

TTNCF5

TomTom Navigator SDK Wrapper for the Microsoft .NET Compact Framework

Version 5.2

*for TomTom Navigator SDK V 5.2
(for Windows Powered Pocket PCs)*

***Using the TomTom SDK5 with the
Microsoft® .NET Compact Framework***

Copyright by

PPP – Pitsch & Pohler Professional KEG
Rohrauerweg 10
4030 Linz
Austria

Fax: +43 732 315935 40

Web: <http://www.pp-p.com> (only german)

Product Web: <http://ttncf.pp-p.com>

Customer Web: <http://ttncf.pp-p.net>

Contact: TTNCF@pp-p.com

1.Introduction

1.1. Version changes

For users of older versions we added a chapter [Version changes](#) at the end of this document.
Go there to find the most current information for this particular version.

1.2. Components of this software

This software consists of 4 Parts.

- TTNCFCGen a tool to update and maintain the TTNCF5 DLLs
- native (C++) DLLs **PPPTT_{x_x}Helper.dll** (ARM or ARMDLL directory / (**x_x is the version number**)
- a Microsoft® .NET CF 1.0 Component in **PPPTTCF_{x_x}.dll** (**x_x is the version number**)
- a Microsoft® .NET CF 2.0 Component in **PPPTTCF_{x_x}N2.dll** (**x_x is the version number**)
- documentation (this and one for TTNCFCGen)

1.3. Prerequisites

To use this DLL you need the TomTom SDK V 5.0 for licensing purposes.
With this SDK you get a file certain files – one of them needed with TTNCF5.

1.3.1.TTSDK5Link.dll

You need this DLL if you decide to use **ARMDLL_....\PPPTT_{x_x}Helper.dll** in your projects.

It's on you to decide what to use – the dynamic link (**ARMDLL\PPPTT_{x_x}Helper.dll**)
or the static one (**ARM_....\PPPTT_{x_x}Helper.dll**).

Make your choice depending on the “footprint” which means the size and number of the files to deploy with your application.

1.3.2.TTNCFCGen

To install and run TTNCFCGen you need a desktop PC with Microsoft .NET Framework installed. To activate TTNGen and download updated the PC needs an internet connection.

1.4. Installation

TTNCF5 is installed with TTNCFCGen a branding module for the use with this Wrapper.
You will find the here documented files in the *InstallationDir*\Products directory of TTNCFCGen.

Take a look at [Graphical information about file location](#) to see the files locations.

1.5. The files from TTNCFCGen

You should always use the files from the TTNCFCGen [Working directory](#).

Although the [Products folder](#) does also contain a **PPPTTCF_{x_x}.dll** the version in the [Working directory](#) holds the update versions of the component.

The files from the [Products folder](#) shall only be used, when you have problems with an update!

1.6. Program with the component

1.6.1. Add the component to your project

On the project view right click references.

Choose "Add reference".

Under .Net choose browse and select the **PPPTTCF_{x_x}.dll**.

On the project view right click the project name and choose "Add existing element".

Add the **PPPTT_{x_x}Helper.dll**.

1.6.1.1. With the dynamic link version also

On the project view right click the project name and choose "Add existing element".

Add the **TTSDK5Link.dll** from the TomTom SDK.

1.6.2. Tip for faster deployment while developing

To speed up the deployment, run your program once on the device.

This will copy the DLLs to the device.

Then mark the **PPPTT_{x_x}Helper.dll** (and if working with the dynamic link version

TTSDK5Link.dll) right click and choose "exclude from project"

This will prevent from copying the files to the device on every build.

Don't forget to "reinclude" those DLLs before you build the CAB!!

1.6.3. Use the Component in your program

You can now call the function

- a.) With the full namespace
PPPTTCF_{x_x}.TTN.GetSdkVersion(....)
- b.) Include the namespace in your project
C#: `using PPPTTCFx_x;`
VB: `import PPPTTCFx_x`
Call: TTN.GedSdkVersion(.....)

b.) is better if you use the component more often in your source, since you don't have to write always PPPTTCF_{x_x}.TTN.....

1.7. Changes from the TomTom SDK

We changed the **long** values to **int**.

In some cases the coordinates (Longitude, Latitude) have been changed (additional functions) to double. The integer part of the number represents the degrees!

So you can easier write coordinates: **Instead of 4820400 you can use 48.204.**

We also built in double properties for longitude and latitude in the structures.

WARNING: TomTom changed WGS84 format from a millionth of a degree to a 100,000ths (onehundredthousandths). Take car of this when you use the integer versions of the functions.

Structure members allowing enumerated values are replaced by the appropriate enumeration types.

The TomTom SDK uses C++ structures with char or TCHAR arrays.

Since there is no need for this in the CF we changed those arrays to strings.

Attention: this does not eliminate the presences of the TomTom SDK structures.

Therefore in input parameters strings are cut to the length of the SDK arrays which means that a string transferred to a TCHAR [64] array must only be 63 (one for the terminating zero) characters long!

The rest will be cut off!!

1.8. Conventions of this Wrapper

The calls in this wrapper have the same name and functionality as those in the TomTom SDK. One big difference is that the names of the structure members change.

Instead of always using iXXXX we use the following notation for better readability:

- **strXXnn** for strings (**nn** stands for the length in the TomTom SDK)
- **nXXX** for integers
- **lXXX** for long values
- **dXXX** for doubles
- **eXXX** for enumerations

1.9. Conventions of this document

The samples in this documentation are in C# - we chose this language, since the TomTom SDK comes with C++ DLL and therefore we think (hope) that most of you can understand C#.

By the nature of VB.NET versus C# it should be easy to translate the code to VB.NET!

If something is not very clear, please refer to the TomTom SDK documentation.

We kept the names and number the same, so this task should be pretty easy!

The function **5.13** for an example can be found in the TomTom SDK documentation as **2.13**.

1.9.1. Version information

Every entry has information in the Form TTN: x.xx TTNCf y.y.

This tells you about the navigator version needed for the call and the TTNCf version implementing this element.

1.10. Return values

As you can see in the TomTom SDK documentation every call has a structure **TError** as parameter. To provide higher compatibility with older versions of TTNCf we have allway two versions of our functions differing only in the presence (or absence) of this structure.

There are two special return values:

1.10.1. return -1000

Only in asynchronous calls when the **nSession** parameter is zero,

1.10.2. return -2000

If an exception occurs in the C++ dll we return this code.

1.11. Versions without TError

The functions without **TError** return the content of **TError**. As you can see in the TomTom SDK documentation every call has a structure **TError.iError** as the return value.

A lot of functions from the SDK return 0 even if there is a problem. In this case you will find error information in **TError.iError**.

Following this rule – if the call returns 0 we return **TError.iError** if not we return the return value.

1.12. Versions on functions

We provide Version information for every function in the SDK.

There are a lot of functions available in older TTNC5 and TomTom Navigator Versions.

Although they were available you can't use TTNC5 (or SDK5) to work with older version.

We provide this information to enable you to build application that are "source compatible".

For an example – if you use "BringNavigatorToForeground" this would work (if not using the TError call) with TTNC5 and TTNC3 also. **But you have to deploy different versions of PPPTCF.dll with your app!**

1.13. A last few words

Like the TomTom SDK this wrapper uses static methods. The only things you will have to instantiate are the structures.

All error codes are directly passed from the TomTom SDK, so you will get exact the same results, as if you would use C++ to call the SDK functions.

2. Asynchronous calls

With SDK5 TomTom introduces asynchronous calls.

We implemented these calls and follow a simple schema:

2.1. ***FunctionAsync(ref int pnSession, int nWantResult, [input params])***

With this call you initiated the function call.

If nWantResult is set to 0 – you will not any kind of return. So this is useless for functions like “GeoCode” but it makes sense for functions like “StopApplication”.

At the moment the TomTom SDK5 documentation states “nWantResult must be true” (not zero).

You have to keep nSession in your application for calls to the other 2 functions!

These functions will not start the navigator!

Some functions fail when they are processed immediately after a start of the application. You can think this as a bug of the SDK or not – anyhow it is a matter of facts that they fail when you call them before the Navigator is running.

We think this has something to do with the initialization of the Navigator.

For an example Geocode fails (location not found) because (we think) the function is processed before the maps are loaded.

2.2. ***FunctionAsyncRetrieve(int nSession, ref TError tErr, [output params])***

This call tries to get a result from the previous call to FunctionAsync.

If the navigator is not already running this function call will start it.

A call to FunctionAsync followed by FunctionAsyncRetrieve is almost the same as a call to the synchronous function.

WARNING: this function does not start the TomTom Navigator. If the navigator is not running and no return is available it will block your application!!

This was different in earlier TomTom SDK5 version and changed with release.

Until TomTom does not fix this use extreme caution with the retrieve.

Best is to precede this call with [ReturnAvailable](#).

2.3. ***ReturnAvailable(int nSession)***

This call tells you if a previous FunctionAsync call is has finished.

0 means “Not Processed”, negative numbers mean a failure.

A positive number means – the asynchronous call has finished and you can call FunctionAsyncRetrieve without blocking the application.

Take a closer look at [ReturnAvailable](#).

3.Supporting functions

To keep the following section equal with the TomTom documentation numbering we add this section for functions not available in the C++ API of the SDK 5.
We added functions needed for the use with SDK 5 as well as some functions to overcome limitations of the CF. These functions are listed below.

3.1. *GetInternalVersions*

3.1.1.Version

TTN: **NotNeeded**

TTNCF: **3.0** (differs by new structure and return)

3.1.2.Declaration(s)

int GetInternalVersions(**ref TInternalVersion** tIV)

3.1.3.Parameter

- o **TInternalVersion** tIV structure to fill with the information

3.1.4.Returns

1 if OK
-2000 if a problem occurs in PPPTTHelper.dll.
-1 if an error in occurred PPPTTCF.dll.
Values of zero in the versions indicate also an error.

3.1.5.Action

Get version information of TTNCF.

3.2. *SetForegroundWindow*

3.2.1.Version

TTN: **NotNeeded**

TTNCF: **1.0**

3.2.2.Declaration(s)

bool SetForegroundWindow(**System.Windows.Forms.Form** theForm)

3.2.3.Parameters

- o **System.Windows.Forms.Form** theForm the form you want to show

3.2.4.Returns

Success of setting the foreground window.

3.2.5.Action

Since the CF does not offer a method to bring your application to the foreground, we built this function. You must use a "topmost" form that is already open and can be made your applications view.

3.2.5.1. Hint

With this function you can (for an example) switch to your application when some event occurs while running the navigator.

Normally you would use your applications main form or the current active dialog in your application!

3.3. *SendNMEAData*

3.3.1.Version

TTN: 5.0

TTNCF: 5.0

3.3.2.Declaration(s)

int SendNMEAData(**TExternalGPSResult** tGPS, **string** strNMEAMax2500)

3.3.3.Parameters

- **TExternalGPSResult** tGPS the value returned from EnableExternalGPSInput
- **string** strNMEAMax2500 NMEA data (single sentence or part of it or more sentences) for performance reasons length is limited to 2500 characters . (this is more than 10 max. filled sentences)

3.3.4.Returns

0 if successful.
Otherwise a negative value.
-1000 nResult in tGPS was zero
-2000 a catchable error occurred
-3000 the string was too long

3.3.5.Action

Transfers the NMEA sentence (a part of a sentence / some sentences) to TomTom-

3.3.5.1. Hint

You should include the trailing “\r\n” within your sentences.
Since there are some timing problems with the current version of GPS-Driver support in TTN5 we built to [SendNMEASentenceBuffered](#) avoid the problems.

3.4. *SendNMEASentenceBuffered*

3.4.1.Version

TTN: 5.0

TTNCF: 5.0

5.1 filters sentences for RMC and GGA

3.4.2.Declaration(s)

int SendNMEASentenceBuffered(**TExternalGPSResult** tGPS, **string** strNMEASentence)

3.4.3.Parameters

- **TExternalGPSResult** tGPS the value returned from EnableExternalGPSInput
- **string** strNMEASentence a single NMEA sentence

3.4.4.Returns

0 if successful.
1000 signals the buffering (nothing was sent to TTN)
Otherwise a negative value.
-1000 nResult in tGPS was zero
-2000 a catchable error occurred
-3000 the string was too long
-100 sentence was not buffered (new in 5.1 – only GPGGA and GPRMC are buffered)

3.4.5.Action

Collects NMEA sentences. Every time a GPRMC sentence is found the buffered sentences are transferred to TTN as a whole. If a sentence does not end with “\r\n” this sequence will be added to it.

If no GPRMC sentence arrives within 2200 characters – the buffer is transferred to TTN.

New in version 5.1: Only GPRMC and GPGGA sentences are transmitted.

3.4.5.1. Hint

We think the easiest way to transfer NMEA to TTN while the “timing problem” exists. This function is (like all the others) thread safe! BUT: it is achieved by locking – so take care if you use different threads calling this function!

3.5. ReturnAvailable

3.5.1.Version

TTN: 5.0

TTNCF: 5.0

3.5.2.Declaration(s)

int ReturnAvailable(**int** nSession)

3.5.3.Parameters

- **int** nSession the value you got by calling xxxxAsync

3.5.4.Returns

Any negative number indicates a problem.

-1000 the nSession was zero

-2000 a catchable error occurred

0 the call is not finished

1 the call is finished

3.5.5.Action

Checks if an asynchronous call is finished.

3.5.5.1. Hint

Use this function to determine if a previous asynchronous call has finished.

3.6. CheckCMDHandler

3.6.1.Version

TTN: 5.0

TTNCF: 5.1

New in 5.1

3.6.2.Declaration(s)

int CheckCMDHandler(**string** strAppName, **ref string** strResult)

3.6.3.Parameters

- **string** strAppName name of the application like given in the CAP file
- **ref string** strResult holds the "command name" from the CAP file

3.6.4.Returns

Any negative number indicates a problem.

-2000 a catchable error occurred

0 no command was found

>1 a command was given (the number is the length of the command string)

3.6.5.Action

Checks if a command was fired from the TomTom Menu.

3.6.5.1. Hint

Use this function with a timer to check periodically if some command was sent to your app.
strApplication is normally the Executable file name without extension.

```
private void tmCmdHandler_Tick(object sender, System.EventArgs e) {
    tmCmdHandler.Enabled=false; //ensure that no call appears
                                //while handling the current
    string strErg="";
    int nRet=TTN.CheckCMDHandler("TourLogger",ref strErg);
    if(nRet>0) { //we have found a Command
        HandleCommad(strErg);
    }
    tmCmdHandler.Enabled=true; //reenable the timer
}
```

3.7. StartTTN

3.7.1.Version

TTN: 5.0

TTNCF: 5.2

3.7.2.Declaration(s)

bool StartTTN(**System.Windows.Forms.Form** theForm, **int** nWaitForStartSecs, **int** nWaitForFinishSecs)

3.7.3.Parameters

- **System.Windows.Forms.Form** **theForm** the form you want to show after start
- **int** **nWaitForStartSecs** Time to wait for start
- **int** **nWaitForFinishSecs** Time to wait after start

3.7.4.Returns

Success of starting TTN.

3.7.5.Action

This is a helper for a common task.

Your app wants to start TTN and stay in foreground (be in foreground after start).

Since TTN takes (like every PPC program) takes the focus after start this function starts TTN.

After the start command it waits max **nWaitForStartSecs** for TTN to run.

If TTN runs the waiting is stopped.

Next when TTN signals that it runs it is not fully running which means that it does some "screen things" – after these things it takes the focus again.

So this function waits **nWaitForFinishSecs** and then it calls SetForegroundWindow with the given form.

3.7.5.1. Hint

On all devices we tested with the values 15 for **nWaitForStartSecs** and 3 for **nWaitForFinishSecs** did the job.

3.8. TerminateWrapper

3.8.1.Version

TTN: 5.0

TTNCF: 5.2

3.8.2.Declaration(s)

void TerminateWrapper (**bool** bStopNavigator)

3.8.3.Parameters

- **bool** **bStopNavigator** if true the navigator is also stopped

3.8.4.Returns

None

3.8.5.Action

This function terminates all potential running things in TTNCF.

It disables external GPS feed if running, disables raw GPS output if running and finally terminates the watch thread.

If TTN is not running it will NOT be restarted for terminating these things.

After a call to this function you can no longer use TTNCF till restart.

3.8.5.1. Hint

If KeepAlive is running this function takes about 2 seconds to finish.

4.Events

4.1. OnGPSMessage

4.1.1.Version

TTN: 5.2

TTNCF: 5.2

4.1.2.Declaration(s)

public static event TTNCFGPSMsgHandler OnGPSMessage

4.1.3.Type

delegate void TTNCFGPSMsgHandler (string strNMEA)

4.1.3.1. Parameters

- **string strNMEA** NMEA sentences (parts possible) from TTN

4.1.4.Returns

None

4.1.5.Action

The event is fired when Raw GPS data is received from the TomTom Navigator. The sentences are transmitted whenever TomTom likes to do it. So it is very likely that sentences are not complete. Think about it, when implementing your GPS parser. Assume that a complete sentence can arrive with 2 or more events.

It is also possible, that one message sends more than one GPS Sentence.

This event is raised from an own thread – so be sure to use Control.Invoke if you deal with the UI. And remember this is “chained” – so if you feed a NMEA parser in this event and this parser raises an event there is also Control.Invoke needed!

4.1.5.1. Hint

Since TomTom transfers GPS Data via WM_COPYDATA (which is sent not posted) the handling of this call blocks the navigator. So ensure that you handle this event very fast. Never do such things as calling MessageBox or other user interaction in such an event.

4.1.6.See also

[EnableRawGPSOutput](#), [IGPSInput](#)

4.2. OnTTNRestarted

4.2.1.Version

TTN: 5.2

TTNCF: 5.2

4.2.2.Declaration(s)

public static event TTNCFStartNavMsgHandler OnTTNRestarted

4.2.3.Type

delegate void TTNCFStartNavMsgHandler (bool bStarted)

4.2.3.1. Parameters

- **bool bStarted** true if TTN was successfully started – otherwise false

4.2.4.Returns

None

4.2.5.Action

The event is fired when KeepAlive restarts (tries to restart) the navigator.

If bStarted is true TTN is running again. You can use this event to do special handling of this situation. Example: you feed GPS Data to TTN or get Raw GPS data from the Navigator – these things break when the Navigator terminates and have to be reinitiated!

This event is raised from an own thread – so be sure to use Control.Invoke if you deal with the UI.

4.2.6.See also

[KACurrentForm](#)

5.Properties

5.1. *IGPSInput*

5.1.1.Version

TTN: 5.2

TTNCF: 5.2

5.1.2.Declaration(s)

public static INMEAInput IGPSInput

5.1.3.Implementation

```
public interface INMEAInput {  
    void AddNMEADData(string strNMEA);  
}
```

5.1.3.1. Parameters

- **string** strNMEA NMEA sentences (parts possible)

5.1.4.Use

When set the AddNMEADData method is called when raw GPS output arrives from TTN.

This Interface is acalled an own thread – so if your NMEA handler raises events be sure to use Control.Invoke if you deal with the UI!

5.1.4.1. Hint

Since TomTom transfers GPS Data via WM_COPYDATA (which is sent not posted) the handling of this call blocks the navigator.

So ensure that you act pretty fast in your NMEA handler.

Never do such things as calling MessageBox or other user interaction in such an event.

5.1.5.See also

[EnableRawGPSOutput](#), [OnGPSMessage](#)

5.2. *KACurrentForm*

5.2.1.Version

TTN: 5.2

TTNCF: 5.2

5.2.2.Declaration(s)

public System.Windows.Form KACurrentForm

5.2.3.Use

Whe KeepAlive must resart the navigator you would loose the focus.

To keep the focus just set this property to you current visible form.

For a custom handling of TTN restarts use [OnTTNRestarted](#).

5.2.3.1. Hint

If you loose this blank TTN will stay in foreground after a restart.

5.2.4.See also

[OnTTNRestarted](#)

5.3. *KACheckCallsForTTNRunning*

5.3.1.Version

TTN: 5.2

TTNCF: 5.2

5.3.2.Declaration(s)

```
public bool KACheckCallsForTTNRunning
```

5.3.3.Use

When this value is set to true than most blocking call is preceeded with a check if the Navigator runs – and if not the system tries to restart it.

This behaviour makes use of [KACurrentForm](#) if set.

For a custom handling of TTN restarts use [OnTTNRestarted](#).

5.3.3.1. Hint

You can use this instead of setting [KAMode](#) to a watching value or in combination with it.

Take care – setting this value to true preceeds the most calls to blocking functions with IsApplicationRunning!

5.3.4.See also

[OnTTNRestarted](#)

5.4. *KAMode*

5.4.1.Version

TTN: 5.2

TTNCF: 5.2

5.4.2.Declaration(s)

```
public EKAMode KAMode
```

5.4.3.Implementation

```
public enum EKAMode {  
    KeepLiving,           // watches if TTN is running – if not it tries to restart  
    WatchFunctions,      // watches if a call is pending and TTN is not running–  
                        // if so it tries to restart TTN  
    Suspended            //deactivates the system  
}
```

5.4.4.Use

When this property is set to a value diffent then Suspended a thread is started which check if TTN is alive. The check is done periodically depending on [KAWatchIntervalSec](#).

This behaviour makes use of [KACurrentForm](#) if set.

For a custom handling of TTN restarts use [OnTTNRestarted](#).

5.4.4.1. Hint

Be sure to call [TerminateWrapper](#) if you once set this property to a value different than Suspended. This ensures that the thread terminates at the end of your program.

5.5. *KAWatchIntervalSec*

5.5.1.Version

TTN: 5.2

TTNCF: 5.2

5.5.2.Declaration(s)

```
public int KAWatchIntervalSec
```

5.5.3.Use

Determines how often the KeepAlive System shall check if the Navigator is running. This value allows a minimum of 0 seconds – if you specify a lower value it is corrected.

5.5.3.1. Hint

You can a large value in combination with [KACheckCallsForTTNRunning](#) so you don't stress the system to much and on the other hand you can be sure that blocking call will run as fast as possible.

5.5.4.See also

[OnTTNRestarted](#)

6. Supported API calls

6.1. *GetSdkVersion*

6.1.1. Version

TTN: 1.42

TTNCF: 1.0 (formerly called GetSdkVersionInfoV01)

6.1.2. Declaration(s)

```
int GetSdkVersion(ref TError tErr, ref TVersion tnV)
int GetSdkVersion(ref TVersion tnV)
```

6.1.2.1. Asynchronous

Not available.

6.1.3. Parameters

- TError tErr Error structure to fill
- TVersion tnV version structure to fill

6.1.4. Returns

0 if successful, non-zero value otherwise.

-2000 a catchable error occurred

6.1.5. Action

Returns the SDK version information (version string and build number)

6.1.5.1. Hint

Use this call to determine the presence of the SDK (if using DLL version) and Helper DLL. The call will catch the Error returning -1 and the Error Message in the structure.

6.2. *GetApplicationVersion*

6.2.1. Version

TTN: 1.42

TTNCF: 1.0 (formerly called GetNavigatorVersionInfoV01)

6.2.2. Declaration(s)

```
int GetApplicationVersion(ref TError tErr, ref TVersion tnV)
int GetApplicationVersion(ref TVersion tnV)
```

6.2.2.1. Asynchronous

```
int GetApplicationVersionAsync(ref int pnSession, int nWantResult)
int GetApplicationVersionAsyncRetrieve(int nSession, ref TError tErr, ref TVersion tnV)
int GetApplicationVersionAsyncRetrieve(int nSession, ref TVersion tnV)
```

6.2.3. Parameters

- TError tErr Error structure to fill
- TVersion tnV version structure to fill

6.2.4. Returns

0 if successful, non-zero value otherwise.

6.2.5. Action

Returns the TomTom Navigator version information (version string and build number)

6.2.5.1. Hint

Use this call to determine the presence of SDK, Helper DLL, and the Navigator. The call will catch the Error returning -1 and the Error Message in the structure.

6.3. *StartApplication*

6.3.1.Version

TTN: 5.0

TTNCF: 5.0

6.3.2.Declaration(s)

int StartApplication(**ref** TError tErr)

int StartApplication()

6.3.2.1. Asynchronous

Not available.

6.3.3.Parameter

- **TError** **tErr** Error structure to fill

6.3.4.Returns

0 if successful, non-zero value otherwise.

6.3.5.Action

Starts TomTom Navigator. If TTN is already running this function returns immediately with 0.

6.3.5.1. Hint

Since this call would bring TomTom Navigator to foreground in most cases be prepared that your application is no longer visible after that call.

We built SetForegroundWindow to bring your application back to foreground.

Since the call returns before the Navigator has finished starting – use a

System.Threading.Thread.Sleep(3000) to ensure that the start process has finished before you bring your app to foreground.

If you miss the sleep – your app comes to foreground – but finishing the start of Navigator results in a “switch back” to the Navigator.

6.4. *IsApplicationRunning*

6.4.1.Version

TTN: 5.0

TTNCF: 5.0

6.4.2.Declaration(s)

int IsApplicationRunning(**ref** TError tErr)

int IsApplicationRunning()

6.4.2.1. Asynchronous

Not available.

6.4.3.Parameter

- **TError** **tErr** Error structure to fill

6.4.4.Returns

0 if not running, 1 if running.

A negative value signals an error.

6.4.5.Action

Checks if TomTom Navigator is running

6.4.5.1. Hint

Use this function to determine that Navigator is running before you call xxxAsyncRetrieve.

6.5. *BringNavigatorToForeground*

6.5.1.Version

TTN: 1.42

TTNCF: 1.0

6.5.2.Declaration(s)

```
int BringNavigatorToForeground(ref TError tErr)
```

```
int BringNavigatorToForeground ()
```

6.5.2.1. Asynchronous

```
int BringNavigatorToForegroundAsync(ref int pnSession, int nWantResult)
```

```
int BringNavigatorToForegroundAsyncRetrieve(int nSession, ref TError tErr)
```

```
int BringNavigatorToForegroundAsyncRetrieve(int nSession)
```

6.5.3.Parameter

- TError tErr Error structure to fill

6.5.4.Returns

0 if successful, non-zero value otherwise.

6.5.5.Action

This call brings the navigator into the foreground. If it is not running it starts the navigator.

6.5.5.1. Hint

There is no "BringNavigatorToBackground".

Since a call like this would bring "any window" to foreground just use a call that brings your application to foreground!!

We built SetForegroundWindow for this.

6.6. *TomTomApplicationToForeground*

6.6.1.Version

TTN: 5.0

TTNCF: 5.0

6.6.2.Declaration(s)

```
int TomTomApplicationToForeground()
```

6.6.2.1. Asynchronous

Not available.

6.6.3.Parameter

- None

6.6.4.Returns

0 if successful. A negative value signals an error.

-5 INavigator is not running

6.6.5.Action

Brings the TomTom Navigator to foreground if it is running.

6.6.5.1. Hint

This call is faster than BringNavigatorToForeground but it does not start the Navigator if it is not running.

6.7. StopApplication

6.7.1.Version

TTN: 5.0

TTNCF: 5.0

6.7.2.Declaration(s)

int StopApplication(**ref** TError tErr)

int StopApplication()

6.7.2.1. Asynchronous

int StopApplicationAsync(**ref** **int** pnSession, **int** nWantResult)

int StopApplicationAsyncRetrieve(**int** nSession, **ref** TError tErr)

int GetApplicationVersionAsyncRetrieve(**int** nSession)

6.7.3.Parameter

- TError tErr Error structure to fill

6.7.4.Returns

0 if successful, non-zero value otherwise.

6.7.5.Action

Stops the TomTom Navigator Application.

6.8. SwitchToNavigatorView

6.8.1.Version

TTN: 1.42

TTNCF: 1.0

KEEPALIVE

6.8.2.Declaration(s)

int SwitchToNavigatorView(**ref** TError tErr)

int SwitchToNavigatorView()

6.8.2.1. Asynchronous

int SwitchToNavigatorViewAsync(**ref** **int** pnSession, **int** nWantResult)

int SwitchToNavigatorViewAsyncRetrieve(**int** nSession, **ref** TError tErr)

int SwitchToNavigatorViewAsyncRetrieve(**int** nSession)

6.8.3.Parameter

- TError tErr Error structure to fill

6.8.4.Returns

0 if successful, non-zero value otherwise.

6.8.5.Action

Switches to the navigation view.

6.9. FlashMessage

6.9.1. Version

TTN: 5.0

TTNCF: 5.0 **KEEPALIVE**

6.9.2. Declaration(s)

```
int FlashMessage(ref TError tErr, string strMessage, int nMilliseconds)
```

```
int FlashMessage(string strMessage, int nMilliseconds)
```

6.9.2.1. Asynchronous

```
int FlashMessageAsync(ref int pnSession, int nWantResult, string strMessage, int nMilliseconds)
```

```
int FlashMessageAsyncRetrieve(int nSession, ref TError tErr)
```

```
int FlashMessageAsyncRetrieve(int nSession)
```

6.9.3. Parameters

- TError tErr Error structure to fill
- string strMessage Message to display
- int nMilliseconds duration of displaying the message

6.9.4. Returns

0 if successful, non-zero value otherwise.

6.9.5. Action

Displays a message on top of the navigator. Uses the same kind of output you see when starting the Navigator (Welcome message).

6.10. GetDataSetInfo

6.10.1. Version

TTN: 2.0

TTNCF: 1.0 **KEEPALIVE**

6.10.2. Declaration(s)

```
int GetDataSetInfo(ref TError tErr, ref TDataSetInfo DSI)
```

```
int GetDataSetInfo(ref TDataSetInfo DSI)
```

6.10.2.1. Asynchronous

```
int GetDataSetInfoAsync(ref int pnSession, int nWantResult)
```

```
int GetDataSetInfoAsyncRetrieve(int nSession, ref TError tErr, ref TDataSetInfo DSI)
```

```
int GetDataSetInfoAsyncRetrieve(int nSession, ref TDataSetInfo DSI)
```

6.10.3. Parameters

- TError tErr Error structure to fill
- TDataSetInfo DSI data set information

6.10.4. Returns

0 if successful, non-zero value otherwise.

6.10.5. Action

Get information about the currently selected data set (Map).

6.10.5.1. Hint

Since this call also returns the path to the dataset it is useful to place certain files there.

6.11. GetRouteInfo

6.11.1. Version

TTN: 2.0 TTNCf: 1.0 **KEEPALIVE**

6.11.2. Declaration(s)

```
int GetRouteInfo(ref TError tErr, ref TRouteInfo RI)
int GetRouteInfo(ref TRouteInfo RI)
```

6.11.2.1. Asynchronous

```
int GetRouteInfoAsync(ref int pnSession, int nWantResult)
int GetRouteInfoAsyncRetrieve(int nSession, ref TError tErr, ref TRouteInfo RI)
int GetRouteInfoAsyncRetrieve(int nSession, ref TRouteInfo RI)
```

6.11.3. Parameters

- TError tErr Error structure to fill
- TRouteInfo RI route information

6.11.4. Returns

0 if successful, non-zero value otherwise.

6.11.5. Action

Get information about the current route.

6.12. GetRouteCoordinates

6.12.1. Version

TTN: 3.0 TTNCf: 3.0 **KEEPALIVE**

6.12.2. Declaration(s)

```
int GetRouteCoordinates(ref TError tErr, int nMaxNrCoords, ref TRouteCoordinates rCs)
int GetRouteCoordinates(int nMaxNrCoords, ref TRouteCoordinates rCs)
int GetRouteCoordinates(ref TError tErr, ref TRouteCoordinates rCs)
int GetRouteCoordinates(ref TRouteCoordinates rCs)
```

6.12.2.1. Asynchronous

```
int GetRouteCoordinatesAsync(ref int pnSession, int nWantResult)
int GetRouteCoordinatesAsyncRetrieve(int nSession, ref TError tErr, int
nMaxNrCoords, ref TRouteCoordinates rCs)
int GetRouteCoordinatesAsyncRetrieve(int nSession, int nMaxNrCoords, ref
TRouteCoordinates rCs)
int GetRouteCoordinatesAsyncRetrieve(int nSession, ref TError tErr, ref
TRouteCoordinates rCs)
int GetRouteCoordinatesAsyncRetrieve(int nSession, ref TRouteCoordinates rCs)
```

6.12.3. Parameters

- TError tErr Error structure to fill
- int nMaxNrCoords limit number of coordinates returned (default == 1000)
- TRouteCoordinates rCs route information to retrieve

6.12.4. Returns

0 if successful, non-zero value otherwise.

6.12.5. Action

Get the coordinates of the current route.

6.13. *GetLocationInfo*

6.13.1. Version

TTN: 2.25

TTNCF: 3.0 **KEEPALIVE**

6.13.2. Declaration(s)

```
int GetLocationInfo(int nLongi, int nLati, ref TLocationInfo L)
int GetLocationInfo(ref TError tErr, int nLongitude, int nLatitude, ref TLocationInfo LI)
int GetLocationInfo(double dLongi, double dLati, ref TLocationInfo LI)
int GetLocationInfo(ref TError tErr, double dLongitude, double dLatitude, ref
TLocationInfo LI)
```

6.13.2.1. Asynchronous

```
int GetLocationInfoAsync(ref int pnSession, int nWantResult, int nLatitude, int
nLongitude)
int GetLocationInfoAsync(ref int pnSession, int nWantResult, double dLatitude,
double dLongitude)
int GetLocationInfoAsyncRetrieve(int nSession, ref TLocationInfo LI)
int GetLocationInfoAsyncRetrieve(int nSession, ref TError tErr, ref TLocationInfo LI)
```

6.13.3. Parameters

- **TError** tErr Error structure to fill
- **int** nLongitude longitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nLatitude latitude specified in WGS84 format (in 100,000ths of a degree)
- **TLocationInfo** LI structure that will be filled with the location information

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.13.4. Returns

0 if successful, non-zero value otherwise.

6.13.5. Action

Get information about the location closest to the given point. In other words, perform reverse geocoding, or 'geodecoding'.

6.14. *GetLocationInfoEx*

6.14.1. Version

TTN: 5.0 TTNCf: 5.0 **KEEPALIVE**

6.14.2. Declaration(s)

```
int GetLocationInfoEx(int nLongitude, int nLatitude, ref TLocationInfoEx Llx)
int GetLocationInfoEx(ref TError tErr, int nLongitude, int nLatitude, ref TLocationInfoEx Llx)
int GetLocationInfoEx(double dLongitude, double dLatitude, ref TLocationInfoEx Llx)
int GetLocationInfoEx(ref TError tErr, double dLongitude, double dLatitude, ref TLocationInfoEx Llx)
```

6.14.2.1. Asynchronous

```
int GetLocationInfoExAsync(ref int pnSession, int nWantResult, int nLatitude, int nLongitude)
int GetLocationInfoExAsync(ref int pnSession, int nWantResult, double dLatitude, double dLongitude)
int GetLocationInfoExAsyncRetrieve(int nSession, ref TLocationInfoEx Llx)
int GetLocationInfoExAsyncRetrieve(int nSession, ref TError tErr, ref TLocationInfoEx Llx)
```

6.14.3. Parameters

- **TError** tErr Error structure to fill
- **int** nLongitude longitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nLatitude latitude specified in WGS84 format (in 100,000ths of a degree)
- **TLocationInfoEx** Llx structure that will be filled with the location information

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.14.4. Returns

0 if successful, non-zero value otherwise.

6.14.5. Action

Get information about the location closest to the given point. In other words, perform reverse geocoding, or 'geodecoding'. With this function you will also get the **CityName**.

6.15. *GetUniqueDeviceId*

6.15.1. Version

TTN: 5.0 TTNCf: 5.0 **KEEPALIVE**

6.15.2. Declaration(s)

```
int GetUniqueDeviceId(ref TDeviceId DI)
int GetUniqueDeviceId(ref TError tErr, ref TDeviceId DI)
```

6.15.2.1. Asynchronous

```
int GetUniqueDeviceIdAsync(ref int pnSession, int nWantResult)
int GetUniqueDeviceIdAsyncRetrieve(int nSession, ref TDeviceId DI)
GetUniqueDeviceIdAsyncRetrieve(int nSession, ref TError tErr, ref TDeviceId DI)
```

6.15.3. Parameters

- **TError** tErr Error structure to fill
- **TDeviceId** DI the unique device ID where TTN runs

6.15.4. Returns

0 if successful, non-zero value otherwise.

6.15.5. Action

Retrieves an ID identifying the device.

6.15.5.1. Hint

You can use this ID for „device fixing“ of your software.

6.16. *NavigateToCoordinates*

6.16.1. Version

TTN: 1.42 TTNCf: 1.0 **KEEPALIVE**

6.16.2. Declaration(s)

```
int NavigateToCoordinates(int nLongitude, int nLatitude, string strName)
int NavigateToCoordinates(ref TError tErr, int nLongitude, int nLatitude, string strName)
int NavigateToCoordinates(double dLongitude, double dLatitude, string strName)
int NavigateToCoordinates(ref TError tErr, double dLongitude, double dLatitude, string strName)
```

6.16.2.1. Asynchronous

```
Int NavigateToCoordinatesAsync(ref int pnSession, int nWantResult, int nLatitude, int nLongitude, string strName)
int NavigateToCoordinatesAsync(ref int pnSession, int nWantResult, double dLatitude, double dLongitude, string strName)
int NavigateToCoordinatesAsyncRetrieve(int nSession)
int NavigateToCoordinatesAsyncRetrieve(int nSession, ref TError tErr)
```

6.16.3. Parameters

- o **TError** tErr Error structure to fill
- o **int** nLongitude longitude spec. in WGS84 format (in 100,000ths of a degree)
- o **int** nLatitude latitude specified in WGS84 format (in 100,000ths of a degree)
- o **string** strName name of the location (shown in the route description)

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.16.4. Returns

0 if successful, non-zero value otherwise.

6.16.5. Action

Navigate to a location

6.16.5.1. Hint

If you use the long version be sure to have 5 digits after the degrees!!!

6.16.6. See also

[NavigateToFavorite](#), [ShowCoordinatesOnMap](#)

6.17. *NavigateToFavorite*

6.17.1. Version

TTN: 1.42 TTNCf: 1.0 **KEEPALIVE**

6.17.2. Declaration(s)

```
int NavigateToFavorite(int nIDX)
int NavigateToFavorite(ref TError tErr, int nIDX)
```

6.17.2.1. Asynchronous

```
Int NavigateToFavoriteAsync(ref int pnSession, int nWantResult, int nIDX)
int NavigateToFavoriteAsyncRetrieve(int nSession)
int NavigateToFavoriteAsyncRetrieve(int nSession, ref TError tErr)
```

6.17.3. Parameter

- o **TError** tErr Error structure to fill
- o **int** nIDX index of favorite (base zero)

6.17.4. Returns

0 if successful, non-zero value otherwise.

6.17.5. Action

Navigates to a specific favorite with the index nIndex (base zero).

6.17.6. See also

[SetFavorite](#), [GetFavorite](#), [ClearFavorite](#)

6.18. NavigateToAddress

6.18.1. Version

TTN: 5.0 TTNCf: 5.0 **KEEPALIVE**

6.18.2. Declaration(s)

```
int NavigateToAddress(string strCity, string strStreet, string strHouseNr, string  
strPostcode)  
int NavigateToAddress(ref TError tErr, string strCity, string strStreet, string strHouseNr,  
string strPostcode)
```

6.18.2.1. Asynchronous

```
int NavigateToAddressAsync(ref int pnSession, int nWantResult, string strCity, string  
strStreet, string strHouseNr, string strPostcode)  
int NavigateToAddressAsyncRetrieve(int nSession)  
int NavigateToAddressAsyncRetrieve(int nSession, ref TError tErr)
```

6.18.3. Parameters

- **TError** tErr Error structure to fill
- **string** strCity city name
- **string** strStreet street name
- **string** strHouseNr house number
- **string** strPostcode postcode of the destination

6.18.4. Returns

0 if successful, non-zero value otherwise.

6.18.5. Action

Navigates to a specific address.

6.19. NavigateToPostalCode

6.19.1. Version

TTN: 5.2 TTNCf: 5.2 **KEEPALIVE**

6.19.2. Declaration(s)

```
int NavigateToPostalCode(string strCountry, string strPostcode, string strHouseNr)  
int NavigateToPostalCode(ref TError tErr, string strCountry, string strPostcode, string  
strHouseNr)
```

6.19.2.1. Asynchronous

```
int NavigateToPostalCodeAsync(ref int pnSession, int nWantResult, string strCountry,  
string strPostcode, string strHouseNr)  
int NavigateToPostalCodeAsyncRetrieve(int nSession)  
int NavigateToPostalCodeAsyncRetrieve(int nSession, ref TError tErr)
```

6.19.3. Parameters

- **TError** tErr Error structure to fill
- **string** strCountry Country name
- **string** strPostcode postcode of the destination
- **string** strHouseNr house number

6.19.4. Returns

0 if successful, non-zero value otherwise.

6.19.5. Action

Navigates to a specific PostalCode – at the moment only supported for UK and NL.

6.20. SetRouteSummaryVisible

6.20.1. Version

TTN: 5.20

TTNCF: 5.2 KEEPALIVE

6.20.2. Declaration(s)

int SetRouteSummaryVisible(**bool** bVisible)

int SetRouteSummaryVisible(**ref** TError tErr, **bool** bVisible)

6.20.2.1. Asynchronous

int SetRouteSummaryVisibleAsync(**ref int** pnSession, **int** nWantResult, **bool** bVisible)

int SetRouteSummaryVisibleAsyncRetrieve(**int** nSession)

int SetRouteSummaryVisibleAsyncRetrieve(**int** nSession, **ref TError** tErr)

6.20.3. Parameters

- **TError** tErr Error structure to fill
- **bool** bVisible wheter the summary window should be shown or not

6.20.4. Returns

0 if successful, non-zero value otherwise.

6.20.5. Action

Disables or enables the display of the summary window after route calculation.

6.21. ClearRoute

6.21.1. Version

TTN: 5.20

TTNCF: 5.2 KEEPALIVE

6.21.2. Declaration(s)

int ClearRoute()

int ClearRoute(**ref** TError tErr)

6.21.2.1. Asynchronous

int ClearRouteAsync(**ref int** pnSession, **int** nWantResult)

int ClearRouteAsyncRetrieve(**int** nSession)

int ClearRouteAsyncRetrieve(**int** nSession, **ref TError** tErr)

6.21.3. Parameters

- **TError** tErr Error structure to fill

6.21.4. Returns

0 if successful, non-zero value otherwise.

6.21.5. Action

Clears the current active route.

6.21.5.1. Hint

This function is usefull if a driver almost reaches his destination and then stops. Most common – he found a parking place. Later when he states again (without that call) and leaves the Navigator still tries to guid him to the destination.

6.22. ShowCoordinatesOnMap

6.22.1. Version

TTN: 1.42

TTNCF: 1.0 **KEEPALIVE**

6.22.2. Declaration(s)

```
int ShowCoordinatesOnMap(int nLongitude, int nLatitude)
int ShowCoordinatesOnMap(double dLongitude, double dLatitude)
int ShowCoordinatesOnMap(ref TError tErr, int nLongitude, int nLatitude)
int ShowCoordinatesOnMap(ref TError tErr, double dLongitude, double dLatitude)
```

6.22.2.1. Asynchronous

```
int ShowCoordinatesOnMapAsync(ref int pnSession, int nWantResult, int nLatitude, int
nLongitude)
int ShowCoordinatesOnMapAsync(ref int pnSession, int nWantResult, double dLatitude,
double dLongitude)
int ShowCoordinatesOnMapAsyncRetrieve(int nSession)
int ShowCoordinatesOnMapAsyncRetrieve(int nSession, ref TError tErr)
```

6.22.3. Parameters

- **TError** tErr Error structure to fill
- **int** nLongitude longitude spec. in WGS84 format (in 100,000ths of a degree)
- **int** nLatitude latitude specified in WGS84 format (in 100,000ths of a degree)

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.22.4. Returns

0 if successful, non-zero value otherwise.

6.22.5. Action

Centers the map on the given coordinate (zoomed in).

6.22.6. See also

[NavigateToCoordinate](#), [ShowRectangleOnMap](#)

6.23. ShowRectangleOnMap

6.23.1. Version

TTN: 3.0

TTNCF: 3.0 **KEEPALIVE**

6.23.2. Declaration(s)

```
int ShowRectangleOnMap(int nLongiW, int nLatiS, int nLongiE, int nLatiN)
int ShowRectangleOnMap(double dLongiW, double dLatiS, double dLongiE, double
dLatiN)
int ShowRectangleOnMap(ref TError tErr, int nLongiW, int nLatiS, int nLongiE, int
nLatiN)
int ShowRectangleOnMap(ref TError tErr, double dLongiW, double dLatiS, double
dLongiE, double dLatiN)
```

6.23.2.1. Asynchronous

```
Int ShowRectangleOnMapAsync(ref int pnSession, int nWantResult, int nLongiW, int
nLatiS, int nLongiE, int nLatiN)
int ShowRectangleOnMapAsync(ref int pnSession, int nWantResult, double dLongiW,
double dLatiS, double dLongiE, double dLatiN)
int ShowRectangleOnMapAsyncRetrieve(int nSession)
int ShowRectangleOnMapAsyncRetrieve(int nSession, ref TError tErr)
```

6.23.3. Parameters

- **TError** tErr Error structure to fill
- **int** nLongiW western longitude spec.in WGS84 format (in 100,000ths of a degree)
- **int** nLatiS southern latitude spec.in WGS84 format (in 100,000ths of a degree)
- **int** nLongiE eastern longitude spec.in WGS84 format (in 100,000ths of a degree)
- **int** nLatiN northern latitude spec.in WGS84 format (in 100,000ths of a degree)

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.23.4. Returns

0 if successful, non-zero value otherwise.

6.23.5. Action

Show the given rectangle of the map.

6.23.6. See also

[ShowCoordinatesOnMap](#)

6.24. ShowAddressOnMap

6.24.1. Version

TTN: 5.0 TTNCf: 5.0 **KEEPALIVE**

6.24.2. Declaration(s)

```
int ShowAddressOnMap(string strCity, string strStreet, string strHouseNr, string  
strPostcode)  
int ShowAddressOnMap(ref TError tErr, string strCity, string strStreet, string  
strHouseNr, string strPostcode)
```

6.24.2.1. Asynchronous

```
int ShowAddressOnMapAsync(ref int pnSession, int nWantResult, string strCity, string  
strStreet, string strHouseNr, string strPostcode)  
int ShowAddressOnMapAsyncRetrieve(int nSession)  
int ShowAddressOnMapAsyncRetrieve(int nSession, ref TError tErr)
```

6.24.3. Parameters

- **TError** tErr Error structure to fill
- **string** strCity city name
- **string** strStreet street name
- **string** strHouseNr house number
- **string** strPostcode the postcode of the address

6.24.4. Returns

0 if successful, non-zero value otherwise.

6.24.5. Action

Shows a specific address on the map.

6.25. ShowPostalCodeOnMap

6.25.1. Version

TTN: 5.2 TTNCf: 5.2 **KEEPALIVE**

6.25.2. Declaration(s)

```
int ShowPostalCodeOnMap(string strCountry, string strPostcode, string strHouseNr)  
int ShowPostalCodeOnMap(ref TError tErr, string strCountry, string strPostcode, string  
strHouseNr)
```

6.25.2.1. Asynchronous

```
int ShowPostalCodeOnMapAsync(ref int pnSession, int nWantResult, string strCountry,  
string strPostcode, string strHouseNr)  
int ShowPostalCodeOnMapAsyncRetrieve(int nSession)  
int ShowPostalCodeOnMapAsyncRetrieve(int nSession, ref TError tErr)
```

6.25.3. Parameters

- **TError** tErr Error structure to fill
- **string** strCountry Country name
- **string** strPostcode postcode of the destination
- **string** strHouseNr house number

6.25.4. Returns

0 if successful, non-zero value otherwise.

6.25.5. Action

Shows a specific PostalCode on the map – at the moment only supported for UK and NL.

6.26. ChangeSetting

6.26.1. Version

TTN: 5.0 TTNCf: 5.0 KEEPALIVE

6.26.2. Declaration(s)

```
int ChangeSetting(ESettingAndValue eSV)
int ChangeSetting(ref TError tErr, ESettingAndValue eSV)
```

6.26.2.1. Asynchronous

```
int ChangeSettingAsync(ref int pnSession, int nWantResult, ESettingAndValue eSV)
int ChangeSettingAsyncRetrieve(int nSession)
int ChangeSettingAsyncRetrieve(int nSession, ref TError tErr)
```

6.26.3. Parameters

- TError tErr Error structure to fill
- ESettingAndValue eSV the structur to fill

6.26.4. Returns

0 if successful, non-zero value otherwise.

6.26.5. Action

Changes one of the supported settings, like sound on, show map, etc.

6.27. SwitchMap

6.27.1. Version

TTN: 3.0 TTNCf: 3.0 KEEPALIVE

6.27.2. Declaration(s)

```
int SwitchMap(string strPath)
int SwitchMap(ref TError tErr, string strPath)
```

6.27.2.1. Asynchronous

```
int SwitchMapAsync(ref int pnSession, int nWantResult, string strPath)
int SwitchMapAsyncRetrieve(int nSession)
int SwitchMapAsyncRetrieve(int nSession, ref TError tErr)
```

6.27.3. Parameters

- TError tErr Error structure to fill
- string strPath Full path and Name of the PNA file

6.27.4. Returns

0 if successful, non-zero value otherwise.

6.27.5. Action

Switch to the map specified by the full name and path of the .PNA file.

6.28. SetNavigatorProperty

6.28.1. Version

TTN: 3.0

TTNCF: 3.0 **KEEPALIVE**

6.28.2. Declaration(s)

```
int SetNavigatorProperty(EProperty eProp, int nValue)
int SetNavigatorProperty(EProperty eProp, EPVColor ePVC)
int SetNavigatorProperty(ref TError tErr, EProperty eProp, int nValue)
int SetNavigatorProperty(ref TError tErr, EProperty eProp, EPVColor ePVC)
```

6.28.2.1. Asynchronous

```
int SetNavigatorPropertyAsync(ref int pnSession, int nWantResult, EProperty eProp, int nValue)
int SetNavigatorPropertyAsync(ref int pnSession, int nWantResult, EProperty eProp, EPVColor ePVC)
int SetNavigatorPropertyAsyncRetrieve(int nSession)
int SetNavigatorPropertyAsyncRetrieve(int nSession, ref TError tErr)
```

6.28.3. Parameters

- **TError** tErr Error structure to fill
- **EProperty** eProp property to set
- **int** nValue pure numeric value of the property
- **EPVColor** ePVC enumeration for color values

6.28.4. Returns

0 if successful, non-zero value otherwise.

6.28.5. Action

Sets the value for one of the properties (user settings), like current color schemes, voice volume, etc.

6.28.5.1. Hint

We built an extra functions with **EPVColor** as parameter to make the use more save.

6.29. GetCurrentPosition

6.29.1. Version

TTN: 1.42 TTNCf: 1.0 **KEEPALIVE**

6.29.2. Declaration(s)

```
int GetCurrentPosition(ref TCurrentPositionInfo tCP)
int GetCurrentPosition(ref TError tErr, ref TCurrentPositionInfo tCP)
```

6.29.2.1. Asynchronous

```
int GetCurrentPositionAsync(ref int pnSession, int nWantResult)
int GetCurrentPositionAsyncRetrieve(int nSession, ref TCurrentPositionInfo tCP)
int GetCurrentPositionAsyncRetrieve(int nSession, ref TError tErr, ref
TCurrentPositionInfo tCP)
```

6.29.3. Parameters

- TError tErr Error structure to fill
- TCurrentPositionInfo tCP structure to fill with the values of the current position

6.29.4. Returns

0 if successful, non-zero value otherwise.

6.29.5. Action

Fills the structure passed in with current position information from GPS.

6.30. Geocode

6.30.1. Version

TTN: 1.51 TTNCf: 1.0 **KEEPALIVE**

6.30.2. Declaration(s)

```
int Geocode(string strCity, string strStreet, string strHouseNr, ref TGeocodeResult tGR)
int Geocode(ref TError tErr, string strCity, string strStreet, string strHouseNr, ref
TGeocodeResult tGR)
```

6.30.2.1. Asynchronous

```
int GeocodeAsync(ref int pnSession, int nWantResult, string strCity, string strStreet,
string strHouseNr)
int GeocodeAsyncRetrieve(int nSession, ref TGeocodeResult tGR)
int GeocodeAsyncRetrieve(int nSession, ref TError tErr, ref TGeocodeResult tGR)
```

6.30.3. Parameters

- TError tErr Error structure to fill
- string strCity city name
- string strStreet street name
- string strHouseNr house number
- TGeocodeResult tGR filled with the results of geocoding

6.30.4. Returns

0 if successful, non-zero value otherwise.

6.30.5. Action

Returns geocode (latitude and longitude) for the given address, specified by the city name, street name and house number arguments, and returns the co-ordinates of the found location and its complete city and street names in the geocode information argument. If the street name argument is empty, the function will try to find the center of the city specified by the city name argument. If no location is found with city and street names exactly matching the arguments, then a city and a street will be chosen with names as similar as possible to the given ones. This will be indicated in the status field of the geocode information argument.

6.31. GeocodeExtended

6.31.1. Version

TTN: 2.22

TTNCF: 3.0 **KEEPALIVE**

6.31.2. Declaration(s)

```
int GeocodeExtended(string strCity, string strStreet, string strHouseNr, string  
strPostcode, ref TGeocodeResult tGR)  
int GeocodeExtended(ref TError tErr, string strCity, string strStreet, string strHouseNr,  
string strPostcode, ref TGeocodeResult tGR)
```

6.31.2.1. Asynchronous

```
int GeocodeExtendedAsync(ref int pnSession, int nWantResult, string strCity, string  
strStreet, string strHouseNr, string strPostcode)  
int GeocodeExtendedAsyncRetrieve(int nSession, ref TGeocodeResult tGR)  
int GeocodeExtendedAsyncRetrieve(int nSession, ref TError tErr, ref TGeocodeResult  
tGR)
```

6.31.3. Parameters

- | | | |
|-------------------------|------------|--------------------------------------|
| ○ <u>TError</u> | tErr | Error structure to fill |
| ○ string | strCity | city name |
| ○ string | strStreet | street name |
| ○ string | strHouseNr | house number |
| ○ <u>TGeocodeResult</u> | tGR | filled with the results of geocoding |

6.31.4. Returns

0 if successful, non-zero value otherwise.

6.31.5. Action

Returns geocode (latitude and longitude) for the given address, specified by the city name, street name, house number and postal code arguments, and return the co-ordinates of the found location and its complete city and street names in the geocode information argument. If the street name argument is empty, the function will try to find the center of the city specified by the city name argument. . If no location is found with city and street names exactly matching the arguments, then a city and a street will be chosen with names as similar as possible to the given ones. This will be indicated in the status field of the geocode information argument.

6.31.5.1. Hint

This extended version accepts also the postcode in geocoding. This brings better results as Geocode (especially with map which have a postcode file included – for an example UK).

6.32. GeocodeByPostalCode

6.32.1. Version

TTN: 5.2 TTNCf: 5.2 **KEEPALIVE**

6.32.2. Declaration(s)

```
int GeocodeByPostalCode(string strCountry, string strPostcode, string strHouseNr,  
ref TGeocodeResult tGR)
```

```
int GeocodeByPostalCode(ref TError tErr, string strCountry, string strPostcode, string  
strHouseNr, ref TGeocodeResult tGR)
```

6.32.2.1. Asynchronous

```
int GeocodeByPostalCodeAsync(ref int pnSession, int nWantResult, string strCountry,  
string strHouseNr, string strPostcode)
```

```
int GeocodeByPostalCodeAsyncRetrieve(int nSession, ref TGeocodeResult tGR)
```

```
int GeocodeByPostalCodeAsyncRetrieve(int nSession, ref TError tErr, ref  
TGeocodeResult tGR)
```

6.32.3. Parameters

- TError tErr Error structure to fill
- string strCountry country name
- string strPostcode postcode
- string strHouseNr house number
- TGeocodeResult tGR filled with the results of geocoding

6.32.4. Returns

0 if successful, non-zero value otherwise.

6.32.5. Action

Returns geocode (latitude and longitude) for the given address, specified by the city name, postal code, and house number arguments, and return the co-ordinates of the found location and its complete city and street names in the geocode information argument.

At the moment only supported for UK and NL

6.33. ClearFavorite

6.33.1. Version

TTN: 1.42 TTNCf: 1.0 **KEEPALIVE**

6.33.2. Declaration(s)

```
int ClearFavorite(int nIDX)
```

```
int ClearFavorite(ref TError tErr, int nIDX)
```

6.33.2.1. Asynchronous

```
int ClearFavoriteAsync(ref int pnSession, int nWantResult, int nIDX)
```

```
int ClearFavoriteAsyncRetrieve(int nSession)
```

```
int ClearFavoriteAsyncRetrieve(int nSession, ref TError tErr)
```

6.33.3. Parameters

- TError tErr Error structure to fill
- int nIDX index of favorite (base zero)

6.33.4. Returns

0 if successful, non-zero value otherwise.

6.33.5. Action

Clears a favorite.

6.33.6. See also

SetFavorite, GetFavorite, NavigateToFavorite

6.34. SetFavorite

6.34.1. Version

TTN: 1.42

TTNCF: 1.0 **KEEPALIVE**

6.34.2. Declaration(s)

```
int SetFavorite(int nIDX, string strName, string strDescription, int nLongitude, int nLatitude)
int SetFavorite(ref TError tErr, int nIDX, string strName, string strDescription, int nLongitude, int nLatitude)
int SetFavorite(int nIDX, string strName, string strDescription, double dLongitude, double dLatitude)
int SetFavorite(ref TError tErr, int nIDX, string strName, string strDescription, double dLongitude, double dLatitude)
int SetFavorite(int nIDX, TFavoriteInfo fi)
int SetFavorite(ref TError tErr, int nIDX, TFavoriteInfo fi)
```

6.34.2.1. Asynchronous

```
Int SetFavoriteAsync(ref int pnSession, int nWantResult, int nIDX, string strName, string strDescription, int nLongitude, int nLatitude)
int SetFavoriteAsync(ref int pnSession, int nWantResult, int nIDX, string strName, string strDescription, double dLongitude, double dLatitude)
int SetFavoriteAsyncRetrieve(int nSession)
int SetFavoriteAsyncRetrieve(int nSession, ref TError tErr)
```

6.34.3. Parameters

- **TError** tErr Error structure to fill
- **int** nIndex index of favorite (base zero)
- **string** strName Name the favorite should get
- **string** strDescription Description of the favorite
- **int** nLongitude longitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nLatitude latitude specified in WGS84 format (in 100,000ths of a degree)
- **TFavoriteInfo** fi struture containing the information

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.34.4. Returns

0 if successful, non-zero value otherwise.

6.34.5. Action

Assigns the data to the element from the favourites array with the given index (zero-based).

6.34.5.1. Hint

For you convinience we added a version where you can pass the values in a [TFavoriteInfo](#) structure. This makes it easy to use the result of [GetFavorite](#).

6.34.6. See also

[GetFavorite](#), [ClearFavorite](#), [NavigateToFavorite](#)

6.35. GetFavorite

6.35.1. Version

TTN: 1.42

TTNCF: 1.0 **KEEPALIVE**

6.35.2. Declaration(s)

int GetFavorite(**ref** **TError** tErr, **int** nIDX, **ref** **TFavoriteInfo** tFI)

int GetFavorite(**ref** **TError** tErr, **int** nIDX, **ref** **TFavoriteInfo** tFI)

6.35.2.1. Asynchronous

int GetFavoriteAsync(**ref** **int** pnSession, **int** nWantResult, **int** nIDX)

int GetFavoriteAsyncRetrieve(**int** nSession, **ref** **TFavoriteInfo** tFI)

int GetFavoriteAsyncRetrieve(**int** nSession, **ref** **TError** tErr, **ref** **TFavoriteInfo** tFI)

6.35.3. Parameters

- **TError** tErr Error structure to fill
- **int** nIDX index of favorite (base zero)
- **TFavoriteInfo** tFI struture to retrieve the information

6.35.4. Returns

0 if successful, non-zero value otherwise.

6.35.5. Action

Get favorite information.

6.35.6. See also

[SetFavorite](#), [ClearFavorite](#), [NavigateToFavorite](#)

6.36. MakePoiVisible

6.36.1. Version

TTN: 1.42

TTNCF: 1.0 **KEEPALIVE**

6.36.2. Declaration(s)

int MakePoiVisible(**EPoiTypeCode** eTypeCode, **int** nVisibility)

int MakePoiVisible(**ref** **TError** tErr, **EPoiTypeCode** eTypeCode, **int** nVisibility)

6.36.2.1. Asynchronous

int MakePoiVisibleAsync(**ref** **int** pnSession, **int** nWantResult, **EPoiTypeCode** eTypeCode, **int** nVisibility)

int MakePoiVisibleAsyncRetrieve(**int** nSession)

int MakePoiVisibleAsyncRetrieve(**int** nSession, **ref** **TError** tErr)

6.36.3. Parameters

- **TError** tErr Error structure to fill
- **EPoiTypeCode** ePoiID POI unique type identifier
- **int** nVisibility show or hide this POI type on the map

6.36.4. Returns

0 if successful, non-zero value otherwise.

6.36.5. Action

Enable or disable displaying a specific POI type.

6.36.6. See also

[AddPoi](#), [DeleteAllPoi](#), [DeleteClosestPoi](#), [MakeUserPoiVisible](#), [MoveClosestPoi](#)

6.37. AddPoi

6.37.1. Version

TTN: 1.51

TTNCF: 1.0 **KEEPALIVE**

6.37.2. Declaration(s)

```
int AddPoi(string strFName, int nLongi, int nLati, string strName, string strID)
int AddPoi(string strFName, int nLongi, int nLati, string strName)
int AddPoi(string strFName, double dLongi, double dLati, string strName, string strID)
int AddPoi(string strFName, double dLongi, double dLati, string strName)
int AddPoi(ref TError tErr, string strFName, int nLongi, int nLati, string strName, string strID)
int AddPoi(ref TError tErr, string strFName, int nLongi, int nLati, string strName)
int AddPoi(ref TError tErr, string strFName, double dLongi, double dLati, string strName, string strID)
int AddPoi(ref TError tErr, string strFName, double dLongi, double dLati, string strName)
int AddPoi(string strFName, double dLongi, double dLati, string strName)
```

6.37.2.1. Asynchronous

```
int AddPoiAsync(ref int pnSession, int nWantResult, string strFName, int nLongi, int nLati, string strName, string strID)
int AddPoiAsync(ref int pnSession, int nWantResult, string strFName, int nLongi, int nLati, string strName)
int AddPoiAsync(ref int pnSession, int nWantResult, string strFName, double dLongi, double dLati, string strName, string strID)
int AddPoiAsync(ref int pnSession, int nWantResult, string strFName, double dLongi, double dLati, string strName)
int AddPoiAsyncRetrieve(int nSession)
int AddPoiAsyncRetrieve(int nSession, ref TError tErr)
```

6.37.3. Parameters:

- **TError** tErr Error structure to fill
- **string** strFName name of the POI category
- **int** nLongi longitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nLati latitude specified in WGS84 format (in 100,000ths of a degree)
- **string** strName name of the POI
- **string** strID **optional** external identifier of the POI

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.37.4. Returns

0 if successful, non-zero value otherwise.

6.37.5. Action

Add a point of interest to the .OV2 file specified by the POI category argument. The point of interest will have the name and co-ordinates given by the corresponding arguments. An external identifier string can also be attached to the point of interest.

When the .OV2 file does not yet exist, it is created.

6.37.6. See also

[DeleteAllPoi](#), [DeleteClosestPoi](#), [MakePoiVisible](#), [MakeUserPoiVisible](#), [MoveClosestPoi](#)

6.38. DeleteClosestPoi

6.38.1. Version

TTN: 1.51 TTNCf: 1.0 **KEEPALIVE**

6.38.2. Declaration(s)

```
int DeleteClosestPoi(string strFName, int nLongi, int nLati)
int DeleteClosestPoi(string strFName, double dLongi, double dLati)
int DeleteClosestPoi(ref TError tErr, string strFName, int nLongi, int nLati)
int DeleteClosestPoi(ref TError tErr, string strFName, double dLongi, double dLati)
```

6.38.2.1. Asynchronous

```
int DeleteClosestPoiAsync(ref int pnSession, int nWantResult, string strFName, int
nLongi, int nLati)
int DeleteClosestPoiAsync(ref int pnSession, int nWantResult, string strFName, double
dLongi, double dLati)
int DeleteClosestPoiAsyncRetrieve(int nSession)
int DeleteClosestPoiAsyncRetrieve(int nSession, ref TError tErr)
```

6.38.3. Parameters

- **TError** tErr Error structure to fill
- **string** strFName name of the POI category
- **int** nLongi longitude spec. in WGS84 format (in 100,000ths of a degree)
- **int** nLati latitude specified in WGS84 format (in 100,000ths of a degree)

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.38.4. Returns

0 if successful, non-zero value otherwise.

6.38.5. Action

Delete a point of interest from the .OV2 file specified by the POI category argument. The point of interest to be deleted is the closest one to the given co-ordinates among the points of interest that are within a distance of 100 meters from there.

6.38.6. See also

[AddPoi](#), [DeleteAllPoi](#), [MakePoiVisible](#), [MakeUserPoiVisible](#), [MoveClosestPoi](#)

6.39. DeleteAllPoi

6.39.1. Version

TTN: 1.51 TTNCf: 1.0 **KEEPALIVE**

6.39.2. Declaration(s)

```
int DeleteAllPoi(string strFName)
int DeleteAllPoi(ref TError tErr, string strFName)
```

6.39.2.1. Asynchronous

```
int DeleteAllPoiAsync(ref int pnSession, int nWantResult, string strFName)
int DeleteAllPoiAsyncRetrieve(int nSession)
int DeleteAllPoiAsyncRetrieve(int nSession, ref TError tErr)
```

6.39.3. Parameter

- **TError** tErr Error structure to fill
- **string** strFName name of the POI category

6.39.4. Returns

0 if successful, non-zero value otherwise.

6.39.5. Action

Delete all points of interest from the .OV2 file specified by the POI category argument. The POI category itself is also removed after that.

6.39.6. See also

[AddPoi](#), [DeleteClosestPoi](#), [MakePoiVisible](#), [MakeUserPoiVisible](#), [MoveClosestPoi](#)

6.40. MakeUserPoiVisible

6.40.1. Version

TTN: 2.0

TTNCF: 1.0 **KEEPALIVE**

6.40.2. Declaration(s)

int MakeUserPoiVisible(**string** strFName, **int** nVisibility)

int MakeUserPoiVisible(**ref** **TError** tErr, **string** strFName, **int** nVisibility)

6.40.2.1. Asynchronous

int MakeUserPoiVisibleAsync(**ref** **int** pnSession, **int** nWantResult, **string** strFNam, **int** nVisibility)

int MakeUserPoiVisibleAsyncRetrieve(**int** nSession)

int MakeUserPoiVisibleAsyncRetrieve(**int** nSession, **ref** **TError** tErr)

6.40.3. Parameters:

- **TError** **tErr** Error structure to fill
- **string** **strFName** name of the POI category
- **int** **nVisibility** whether to show or hide this POI type on the map

6.40.4. Returns

0 if successful, non-zero value otherwise.

6.40.5. Action

Enable or disable displaying a specific user-created POI type.

6.40.6. See also

[AddPoi](#), [DeleteAllPoi](#), [DeleteClosestPoi](#), [MakePoiVisible](#), [MoveClosestPoi](#)

6.41. MoveClosestPoi

6.41.1. Version

TTN: 1.51

TTNCF: 1.0 **KEEPALIVE**

6.41.2. Declaration(s)

```
int MoveClosestPoi(string strFName, int nLongi, int nLati, int nNewLongi, int nNewLati)
int MoveClosestPoi(string strFName, double dLongi, double dLati, double dNewLongi,
double dNewLati)
int MoveClosestPoi(ref TError tErr, string strFName, int nLongi, int nLati, int nNewLongi,
int nNewLati)
int MoveClosestPoi(ref TError tErr, string strFName, double dLongi, double dLati,
double dNewLongi, double dNewLati)
```

6.41.2.1. Asynchronous

```
int MoveClosestPoiAsync(ref int pnSession, int nWantResult, string strFNam, int nLongi
, int nLati, int nNewLongi, int nNewLati))
int MoveClosestPoiAsync(ref int pnSession, int nWantResult, string strFNam, double
dLongi , double dLati, double dNewLongi, double dNewLati)
int MoveClosestPoiAsyncRetrieve(int nSession)
int MoveClosestPoiAsyncRetrieve(int nSession, ref TError tErr)
```

6.41.3. Parameters

- **TError** tErr Error structure to fill
- **string** strFName name of the POI category
- **int** nLongi old longitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nLati old latitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nNewLongi new longitude specified in WGS84 format (in 100,000ths of a degree)
- **int** nNewLati new latitude specified in WGS84 format (in 100,000ths of a degree)

The double version makes it easier to write constant coordinates.

See: [Changes from the TomTom SDK](#)

6.41.4. Returns

0 if successful, non-zero value otherwise.

6.41.5. Action

Move a point of interest from the .OV2 file specified by the POI category argument to a new position. The point of interest to be moved is the closest one to the given old co-ordinates among the points of interest that are within a distance of 100 meters from there.

6.41.6. See also

[AddPoi](#), [DeleteAllPoi](#), [DeleteClosestPoi](#), [MakePoiVisible](#), [MakeUserPoiVisible](#)

6.42. *EnableExternalGPSInput*

6.42.1. Version

TTN: 5.0

TTNCF: 5.0

6.42.2. Declaration(s)

`int EnableExternalGPSInput(ref TExternalGPSResult tGPS)`

`int EnableExternalGPSInput(ref TError tErr, ref TExternalGPSResult tGPS)`

6.42.2.1. Asynchronous

`int EnableExternalGPSInputAsync(ref int pnSession, int nWantResult)`

`int EnableExternalGPSInputAsyncRetrieve(int nSession, ref TExternalGPSResult tGPS)`

`int EnableExternalGPSInputAsyncRetrieve(int nSession, ref TError tErr, ref TExternalGPSResult tGPS)`

6.42.3. Parameters

- TError tErr Error structure to fill
- TExternalGPSResult tGPS holds the "access handle" later used in SendNMEAData

6.42.4. Returns

0 if successful, non-zero value otherwise.

6.42.5. Action

Enables external GPS input coming from the user instead of GPS input supplied via the Navigator itself. Use TerminateWrapper at the end of your program to disable the external GPS input or call DisableExternalGPSInput instead.

6.42.5.1. Hint

After a call to this function the navigator will close the used port.
Add a short delay before you open the port with your application.
The delay ensures that the resource is freed by the navigator.

6.43. *DisableExternalGPSInput*

6.43.1. Version

TTN: 5.0

TTNCF: 5.0

6.43.2. Declaration(s)

`int DisableExternalGPSInput()`

`int DisableExternalGPSInput(ref TError tErr)`

6.43.2.1. Asynchronous

`int DisableExternalGPSInputAsync(ref int pnSession, int nWantResult)`

`int DisableExternalGPSInputAsyncRetrieve(int nSession)`

`int DisableExternalGPSInputAsyncRetrieve(int nSession, ref TError tErr)`

6.43.3. Parameters

- TError tErr Error structure to fill

6.43.4. Returns

0 if successful, non-zero value otherwise.

6.43.5. Action

Disables external GPS input from the user and reverts to GPS input from the Navigator.

6.43.5.1. Hint

Close the port before calling this function. So the Navigator can reopen it for internal use.
Adding a short delay after closing the port ensures that the resource is freed.

6.44. *EnableRawGPSOutput*

6.44.1. Version

TTN: 5.2

TTNCF: 5.2

6.44.2. Declaration(s)

```
int EnableRawGPSOutput()
```

```
int EnableRawGPSOutput (ref TError tErr)
```

6.44.2.1. Asynchronous

Not available.

6.44.3. Parameters

- o [TError](#) tErr Error structure to fill

6.44.4. Returns

Any negative number indicates a problem.

-2000 a catchable error occurred

-3000 the RawGPSHandling system did not initialize

0 Successfull

6.44.5. Action

This enables the reveiving of raw GPS Data from the TomTom navigator.

You can handle the data via [OnGPSMessage](#) or by providing your own NEMA handler by implementing the interface described at [IGPSInput](#).

6.44.5.1. Hint

Turn of the system before your application terminates by calling [DisableRawGPSOutput](#) or simply use [TerminateWrapper](#).

Only one receiver at a time is allowed by the TomTom Navigator!

6.44.6. See also

[OnGPSMessage](#), [IGPSInput](#)

6.45. *DisableRawGPSOutput*

6.45.1. Version

TTN: 5.2

TTNCF: 5.2

6.45.2. Declaration(s)

```
int DisableRawGPSOutput()
```

```
int DisableRawGPSOutput(ref TError tErr)
```

6.45.2.1. Asynchronous

Not available.

6.45.3. Parameters

- o [TError](#) tErr Error structure to fill

6.45.4. Returns

0 if successful, non-zero value otherwise.

6.45.5. Action

Disables the output of raw GPS data from the TomTom Navigator.

6.45.5.1. Hint

Always call this function before your terminate your application or simply use [TerminateWrapper](#) for general cleanup.

7. Structures and enumerations from the API

All structures differ from the TomTom SDK by the use of str for string, n for Integers, l for longs and finally a length name suffix at some string values! **The sub-numbering is the same as in the SDK 5 documentation.**

7.1. TVersion structure

```
public struct TVersion {  
    public int nBuildNumber;  
    public string strVersion16;  
}
```

7.2. TError structure

```
public struct TError {  
    public int nError;  
}
```

7.3. TDataSetInfo structure

```
public struct TDataSetInfo {  
    public string strDataSetName128;    // name of the dataset  
    public string strDataSetPath128;    // path to the dataset  
    public EDatasetFlags eFlags;  
    public int nBuildNumber;  
}
```

7.3.1. The EDatasetFlags enumeration

Possible values in eFlags of the [TDataSetInfo](#).

```
public enum EDatasetFlags {  
    KHouseNrPrefixed=1, // house numbers are written before street name  
    KLeftSideDriving=2  // indicates left-sided driving  
}
```

7.4. TLocationInfo structure

```
public struct TLocationInfo {  
    public long lLongitude;  
    public long lLatitude;  
    public double dLongitude;    // double accessor for lLongitude  
    public double dLatitude;    // double accessor for lLatitude  
    public ELocDataType eType;  
    public string strStreetName128;  
    public int lnHouseNr1;  
    public int lnHouseNr2;  
}
```

7.5. TLocationInfoEx structure

```
public struct TLocationInfoEx {  
    public long lLongitude;  
    public long lLatitude;  
    public double dLongitude;    // double accessor for lLongitude  
    public double dLatitude;    // double accessor for lLatitude  
    public ELocDataType eType;  
    public string strStreetName128;  
    public int lnHouseNr1;  
    public int lnHouseNr2;  
    public string strCityName128;  
}
```

7.6. *ELocDataType* enumeration

Possible values in eType in [TLocationInfo](#) and in [TLocationInfoEx](#)

```
public enum ELocDataType {
    KMapPosition = 0,      // the location is a position on the map
    KNode = 1              // the location is a singled node (e.g. city center or a pos. selected on map)
    KCrossing = 2          // the location is a crossing
    KStreetPos = 3         // the location is somewhere on a street
    KStreet = 4,           // the location is a street
    KBackToRoute = 5       // found in header file
}
```

7.7. *TRouteInfo* structure

```
public struct TRouteInfo {
    public ERouteInfoStatus eStatus;
    public TLocation Departure; // coordinates for departure and destination
    public TLocation Destination;
                                // rectangle enclosing the route
    public long lEnclosingLongitudeW;
    public double dEnclosingLongitudeW; // double accessor
    public long lEnclosingLatitudeS;
    public double dEnclosingLatitudeS; // double accessor
    public long lEnclosingLongitudeE;
    public double dEnclosingLongitudeE; // double accessor
    public long lEnclosingLatitudeN;
    public double dEnclosingLatitudeN; // double accessor
    public long lLength; // length of route in meters
    public long lDuration; // duration of route in seconds
    public long lDistanceToDestination; // distance current pos to dest in meters
    public long lTimeToDestination; // time from current pos to dest in seconds
}
```

7.7.1. The *ERouteInfoStatus* enumeration

Possible values in eStatus of the [TRouteInfo](#).

```
public enum ERouteInfoStatus { //combined values are possible
    KDepartureSet = 1,          //the departure point has been set
    KDestinationSet = 2,        //the destination point has been set
    KCalculatingRoute = 4,      //in the process of calculating a route
    KRoutePlanned = 8,          //there is a valid route planned
    KLeftToDestinationAvailable = 16 //time + distance to destination available
}
```

7.8. *TLocation* structure

Used in [TRouteInfo](#)

```
public struct TLocation {
    public long lLongitude;
    public long lLatitude;
    public double dLongitude; // double accessor for lLongitude
    public double dLatitude;  // double accessor for lLatitude
}
```

7.9. *TDeviceID* structure

```
public struct TDeviceID {
    public string strDeviceId32;
    public int nDeviceIDLength; // length of the string in strDeviceID32
}
```

7.10. TCurrentPositionInfo structure

This structure differs from the TomTom SDK structure by the double accessors.

```
public struct TCurrentPositionInfo {
    public EGpsStatus eGpsStat;
    public long lLongitude;
    public long lLatitude;
    public double dLongitude; // double accessor for lLongitude
    public double dLatitude; // double accessor for lLatitude
    public int nDirection; // Direction: zero is north, clockwise in degrees
    public int nSpeedKMh;
}
```

7.11. TGeocodeResult structure

```
public struct TGeocodeResult {
    public EGeoCodeStatus eStatus;
    public long lLongitude;
    public long lLatitude;
    public double dLongitude // double accessor for lLongitude
    public double dLatitude; // double accessor for lLatitude
    public string strCityName128;
    public string strStreetName128;
}
```

7.11.1. EGeoCodeStatus enumeration

Possible values in eStatus in [TGeocodeResult](#)

```
public enum EGeoCodeStatus {
    KGeocodeHouseNrIgnored=1, // // the house number was ignored, either because there
                                // was no house number data available or because the given
                                // house number was not found
    KGeocodeStreetAmbiguous = 2, // g. street name ambiguous, tried to make smart selection
    KGeocodeCityAmbiguous = 4, // the given city name was ambiguous, random
                                // (possibly wrong) selection made
    KGeocodeFailure = 8 // no location found at all, the rest of the fields are invalid
}
```

7.12. TFavoriteInfo structure

This structure differs from the TomTom SDK structure by the double access and the limited (programmatically cut of) strings!

```
public struct TFavoriteInfo {
    public string strName64; // limited to 63 characters text
    public string strDescription128; // limited to 128 characters text
    public long lLongitude;
    public long lLatitude;
    public double dLongitude; // double accessor for lLongitude
    public double dLatitude; // double accessor for lLatitude
}
```

7.13. TRouteCoordinates structure

```
public struct TRouteCoordinates {
    public int nNrCoordinates; // number of coordinates in the route
    public TLocation [] aCoordinates; // array with coordinates
}
```

7.14. TExternalGPSResult structure

```
public struct TExternalGPSResult {  
    public int nResult;    // "handle" for GPS calls  
}
```

7.15. ESettingAndValue enumeration

```
public enum ESettingAndValue {  
    KCommandSoundOn= 0,    // turn spoken voice instructions on  
    KCommandSoundOff= 1,   // turn spoken voice instructions off  
    KCommandNightColors=2, // show map in night colors  
    KCommandDayColors=3,   // show map in day colors  
    KCommandShowMap=4,     // show map and instructions while driving  
    KCommandHideMap=5      // show "instructions only" view while driving  
}
```

7.16. EProperty enumeration

```
public enum EProperty {  
    KPropertyColorStyle= 0,    // colour style used in daylight mode -- see EPVColor  
    KPropertyNightColorStyle= 1, // the colour style used in night mode -- see EPVColor  
    KPropertyVoiceVolume=2,    // the volume of the voice prompts  
    KPropertySafetySpeed=3,    // show map in day colors  
    KPropertyMaxPoiOnMap=4     // found in header file  
}
```

7.16.1. The EPVColor enumeration

```
public enum EPVColor {    // map color styles  
    KColorStyleBelgica=1027,  
    KColorStyleBrittanica=1028,  
    KColorStyleGermanica=1029,  
    KColorStyleAustralia=1030,  
    KColorStyleDeuteranopia=1031,  
    KColorStyleGreys=1032,  
    KColorStyleAmerica=1033,  
    KColorStyleAstra=1034,  
    KColorStyleAfrica=1035,  
    KColorStyleAntarctica=1036,  
    KColorStyleTraffic=1037,  
    KColorStyleCustom=1100  
}
```

7.17. EGpsStatus enumeration

Possible values for eGpsStat in [TCurrentPositionInfo](#)

```
public enum EGpsStatus {  
    KGpsStateGeneralError = -1, // error == position invalid  
    KGpsStateGeneral = 0        // position valid (documentation)  
    KGpsStateReceiving = 1      // position valid  
}
```

7.18. EPoiTypeCode enumeration

```
public enum EPoiTypeCode {
    KPoiTypeGovernmentOffice = 0 ,
    KPoiTypeOpenParking = 2 ,
    KPoiTypePetrolStation = 4 ,
    KPoiTypeRestArea = 6 ,
    KPoiTypeCarDealer = 8 ,
    KPoiTypeCasino = 9 ,
    KPoiTypeChurch = 10 ,
    KPoiTypeCinema = 11 ,
    KPoiTypeCityCenter = 12 ,
    KPoiTypeCompany = 13 ,
    KPoiTypeConcertHall = 14 ,
    KPoiTypeCourthouse = 15 ,
    KPoiTypeCulturalCenter = 16 ,
    KPoiTypeExhibitionCenter = 17 ,
    KPoiTypeFerryTerminal = 18 ,
    KPoiTypeFrontierCrossing = 19 ,
    KPoiTypeGolfCourse = 20 ,
    KPoiTypeHospitalPolyclinic = 21 ,
    KPoiTypeHotelMotel = 22 ,
    KPoiTypeTouristAttraction = 23 ,
    KPoiTypeMountainPass = 24 ,
    KPoiTypeMuseum = 25 ,
    KPoiTypeOpera = 26 ,
    KPoiTypePlaceofWorship = 27 ,
    KPoiTypePostOffice = 28 ,
    KPoiTypeRentCarFacility = 29 ,
    KPoiTypeRentCarParking = 30 ,
    KPoiTypeRestaurant = 31 ,
    KPoiTypeShop = 32 ,
    KPoiTypeShoppingCenter = 33 ,
    KPoiTypeStadium = 34 ,
    KPoiTypeTheatre = 35 ,
    KPoiTypeTouristInformationOffice = 36 ,
    KPoiTypeZoo = 37 ,
    KPoiTypeSportsCentre = 38 ,
    KPoiTypePoliceStation = 39 ,
    KPoiTypeEmbassy = 40 ,
    KPoiTypeCollegeUniversity = 41 ,
    KPoiTypeCashDispenser = 42 ,
    KPoiTypeBeach = 43 ,
    KPoiTypeIceSkatingRing = 44 ,
    KPoiTypeTennisCourt = 45 ,
    KPoiTypeWaterSport = 46 ,
    KPoiTypeDoctor = 47 ,
    KPoiTypeDentist = 48 ,
    KPoiTypeVeterinarian = 49 ,
    KPoiTypeNightlife = 50 ,
    KPoiTypeAmusementPark = 51 ,
    KPoiTypeLibrary = 52 ,
    KPoiTypeCarRepairFacility = 53 ,
    KPoiTypePharmacy = 54 ,
    KPoiTypeScenicPanoramicView = 55 ,
    KPoiTypeSwimmingPool = 56 ,
    KPoiTypeWinery = 57 ,
    KPoiTypeCampingGround = 58 ,
    KPoiTypeParkRecreationArea = 59 ,
    KPoiTypeConventionCentre = 60 ,
    KPoiTypeLeisureCentre = 61 ,
    KPoiTypeYachtBasin = 62
};
```

```
    KPoiTypeMountainPeak = 1 ,
    KPoiTypeParkingGarage = 3 ,
    KPoiTypeRailwayStation = 5 ,
    KPoiTypeAirport = 7 ,
```

7.19. TInternalVersion structure

This structure is use by the [GetInternalVersions](#) function.

```
public struct TInternalVersion {  
    public int nMajor;           //major version of helper Dll  
    public int nMinor;          //minor version of helper Dll  
    public int nRelease;        //release of helper Dll  
    public string strBrand;      //brand of helper Dll  
    public System.Version vPPPTTCFVersion; //version of PPTTCF.dll (can be null)  
}
```

8. Error codes

The following error codes are currently defined in TomTom API:

8.1. *ESDKErrors enumeration*

```
public enum ESDKErrors {  
    KErrSDKNone= 0,      //returned if the command was processed successfully  
    KErrSDKNoSession= -1, //no session was created  
    KErrSDKNoResponse= -2, //no response from the server to the client  
    KErrSDKNoMethod= -3,  //command not supported by TomTomNavigationServer  
    KErrSDKBadParams= -4, //number of params for the command does not match  
    KErrSDKMethodFailed= -5, //internal error during processing of the command  
    KErrSDKEngineBusy= -6, //server busy, command was ignored  
    KErrSDKNoMapOpen= -7, //no map was currently open (could happen when the map  
                          //was not activated)  
    KErrSDKUnknownError= -100, //unspecified error  
}
```

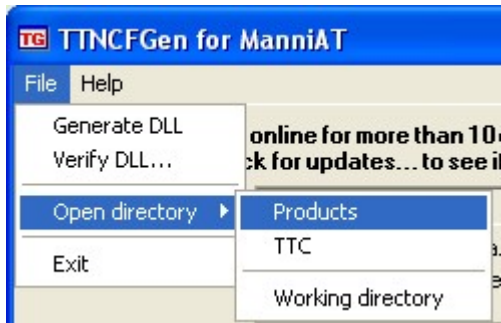
Note that these are the error codes returned from the navigation application as stored in TError. The API calls themselves return a subset of the above as a result:

- KErrSDKNone - if the response for a given request has been received;
- KErrSDKNoSession - if the client session could not be created;
- KErrSDKNoResponse - if the response was never received or did not have a valid format.

9. Graphical information about file location.

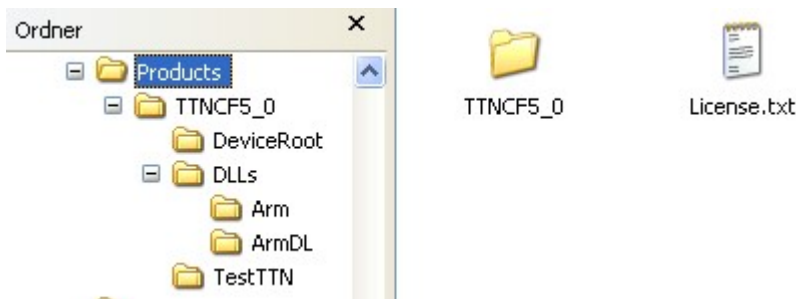
9.1. Access Products and Working Directory via TTNCGen

In the File Dialog of TTNCGen you will find an “Open Directory\Products” item.



This opens the products folder. Here you can also open the Working directory.

9.2. The Products Folder



9.2.1. Device Root

Files for the sample which must be copied to the Device Root.

9.2.2. DLLs

All DLLs which are not produced via TTNCGen are located in that folder.

But normally you will access those files from the Working directory, since updated version will be located there.

So this dir contains only the “installation time versions” of the DLLs.

9.3. The Working directory



This directory contains (next to other files) all you need to create your projects.

9.3.1. PPPTTCF.dll

The most current version of the .NET component.

9.3.2. ARM_.....

These directories contain the Helper DLLs you created via TTNCGen.

10. Version changes

10.1. Version 5.1

With 5.1 we introduced some new things which enhance the usability of TTNCF5.

10.1.1. PPPTTCF.dll name change

We change the name of PPPTTCF.dll to PPPTTCF~~xx~~.dll.

~~XX~~ reflects the version number. So for TTNCF 5.1 the file is name **PPPTTCF5_1.dll**

We did this to avoid version conflicts.

The namespace also changed from **PPPTTCF** to **PPPTTCF5_1**.

10.1.2. CheckCMDHandler

We implemented the ability to handle TTN commands from CAP files.

We decided not to use events for this. Instead we created a polling handler.

This is easier to handle because polling has two positive effects:

- 1.) You choose the frequency of checks
- 2.) The handler runs in the context of your form (no control invoke needed)

Take a closer look at this feature here: [CheckCMDHandler](#)

10.1.3. Changes in [SendNMEASentenceBuffered](#)

Since there are still timing problems with GPS handling we implemented a filter in this function. Before we thought TomTom would solve this problem soon – now they told us that this will be fixed in a later release.

So we decided to make the GPS handling usable by filtering only the needed sentences.

10.2. Version 5.2

With 5.2 we introduced some new things which enhance the usability of TTNCf5 and the whole functions of the new TomTom SDK. Release 5.2 (Q4 2005).

10.2.1. PPPTT~~x~~_xHelper.dll name change

We change the name of **PPPTT5Helper.dll** to **PPPTT~~x~~_xHelper.dll**.

~~x~~_x reflects the version number. So for TTNCf 5.2 the file is name **PPPTTCF5_2.dll**

We did this to avoid version conflicts.

10.2.2. Native .NET CF 2.0 support

We ship TTNCf~~x~~_x from this version on with support for .NET CF 1.0 and 2.0.

The 2.0 versions simply add the extension N2 to filename and namespace.

So PPPTTCF5_2 is the 1.1 version and PPPTTCF5_2N2 the one for .NET CF 2.0.

10.2.2.1. Enhanced TestTTN for .NET 2.0

Our test application TestTTN ships now in two versions.

The TestTTN2 is written as .NET CF 2.0 application.

It uses enhancements of the new CF as for example automatic layouts on change of the screen orientation, Colored TabPages, generic collections....

10.2.3. New internal functions

[StartTTN](#)

[TerminateWrapper](#)

10.2.4. New SDK functions

We implemented the functions of the new Q4 2005 TomTom SDK release.

[NavigateToPostalCode](#)

[SetRouteSummaryVisible](#)

[ClearRoute](#)

[ShowPostalCodeOnMap](#)

[GeocodeByPostalCode](#)

[EnableRawGPSOutput](#)

[DisableRawGPSOutput](#)

10.2.5. Support for NMEA Interface

To enable a tight integration of your NMEA parser (handler) we provide an interface for the most common GPS handling.

[IGPSInput](#)

With TestTTN2 we provide a “reduced version” of our NMEA helper to demonstrate the use of this interface.

10.2.6. New Events

[OnGPSMessage](#)

[OnTTNRestarted](#)

10.2.7. The KeepAlive System (general)

Since TomTom removed the “Autostart” from their SDK 5.0 we implemented a system which ensures that a termination of the Navigator does not block TTNCf calls.

Most blocking calls are included in this system. You can identify them by the KEEPALIVE marker next to the version in this documentation.

10.2.7.1. Version (Example)

TTN: 2.0

TTNCf: 1.0

KEEPALIVE

Some functions are not secured.

The system works by intention only for synchronous functions!

Asynchronous calls can block in the “xxxxxRetrieveAsync”.

10.2.8. The KeepAlive System modes

The KeepAlive System knows 3 modes (see [KAMode](#)).

10.2.8.1. KeepLiving,

In this mode the system check periodically if the Navigator is running.
If not – it tries to restart it.

10.2.8.2. WatchFunctions

In this mode a user can terminate the Navigator. When you call a checked function this will block your software. The Keep Alive System checks periodically if you got stuck in such a call. If so it tries to restart the Navigator.

10.2.8.3. Suspended

This is the default – the system is not working.

10.2.9. KeppAlive System Checker

Even with the suspended mode you can preceed checked functions with a call to `IsApplicationRunning` followed by a `StartTTN` if it does not run.

To enable this set [KACheckCallsForTTNRunning](#) to true.

If the starting of the Navigator fails – your application gets blocked!!

This checker can also be used in combination with other KAModes.

10.2.10. KeppAlive related properties

The KeepAlive System uses a nubmer of public variables to contorl its behaviour.

10.2.10.1. [KAMode](#)

Set the mode of the system.

10.2.10.2. [KAWatchIntervalSec](#)

This property sets the interval in sevonds for checking the Navigator.

10.2.10.3. [KACurrentForm](#)

If the Navigator needs to be restarted your application will lose the focus. When this property is set the focus is set to this form.

You can also (for a more specific handling) bind the event [OnTTNRestarted](#).

10.2.10.4. [KACheckCallsForTTNRunning](#)

This property allows you to preceed checked calls with a check if the Navigator is running.

10.2.11. KeppAlive capabilities

The system is designed as a “silent helper” – so if it can’t work (can’t start the navigator) your calls will still block. This is done by intention because a functional system will start the Navigator and if this is not possible you have serious problems.

10.2.12. [TerminateWrapper](#) **IMPORTANT**

This function must be called after [KAMode](#) is once set to a mode different than suspended. Setting the mode to other values starts a thread. When you switch back the thread is not terminated – it only stops doing his work.

So at programm termination you will have a running thread.

[TerminateWrapper](#) ends this thread (and does other needfull thins) so it is ensured that your application is clean terminated.

10.2.13. KeepAlive is not needed?

At the moment we work with a beta of the TomTom Navigator SDK which starts the navigator on the most functions if it is not running.

But this was also like now in prevoious betas – and the it was removed at release time!

There are two reasons why we implemented it althoug it looks obsolete.

First is the danger that TomTom removes the feature again.

Second is that KeepLiving ensures that TTN is always running as long as your app runs without a need to check it manually or call a function from the SDK.