

REV 1.4

## Application notes

# Implementing AGPS for iSuite™ 3 Based GPS Receivers

This document briefly describes the different types of data that are needed for acquiring GPS signals and calculating receiver position and time. The focus is on such kind of information that can be provided from external sources in order to speed up TTFF and improve the weak signal capability of the iTrax receiver. This document also describes, in detail, how to provide this information to the iTrax and what effect the accuracy of the aiding information is on the performance of the iTrax.

January 17, 2006

Fastrax Ltd



## TRADEMARKS

iTrax™, iSuite™, iCore™ and iTalk™ are trademarks of Fastrax Ltd.

VS\_DSP™ is a trademark of VLSI Solution Oy.

Microsoft Windows NT®, Microsoft Windows 2000®, Microsoft Windows XP®, and Microsoft Visual C++® are registered trademarks of Microsoft® Corporation.

MATLAB® is a registered trademark of MathWorks, Inc.

All other trademarks are trademarks or registered trademarks of their respective holders.

## Change log

<b>Rev.</b>	<b>Notes</b>	<b>Date</b>
1.0	Initial revision	20-04-2005
1.1	Added iSuite SDK sample	12-12-2005
1.2	Corrected some typing	16-01-2006
1.3	Fixed serial setup for AGPS demo	25-10-2006
1.4	Fixed IF frequence	16-01-2007

## CONTENTS

REV 1.4 .....	1
1. COMPLEMENTARY READING .....	6
PRO_ITALK.HTML.....	6
ITALK PROTOCOL SPECIFICATION(*) .....	6
PRO_NMEA.HTML.....	6
NMEA PROTOCOL SPECIFICATIONS(*) .....	6
2. VOCABULARY .....	7
3. INTRODUCTION.....	8
4. PROVIDING AIDING INFORMATION .....	9
4.1 GPS time .....	9
4.1.1 Providing GPS time estimate to the iTrax .....	9
4.2 Receiver position estimate .....	10
4.2.1 Providing GPS position estimate to iTrax receiver .....	11
4.3 Ephemeris and almanac data .....	12
4.4 Other aiding information.....	14
4.4.1 IF frequency .....	14
4.4.2 UTC_IONO .....	14
4.5 Recommended procedure.....	15
4.5.1 iTalk .....	15
4.6 NMEA.....	15
5. EXAMPLE: USING ISUITE™ SDK TO IMPLEMENT A SIMPLE AGPS SERVER	16
5.1 Simple AGPS server.....	16
5.2 Modifying SYSTEM library .....	17
5.3 Modifying your default user-project .....	20
5.4 Creating server project using iSuite Builder.....	21
5.5 Implementing AGPS service .....	23
5.6 Simple ephemeris service.....	24
5.7 Providing time and position assistance .....	26
5.8 Source code listing of Simple server user_task.c .....	29
6. Q&A ABOUT IMPLEMENTING AGPS.....	35

## 1. COMPLEMENTARY READING

The following reference documents are complementary reading for this document:

Ref . #	File name	Document name
1	PRO_italk.html	ITalk protocol specification(*)
2	PRO_nmea.html	NMEA protocol specifications(*)
	(*) Documents available on <a href="http://suite.fastrax.fi">http://suite.fastrax.fi</a>	

## 2. VOCABULARY

Term	Explanation
RTC	Real-Time Clock, a clock that maintains the current time.
Ephemeris	Set of satellite orbit parameters that are used to calculate precise satellite position at any given time instant. Ephemeris are downloaded from GPS satellite navigation data and remain valid for about 4 hours at a time.
Almanac	Stripped-down version of satellite orbit parameters, not precise enough for navigation but useful for predicting approximate satellite locations to speed up satellite search.
TTF	Time To First Fix, how much time does it take to acquire a navigation fix since the navigation start command.
Acquisition	Process of searching GPS satellite signals and locking on them.
Doppler Frequency	Shifting of the satellite radio signal frequency due to satellite movement in relation with receiver.
Pseudorange	Distance measurement between GPS satellite and receiver.
Ionospheric delay	Error caused by earth's ionosphere in pseudoranges. Can be eliminated using a mathematical model if the model parameters are available.

### 3. INTRODUCTION

The GPS receiver uses such data as satellite ephemeris and almanacs, GPS time and estimated receiver position for acquiring GPS signals and updating receiver position. Satellite information is normally extracted from the GPS signal itself, but for weak signals this may not be possible.

GPS time can be provided from the external RTC on the iTrax module and the last known good position can be used as a position estimate.

However, there are situations when the available information from these sources are missing or invalid. Satellite ephemeris data may have expired, the RTC might not be available, we do not have any last known good position or we have moved the receiver far away from our last position etc. Downloading satellite ephemeris from the GPS signals also take some time and requires good quality GPS signals. This means that in order to calculate navigation fix using a very weak signal, satellite ephemeris data has to be provided to the receiver from external source such as a cellular network.

By providing assistance information to the iTrax including satellite ephemeris, time and position estimate means that time to first fix, TTFF, can be shortened, the position accuracy and integrity can be improved and weak signal acquisition and navigation can be enabled.

**NOTE:** This document is valid for iTrax receivers that use **iSuite 3 firmware**, version 3.11 or newer.

## 4. PROVIDING AIDING INFORMATION

Aiding information can be provided through both serial protocols supported by the iTrax, the proprietary iTalk binary protocol [1] and proprietary NMEA commands [2]. All aiding information can be sent with the iTalk protocol, whereas only a limited subset can be sent over NMEA. If the iTalk protocol is used the assistance data should be sent to SYSTEM\_TASK on iTrax (see attached SDK example). To achieve the best AGPS performance, use of iTalk protocol is highly recommended.

### 4.1 GPS time

Exact receiver GPS time is obtained when the GPS receiver calculates a position fix. The Real Time Clock, RTC, is synchronized to this time frame and if the receiver is stopped, put to sleep, or not able to produce a fix, the RTC will uphold the time estimate for future use. If the power is switched off the RTC will be reset and GPS time will be lost. The RTC time will also diverge from the real GPS time if for some reason no position fix is available for an extended period of time.

When restarting the receiver or reacquiring a position fix the time estimate (from RTC or other source) is mainly used for three purposes. Estimating satellite Doppler frequencies for signal acquisition, estimating bit boundaries from the 50 baud navigation signal for coherent signal integration, and estimating satellite positions for calculating pseudo ranges.

In order to acquire the GPS signals as quickly as possible we need a good estimate of the satellite doppler frequencies. They change very slowly, about 30 Hz per minute, and therefore the time estimate does not need to be very accurate for this purpose. Within 10-15 minutes of true time is usually sufficient although a better time estimate is likely to produce faster signal acquisition.

For navigational purposes timing errors of less than 100 ms are acceptable although the position accuracy might suffer already at much smaller errors. Sending time estimates with this kind of accuracy requires good knowledge of time delays in the system.

#### 4.1.1 Providing GPS time estimate to the iTrax

The GPS time estimate can be provided to the iTrax utilizing the iTalk ASSISTANCE message [1] and the substructure INT\_GPS\_TIME with the fields wWeek - GPS week number, and dwTowMs - GPS time of week in ms. Other time related assistance is currently not supported.

The recommended way to give time estimate to the iTrax is:

- Set GpsTow->wWeek to GPS week (including first 1023 GPS weeks)
- Set GpsTow->dwTowMs to estimated GPS tow.
- Set the flag: ASSISTANCE\_TIME\_VALID (third bit) in the swFlags field.
- Send the message to iTrax.

Time estimate can also be provided with the proprietary NMEA configuration command[2]

`$PFST,INITAID,<time>,<date>,<lat>,<N/S>,<long>,<E/W>,<altitude>`

where the time and date are in UTC format. Notice that the position information can be omitted from the INITAID command if it's desirable to give only the time aiding; in such case the last known good position will be used as aiding position.

GPS time aid can be provided before or after the iTrax has been started. When iTrax receives the time aiding command, iTrax sets its internal RTC to the given time so that the time will be kept up-to-date even if the navigation START command doesn't follow immediately. However, if iTrax is navigating the RTC will be set to the correct time at the next valid position fix and therefore the time aid is irrelevant under such conditions.

The week number in the GPS system is the number of weeks starting from January 6, 1980. Since the GPS system only uses 10 bits for identifying the week number the maximum number is 1023. The last week roll over occurred on August 22 1999. The iSuite 3, however, includes the first 1023 weeks in it's week number. When providing week estimate you should therefore use numbers always > 1023.

The GPS week starts at midnight on the Saturday/Sunday night (0 hours, 0 minutes and 0 milliseconds) and the GPS tow is the time, in msec, from that moment.

## 4.2 Receiver position estimate

Receiver position is calculated whenever there are enough useful satellites visible to the iTrax and this position is stored for later use. The last known good position will not be erased even if the receiver is completely powered down.

If the receiver is turned off or unable to produce position fixes there are, unlike with time and RTC, no means of upholding a position estimate. When the receiver is reinitialized or tries to reacquire a position fix the last known good fix is used if available.

An approximate user position estimate is needed for example to estimate satellite Doppler frequencies for GPS signal acquisition.

If the receiver has been moved to a completely different location or if the receiver is started for the first time the position estimate may not exist or is wrong. This can affect the TTFF and receiver integrity significantly.

#### 4.2.1 Providing GPS position estimate to iTrax receiver

The GPS position estimate can be provided to the iTrax utilizing the iTalk ASSISTANCE message[1] and the substructure INT\_LLA\_POS and the fields ILat – WGS84 latitude in units of 0.0000001 degrees, ILon – WGS84 longitude in units of 0.0000001 degrees, IAlt – WGS84 altitude in units of 0.001 meters. When providing position information the two flags: ASSISTANCE\_ALT\_VALID and ASSISTANCE\_LATLONG\_VALID should be set.

The recommended way to give position estimate to the iTrax is:

- Set the GPS time as described in section 4.1.1, and the clock drift as described in section **Error! Reference source not found.**
- Set Lla->ILat to estimated receiver latitude.
- Set Lla->ILon to estimated receiver longitude..
- Set Lla->IAlt to estimated receiver altitude.
- Set the flags: ASSISTANCE\_ALT\_VALID (first bit) and ASSISTANCE\_LATLONG\_VALID (second bit) in the swFlags field.
- Send the message to iTrax.

Position estimate can also be provided with the proprietary NMEA configuration command[2]

`$PFST,INITAID,<time>,<date>,<lat>,<N/S>,<long>,<E/W>,<altitude>`

A good rule of thumb is that an average error of 4 Hz in satellite doppler frequency is introduced for every 10 km offset in position. It is recommended

that the position estimate should be within 1000 km from the correct receiver position. If this is not the case the signal acquisition can actually perform worse than without an estimate.

The position aid has to be provided before the iTrax is started. It will have no effect after this.

### 4.3 Ephemeris and almanac data

Ephemeris and almanac data are normally extracted from the GPS signals. Ephemeris data are very accurate satellite orbit parameters that are used for calculating pseudo ranges for navigation. Almanacs are like stripped-down ephemeris information, not precise enough for navigation but useful for speeding up the satellite search at navigation start. The validity period of ephemeris and almanac data is twofold.

1. Ephemeris are valid for navigation only for a limited time and almanacs are not recommended to be used for navigation at all. A good rule of thumb is that one downloaded ephemeris from a satellite rising above the horizon will be valid for the visibility period of that satellite when the receiver is static (approximately 4 hours). If we have ephemeris from the previous day for example, they can not be used for navigation and a new ephemeris has to be downloaded for that satellite.
2. Almanacs and old ephemeris are valid for satellite Doppler predictions for a much longer time, usually weeks. An advantage is that the almanacs for all satellites are broadcasted from each and every satellite (they are almost identical in each satellite signal). The disadvantage is that it takes a long time to download them.

Ephemeris and almanacs are stored to flash memory on the iTrax and will therefore be available even after a power down. Almanacs are also stored into memory during the production process and are therefore available for the first “out-of-the-box” start. Notice that ephemeris and/or almanacs information are discarded when starting navigation in the “cold start” mode.

Ephemeris and almanac data can be provided to the iTrax utilizing the iTalk RAW\_EPHEMERIS and RAW\_ALMANAC messages.

Ephemeris and almanac data can be provided to the iTrax any time. However, the data downloaded from GPS signals will override this aiding data.

When sending ephemeris data with the RAW\_EPHEMERIS message, the complete set of data should be provided, with the same contents as the GPS subframes 1, 2 and 3. The data should be prepared as follows:

- Set the data contents. The RAW\_EPHEMERIS message contains the same data fields as the GPS subframes, but is conveniently aligned to full-word boundaries. If GPS subframes are available from another source, the data can be copied directly from the GPS subframes, except for those fields mentioned below.
- Set the clock reference time (TOC) to the ClockSub.Toc field. The week number should be the full GPS week (> 1024 in 2005), and the dwTowMs is in milliseconds.
- Set the the time of ephemeris to the OrbitExt.dwToe field. This is given in seconds.
- Set the EPH\_DATA\_VALID and EPH\_FULL\_DATA bits in the wFlags field to indicate that the whole data is valid, not just the almanac subset.
- Note: the other wFlags bits are reserved for future use and do not have to be set currently.
- Set the wFitPeriod to the number of hours that the ephemeris will remain valid. For ephemerides broadcast by the GPS satellites, 4 hours is the recommended value.

Almanac data can be sent with almost the same procedure:

- Set the data contents. These are a subset of the RAW\_EPHEMERIS contents.
- Set the almanac reference time in the ClockSub.Toc field. The week number must be set to the full GPS week (values > 1024 in 2005).
- Set the bits EPH\_DATA\_VALID and EPH\_WEEK\_VALID in the wFlags field. Do not set the EPH\_FULL\_DATA flag.

It should be noted, that if assistance ephemeris data is acquired from an other iTrax receiver using iTalk protocol, data conversions are not required and ephemeris data can be passed on directly to client GPS receiver in iTalk format (see attached iSuite 3 SDK example).

## 4.4 Other aiding information

There are some additional aiding information that also can be provided for iTrax receivers using iSuite 3 firmware to improve overall performance.

### 4.4.1 IF frequency

Each individual GPS receiver have a characteristic IF frequency. In the optimal case the IF frequency is equal to what we call the nominal IF frequency, but in most cases the true IF frequency will be offset from this by an unknown value. The frequency offset is needed by the iTrax to estimate satellite frequencies and making a so called aided signal search.

Normally, iTrax receiver stores the IF frequency offset to persistence storage with last know good fix. Since IF frequency changes very slowly for a receiver this value is usually sufficient.

The estimated IF frequency offset can also be given in the ASSISTANCE message [1]. It is given as an offset from the nominal IF by:

- Set IFreqOffset in ASSISTANCE message (units of mHz)
- Set the ASSISTANCE\_DRIFT\_VALID (bit 5 in swFlags).
- Send message to iTrax.

Giving a IF frequency with the ASSISTANCE message involves a challenge of knowing and monitoring all client receivers and their individual IF frequencies. If this facility cannot be implemented by the AGPS service, it is recommended that IF frequency is not given in the ASSISTANCE message.

### 4.4.2 UTC\_IONO

The corrections for the ionospheric delay are downloaded from the almanac subframes in the GPS signal. This requires strong enough signals for data downloading and it will take some time to actually download them. Until new corrections are available the iTrax is using corrections that might not be up to date. New corrections can also be provided after starting the iTrax by sending the iTalk UTC\_IONO\_MODEL message[1] to iTrax and they will then immediately be used to correct pseudo ranges.

## 4.5 Recommended procedure

Although some information, i.e. ephemeris and almanacs, can be provided at any time aiding information is typically provided as a part of the startup sequence of the iTrax. It is advantageous to send the least time critical information first.

### 4.5.1 iTalk

A typical startup sequence utilizing iTalk would then be:

- Power on with autostart parameter turned off
- Send ephemeris and/or almanac data using the RAW\_EPHEMERIS and RAW\_ALMANAC messages. If amount of aiding data to be transmitted is critical, it's sufficient to send ephemeris data for those satellites only that are currently visible for the iTrax receiver. Notice also that as almanacs are valid for predictions for several weeks, there's no use sending the almanacs every time if iTrax has the corresponding data in flash memory already.
- Send valid position and time estimate with the ASSISTANCE message
- Send the START message with the auto start mode selected.

The ionospheric corrections can be sent separately as described earlier if required.

## 4.6 NMEA

A typical startup sequence utilizing NMEA would then be:

- Power on with auto start turned off
- Send position and time estimates using \$PFST,INITAID command
- Start the iTrax with \$PFST,START,0 (AutoStart)

Note: Ephemeris and/or almanac information can't be sent to iTrax using NMEA protocol.

## 5. **EXAMPLE: USING iSuite™ SDK TO IMPLEMENT A SIMPLE AGPS SERVER**

Implementing AGPS service is usually a very demanding project and requires maximum control over GPS receiver features. For this reason use of iSuite 3 Software Development Kit is highly recommended in AGPS projects.

For details about iSuite 3 SDK, please refer to our iSuite website at <http://isuite.fastrax.fi>

This example implements a simple AGPS service using 2 iTrax Evaluation Kits and a serial line between the kits to provide the assistance data in iTalk format.

iTalk protocol can be implemented on any reasonable processor environment. Details on iTalk protocol can be found from <http://isuite.fastrax.fi>. Also, iTalk 3 reference implementation software (iTalkRIS) is available from Fastrax with full source code. iTalkRIS is free of charge for all iTrax receiver customers.

**To acquire free iTalkRIS license, please email to [isuite@fastrax.fi](mailto:isuite@fastrax.fi) and request for “iTalkRIS package”.**

### 5.1 Simple AGPS server

Implementing a simple AGPS server with iSuite 3 SDK is the easiest way to demonstrate AGPS capability of iTrax receivers. If we are using an other iTrax receivers to provide the assistance data, we do not need to deal with ephemeris data conversions but can sent the data to client receiver as such in plain iTalk format.

iSuite 3 compatible receivers are able to receive all assistance data described earlier in his document using iTalk protocol. If you are providing Talk data directly from iTalk compatible environment (iTrax receiver or iSuite host, i.e. a PC) sending aiding data is very easy: Just pass the data to client receivers SYSTEM tasks:

```
ITK_SendMsg(pMsg, TASK_SYSTEM|NODE_HOST);
```

NODE\_HOST tells iTalk to pass message out of the current node, otherwise the message would end up to current nodes (server itself) SYSTEM task and this is not our intention.

With iSuite 3 SDK this can be implemented in a matter of minutes just by writing couple of lines of server code

Note that iTalk host can also be implemented using iTalkRIS and do not require the use of iSuite 3 SDK. In this case a separate server PC is required to work as a server and acquiring assistance data from an other iTrax receiver.

## 5.2 Modifying SYSTEM library

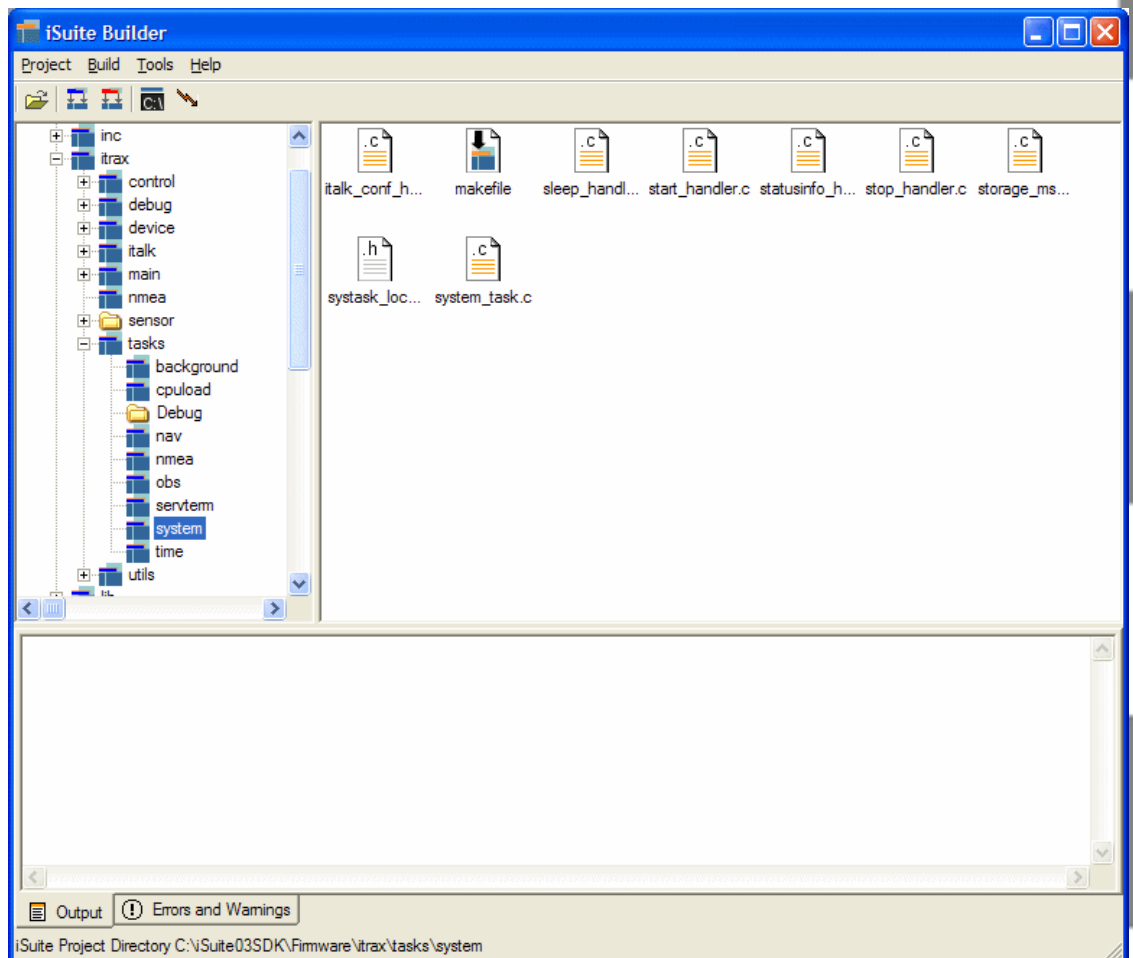
**ATTN:** This example involves changing slightly your iSuite libraries. This means that you need to select “Additional Source Code”- option when installing iSuite 3 SDK. Note that while version 3.11 requires you to do this, later versions may support this change without the need to modify your “system”-library.

We will implement the AGPS service in our “user\_task” of iTrax03 receiver using iSuite 3 SDK. This will require some special changes to our SYSTEM library.

Acquiring Ephemeris data for our “user\_task” requires us to pass the data from SYSTEM task to USER1 task. This is not normally done in iSuite 3 System library. Please note that this change becomes a PERMANENT FEATURE for all further created user tasks.

To make the required change to SYSTEM library and recompile it, do the following.

Select “iTrax->Tasks->System” from iSuite Builder directory view.



Open the "storage\_msghandlers.c" source code file and modify the function SYS\_RawEphMsgHandler as follows:

```
/// Message handler for RAW_EPHEMERIS message.
void FAR_CODE SYS_RawEphMsgHandler(RAW_EPHEMERIS_MSG *pMsg)
{
    WORD wTimeQuality;

    wTimeQuality = STORAGE_GetTimeAvailability(g_pStore);

    // This is the place where we get our valid week number.
    STORAGE_SetWeek(g_pStore, &pMsg->Payload.ClockSub.Toc);
    // Store the ephemeris
    STORAGE_StoreEphemeris(g_pStore, &pMsg->Payload);

    if ((wTimeQuality & TIME_FLAG_WEEK_VALID) == 0)
    {
        // If we didn't have time so far, signal the background task to update
        // all predictions.
        ISYS_Signal(TASK_BACKGROUND, SIG_BG_UPDATE_ALL_PREDS);
    }
}
```

```
//  
// Modified 2005-11-23/TY  
//  
// In order to keep consistent, default USER project has been also modified  
// to further route to MONITOR  
ITK_SendMsg(pMsg, _TASK_USER_1);  
}
```

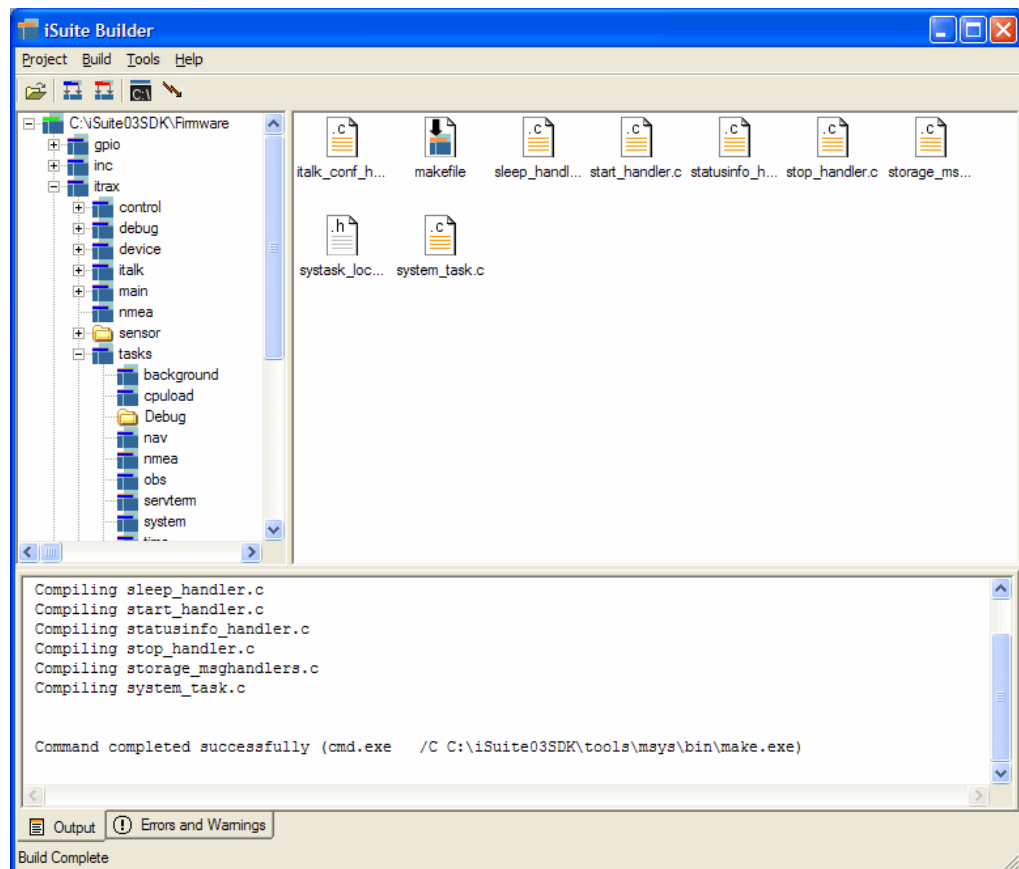
Modify the last line of the function to:

```
ITK_SendMsg(pMsg, _TASK_USER_1);
```

This will send the ephemeris data to our USER\_1 task instead of MONITOR which is the default.

As stated also in source code comment above, this causes a permanent change to our system library and we should also modify our default “user”-task template to reflect the change and handle the RAW\_EPHEMERIS\_MSG accordingly. iSuite Builder uses the default project in directory <ISUITE03\_ROOT>\firmware\user as a basis for new user projects, so modifying the default user project ensures that the changes in system library are propagated consistently to all new user projects and new user tasks remain compatible with changed SYSTEM task.

Last step is to rebuild your SYSTEM task to reflect the changes in user projects build. To rebuild, select “iTrax->Tasks->System” from iSuite Builder directory view and then select “Rebuild” from “Build”-menu:



This will rebuild **libtasks.a** library in **libvdsdp-** directory and change the default behavior of you iSuite 3 libraries. You need to modify the default user tasks to ensure system integrity as described in the next chapter.

### 5.3 Modifying your default user-project

This step is required to ensure consistency in all future projects. If this is not done, you default user project is not able to handle the ephemeris data sent by SYSTEM task. To modify the default user project, open the source code file:

<ISUITE03\_ROOT>\firmware\user\user\_task\user\_task.c and do the following modifications:

Add a handler function:

```
void FAR_CODE USER_RawEphMsgHandler(RAW_EPHEMERIS_MSG* pMsg)
{
    // Send to monitor
    ITK_SendMsg(pMsg, TASK_MONITOR);
}
```

This can be added to the beginning of the file as a first function. Also, add entry to iTalk handler table in user\_task.c:

```
/// iTalk message handler table for the user task. This table defines handler
/// functions for
/// each iTalk message we wish to process in the user task.
STATIC const __far ITALK_MSG_HANDLER _UserMsgHandlerArray[] =
{
    { NAV_FIX_MSG_ID,          (ITALK_MSG_FUNC)USER_NavFixMsgHandler },
    { PSEUDO_MSG_ID,         (ITALK_MSG_FUNC)USER_PseudoMsgHandler },
    { BIT_STREAM_MSG_ID,     (ITALK_MSG_FUNC)USER_BitStreamMsgHandler},
    { START_MSG_ID,         (ITALK_MSG_FUNC)USER_StartMsgHandler },
    { STOP_MSG_ID,          (ITALK_MSG_FUNC)USER_StopMsgHandler },
    { START_COMPLETED_MSG_ID, (ITALK_MSG_FUNC)USER_StartCompletedMsgHandler },
    { STOP_COMPLETED_MSG_ID, (ITALK_MSG_FUNC)USER_StopCompletedMsgHandler },
    { RAW_EPHEMERIS_MSG_ID,  (ITALK_MSG_FUNC)USER_RawEphMsgHandler },
    {0, NULL}
};
```

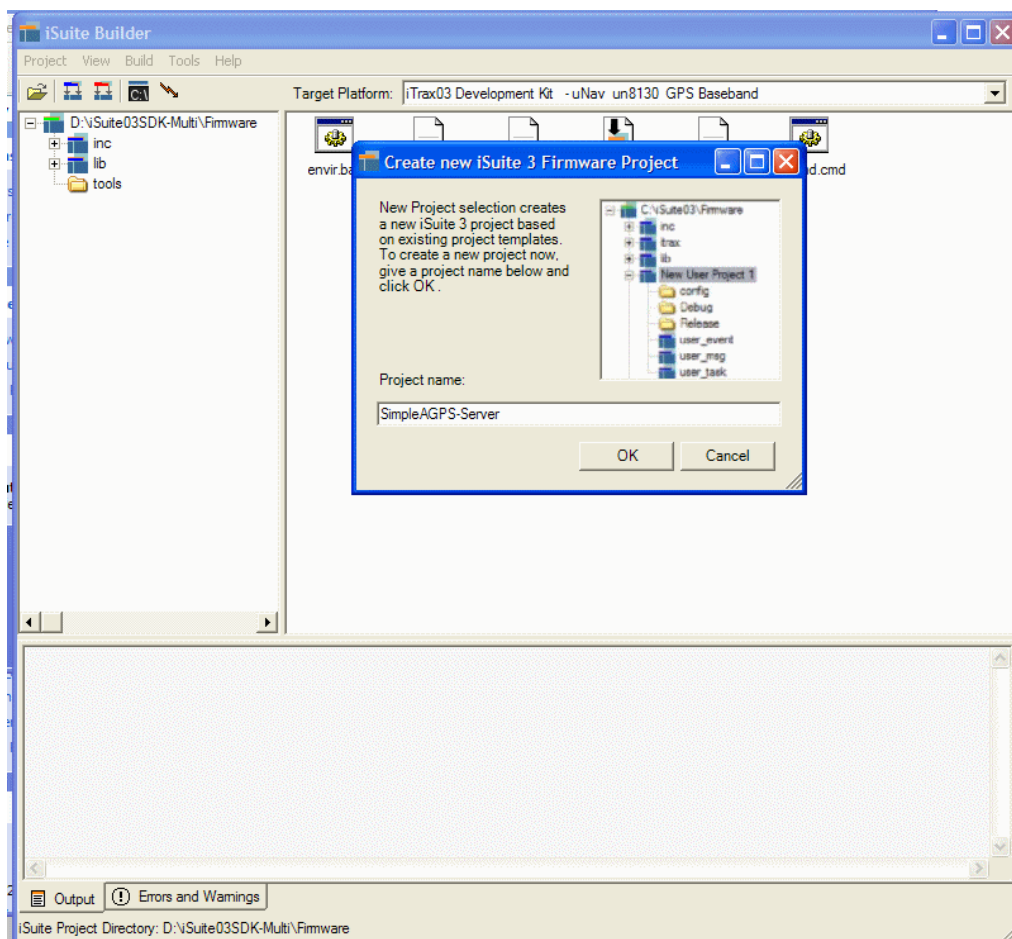
Where the last entry in the table is our new handler function. This change to default user task restores the default behavior of ephemeris messages passing on to MONITOR task (which is usually GPS Workbench) while we still can intercept the message in user\_task.c to implement our AGPS service.

#### 5.4 Creating server project using iSuite Builder

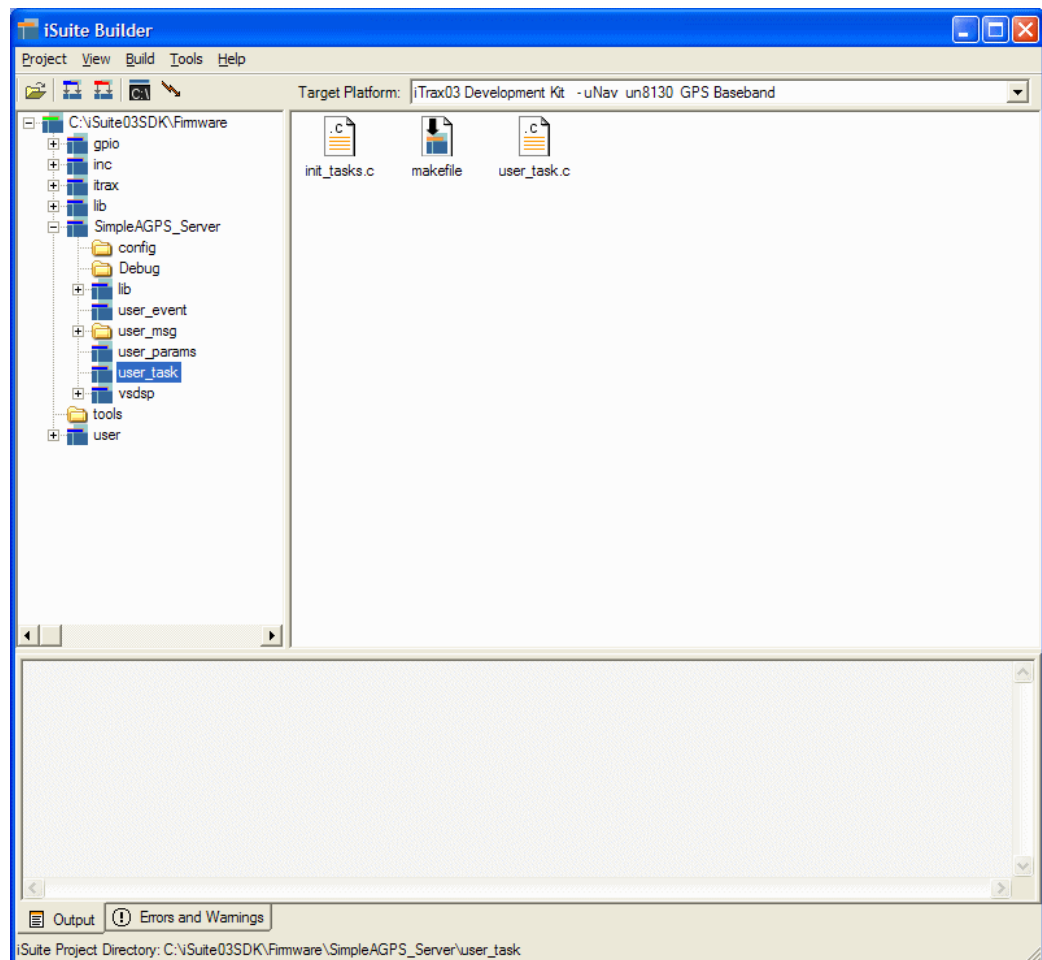
In order to implement simple AGPS demo, we need 2 iTrax evaluation kits and iSuite 3 SDK. In this case we assume the receivers used will be iTrax03 receivers.

We will create a simple “push” type AGPS server firmware for one of the iTrax03 receivers (Server). Client receiver firmware does not have to be modified in this demo. Note that this is not likely a true world implementation since the client does not participate in controlling the process in any way. Normally client side would control calling the service network and request the assistance from the server while we only simply “push” the assistance data to client receiver when it is available.

To create a new server project start iSuite Builder and select “**New user project**” from “**Project**”- menu.



Give the name of your project, in this case “SimpleAGPS-Server” and click OK.

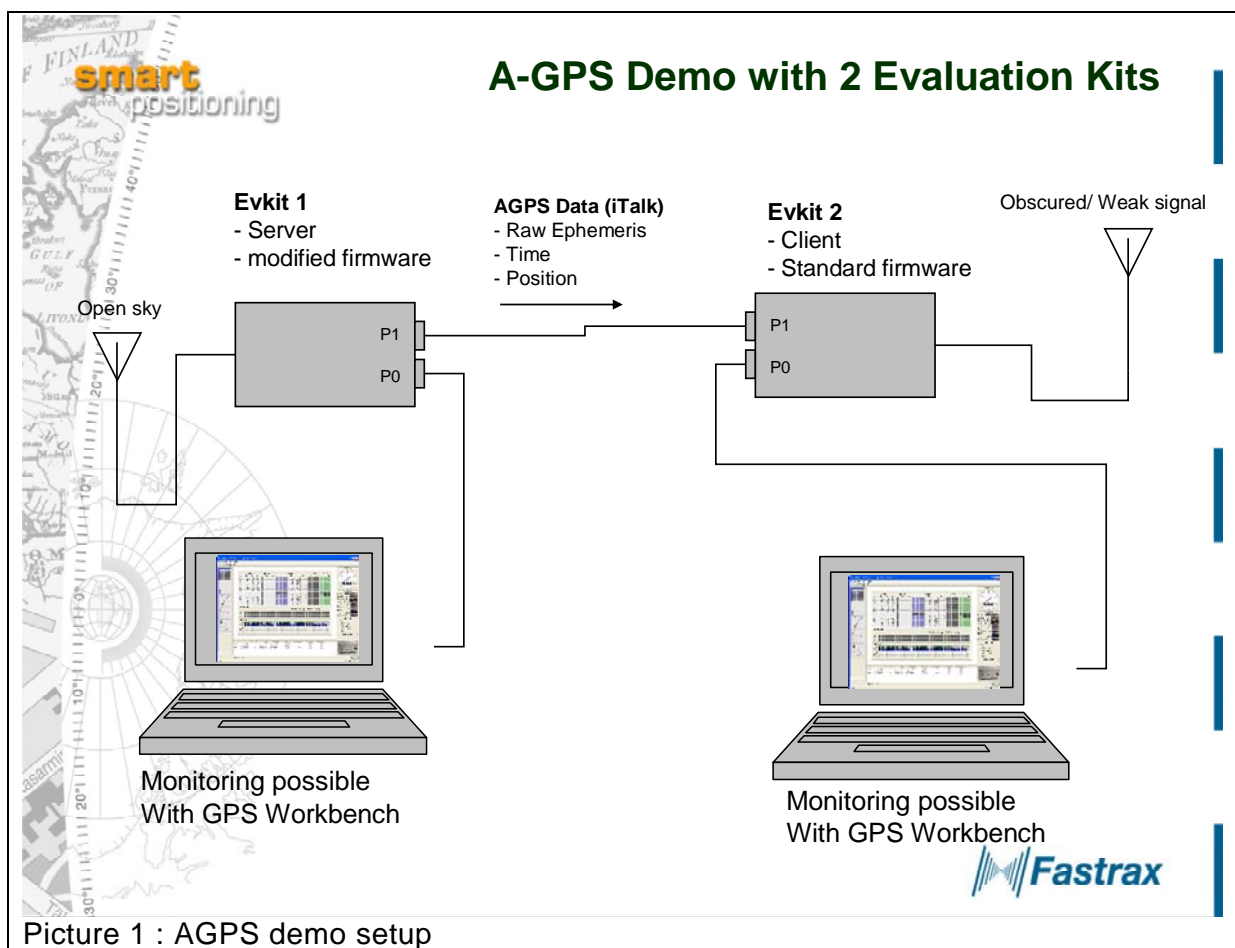


## 5.5 Implementing AGPS service

Following steps assume that you have completed the following as described earlier:

1. Implemented ephemeris data re-routing to USER\_1 task from SYSTEM task
2. Recompiled the SYSTEM library.
3. Modified your default user-task as described
4. Created a new user project called SimpleAGPS-Server

Next step is to implement our AGPS service. We start by setting up our two evaluation kit system as shown in the picture 1.



Picture 1 : AGPS demo setup

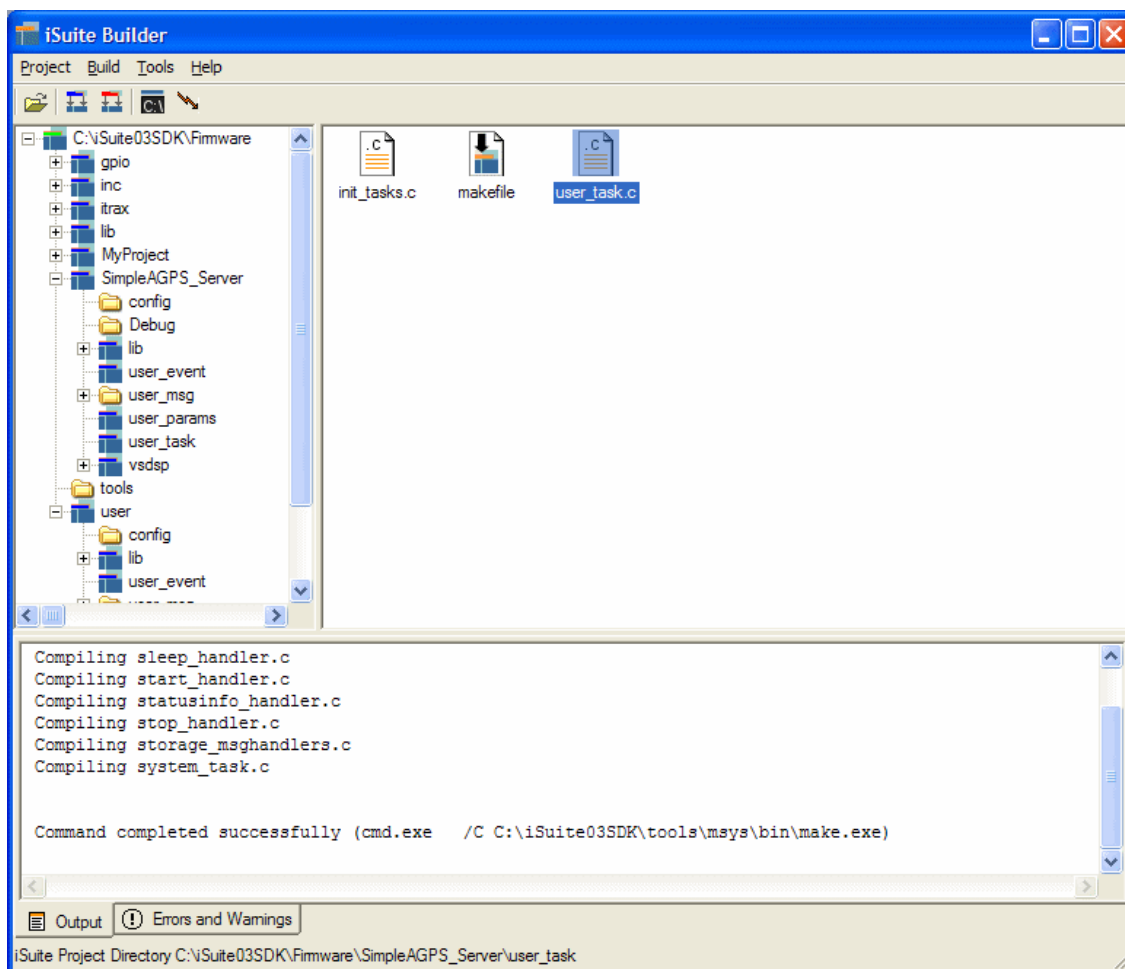
It is possible to connect also GPS Workbench to both evaluation kits for monitoring. Note that if you have 2 serial ports on 1 PC, you can run 2 instances of GPS Workbench on a single PC.

**NOTE:** Evaluation kit 1 antenna should be placed under open sky view, while evaluation kit 2 antenna should be placed in obscured/weak signal conditions to demonstrate AGPS capability.

## 5.6 Simple ephemeris service

Since both our server and client system are complete iTalk systems and connected with serial cable, implementing ephemeris service is extremely easy.

To achieve this, open `user_task.c` under your `SimpleAGPS_Server` project:



Modify function `USER_RawEphMsgHandler`. Note that this will only exist if you have completed all the steps above and modified your default user task.

```
// Added 2005-11-23/TY
void FAR_CODE USER_RawEphMsgHandler(RAW_EPHEMERIS_MSG* pMsg)
{
    DBG_TraceString("Ephe");
    // Send to system task
    ITK_SendMsg(pMsg, TASK_SYSTEM|NODE_HOST);
}
```

The modification sends the ephemeris to `TASK_SYSTEM` instead of `TASK_MONITOR` on the node `NODE_HOST`. `NODE_HOST` tells that the message will be sent through serial line away from our current node. Without this the message would end up to our current (server) node `TASK_SYSTEM` and an infinite loop would be created.

We have also added a simple debug message that would be visible on GPS Workbench trace-view.

You can now rebuild you simple server project and flash the modified firmware to your evaluation kit 1 (server).

To monitor that ephemeris will be sent through, you may want to stop and start the client receiver using GPS Workbench while selecting the “Clear Ephemeris” flag above the “Stop/Start button”.

Now start the client and server evaluation kits (with client antenna in weak signal conditions). When server has downloaded the ephemeris data, it should immediately appear on client side GPS workbench indicating that data have been received by the client receiver.

This only implements the ephemeris service, but we would like to give time and position assistance as well.

## 5.7 Providing time and position assistance

To add time and position aiding, we need to add a code like this:

```
// Added 2005-11-23/TY
void _SendAiding()
{
    STOP_MSG *pStopMsg;

    pStopMsg = ITK_AllocMsgType(STOP);
    if (pStopMsg)
    {
        ITK_SendMsg(pStopMsg, TASK_SYSTEM|NODE_HOST);
    }

    {
        ASSISTANCE_MSG* pMsg = ITK_AllocMsgType(ASSISTANCE);

        if (pMsg)
        {
            GPS_TIME Time;
            double dDum;
            STORAGE_GetCurrentTime(g_pStore, &Time, &dDum);

            pMsg->Payload.GpsTow.t.wWeek = Time.wWeek;
            pMsg->Payload.GpsTow.t.dwTowMs = Time.dwTowMs;
            pMsg->Payload.swFlags = 0;
            pMsg->Payload.swFlags |= ASSISTANCE_TIME_VALID;

            pMsg->Payload.Lla.lLat = _fixLast.Pos.Lla.lLat;
            pMsg->Payload.Lla.lLon = _fixLast.Pos.Lla.lLon;
            pMsg->Payload.swFlags |= ASSISTANCE_LATLONG_VALID;
            ITK_SendMsg(pMsg, TASK_SYSTEM|NODE_HOST);

            DBG_TraceString("ASSISTANCE_MSG");
        }
    }

    {
        START_MSG *pStartMsg = ITK_AllocMsgType(START);
        pStartMsg->Payload.wStartMode = 0xFFFF;
        pStartMsg->Payload.wFlags = 0xFFFF;
        ITK_SendMsg(pStartMsg, TASK_SYSTEM|NODE_HOST);
    }
}
```

```
}
```

This code should be trickered in an appropriate place where we want to send ASSISTANCE data.

The code does the following:

- 1) Send stop message to stop acquisition on client receiver
- 2) Fill and send ASSISTANCE message. Note that pMsg->Payload.swFlags should be used consistently with available data.
- 3) Send START message to start assisted search on client receiver

For easiness, we add call to this function when first fix is received by the server:

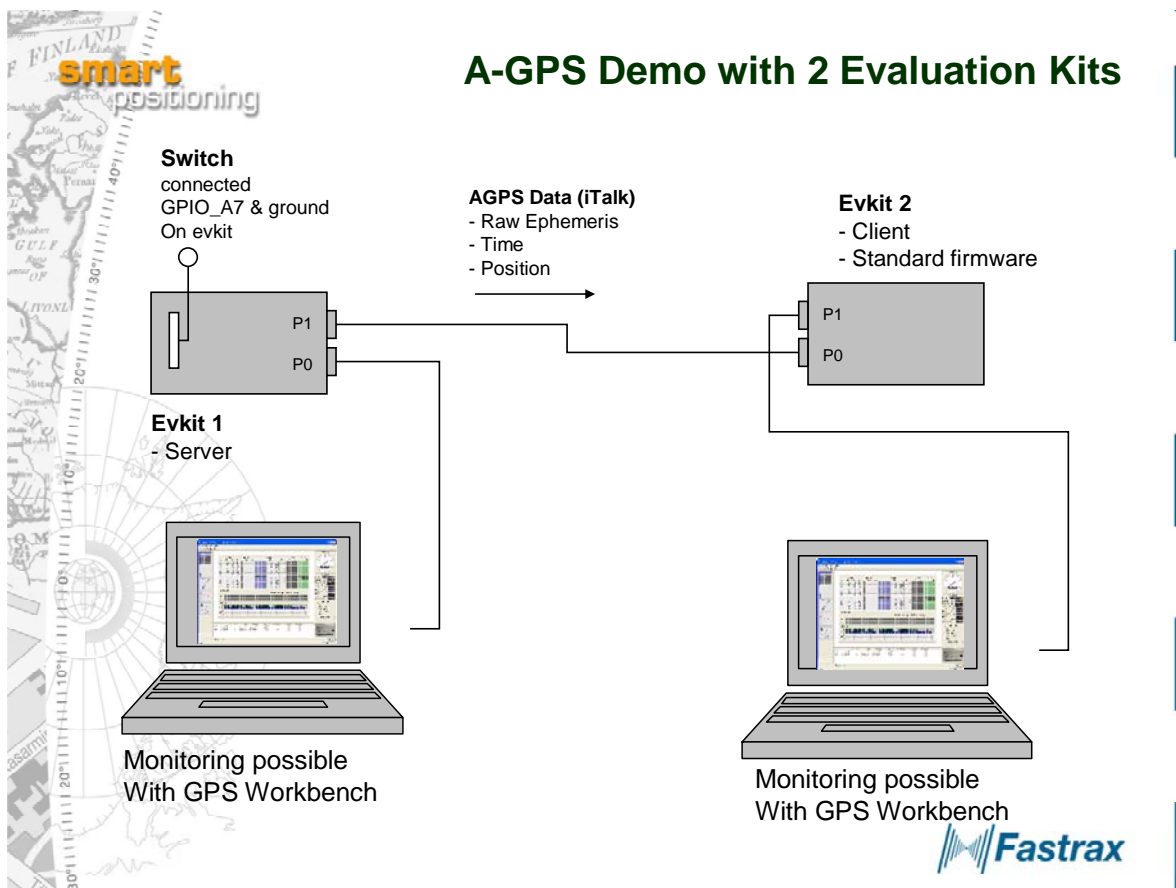
```
/// NAV_FIX message handler function
void FAR_CODE USER_NavFixMsgHandler(NAV_FIX_MSG *pMsg)
{
    // Perform user navfix events here.

    if ((pMsg->Payload.swFlags & FIXINFO_FLAG_NEW_FIX) == FIXINFO_FLAG_NEW_FIX)
    {
        if (!_bAidingSent)
        {
            _fixLast = pMsg->Payload;
            _SendAiding();
            _bAidingSent = TRUE;
        }
    }

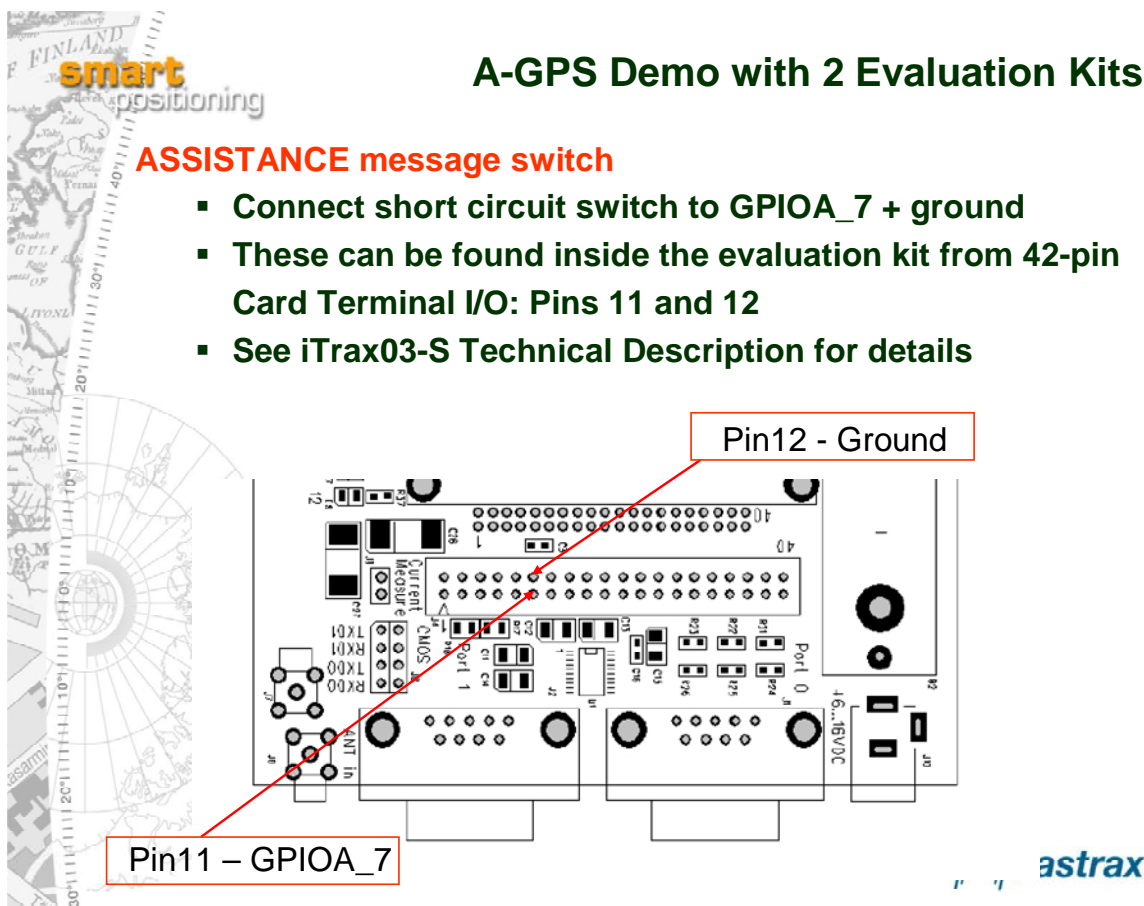
    // Forward the message to the monitor task.
    ITK_SendMsg(pMsg, TASK_MONITOR);
}
```

To the global level, we have also defined a variable `_fixLast` of type `NAV_FIX` to store last good fix and `bAidingSent` variable to flag that ASSISTANCE has been sent. Alternatively, you can use STORAGE API to retrieve last known position fix on server.

You would also like the ASSISTANCE message to be send on demand, so we could add for example GPIO control to do this. Attached source code listing uses GPIOA\_7 switch to trigger the sending of ASSISTANCE message.



Picture 2: Using GPIOA\_7 switch to trigger ASSISTANCE message



Picture 3: ASSISTANCE message switch setup on server evaluation kit

## 5.8 Source code listing of Simple server user\_task.c

This is a complete source code listing of user\_task.c for simple AGPS server. If you do not have iSuite 3 SDK and want to try the demo, the firmware is available from Fastrax. Please email to [isuite@fastrax.fi](mailto:isuite@fastrax.fi) and request for "Simple AGPS demo firmware".

Similar functionality can also be implemented using free iTalkRIS package, 2 evaluation kits and a PC to work as a server.

```
//-----  
//  
// Copyright (C) Fastrax Oy  
//  
// This source file is property of Fastrax and may not be  
// copied, modified or distributed without a written permission  
// from the copyright holder.  
//  
// Contact information:  
//-----  
//
```

```
// email: support@fastrax.fi
//
//=====
/// \file
/// User task main routine
///
/// $Revision: 1.28 $
///
/// $Author: op $
///
/// $Logfile: W:/iSuite3/archives/firmware/user/user_task/user_task.c-arc $
///
/// $Date: Jan 28 2005 14:59:48 $
///
//=====

#include "isys.h"
#include "italk.h"
#include "param.h"
#include "user.h"
#include "trace.h"
#include "data_storage.h"
#include "gpio.h" // for the GPIO control functions

// iTalk message handler function prototypes
void FAR_CODE USER_BitStreamMsgHandler(BIT_STREAM_MSG *pMsg);
void FAR_CODE USER_StartMsgHandler(START_MSG *pMsg);
void FAR_CODE USER_StopMsgHandler(STOP_MSG *pMsg);
void FAR_CODE USER_NavFixMsgHandler(NAV_FIX_MSG *pMsg);
void FAR_CODE USER_PseudoMsgHandler(PSEUDO_MSG *pMsg);

void FAR_CODE USER_RawEphMsgHandler(RAW_EPHEMERIS_MSG* pMsg);

#define SIGF_USER_GPIO (1 << 9)

/// User task timer.
STATIC ISYS_TIMER _UserTaskTimer;

BOOL _bAidingSent = FALSE;
NAV_FIX _fixLast;

// ADded 2005-11-23/TY
void _SendAiding()
{
    STOP_MSG *pStopMsg;

    pStopMsg = ITK_AllocMsgType(STOP);
    if (pStopMsg)
    {
        ITK_SendMsg(pStopMsg, TASK_SYSTEM|NODE_HOST);
        ISYS_Delay(1000);
    }
}

ASSISTANCE_MSG* pMsg = ITK_AllocMsgType(ASSISTANCE);

if (pMsg)
{
    GPS_TIME Time;
    double dDum;
    STORAGE_GetCurrentTime(g_pStore, &Time, &dDum);

    pMsg->Payload.GpsTow.t.wWeek = Time.wWeek;
    pMsg->Payload.GpsTow.t.dwTowMs = Time.dwTowMs;
    pMsg->Payload.swFlags = 0;
    pMsg->Payload.swFlags |= ASSISTANCE_TIME_VALID;

    pMsg->Payload.L1a.lLat = _fixLast.Pos.L1a.lLat;
}
```

```
        pMsg->Payload.Lla.lLon = _fixLast.Pos.Lla.lLon;
        pMsg->Payload.swFlags |= ASSISTANCE_LATLONG_VALID;
        ITK_SendMsg(pMsg, TASK_SYSTEM|NODE_HOST);
        ISYS_Delay(100);

        DBG_TraceString("ASSISTANCE_MSG");
    }
}

{
    START_MSG *pStartMsg = ITK_AllocMsgType(START);
    pStartMsg->Payload.wStartMode = 0xFFFF;
    pStartMsg->Payload.wFlags = 0xFFFF;
    ITK_SendMsg(pStartMsg, TASK_SYSTEM|NODE_HOST);
}

}

// ADded 2005-11-23/TY
void FAR_CODE USER_RawEphMsgHandler(RAW_EPHEMERIS_MSG* pMsg)
{
    DBG_TraceString("Ephe");
    // Send to SYTEM task on client receiver
    ITK_SendMsg(pMsg, TASK_SYSTEM|NODE_HOST);
}

void FAR_CODE USER_BitStreamMsgHandler(BIT_STREAM_MSG *pMsg)
{
    ITK_SendMsg(pMsg, TASK_MONITOR);
}

/// User task message handler for START message. User can do here start related
/// operations.
///
/// Notice that this message is only a part of the start procedure, so receiving
/// this message doesn't yet mean that start has been completed succesfully.
/// Instead, START_COMPLETED message can be used to detect when the navigation
/// start procedure have really been completed and if it was succesful.
void FAR_CODE USER_StartMsgHandler(START_MSG *pMsg)
{
    // Perform user start events here.

    // Respond to the message
    ITK_SendAnswer(pMsg);
}

/// User task message handler for STOP message. User can do here stop related
/// operations.
///
/// Notice that this message is only a part of the stop procedure, so receiving
/// this message doesn't yet mean that stop has been completed succesfully.
/// Instead, STOP_COMPLETED message can be used to detect when the navigation
/// stop procedure have really been completed and if it was succesful.
void FAR_CODE USER_StopMsgHandler(STOP_MSG *pMsg)
{
    // Perform user stop events here.

    // Respond to the message
    ITK_SendAnswer(pMsg);
}
}
```

```
/// Message handler for START_COMPLETED message. This message is received
/// after the navigation start procedure is completed. This message can
/// be used to detect if navigation was successfully started.
void FAR_CODE USER_StartCompletedMsgHandler(START_COMPLETED_MSG *pMsg)
{
    // perform user start completed events here.

    // forward to monitor
    ITK_SendMsg(pMsg, TASK_MONITOR);
}

/// Message handler for STOP_COMPLETED message. This message is received
/// after the navigation STOP procedure is completed. This message can
/// be used to detect if navigation was successfully stopped
void FAR_CODE USER_StopCompletedMsgHandler(STOP_COMPLETED_MSG *pMsg)
{
    // perform user stop completed events here.

    // forward to monitor
    ITK_SendMsg(pMsg, TASK_MONITOR);
}

/// NAV_FIX message handler function
void FAR_CODE USER_NavFixMsgHandler(NAV_FIX_MSG *pMsg)
{
    // Perform user navfix events here.

    if ((pMsg->Payload.swFlags & FIXINFO_FLAG_NEW_FIX) == FIXINFO_FLAG_NEW_FIX)
    {
        if (!_bAidingSent)
        {
            _fixLast = pMsg->Payload;
            _SendAiding();
            _bAidingSent = TRUE;
        }
    }

    // Forward the message to the monitor task.
    ITK_SendMsg(pMsg, TASK_MONITOR);
}

/// PSEUDO message handler function
void FAR_CODE USER_PseudoMsgHandler(PSEUDO_MSG *pMsg)
{
    // Perform user pseudomsg events here.

    // Forward the message to the monitor task.
    ITK_SendMsg(pMsg, TASK_MONITOR);
}

/// Function for user task initializations
STATIC void FAR_CODE _USER_Init(void)
{
    // Do user task initializations here
    // Reserve GPIO line A7 as an input.
    // We'll set it up as a pull-up input - then we'll be able to connect it to
    // the ground, which will be convenient in the Evaluation Kit.
    GPIO_Reserve(GPIO_A7, GPIO_PULLUP_INPUT|GPIO_FALL_INT);

    // And set it to signal the user task with the SIGF_USER_GPIO signal.
    GPIO_SetTaskSignal(TRUE, TASK_USER_1, SIGF_USER_GPIO);
}
```

```
/// Function for user task resource freeing
STATIC void FAR_CODE _USER_Free(void)
{
    // Free other user task resources here if any
}

/// iTalk message handler table for the user task. This table defines handler functions for
/// each iTalk message we wish to process in the user task.
STATIC const __far ITALK_MSG_HANDLER _UserMsgHandlerArray[] =
{
    { NAV_FIX_MSG_ID,          (ITALK_MSG_FUNC)USER_NavFixMsgHandler },
    { PSEUDO_MSG_ID,         (ITALK_MSG_FUNC)USER_PseudoMsgHandler },
    { BIT_STREAM_MSG_ID,     (ITALK_MSG_FUNC)USER_BitStreamMsgHandler},
    { START_MSG_ID,          (ITALK_MSG_FUNC)USER_StartMsgHandler },
    { STOP_MSG_ID,           (ITALK_MSG_FUNC)USER_StopMsgHandler },
    { START_COMPLETED_MSG_ID, (ITALK_MSG_FUNC)USER_StartCompletedMsgHandler },
    { STOP_COMPLETED_MSG_ID, (ITALK_MSG_FUNC)USER_StopCompletedMsgHandler },
    { RAW_EPHEMERIS_MSG_ID,  (ITALK_MSG_FUNC)USER_RawEphMsgHandler },
    {0, NULL}
};

/// User task main function.
void FAR_CODE USER_Task()
{
    // Alloc & init user task resources
    _USER_Init();

    // Wait until all tasks have been started
    ISYS_WaitStartSync();

    // Set up the time so that we are triggered every second.
    ISYS_InitTimer(&UserTaskTimer, 1000, TRUE, SIGF_USER_1);

    // Iterate until a STOP_TASK message is received.
    while(1)
    {
        WORD wSig;

        // Handle received iTalk messages. This function returns either when a desired signal is
        // received, or when a STOP_TASK message has been received and the task should close
        itself.
        if (ITK_HandleMessages(_UserMsgHandlerArray, SIGF_USER_1|SIGF_USER_GPIO) ==
        E_ITK_STOP_TASK)
        {
            // The task has received a STOP_TASK message, so break the loop & exit the task routine
            break;
        }

        // When the ITK_HandleMessages returns, we know that there is
        // a signal waiting.
        wSig = ISYS_WaitSignal(SIGF_USER_1|SIGF_USER_GPIO);

        if(wSig & SIGF_USER_GPIO)
        {
            // Do brief delay in order to avoid multiple signal due to
            // switching ripples.
            ISYS_Delay(100);

            // If the source for the GPIO interrupt was A7...
            if(GPIO_GetSourcePin() == GPIO_A7)
            {
                // ... and the GPIO is now low...
                if(GPIO_GetSingle(GPIO_A7) == 0)
                {
                    // .. then we'll print a message.
                }
            }
        }
    }
}

```

```
        _SendAiding();  
    }  
}  
  
// Clear any pending SIGF_USER_GPIO signals, because these  
// were probably due to switching ripples.  
ISYS_SetSignal(0, SIGF_USER_GPIO);  
}  
  
}  
  
// Free user task resources  
ISYS_RemoveTimer(&_UserTaskTimer);  
  
_USER_Free();  
}
```

## 6. Q&A ABOUT IMPLEMENTING AGPS

The above implementation assumed that assistance data like EPHEMERIS was available directly in iTalk format.

This chapter list some frequently asked questions about implementing generic AGPS with iSuite compatible iTrax receivers.

---

**Q:** In the RAW\_EPHEMERIS message, the term ClockExt.wIODE is not part of incoming RRLP assistance data (not specified in [http://www.3gpp.org/ftp/Specs/archive/44\\_series/44.031/44031-5b0.zip](http://www.3gpp.org/ftp/Specs/archive/44_series/44.031/44031-5b0.zip)); may this field be set to 0?

**A:** The IODE value can be found from the 8 LSBs of the IODC field. Thus, decode IODC from the message coming from the network and take 8 LSBs for IODE.

---

**Q:** Which flags shall I set in the wFlags field ?

**A:** Note the difference of the ALMANAC and EPHEMERIS messages. The ephemeris contains the higher order terms of the clock data that almanacs doesn't have. Since you have filled up also the ephemeris fields, set the flags

```
pEphemMsg->Payload.wFlags |= EPH_FULL_DATA; Payload.wFlags |=  
pEphemMsg->EPH_DATA_VALID;
```

If you have the knowledge of the full week number  
pEphemMsg->Payload.ClockSub.Toc.wWeek, you should mark it valid also  
otherwise, omit it..  
pEphemMsg->Payload.wFlags |= EPH\_WEEK\_VALID;

---

**Q:** A field named User Range Accuracy Index is part of incoming RRLP assistance data; may this data be usefull to iTrax03 ? If yes, is there a way to send it using iTalk ?

**A:** User Range Accuracy Index is not used at the moment in iSuite 3.31

---

**Q:** In the incoming RRLP data, the field Fit Interval Flag (boolean) specifies if the related ephemeris data is considered valid for 4 hours or not; may it be usefull to map it in the wFitPeriod ?

**A:** Yes. You can map the Fit Interval Flag to wFitPeriod. If you know the time in hours the ephemeris are valid, you can provide this information in hours.

---

**Q:** In the iTalk Ionospheric model structure there's a field gTime to provide approximate time of download; is this field mandatory ?

**A:** The GPS time is mainly used for verifying the message reception time within hours. However, it is suggested that you give some time information. The value doesn't need to be very accurate (e.g. +-2 hours is enough). You can use the same time information as given in ephemeris data.

---