Programmer’s Reference for Garmin iQue 3600 Handheld

Table of Contents

1

Overview

Purpose of This Document

Programmer's Reference for Garmin iQue 3600 Handheld is a part of the Garmin Software Development Kit. This document details the information necessary for software development for the Garmin iQue™ 3600 handheld.

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

CodeWarrior for Palm OS® Platform

This contains the Integrated Development Environment (IDE) and all the tools required to develop Palm OS® applications. The development of the applications for the iQue 3600 was performed using CodeWarrior for Palm OS® Platform version 8.3. For more information, visit the Metrowerks web site at http://www.metrowerks.com.

Palm OS® 5.0 SDK

For basic development information for Palm OS® applications, including the Palm OS® 5.0 SDK, visit http://www.palmos.com.

Palm OS® Simulator

This simulates a Palm OS® device. It allows for the testing and debugging of applications. This may also be found at http://www.palmos.com.

Garmin SDK

Components

GarminSimulator.zip includes new PalmSim.exe and DAL.dll files, as well as other DLL files to implement the Garmin extensions. It also includes in the AutoLoad folder the necessary PRCs for the Garmin extensions, as well as prebuilt PRCs for the Garmin Examples (GPSInfo.prc and PINMgrExample.prc).

GarminExamples.zip contains the source code for the two Garmin Examples (GPSInfo and PINMgrExample).

GarminSupport.zip contains the Garmin-specific include files.

Unpacking the SDK

1. If you have not done so already, get the Palm OS® 5.2 debug simulator (Palm_OS_52_Simulator_Dbg.zip) from http://www.palmos.com/dev/tools/simulator/index.html, and unzip that onto your hard drive.
2. Copy the Palm OS® 5.2 Simulator "Debug" folder and all of its contents to a new folder named "GarminDebug".
3. Extract GarminSimulator.zip into this new "GarminDebug" folder.
4. Extract GarminExamples.zip into a convenient folder, such as the "(CodeWarrior Examples)" folder of your CodeWarrior installation.
5. Extract the GarminSupport.zip file into a convenient folder, such as the "Other SDKs" folder of your CodeWarrior installation. This will create a "Garmin" folder under the folder it is extracted into. Remember to add this folder to the access paths of any projects that need to use the Garmin-specific include files.

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 `GPSLib68K.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 GPSLib68K.h` in your application.

Before the GPS Library can be used, it must be found or loaded, using the standard Palm OS® paradigm:

```
/*-----------------------------------------
Find the GPS library. If not found, load it.
-----------------------------------------*/
error = SysLibFind(GPSLibName, &gGPSLibRef);

if (error != errNone)
{
error = SysLibLoad( GPSLibType,
  GPSLibCreator, &gGPSLibRef );
ErrFatalDisplayIf( (error != errNone), "can't
  load GPS Library" );
}
```

The GPS Library normally computes new data once a second. When data is computed, the GPS Library broadcasts the notification `sysNotifyGPSDataEvent`. Once your application has registered for this notification, it 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 and unload the library using the standard Palm OS® paradigm:

```
/*-------------------------------------------
Close the library.
-------------------------------------------*/
err = GPSClose( gGPSLibRef );

/*-------------------------------------------
Unload the GPS Library.
-------------------------------------------*/
if( err != gpsErrStillOpen )
  {
  SysLibRemove( gGPSLibRef );
  }
```

GPS Data and the Palm OS® Simulator

GPS data may be received when using the Palm OS® Simulator by following these steps:

1. Connect a recent model Garmin GPS to a PC serial port.
2. Right-click in the Simulator and select Settings|Communication|Communication ports. Select the Cradle Communication Port and bind it to the COM port to which the Garmin GPS is connected.
3. Turn on the Garmin GPS. All of the GPS information from the external GPS unit will be present in the Palm OS® Simulator.
4. Simulator mode on the Garmin GPS may be used to simulate a position and velocity, or the GPS can be operated normally if the satellite signals are available at your PC.

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 Int8 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 Int8 GPSModeT8; enum
  {
  gpsModeOff      = 0,
  gpsModeNormal   = 1,
  gpsModeBatSaver = 2,
  gpsModeSim      = 3,
  gpsModeExternal = 4
  };
```

Value Descriptions

`gpsModeOff`	GPS is off.
`gpsModeNormal`	Continuous satellite tracking.
`gpsModeBatSaver`	Periodic satellite tracking to conserve battery power.
`gpsModeSim`	Simulated GPS information.

`gpsModeExternal`	External source of GPS information.

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
  {
  Int32       lat;
  Int32       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;
  Int16         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;
  UInt16      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.

GPS Library Constants

GPS Library Error Codes

`gpsErrNone`	No error.
`gpsErrNotOpen`	The GPS Library is not open.
`gpsErrStillOpen`	The GPS Library is still open.
`gpsErrMemory`	Not enough memory.
`gpsErrNoData`	No GPS data available.

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.

`gpsLocationChange`	The GPS position has changed.
`gpsStatusChange`	The GPS status has changed.
`gpsLostFix`	The quality of the GPS position computation has become less than two dimensional.
`gpsSatDataChange`	The GPS satellite data has changed.
`gpsModeChange`	The GPS mode has changed.

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

GPSClose

Purpose Close the GPS Library.

Prototype
```
Err GPSClose( const UInt16 refNum )
```

Parameters

`-> refNum`	Reference number for the library.

Result

`gpsErrNone`	No error.
`gpsErrStillOpen`	Couldn't be closed because the library is still in use by other applications.

Comments Closes the GPS Library and disposes of the global data memory if required. Called by any application or library that's been using the GPS Library and is now finished with it.

This should not be called if `GPSOpen` failed.

If `gpsErrStillOpen` is returned, the calling app should not call `SysLibRemove`.

GPSGetLibAPIVersion

Purpose Get the GPS Library API version.

Prototype
```
UInt16 GPSGetLibAPIVersion
( const UInt16 refNum )
```

Parameters

`-> refNum`	Reference number for the library.

Result The API version of the library.

Comments Can be called without opening the GPS Library first.

GPSGetMaxSatellites

Purpose Get the maximum number of satellites.

Prototype

```
UInt8 GPSGetMaxSatellites
( const UInt16 refNum )
```

Parameters

`-> refNum`	Reference number for the library.

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

```
Err GPSGetPosition( const UInt16 refNum,
GPSPositionDataType *position )
```

Parameters

`-> refNum`	Reference number for the library.
`<- 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

```
Err GPSGetPVT( const UInt16 refNum,
GPSPVTDataType *pvt )
```

Parameters

`-> refNum`	Reference number for the library.
`<- 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

```
Err GPSGetSatellites( const UInt16 refNum,
GPSSatDataType *sat )
```

Parameters

`-> refNum`	Reference number for the library.
`<- sat`	Contains latest satellite information from the GPS. See the comments below.

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.

GPSGetStatus

Purpose Get current status data.

Prototype

```
Err GPSGetStatus( const UInt16 refNum,
GPSStatusDataType *status )
```

Parameters

`-> refNum`	Reference number for the library.
`<- 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

```
Err GPSGetTime( const UInt16 refNum,
GPSTimeDataType *time )
```

Parameters

`-> refNum`	Reference number for the library.
`<- 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
```
Err GPSGetVelocity( const UInt16 refNum,
GPSVelocityDataType *velocity )
```

Parameters

`-> refNum`	Reference number for the library.
`<- 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.

GPSOpen

Purpose Opens the GPS Library.

Prototype
```
Err GPSOpen( const UInt16 refNum )
```

Parameters

`-> refNum`	Reference number for the library.

Result

`gpsErrNone`	No error.
`gpsErrMemory`	Not enough memory to open the library.

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

`GPSOpen` must be called before calling any other GPS Library functions, with the exception of `GPSGetLibAPIVersion`.

3

Pen Input Manager

This chapter describes the Pen Input Manager API declared in the header file `PenInputMgr.h`. It discusses the following topics:

- Introduction to the Pen Input Manager
- Pen Input Manager Data Structures
- Pen Input Manager Constants
- Pen Input Manager Functions.

Introduction to the Pen Input Manager

Pen Input Manager

The Pen Input Manager controls the area of the screen that is traditionally silkscreened onto the device. On the iQue 3600, this area is controlled by software, and it is sometimes referred to as "soft graffiti" or "collapsible graffiti". This area is comprised of two parts. The upper part is the dynamic input area, or graffiti area; the lower part is the status bar. The dynamic input area can be open (shown) or closed (hidden), while the status bar is always shown.

There is a button in the status bar that allows the user to show or hide the dynamic input area. This button is called the "input trigger". It shows a down arrow if the dynamic input area is open, or an up arrow if the dynamic input area is closed.

The input trigger can be enabled or disabled. If the input trigger is enabled, the user can control the state of the dynamic input area; if the input trigger is disabled, the input trigger is grayed out and the user cannot control the state of the dynamic input area.

Dynamic Input Area Concepts

Normally, users are the ones who change the dynamic input area state by tapping the input trigger button in the status bar, but applications also have the ability to set the dynamic input area state and to disable the trigger that allows the user to change the state.

There are two dynamic input area states, open and closed. The function `PINSetInputAreaState()` changes the state of the dynamic input area. Applications may query the dynamic input area state using `PINGetInputAreaState()`.

There are two input trigger states, enabled and disabled. The function `PINSetInputTriggerState()` changes the state of the input trigger. Applications may query the input trigger state using `PINGetInputTriggerState()`.

There are two dynamic input area policies. The default is to have the dynamic input area open and the input trigger disabled. The second policy allows the application and the user to control the dynamic input area state and the input trigger state. Applications should set the form's dynamic input area policy by calling `FrmSetDIAPolicyAttr()` in the `frmLoadEvent`. Each form in an application will use the default policy if `FrmSetDIAPolicyAttr()` is not called by the application.

Applications should register what size they want to be in the `frmLoadEvent` by calling `WinSetConstraintsSize()`.

Pen Input Manager Feature

The Pen Input Manager registers its API version with the feature manager. Use the following feature manager call to determine the Pen Input Manager API version:

```
err = FtrGet( pinCreator, pinFtrAPIVersion,
        &APIVersion );
```

The current Pen Input Manager API version is 1.0, and is fully compatible with the PalmSource™ Pen Input Manager API version 1.0.

If `FtrGet` returns `ftrErrNoSuchFeature,` then the Pen Input Manager is not present and should not be used.

Using the Pen Input Manager

To get access to the Pen Input Manager, `#include PenInputMgr.h` in your 68K application. Since the Pen Input Manager is an extension and not a library, it is available without being found or loaded.

To enable the input trigger and therefore give users the ability to close the dynamic input area, you must make the following calls in the `frmLoadEvent`:

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

```
Set the constraints.
-----------------------------------------*/
WinSetConstraintsSize( WinGetDisplayWindow(),
  160, 160, 0x7FFF, 160, 160, 160 );

/*-----------------------------------------
Set the dynamic input area policy.
-----------------------------------------*/
FrmSetDIAPolicyAttr( FrmGetActiveForm(),
  FrmDIAPolicyCustom );

/*-----------------------------------------
Enable the input trigger.
-----------------------------------------*/
PINSetInputTriggerState
  ( pinInputTriggerEnabled );
```

Determining When the Dynamic Input Area State Changes

Whenever the state of the dynamic input area changes, the Pen Input Manager broadcasts a `sysNotifyDisplayResizedEvent`. Register for this notification if your application needs to know when the dynamic input area changes. If you register, be sure to unregister before your application exits. If you fail to unregister, "the system will crash when the notification is broadcast" (according to the *Palm OS® Programmer's Companion)*.

Determining the Size of the Application Display Area

`WinGetDisplayExtent()` returns the current size of the display window. Typically, at initialization and upon receipt of a `sysNotifyDisplayResizedEvent` notification, your application will get the current size of the display window and adjust the locations of the various user interface items as needed.

The supplied PINMgrExample application is provided to demonstrate the usage of various aspects of the Pen Input Manager.

Pen Input Manager Data Structures

FrmDIAPolicyT16

`FrmDIAPolicyT16` specifies the dynamic input area policy type.

```
typedef UInt16 FrmDIAPolicyT16; enum
  {
  frmDIAPolicyStayOpen,
  frmDIAPolicyCustom
  };
```

Value Descriptions

`frmDIAPolicyStayOpen`	The dynamic input area stays open and the input trigger is disabled. This is the default.
`frmDIAPolicyCustom`	The dynamic input area state and input trigger state may be controlled by the application and the user.

PinInputAreaStateT16

`PinInputAreaStateT16` specifies the dynamic input area state.

```
typedef UInt16 PinInputAreaStateT16; enum
  {
  pinInputAreaOpen,
  pinInputAreaClosed,
  pinInputAreaNone
  };
```

Value Descriptions

`pinInputAreaOpen`	The dynamic input area is displayed. This is the default.
`pinInputAreaClosed`	The dynamic input area is not being displayed.
`pinInputAreaNone`	There is no dynamic input area.

PinInputTriggerStateT16

`PinInputTriggerStateT16` specifies the input trigger state.

```
typedef UInt16 PinInputTriggerStateT16; enum
  {
  pinInputTriggerEnabled,
  pinInputTriggerDisabled,
  pinInputTriggerNone
  };
```

Value Descriptions

`pinInputTriggerEnabled`	The status bar icon is enabled, meaning that the user is allowed to open and close the dynamic input area.
`pinInputTriggerDisabled`	The status bar icon is disabled, meaning that the user is not allowed to open and close the dynamic input area. This is the default.
`pinInputTriggerNone`	There is no dynamic input area.

Pen Input Manager Constants

`pinMaxConstraintSize`	Maximum size for setting constraint sizes.
`pinErrInvalidParam`	An invalid state parameter was entered.

Pen Input Manager Functions

FrmGetDIAPolicyAttr

Purpose Get a form's dynamic input area policy.

Prototype
```
FrmDIAPolicyT16 FrmGetDIAPolicyAttr
( FormPtr formP )
```

Parameters `-> formP` Pointer to a form.

Result The form's dynamic input area policy.

Comments This routine is used to determine a form's dynamic input area policy. The default dynamic input area policy is `frmDIAPolicyStayOpen`.

FrmSetDIAPolicyAttr

Purpose Set a form's dynamic input area policy.

Prototype
```
Err FrmSetDIAPolicyAttr( FomrPtr formP,
const FrmDIAPolicyT16 diaPolicy )
```

Parameters	`-> formP`	Pointer to a form.
	`-> diaPolicy`	The policy to use for this form.
Result	`errNone`	No error.
	`pinErrInvalidParam`	Parameter is not valid.

Comments This routine is used to set a form's dynamic input area policy, which will be used for opening and closing the dynamic input area. Applications should call this function in response to the `frmLoadEvent`. If an application does not call this function, the policy for that application will be `frmDIAPolicyStayOpen`.

PINGetInputAreaState

Purpose Get the current state of the dynamic input area.

Prototype
```
PinInputAreaStateT16 PINGetInputAreaState(void)
```

Parameters None

Result Current state of the dynamic input area.

Comments Call this routine to determine whether the dynamic input area is open or closed.

PINGetInputTriggerState

Purpose Get the current state of the input trigger.

Prototype
```
PinInputTriggerStateT16
PINGetInputTriggerState( void )
```

Parameters None

Result Current state of the input trigger.

Comments Call this routine to determine if the input trigger is enabled or disabled.

PINSetInputAreaState

Purpose Set the state of the dynamic input area.

Prototype
```
Err PINSetInputAreaState
( const PinInputTriggerStateT16 state )
```

Parameters

`-> state`	The desired state of the dynamic input area.

Result

`errNone`	No error .
`pinErrInvalidParam`	Parameter is not valid .

Comments This routine allows the application to set the state of the dynamic input area. Unless the appropriate constraints have been registered and the dynamic input area policy set to custom, the only state allowed is open.

PINSetInputTriggerState

Purpose Set the state of the input trigger.

Prototype
```
Err PINSetInputTriggerState
( const PinInputTriggerStateT16 state )
```

Parameters

`-> state`	The desired state of the input trigger.

Result

`errNone`	No error.
`pinErrInvalidParam`	Parameter is not valid.

Comments This routine enables or disables the input trigger. Unless the appropriate constraints have been registered and the dynamic input area policy set to custom, the only state allowed is disabled.

Normally, the trigger should remain enabled, allowing the user the choice of displaying the dynamic input area or not. In certain circumstances, an application might want to prevent the display of the dynamic input area or ensure the display of the dynamic input area. If the application disables the trigger, it should enable it in response to the `appStopEvent`.

WinSetConstraintSize

Purpose Register an application's size constraints.

Prototype
```
Err WinSetConstraintsSize( WinHandle winHandle,
const Coord minHeight, const Coord  prefHeight,
```

```
const Coord  maxHeight, const Coord  minWidth,
const Coord  prefWidth, const Coord  maxWidth )
```

Parameters

`-> winHandle`	Handle to a window.
`-> minHeight`	The minimum height to which this window can be sized.
`-> prefHeight`	The preferred height for this window.
`-> maxHeight`	The maximum height for this window.
`-> minWidth`	The minimum width for this window.
`-> prefWidth`	The preferred width for this window.
`-> maxWidth`	The maximum width for this window.

Result

`errNone`	No error.

Comments

The values are specified using the standard coordinate system, which refers to the original screen size of 160 x 160.

Currently only the `maxHeight` parameter is used. If your application desires to allow the dynamic input area to be closed, specify the constant `pinMaxConstraintSize` for this parameter.

4

Additional Hardware Buttons

This chapter describes the additional hardware buttons on the Garmin iQue 3600 Handheld. It discusses the following topics:

- Introduction to the Additional Buttons
- Button Activity Reporting
- Button Constants
- Responding to the Additional Buttons

Introduction to the Additional Buttons

Additional Buttons

To help provide support for one-hand applications, additional hardware buttons have been added to the side of the Garmin iQue3600.

The additional Garmin buttons are:

- a Thumbwheel, which can be pressed up, down, or in;
- an Escape button;
- a Record button.

To access these additional hardware buttons, `#include GarminChars.h` in your application.

Garmin Buttons and the Palm OS® Simulator

The Garmin buttons have been mapped to keys in the supplied Palm OS® Simulator as follows:

Thumb Wheel Up:	F6
Thumb Wheel Down:	F8
Thumb Wheel In:	F7
Escape Button:	F9
Record Button:	F11

The Escape and Record button exhibit the "momentarily pressed" and "pressed and held" behavior described below.

Button Activity Reporting

Button activity is reported by `keyDownEvents`. The Escape and Record buttons generate different data depending on whether they are momentarily pressed or pressed and held. If they are momentarily pressed, the `keyDownEvent` is sent when they are released. If they are pressed and held, the `keyDownEvent` is sent after they have been held for a period of time, even if the button has not been released.

The Garmin virtual character codes are sent in the `keyCode` field of the `keyDownEvent` data. The `keyDownEvents` also provide values in the `chr` field, to allow unmodified applications to respond to the additional buttons.

The Thumbwheel can also be held in. This action is dedicated to marking a waypoint at the current GPS position, and is not accessible to third-party developers.

Button Constants

The values sent in the `keyCode` and `chr` fields are defined as follows:

Button	keyCode	chr
Thumbwheel up	`vchrGarminThumbWheelUp`	`vchrPageUp`
Thumbwheel down	`vchrGarminThumbWheelDown`	`vchrPageDown`
Thumbwheel in	`vchrGarminThumbWheelIn`	`chrCarriageReturn`
Escape	`vchrGarminEscape`	`vchrGarminEscape`
Escape held	`vchrGarminEscapeHeld`	`vchrGarminEscapeHeld`
Record	`vchrGarminRecord`	`vchrGarminRecord`
Record held	`vchrGarminRecordHeld`	`vchrGarminRecordHeld`

The values returned by `KeyCurrentState()` for Garmin keys are as follows:

Button	Value
Thumbwheel up	`keyBitGarminThumbWheelUp`
Thumbwheel down	`keyBitGarminThumbWheelDown`
Thumbwheel in	`keyBitGarminThumbWheelIn`
Escape	`keyBitGarminEscape`
Record	`keyBitGarminRecord`

Responding to the Additional Buttons

Typically your application will respond to Garmin buttons by checking for a Garmin `keyDownEvent` before dispatching the event to any other handlers.

```
do
  {
  /*---------------------------------------
  Get an event.
  ---------------------------------------*/
  EvtGetEvent(&event, evtWaitForever);

  /*---------------------------------------
  Send to each handler in order, if not
  already used.
  ---------------------------------------*/
  if ( ! GarminKeyHandleEvent( &event ) )
      {
      if ( ! SysHandleEvent( &event ) )
         {
         if ( ! MenuHandleEvent( 0, &event,
                &error ) )
           {
           if ( ! AppHandleEvent( &event ) )
             {
             FrmDispatchEvent(&event);
             }
           }
         }
      }
  } while ( event.eType != appStopEvent );
```

You should not wait to handle the Garmin button event in `AppHandleEvent`, since the event contains values in the `chr` field and will likely be handled by the system or menu event handler.

The macro `GarminKeyIsGarmin()` in `GarminChars.h` can be used to detect if the `keyDownEvent` is one of the Garmin keys. If you process the event you should not dispatch it to the other event handlers, since the event contains values in the `chr` field and will likely also be handled by the system or menu event handler.