map.

Prototype

```
QueErrT16 QueSelectPointFromMap(
const HWND parent,
const QuePointHandle orig,
QuePointHandle* point)
```

Parameters

`-> parent`	Handle to parent window. NULL if no parent.
`-> orig`	Handle to point to move, it can be queInvalidPointHandle if you don't want to move a point. .
`<- point`	Contains the point handle of the created point.

Result

`queErrNone`	No error.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.

Comments Allow the user to create a point by tapping a location on a displayed map.

If the user cancels the operation an invalid point handle will be returned through the launch code.

QueViewPointDetails

Purpose Displays a modal form containing a map and other details about the point.

Prototype
```
QueErrT16 QueViewPointDetails(
 const QuePointHandle point )
```

Parameters

`-> point`	Point handle to view the details of.

Result

`queErrNone`	No error.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.
`queErrBadArg`	The point handle is not open.

QueViewPointOnMap

Purpose Switches to the QueMap application centered on the point.

Prototype
```
QueErrT16 QueViewPointOnMap(
const QuePointHandle point )
```

Parameters

`-> point`	Point handle to view on map.

Result

`queErrNone`	No error.
`queErrNotOpen`	The library is not open.
`queErrComm`	There was an error communicating with the API.
`queErrBadArg`	The point handle is not open.

Comments This launches the QueMap application centered on the specified point.

4

Change History

This chapter provides the history of changes between released versions of the Que API

Changes from 1.10 to 1.20

- Added GPSGetSatelliteEphInstData() to support position post-processing for high accuracy applications.
- Added isDay value to QueRiseSetType.

Changes from 1.20 to 1.30

- Added QueGetAddressString and deprecated QueGetAddress.
- Added QueGetDrvRteStatusString and deprecated QueGetDrvRteStatus.
- Added QueGetStringFromLocation and deprecated QueGetLocationString.

Changes from 1.30 to 1.40

- Added QueGetNearestAddress

Changes from 1.40 to 1.50

- Added QueLaunchApp.
- Added new QueNotificationT8, QueRouteSortT8, queNavigationEvent.
- Added QueRouteDetour, QueRouteIsActive, QueRouteStop, QueRouteToVias.
- Added GPSSetMode.

Changes from 1.50 to 1.60

- Added queTerminationEvent to QueNotificationT8.
- Added QueRouteInfoType and QueGetRouteInformation.