eRide GPS Core Interface

Contents

Figures

General Notes

Function Naming Conventions

erXxx:	Implemented by eRide (GPS core)
cbXxx:	Callback functions to be implemented by Application

Definitions

Primary UART:	UART connected to GPS chipset
Debug UART:	UART connected to debugging terminal
App:	Application using Generic eRide Core Interface
GPS Core:	eRide GPS core client software
GPS chipset:	eRide GPS chipset
NV data:	Non-volatile data (stored in flash or battery-backed RAM)
PVT:	Position, Velocity, Time
U8:	unsigned character variable
U16:	unsigned 16-bit variable
U32:	unsigned 32-bit variable

System Requirements

The following resources are required:

1. **Memory**
   The GPS core requires approximately 120 kilobytes of RAM and 240 kilobytes of ROM when compiled for an ARM7TDMI processor. The NV data memory requires 8 kilobytes of flash or battery-backed data.

2. **Primary UART**
   The primary UART is for communication between the GPS baseband chip and the host processor that runs the GPS core library. It must be interrupt-driven, and requires the following UART configuration:

   - 57,600 baud
   - 8 data bits, 1 stop bit, odd parity
   - No Flow Control

3. **Debug UART (Optional)**
   The system may optionally use a second UART to enable run-time logging of the GPS core activity. The UART configuration is:

   - 57,600 or 115,200 baud,
   - 8 data bits, 1 stop bit, no parity
   - No Flow Control

Callback Functions

The following callbacks are implemented by the Application.

```
long            (*cbGetMsecTimer) (void);
unsigned short  (*cbPutDebug) (U8 *pBuf, U16 length);
U8              (*cbPutGps) (U8 *pBuf, U8 length);
void            (*cbSleep) (U16 msec);
```

The `cbPutGps` and `cbSleep` callbacks are required. The `cbGetMsecTimer` and `cbPutDebug` callbacks are optional.

GPS States

The GPS core and GPS chipset have three top-level states:

1. **Off** The power to the system is off.
2. **On** Normal operating and navigation mode.
3. **RTC Standby** Low-power standby mode with the GPS real-time clock running. The built-in RTC maintains extremely accurate time and TCXO drift information that enables faster TTFF upon restart.

The following state transition diagram shows the three GPS states and the API functions that cause the transitions:

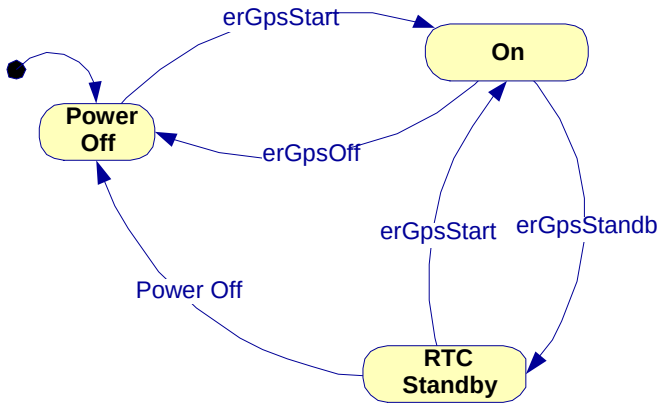

Figure 1: Top Level GPS State Transition Diagram

Application Logic Summary

1. App initializes the primary UART.
2. Call `erRegisterCallbacks`.
3. (Optional) Call `erSetMxMode`, `erSetTcoData`, `erEnableLatencyPositionPropagation`, `erGpsSetTime`, and `erEnablePositionOutagePropagation`.
4. App reads NV data from flash and calls `erSetNvData` to pass the NV Data buffer to the GPS core.
5. App calls `erGpsStart`
    a. GPS core initializes internal memory, including extracting the contents of the NV data buffer
    b. GPS core initializes the GPS chipset by calling `cbPutGps` to send serial packet to the GPS chipset.

6. (while application running)
    a. UART Rx interrupt receives bytes from the GPS chipset and optionally stores bytes in a buffer.
    b. App calls `erGpsCoreTask`
        i. If there is a complete GPS chipset packet available, the GPS core processes the packet and attempts to compute a position fix. Depending on the content of the packet, the GPS core returns bit field that indicate if new data is available. Returns zero if complete GPS chipset packet not available.
        ii. GPS core calls `cbPutGps` to send packet to the GPS chipset.
        iii. (in debug mode) GPS core calls `cbPutDebug` to send debug data to host PC.
    c. App calls `erGetPvt` (and/or any of `erGetTime, erGetSvStatus, erGetFixSet, etc.`) to get PVT and/or satellite data after `erGpsCoreTask` returns non-zero. The exact function to be called depends on the flags returned from `erGpsCoreTask.`
    d. App processes PVT data, e.g., it formats NMEA sentences and outputs the sentences over a serial port.
7. The user presses the power button and the App calls `erGpsStandby` to put the GPS chipset in RTC standby mode.
8. App saves the NV data buffer to flash memory.
9. The user presses the power button and the App calls `erGpsStart` to start the GPS chipset again. The standby-start cycle can be repeated indefinitely.
10. App calls `erGpsOff` before the system goes to power off to shut down the GPS core and chipset.
11. App saves the NV data buffer to flash memory (if not saved upon entering RTC standby).

Synchronization and Reentrancy

The GPS Core functions are not reentrant, so the App must provide synchronization. At any time, there must not be more than one thread executing GPS core functions.

CoCom Restrictions

The GPS core follows limitations imposed by the US Coordinating Committee on Multilateral Export Controls (CoCom). The CoCom limitations are:

- Speed must not exceed 1,000 knots (515 meters per second).
- Altitude must not exceed 60,000 feet (18,300 meters).

If *either* of these conditions are exceeded, the GPS core sets the fix source (`erPvtStruct.pvtFixSource`) to 0 and the position and velocity fields of the `erPvtStruct` to [0,0,0]. The GPS core resumes outputting actual values when the speed and altitude are below the CoCom limitations.

The CoCom limitation comparison uses both the raw and filtered values of speed and altitude. The GPS core considers the limitation to be exceeded if either the raw or filtered value is greater than the threshold.

The fields are:

- Raw speed: magnitude of `erPvtStruct.pvtVel`
- Filtered speed: magnitude of `erPvtStruct.pvtVelFilt`
- Raw altitude: `erPvtStruct.pvtLla[2]`
- Filtered altitude: `erPvtStruct.pvtPosFilt[2]`

API Functions

cbGetMsecTimer

```
long (*cbGetMsecTimer)(void)
```

If a working timer is present in the system, this provides the core with a 32-bit free running timer value that increments every millisecond. This function is used within the core to perform timing measurements, such as TTFF and core execution time..

Returns:
The current value of the millisecond timer. If there is no timing routine available, this callback function should not be registered with `erRegisterCallbacks`.

Notes:
This callback is optional.

cbPutDebug

```
unsigned short (*cbPutDebug)(U8 *pBuf, U16 length)
```

The GPS core calls this function to send debug data to the PC host. This function will copy the data pointed at by `pBuf` into the debug UART output buffer. It returns the actual number of bytes transferred, which could be less than the specified length if the output buffer would otherwise overflow, or there is insufficient memory to allocate an output buffer.

Parameters:
`pBuf` – pointer to buffer containing bytes to send.
`length` – length of the buffer

Returns:
The number of bytes actually transferred.

Notes:
This callback is optional and is used only when the GPS core is in debug mode (see `erSetMxMode`). If `(*cbPutDebug)` returns less than the number of bytes requested, the GPS core calls `(*cbPutDebug)` repeatedly to write the remaining characters. There is no need to sleep or wait between calls.

cbPutGps

```
U8 (*cbPutGps)(U8 *pBuf, U16 length)
```

The GPS core calls this function to send serial packets to the GPS chipset. This function will copy the data pointed at by `pBuf` into the GPS UART output buffer. It returns the actual number of bytes transferred, which could be less than the specified length if the output buffer would otherwise overflow, or there is insufficient memory to allocate an output buffer.

Parameters:
`pBuf` – pointer to buffer containing bytes to send to the GPS chipset.
`length` – length of the buffer

Returns:
The number of bytes actually transferred.

Notes:
This callback is required. If `(*cbPutGps)` returns less than the number of bytes requested, the GPS core calls `(*cbPutGps)` repeatedly to write the remaining characters. There is no need to sleep or wait between calls.

cbSleep

```
void (*cbSleep)(U16 milliseconds)
```

The GPS core calls this function to sleep for a specified interval. This is necessary during initialization of the GPS chipset to avoid overflowing the chipset's Rx buffer.

Parameters:
`milliseconds` – sleep duration in milliseconds.

Notes:
This callback is required. It is only used during initialization and will not effect system performance.

erEnableLatencyPositionPropagation

```
void erEnableLatencyPositionPropagation(int currentTimeOffset)
```

Enables position propagation for system latency.

Parameters:
`currentTimeOffset` – the latency adjustment in milliseconds relative to current time.

The `erPvtStruct.pvtFixType` indicates if the position solution was propagated due to latency (the `ER_PVT_TYPE_LATENCY_PROP` bit will always be set when latency propagation is enabled).

Figure 2 shows the relationship of the `currentTimeOffset` parameter value to the fix time.

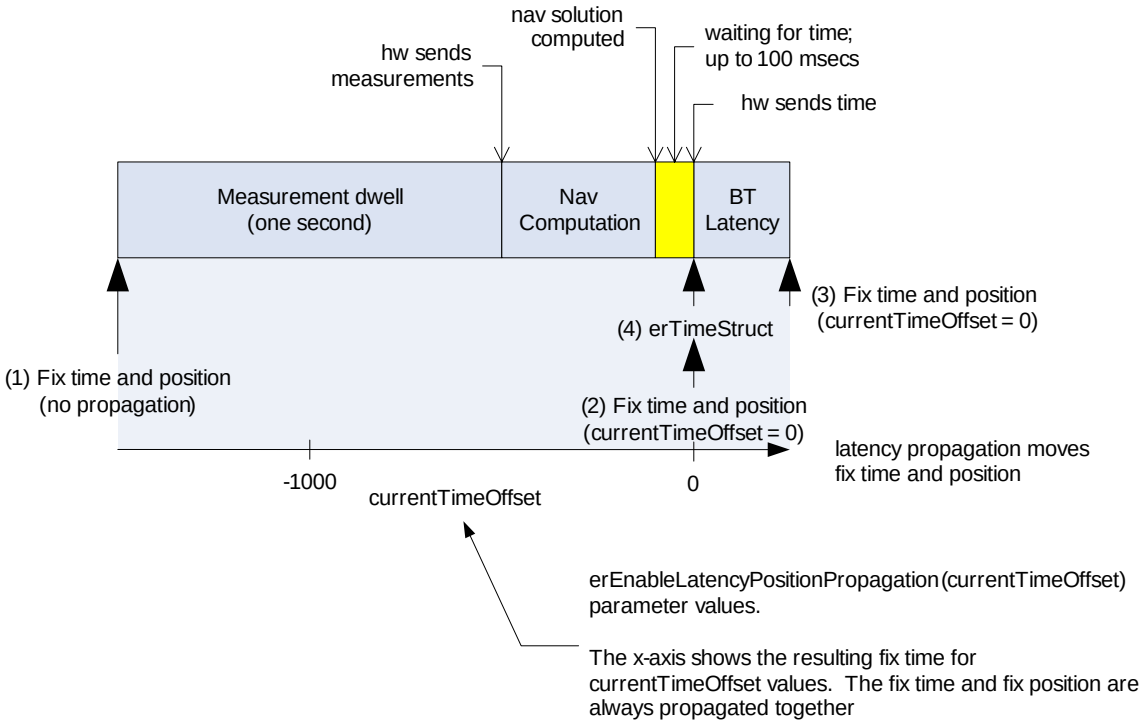

Figure 2: Position Latency Propagation Example

In this example, the product is a GPS/Bluetooth accessory (BT is an abbreviation for Bluetooth), and the processing latency includes Bluetooth communication time before position information is received by the Bluetooth host.

1. If latency propagation is not enabled, the fix time will correspond to the time and position at the start of the measurement dwell.
2. Enabling latency propagation with `currentTimeOffset = 0` causes the GPS library to propagate the fix time and position to the time at which the `erGpsCoreTask` function returns. The total amount of propagations is approximately:

   measurement dwell (1 second) + nav computation time (200 to 600 milliseconds) + wait for next hardware time packet (< 100 milliseconds)
3. To compensate for additional latency, such as Bluetooth, set `currentTimeOffset` to a value greater than zero. For example, if the average Bluetooth latency is 150 milliseconds, using `currentTimeOffset = 150` causes the GPS library to propagate the fix time and position to the approximate time at which the application receives the data over the Bluetooth connection.
4. The values in `erTimeStruct` after the fix will reflect the most recent time report from the GPS hardware. These values will match the fix time (`erPvtStruct` and `erFixSetStruct` members) only when `currentTimeOffset` is zero.

Notes:
Calling this function is optional, but it must be called before calling `erGpsStart`. This function is disabled by default. To disable latency propagation, call the function with a parameter value of –1.

erEnablePositionOutagePropagation

```
void erEnablePositionOutagePropagation(int maxSeconds)
```

Enables position propagation for GPS signal outages.

Parameters:
`maxSeconds` – the maximum number of seconds to propagate position from the previous valid position. 0 will disable this feature.

The `erPvtStruct.pvtFixType` indicates if the position solution was propagated due to an outage by setting the flag `ER_PVT_TYPE_OUTAGE_PROP`.

When there is an outage, all fields in the `erPvtStruct` and `erFixSetStruct` structures are propagated forward in time as if there was a valid nav solution during that second. It is assumed that the velocity remains constant while propagating. Propagation continues from the last valid solution during extended outages up to the number of seconds specified in the `maxSeconds` parameter. There is no maximum value for the propagation parameter.

The outage propagation uses the previous valid position solution as shown in Figure 3:

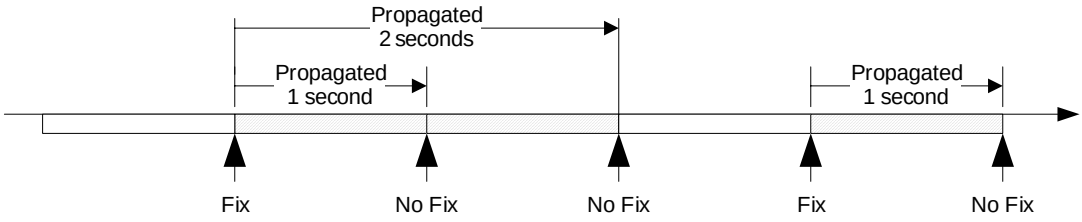

Figure 3: Position Fix Outage Propagation

During an outage, if outage propagation and latency propagation are both enabled, the effect is combined so that the time between fixes remains close to one second.

Notes:
Calling this function is optional, but it must be called before calling `erGpsStart.` The default value is five seconds propagation. Calling this function with a zero parameter disables this feature.

erGetDebugData

```
int erGetDebugData(char *msg)
```

Returns debug strings in NMEA format from the core.

Parameters:
`msg` – string large enough to hold a valid NMEA message (minimum size 84)

Returns:
NMEA-formatted core debug information in the string `msg`.
Return value is 1 if more messages remain; 0 if all data has been sent.

Sample pseudo-code for obtaining and transmitting this data:

```
int sentenceAvailFlag;
do {
    sentenceAvailFlag = erGetDebugData(nmeaStr);
    SendNmea(nmeaStr);
} while (sentenceAvailFlag);
```

Notes:
This function is used to send out system debug data from the core. It is primarily intended to be used to debug serial port problems. This data is primarily intended for eRide engineering use.

This function should be called once per second. Recommended usage is to call this function when the `ER_PVT_DATA_AVAIL` flag is returned from `erGpsCoreTask`.

erGetFixSet

```
erFixSetStruct *erGetFixSet(void)
```

Returns a pointer to the erFixSetStruct structure (see `erFixSetStruct` definition below). The structure contains data for formatting NMEA sentences such as GSA.

Returns:
Pointer to `erFixSetStruct`.

Notes:
The `erFixSetStruct` data values are undefined unless `erGpsCoreTask` indicates that the structure is available (`ER_FIXSET_AVAIL`). The contents of the structure should be processed before the next call to `erGpsCoreTask`.

erGetGpsState

```
int erGetGpsState(void)
```

Returns the current state of the GPS chipset.

Returns:
An integer value that represents the current state of the GPS chipset. Possible values are:

```
ER_GPS_STATE_OFF
ER_GPS_STATE_ON
ER_GPS_STATE_STANDBY
```

Notes:
This function may be called at any time.

erGetPvt

```
erPvtStruct *erGetPvt(void)
```

App calls this function to retrieve the PVT structure (see `erPvtStruct` definition below). The PVT structure contains data for formatting NMEA sentences such as GGA and RMC.

Returns:
Pointer to `erPvtStruct`.

Notes:
The `erPvtStruct` data values are undefined unless `erGpsCoreTask` indicates that the structure is available (`ER_PVT_AVAIL`). The contents of the structure should be processed before the next call to `erGpsCoreTask`.

erGetSvStatus

```
erSvStatusStruct *erGetSvStatus(void)
```

Returns a pointer to the `erSvStatusStruct` structure (see `erSvStatusStruct` definition below). The structure contains data for formatting NMEA sentences such as GSV.

Returns:
Pointer to `erSvStatusStruct`.

Notes:
The `erSvStatusStruct` data values are undefined unless `erGpsCoreTask` indicates that the structure is available (`ER_SVSTAT_AVAIL`). The contents of the structure should be processed before the next call to `erGpsCoreTask`.

erGetTcoFrequency

```
unsigned long erGetTcoFrequency(void)
```

Returns the most recent TCO frequency. The GPS chipset reports TCO frequency every second. Actual frequency values are hardware-dependent. Current evaluation kit modules range from 20,000 at –20°C to 1,900,000 at 85°C, and around 55,000 at room temperature.

Returns:
Unscaled integer TCO frequency.

Notes:
The TCO frequency is valid when `erGpsCoreTask` returns `ER_TCO_AVAIL`.

erGetTemperature

```
int erGetTemperature(void)
```

Returns the current TCO temperature in degrees C.

Returns:
Temperature in degrees C, LSB = 0.

Notes:
The temperature value is a function of the TCO frequency, so this function may be called when `erGpsCoreTask` returns `ER_TCO_AVAIL`.

erGetTestData

```
erGpsMeasStuct *erGetTestData(void)
```

Returns a pointer to the erGpsMeasStruct structure (see `erGpsMeasStruct` definition below). This is intended to be used as part of an automated manufacturing test using a GPS signal simulator. The structure gives detailed measurement data for SV 1 only.

Returns:
Pointer to `erGpsMeasStruct`.

Notes:
The `erGpsMeasStruct` data values are undefined unless `erGpsCoreTask` indicates that the structure is available (`ER_TESTDATA_AVAIL`). The contents of the structure should be processed before the next call to `erGpsCoreTask`.

erGetTime

```
erTimeStruct *erGetTime(void)
```

Returns a pointer to the erTimeStruct structure (see `erTimeStruct` definition below). The structure contains data for formatting NMEA sentences such as RMC and ZDA.

Returns:
Pointer to `erTimeStruct`.

Notes:
The `erTimeStruct` data values are undefined unless `erGpsCoreTask` indicates that the structure is available (`ER_TIME_AVAIL`). The contents of the structure should be processed before the next call to `erGpsCoreTask`.

erGetUTC

```
erUtcStruct *erGetUtc(void)
```

Returns a structure containing the satellite UTC parameters as described in the ICD-GPS-200C specification.

Returns:
Pointer to `erUtcStruct`.

Notes:
This function may be called at any time.

erGetVersion

```
const char *erGetVersion(void)
```

Returns a string containing the eRide GPS library version number. The string is statically defined in the GPS library.

The version string format is:

`V[V].v[v].r[r]-bbbbbbbbb`

Where:

- `V[V]` is 1 or 2 characters representing the major version number.
- `v[v]` is 1 or 2 characters representing the minor version number.
- `r[r]` is 1 or 2 characters representing the release number.
- `bbbbbbbbb` is 9 characters representing the build number.

The maximum length of the string is 18 characters.

Example:

"2.6.6-200604500"

Returns:
Pointer to the version string.

Notes:
The version string is statically defined in ROM, so `erGetVersion` may be called at any time.

erGpsCoreTask

```
int erGpsCoreTask(unsigned char *pData,
                  unsigned short length)
```

Passes UART Rx data to the GPS Core for processing. The GPS core will buffer the data locally, and check to see if there is a complete packet. If a packet is ready, the GPS core loop will be executed one or more times.

The function returns a bit mask that indicates if there is new information to report. Depending on the value of the bit field, the App calls the appropriate function to read the core output values.

Parameters:
`pData` – pointer to buffer containing bytes received from the primary UART.
`length` – length of the buffer.

Returns:
The function returns a bit-field value that indicates which structures contain new data. If the return value is zero there is no new data; otherwise bits are set in the return value as indicated in the table below. The corresponding function returns the indicated structure.

Value	Associated Type	Function	Update Rate*
ER_PVT_AVAIL	erPvtStruct	erGetPvt	1 Hz (if nav solution)
ER_TIME_AVAIL	erTimeStruct	erGetTime	1 Hz
ER_SVSTAT_AVAIL	erSvStatusStruct	erGetSvStatus	1 Hz
ER_FIXSET_AVAIL	erFixSetStruct	erGetFixSet	1 Hz (if nav solution)
ER_TCO_AVAIL	unsigned int int	erGetTcoFrequency erGetTemperature	1 Hz
ER_LCTIME_AVAIL	erTimeStruct	erGetTime	1 Hz. Set if time in week is available but not week number
ER_NVDATA_AVAIL	erNvDataStruct	App-specific	First fix, every 30 minutes thereafter, at core shutdown.
ER_TESTDATA_AVAIL	erGpsMeasStruct	erGetTestData	1 Hz

* Update rates are approximate; “if nav solution” means that the update occurs only when the core calculates a valid nav solution.

Notes:
You must call `erGpsStart` before calling `erGpsCoreTask`.

erGpsOff

```
void erGpsOff(void)
```

This function puts the GPS chipset into a deep sleep mode for minimal power usage. This function should be called just before a system power down, allowing the GPS core library to cleanly shut down the chipset.

Notes:
This function may be called at any time.

This function copies information to the NV data buffer, such as the final position, time, drift, and temperature. The application should save the buffer to NV memory before shutting down.

erGpsSetChipsetMode

```
void erGpsSetChipsetMode(int mode)
```

Provides RF chipset information to the GPS core.

Parameters:
`mode` – integer indicating the type of RF chipset.

Notes:
This function is provided for backward compatibility and should not be used.

erGpsSetTime

```
void erGpsSetTime(erTimeInputStruct *time)
```

Provides time aiding to the GPS core.

Parameters:
`time` – structure containing the current time from the application.

The `erTimeInputStruct.tconf` field specifies the confidence (guaranteed accuracy) of the time passed in the structure:

Confidence	Absolute time accuracy:
0	(none)
1	Within 120 seconds
2	Within 10 seconds

Notes:
Calling this function is optional. It may be called at any time. If GPS time is already decoded from the antenna signal or if it is already available from the Opus 27 MHz RTC, sending this command will have no effect (although it is harmless).

erGpsStandby

```
void erGpsStandby(void)
```

Puts the GPS chipset in RTC standby mode. The GPS core sends the RTC standby command to the chipset, then waits for an RTC status packet as acknowledgement that the transition to RTC standby is complete. When the transition is complete, a call to `erGetGpsState()` returns `ER_GPS_STATE_STANDBY`. If the chipset is unable to enter standby mode, it will shut off (equivalent to calling `erGpsOff()`).

Notes:
This function may be called at any time.

This function copies information to the NV data buffer, such as the final position, time, drift, and temperature. The application should save the buffer to NV memory after calling this function.

erGpsStart

```
void erGpsStart(void)
```

Initializes the GPS core and starts the GPS chipset.

Notes:
This function should be called after power on or a soft reset. Calling this function after the GPS core is running will have no effect.

erPvtIsFix

```
int erPvtIsFix(erPvtStruct *pvt)
```

Returns true if the specified `erPvtStruct` contains a position fix. The `pvtFixSource` field of the `erPvtStruct` contains information about the fix, and indicates whether the position is a true fix or a position estimate. Position estimates occur when there is insufficient information for a true fix.

Returns:
`false` – not enough information for a fix; position is an estimate
`true` – position fix was achieved.

Notes:
This function interprets the value of the `erPvtStruct.pvtFixType` field to determine if the structure contains a position fix. The function returns true even if the fix is a propagated fix.

erPvtIsFixPropagated

```
int erPvtIsFixPropagated(erPvtStruct *pvt)
```

Returns true if the specified `erPvtStruct` contains a propagated position fix. The `pvtFixType` field of the `erPvtStruct` contains information about the fix, and indicates whether the position is a true or propagated fix. Position propagations occur when there is insufficient information for a true fix or when the fix is a delayed estimate from the true fix. See `erEnableLatencyPositionPropagation` and `erEnablePositionOutagePropagation` for more information.

Returns:
`false` – position fix is not propagated
`true` – position fix is propagated.

Notes:
This function interprets the value of the `erPvtStruct.pvtFixSource` field to determine if the structure contains a propagated position fix.

erRegisterCallbacks

```
void erRegisterCallbacks(erCallbackStruct *cb)
```

Registers callbacks for functions to be implemented by the application.

Parameters:
`erCallbackStruct` – list of callback functions.

Notes:
This function must be called before calling `erGpsStart`.

erSetMxMode

```
void erSetMxMode(int mode)
```

Sets the mode of the target muxdem. The muxdem is a debugging component of the GPS core that can be used during development and testing for capturing runtime data to be analyzed by eRide engineers.

Parameters:
`mode` – target muxdem mode setting.

The mode should be one of:

`ER_MX_MODE_RW`	The muxdem is bidirectional, waiting for commands and writing debug data. The GPS core starts executing after the START command is received. This is used for eRide internal testing only.
`ER_MX_MODE_WRONLY`	The muxdem is unidirectional, writing debug data but not reading commands. The GPS core starts immediately when the `erGpsStart` function is called.
`ER_MX_MODE_OFF`	The muxdem is disabled, and the GPS core starts immediately when the `erGpsStart` function is called. This is the default mode that is valid for all applications.

Notes:
Calling this function is optional, but it must be called before calling `erGpsStart`. The default mode is `ER_MX_MODE_OFF`.

erSetNvData

```
void erSetNvData(erNvDataStruct *nvdata)
```

Passes the erNvDataStruct containing the previous run data to the GPS core. The erNvDataStruct memory contains the NV data buffer and is defined by the application. It is shared between the application and the GPS core. This size of the buffer is given as `ER_NVDATA_SIZE`.

Before calling `erGpsStart`, the App loads the NV data buffer from NV memory. The App then sets the isNew flag to indicate to the core that there is modified data in the buffer. During initialization (`erGpsStart`), the GPS core reads the contents of the NV data buffer.

The GPS core updates the NV data buffer when there is new data (e.g., new ephemeris) and on calls to `erGpsOff` and `erGpsStandby`. It also updates the buffer on the first position fix.

The GPS core does not maintain a checksum on the NV data buffer. It is the responsibility of the App to ensure the integrity of the data.

Saving the NV data buffer to NV memory is optional. If for some reason the App is unable to save the NV data buffer, the worst that can happen is reduced performance on the next run.

Figure 4 shows how NV data is managed:

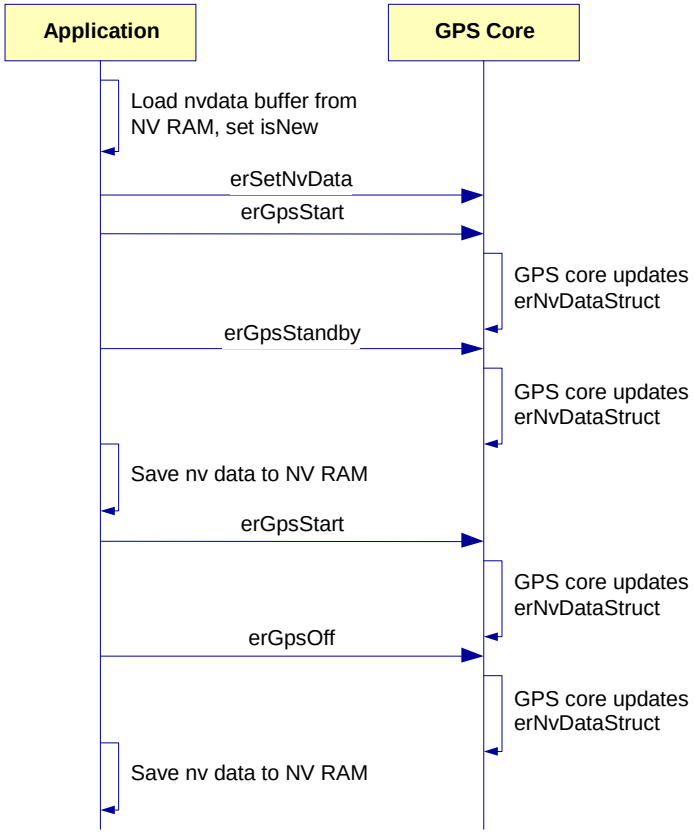

Figure 4: NV data management sequence diagram

Notes:
Calling this function is optional, but it must be called before calling `erGpsStart`.

erSetStaticPinningStrength

```
void erSetStaticPinningStrength(int strength)
```

The App calls this function to enable the position pinning function.

Parameters:
`strength` – how strong the static position pinning should be. Allowable values are `ER_PVPIN_OFF, ER_PVPIN_MEDIUM, ER_PVPIN_STRONG.`

Static pinning forces the core to hold the current position while in static mode. It is enabled by setting the strength to either `ER_PVPIN_MEDIUM` or `ER_PVPIN_STRONG`. When enabled, the core must see a position error of over 20 m between the pinned position and the latest GPS measurement in order to unpin the position. The setting `ER_PVPIN_STRONG` invokes a stronger pinning mode after 20 minutes at the same position.

Static pinning should be disabled (`ER_PVPIN_OFF`) for hand-held personal navigation systems. It should be enabled using `ER_PVPIN_MEDIUM` for car navigation systems.

Notes:
Calling this function is optional. The strength defaults to `ER_PVPIN_STRONG` if the function is not called. This function may be called at any time.

erSetStaticSensitivity

```
void erSetStaticSensitivity(unsigned int entryThreshMps, unsigned int
entryCount, unsigned int exitThreshMps, unsigned int exitCount)
```

The App calls this function to provide the GPS Core with the static entry and exit thresholds in meters per second.

Parameters:

`entryThreshMps` – entry to static mode threshold in meters/second (LSB = -12). Default is 4506 (1.1 m/s).

`entryCount` – number of times that speed is below `entryThreshMps` before entering static mode (default is 2).

`exitThreshMps` – exit from static mode threshold in meters/second (LSB = -12). Default is 4506 (1.1 m/s).

`exitCount` – number of seconds that speed is above `exitThreshMps` before exiting static mode (default is 2).

Static sensitivity sets the conditions under which the core will enter static mode. Separate thresholds are provided for mode entry and exit. While in static mode, the position will hold the same value ("pinned"), and the velocity will go to zero.

Hand-held personal navigation systems will usually want to make it difficult to enter static mode, so `entryThreshMps` should be set to a low value and `entryCount` should be a higher value. The `exitCount` should be low in order to make it easy to show low-speed (walking) movement.

Car navigation systems will usually want to make it relatively easy to enter static mode so that velocity goes to zero quickly when stopping. Set `entryThreshMps` to a slightly higher value (the defaults should be acceptable) and set `entryCount` to a lower value (1 or 2). The `exitCount` will control how long after starting to accelerate that the velocity will be held at zero; it should be kept relatively low.

Notes:

Calling this function is optional. This function may be called at any time.

The parameter `entryThreshMps` must always be less than or equal to `exitThreshMps`. If this condition is not met, `exitThreshMps` will be increased to the `entryThreshMps` value.

erSetTcoData

```
void erSetTcoData(erTcoConstantsStruct *tc, erTcoTableStruct *tt)
```

The App calls this function at initialization to provide information about the TCXO and TCO circuits.

The `tt` parameter points to an array of `erTcoTableStruct` entries. Each entry defines a point of the TCO/temperature transform function. The array length is defined by the application and the value is stored in the `erTcoConstantsStruct.tcTableLength` member.

Parameters:

`tc` – pointer to an `erTcoConstantsStruct`.

`tt` – pointer to an `erTcoTableStruct`.

Notes:

Calling this function is optional, but it must be called before calling `erGpsStart`. The ROM defaults will be used if the function is not called.

Structures

The following structures are used to report information about position, velocity, time, and satellite data to the App.

There are no floating-point data types in the GPS core, but some integer values have an implied LSB scale factor that is specified below.

For example, `erPvtStruct.pvtFixTime` has an implied LSB scale factor of –12. This means that the given value must be multiplied by 2**(-12) to find the actual value.

```
erPvtStruct *pPvtStruct = erGetPvt();
double fixTime = pPvtStruct->pvtFixTime * pow(2, -12);
```

Where there is a valid known range of values for a particular field, it specified using the notation *min…max*.

erCallbackStruct

The `erCallbackStruct` contains the pointers to the callback functions which need to be registered using the `erRegisterCallbacks`() function.

Variable	Type	Description
cbGetMsecTimer	U32(*)(void)	Callback to function providing current count of millisecond timer
cbPutDebug	U16(*)(U8*, U16)	Callback to function sending serial debug data to GPS chip set
cbPutGps	U8(*)(U8*, U8)	Callback to function sending serial data to GPS chip set
cbSleep	void(*)(U16)	Callback to sleep function

erFixSetStruct

The `erFixSetStruct` contains information about the satellites used in the position fix. This can be used for formatting NMEA sentences such as GSA.

Variable	Type	LSB	Description
fixTime	U32	-12	Time of position fix in seconds, 0…604799.
svID[8]	U8	0	Array of satellite IDs, 0 = no data; otherwise 1…32
pdop	U32	-2	PDOP of fix.
hdop	U32	-2	HDOP of fix.
vdop	U32	-2	VDOP of fix.

erNvDataStruct

The `erNvDataStruct` contains the NV data buffer.

Variable	Type	LSB	Description
length	U16	0	Length of the nvData buffer.
IsNew	U8	0	Non-zero to indicate data has been modified; zero otherwise [deprecated]
nvData[length]	U8	0	NV data buffer.

erGpsMeasStruct

The `erGpsMeasStruct` contains measurement information about the satellites. It's intended to be used as part of a manufacturing test. See the application note *eRide Manufacturing Test Procedure*.

Note: The arrays [*] are all of size `ER_MAX_GPSMEAS`. This is fixed at 16 in the current software.

Variable	Type	LSB	Description
gmNumValid	U8	0	Number of valid GPS measurements in this result
gmPrn[*]	U8	0	PRN number associated with each measurement
gmMeasType[*]	U8	0	Indicates the type of each measurement: 0 (ER_MEASTYPE_ODSM): ODSM measurement 1 (ER_MEASTYPE_TSM): TSM measurement 2 (ER_MEASTYPE_IDSM): IDSM measurement
gmTimeTag[*]	U32	-12	GPS Time Tag of the measurement
gmTimeStatus[*]	U8	0	Accuracy of GPS time tag: 0: 20 milliseconds – 10 seconds 1: <20 milliseconds
gmSignalStrength[*]	S16	-2	Estimated signal strength in dB-Hz
gmCodePhaseSigma[*]	U32	-6	Estimated code phase sigma in meters
gmBtt[*]	U8	0	Millisecond location of Nav data bit transition time (0-19 ms)
gmRawCodePhase[*]	S32	-6	Measurement code phase
gmFiltCodePhase[*]	S32	-6	Filtered measurement code phase
gmCcfGain[*]	U8	0	Current gain of the code carrier filter
gmCcfStatus[*]	U8	0	Current state of the code carrier filter (0 = normal, 1 = reset)
gmDoppler[*]	S32	0	Measured satellite Doppler
gmSatPosX[*]	S32	-6	ECEF X position of the satellite
gmSatPosY[*]	S32	-6	ECEF Y position of the satellite
gmSatPosZ[*]	S32	-6	ECEF Z position of the satellite
gmSatVelU[*]	S32	-12	ECEF velocity of the satellite in the X direction
gmSatVelV[*]	S32	-12	ECEF velocity of the satellite in the Y direction
gmSatVelW[*]	S32	-12	ECEF velocity of the satellite in the Z direction
gmPseudoRangeCorr[*]	S32	-6	Measured pseudo-range correction
gmRateRangeCorr[*]	S32	-12	Measured pseudo-range rate correction
gmGpsWeek	S32	0	Current GPS week
gmGpsTime	S32	0	Current GPS time (seconds in the current week)
gmPseudoRR	S32	0	Pseudo range rate
gmSigmaDrift	S32	-12	Clock drift in kHz (not implemented)
gmDriftVar	S32	0	Drift variance in Hz^2
gmSnr	S32	-6	SNR in dBm; To convert to dB-Hz., add 174 (at room temperature).
gmDelPrrSig5	S32	-12	Delta Doppler Sigma (average Doppler is 0)
gmPcm1	S32	0	PCM code to the AGC amplifier (not implemented)
gmPcm4	S32	0	PCM code to the AGC amplifier (not implemented)
gmIm	S32	0	I magnitude counter
gmIs	S32	0	I sign counter
gmQm	S32	0	Q magnitude counter
gmQs	S32	0	Q sign counter
gmTcoCount	S32	0	TCO count from the oscillator
gmImPlus	S32	0	I+ magnitude counter
gmQmPlus	S32	0	Q+ magnitude counter

erPvtStruct

The `erPvtStruct` contains GPS position, velocity, and time data, as well as other data about the position fix.

Variable	Type	LSB	Description
pvtFixTime	U32	-12	Time of position fix in seconds, 0…604799.
pvtFixTimeUTC	U32	0	UTC Time of position fix, seconds since Jan. 1, 1970.
pvtFixTimeUTCms	U16	0	Milliseconds of the fix to be added to UTC seconds.
pvtFixType	U8	0	Bitmasked flag detailing the type of fix calculated: 0x01 (ER_PVT_TYPE_3D): 3D fix 0x02 (ER_PVT_TYPE_OD): OD fix (Over Determined) 0x04 (ER_PVT_TYPE_3x3): 3x3 matrix (Not 4x4 Matrix) 0x08 (ER_PVT_TYPE_FPR): Full Psuedorange (Not Fractional) 0x10 (ER_PVT_TYPE_LATENCY_PROP): Propagated due to latency 0x20 (ER_PVT_TYPE_OUTAGE_PROP): Propagated due to outage 0x30 (ER_PVT_TYPE_PROP): Propagated due to latency or outage 0x80 (ER_PVT_TYPE_HC): High confidence
pvtFixSVs	U8	0	Number of SVs used in fix.
pvtFixSource	U8	0	Indicates source of chosen solution. 0 (ER_PVT_SRC_NOFIX): No Fix Available 1 (ER_PVT_SRC_MASTER): Master Solution 2 (ER_PVT_SRC_SUBSOL): Subsolution 3 (ER_PVT_SRC_KF): Kalman Filter
pvtSubsolnSv	U8	0	If chosen fix is subsolution, indicates which SV was removed to calculate the fix.
pvtAltSource	U8	0	Indicates source of reported altitude. 0: No Table Available 1: Ocean 2: Table 3: 3D Fix
pvtPos[3]	S32	-6	ECEF Position coordinates in meters.
pvtLla[3]	S32	-28 -28 -6	LLA Position coordinates in radians, radians, meters.
pvtPosFilt[3]	S32	-6	Filtered ECEF Position coordinates in meters.
pvtLlaFilt[3]	S32	-28 -28 -6	Filtered LLA Position coordinates in radians, radians, meters.
pvtVel[3]	S32	-12	ECEF Velocity coordinates in meters per second.
pvtEnu[3]	S32	-12	ENU Velocity coordinates in meters per second.
pvtVelFilt[3]	S32	-12	Filtered ECEF Velocity in meters per second.
pvtEnuFilt[3]	S32	-12	Filtered ENU Velocity coordinates in meters per second.
pvtBias	S32	-6	Filtered Clock Bias in meters.
pvtDrift	S32	-12	Filtered Clock Drift in meters per second.
pvtPdop	U32	-2	PDOP of fix.
pvtHdop	U32	-2	HDOP of fix.
pvtVdop	U32	-2	VDOP of fix.
pvtSigmaPos	U32	-6	Accuracy of raw position in m.
pvtSigmaPosFilt	U32	-6	Accuracy of filtered position in m.
pvtSigmaVel	U32	-12	Accuracy of filtered velocity in m/s.
pvtSigmaBias	U32	-6	Accuracy of filtered bias in m.
pvtSigmaDrift	U32	-12	Accuracy of filtered drift in m/s.
pvtSpeed	U32	-12	Horizontal speed in m/s.
pvtSpeedFilt	U32	-12	Horizontal speed in m/s derived from pvtHeadingFilt; may be very delayed
pvtClimb	S32	-12	Vertical speed in m/s.
pvtDirection	S32	-6	Direction of user movement in degrees (0-360, 0 = E), deprecated
pvtHeading	S32	0	Raw direction of user movement in degrees (0-360, 0 = E)
pvtHeadingFilt	S32	0	Filtered direction of user movement in degrees (0-360, 0 = E)

erSvStatusStruct

The `erSvStatusStruct` contains information about the satellites in view. This can be used for formatting NMEA sentences such as GSV.

Variable	Type	LSB	Description
gpsTime	U32	0	GPS Time that the structure was last updated, 0…604799 (this is not the fix time).
svID[12]	U8	0	Array of satellite IDs, 0 = no data; otherwise 1…32.
tracking[12]	U8	0	Boolean; non-zero indicates corresponding svID is in track set.
snr[12]	S32	-6	SNR in dBm; approx -157.5…-120.0, To convert to dB-Hz., add 174.
elevation[12]	S32	-6	Satellite elevation in degrees.
azimuth[12]	S32	-6	Satellite azimuth in degrees to true, -180…180.

erTcoConstantsStruct

The `erTcoConstantsStruct` defines the TCO characteristics.

Variable	Type	LSB	Description
tcMaxAging	U32	-12	TCXO max aging rate in meters per second per year.
tcFreqHysteresis	U32	-12	TCXO max hysteresis in meters per second.
tcTempRefPoint	S32	-12	TCXO Temperature where stability is referenced from in degrees C.
tcMaxTempFull	S32	-12	Max temperature of full range specs in degrees C.
tcMinTempFull	S32	-12	Min temperature of full range specs in degrees C.
tcFreqStabilityFull	U32	-12	Max frequency error over full range in meters per second.
tcFreqSlopeFull	U32	-12	Max frequency (error) slope over full range in meters per second per degrees C.
tcMaxTempRed	S32	-12	Max temperature of reduced range specs in degrees C.
tcMinTempRed	S32	-12	Min temperature of reduced range specs in degrees C.
tcFreqStabilityRed	U32	-12	Max frequency error over reduced range in meters per second.
tcFreqSlopeRed	U32	-12	Max frequency (error) slope over reduced range in meters per second per degrees C.
tcTableLength	U16	0	Number of entries in the erTcoTableStruct.

erTcoTableStruct

The `erTcoTableStruct` defines a TCO/temperature pair. An array of `erTcoTableStruct` entries is passed to the `erSetTcoConstants` function to define the TCO/temperature transfer function.

The `erGetTemperature` function uses the array of `erTcoTableStruct` entries to convert TCO frequency to temperature, using a piecewise interpolation method.

Variable	Type	LSB	Description
tcTableTco	U16	-8	TCO frequency in kilohertz.
tcTableTemp	S8	0	Temperature in degrees C for the corresponding TCO frequency.

erTimeInputStruct

The `erTimeInputStruct` contains the aiding time for the GPS core in GPS time.

Variable	Type	LSB	Description
gpsTime	U32	0	Current GPS Time, 0…604799.
gpsMsec	U16	0	Msec Portion of GPS time of week; 0…999.
gpsWeek	U16	0	Current GPS partial week number; 0…1024.
tConf	U16	0	Confidence of current time 1 – time accuracy < 120 seconds 2 – time accuracy < 10 seconds

erTimeStruct

The `erTimeStruct` contains the most recent time computed by the GPS core. It includes UTC and GPS time, and can be used for formatting NMEA sentences.

Variable	Type	LSB	Description
gpsTime	U32	0	Current GPS Time, 0…604799.
gpsMsec	U16	0	Millisecond portion of GPS time of week; 0…999.
gpsWeek	U16	0	Current GPS partial week number; 0…1024.
utcTime	U32	0	Current UTC time, seconds since January 1, 1970
tConf	U16	0	Confidence of current time 0 – invalid time 1 – time accurate to within 120 seconds 2 – time accurate to within 10 seconds 4 – time accurate to within 20 milliseconds
tWeekConf	U8	0	Confidence of current week 0 – invalid week number 1 – valid week number

erUtcStruct

The `erUtcStruct` contains the UTC satellite parameters per ICD-GPS-200C.

Variable	Type	LSB	Description
tUtcA0	S32	0	A0 term
tUtcA1	S32	0	A1 term
tUtcTot	U8	0	Reference time for UTC data
tUtcWnt	U8	0	Week number
tUtcTLs	S8	0	Leap Seconds
tUtcDn	U8	0	Day number
tUtcTLsf	S8	0	Future Leap Seconds

Change History

Date	Version	Description
10/17/2005	1.11	1. Fixed typos and grammatical errors. 2. Fixed error in table under `erGpsCoreTask` that referred to `erGetTrackStatus`; the correct function is `erGetSvStatus.` 3. Removed `isNew` and all references.
1/24/2006	1.12	1. Added documentation for pinning control functions `erSetStaticPinningParams` and `erSetStaticPinningStrength`. 2. Added documentation on HDOP/VDOP values now being supported. 3. Updated `erPvtStruct` with latest values. 4. Put `isNew` back in. 5. Added new return value `ER_NVDATA_AVAIL` for `erGpsCoreTask`.
3/7/2006	1.13	1. Fixed typos and grammatical errors. 2. Changed `cbGetTimerValue` in `erCallbackStruct` to `cbGetMsecTimer.` 3. Added new return value `ER_TESTDATA_AVAIL` for `erGpsCoreTask`. 4. Added `erGpsMeasStruct`. 5. Removed `erPutDebug, erNotifyError`.
4/3/2006	1.14	1. `cbSleep` is now a required callback function.
4/4/2006	1.15	1. Formatting changes for formal document release.
10/25/2006	1.16	1. Added table of figures. 2. Added parameters to the `erSetStaticSensitivity` function. 3. Added confidence value of 1 for `erGpsSetTime`. 4. Enhanced usage descriptions for `erGpsSetTime`, `erGpsStandby`, `erSetStaticPinningStrength`, `erSetStaticSenstivity`. 5. Added `erGetDebugData`. 6. Changed default value to five seconds for `erEnablePositionOutagePropagation`.
11/13/2006	1.16.01	1. Change figure 1 to show correct transition from RTC Standby state to Power Off state: it is only possible through Power Off, not through erGpsOff().