Host application developing for the RF-family hand terminals

Document history:

28.9.2005 Version 1.0 by Ari Pöyhönen.

Abstract

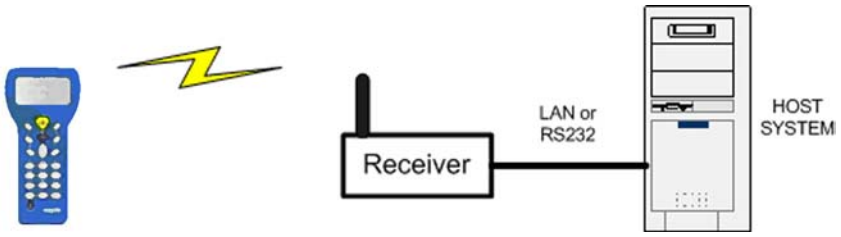
The Nordic ID RF-family hand terminals are easy and flexible solution for real-time and remote control of database. The RF-family hand terminals acts like a terminal client for the backend system so there is no need for programming of the hand terminal. The backend system runs on a server in which the hand terminal connects through the base station(s). The backend system is responsible for the User Interface elements such as Text Buttons and Input fields.

This document is intended for developing the backend system for RF-family hand terminals.

Contents

1 System operation

The system consists of radio hand terminals, which are used to communicate wireless with the host system via a receiver device using a very efficient communications protocol. The host system can be just a PC or a more complex computer system.



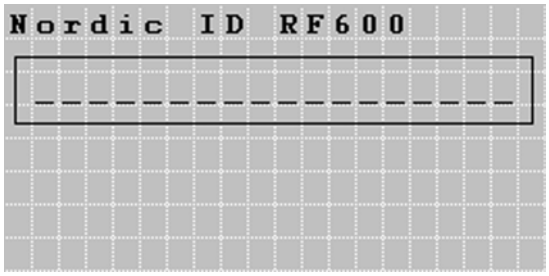
The application software runs by the host system, the handheld appears to it like a wireless keyboard/display/scanner. This eliminates the need for programming the handheld units, thus making software development and updating extremely easy and allowing software to be flexible in application. And more, the communication is rapid because it is real time.

1.1 Operation of the Hand terminal

The Hand terminal is designed to use application specific forms i.e. fields in the virtual display. This helps to make the user interface of the Hand terminal flexible and easy-to-use. The commands used to generate, modify, read and write the forms are described later in this document.

1.2 Initial state

After the batteries have been plugged in to the hand terminal, initial screen appears to the display. Also, when no forms are used, the initial screen prompts.
The hand terminal shows as a prompt a user writeable header and an input field. This field can



be filled with data from the keyboard or from a laser scanner. Any text sent by the Host Computer will clear the screen and show the text that was sent. Any user input (from the keyboard or a laser scanner) will clear the text and the initial screen will be displayed again.

1.3 Starting transaction

The user starts a transaction by using the Hand terminal keyboard to make an entry, or by reading a barcode. The Hand terminal then sends this data to the Host Computer (via the receiver device) and waits for a message from the Host. If it does not receive a correct (check summed) message within the specified time-out period, it will resend the original data as many times as it has been instructed.

By default, the hand terminal sends 3 times between 1 second. If no answer during that time, transaction failure will be generated on the hand terminal.

Note: When no entry is made on the Hand terminal by the user, it will remain in a standby state and will not be able to receive data from the Host Computer. There are some exceptions to this which will be described later.

1.4 How many hand terminals can be used in same system?

In theory, up to 65536 hand terminals can communicate with the host. But in practice, it depends a lot on the radio traffic. In theory, one transaction takes max of 300 ms on the air when using in RF600 system.

In the RF600 system the backend software receives circa three transactions per second depending on the backend system response time is quick enough (<50ms).

In the RF650 system the backend software can receive 21 transactions per second if the backend system is multithreaded and the response time is quick enough (<50ms).

2 Host application

The backend software can be executed in which ever computer and operating system in use. The RF-family hand terminals are not equipment and/or operating system dependent.

What the backend software does, is simply to receive data from the hand terminal, handle it and give response to terminal.

After the hand terminal begins the transaction by sending a data, it always waits for a response from the backend software. For that reason, it is very important to send a response from the backend software as quick as possible. The user can not operate with the hand terminal before the response is received.

2.1 Host application security

Sometimes the signal of the hand terminal overlaps with the area where another RF-system is being used in the same rf-channel. In this case there is possibility that the user interface of the other backend systems is displayed in the hand terminal screen. To avoid this situation follow the instruction below.

1. Always use different channel than the other system uses. The co-operation with the other systems operator has to be made so that there are no same channels used in both systems. Also it is strongly recommended that the consecutive channels are not used.

2. Do not set the backend software so that it can communicate with the unauthorized hand terminals. The backend system has to keep a log of the hand terminals commID which have right to access the system. If the unauthorized hand terminal is trying to communicate with the server, the queries has to be rejected and NO information to the hand held has to be transmited.

   In the backend system, the DataArrived –event at the PLServer event handler should always first authorize the hand terminal before the requests are transmitted any further.

3. Even if the hand terminal is authorized, User login and password authorization scheme has to be used to confirm that the user of the hand terminal has access to the backend software

2.1.1 Radio link security

The system developer has to estimate the importance of the secured radio pathway. Often the transferred information is not so confidential that data encryption is really needed.

The encryption of data is recommended if data includes pin-codes or social security numbers.

The RF600 –devices are operated with 434MHz radio. The radiolink encryption is not very complicated but it takes a lot of technical and special knowledge to decode it

The RF600 hand terminals include a data encryption -feature. The data is encrypted by using a 24-bit encryption key. The same encryption key has to be entered in both hand terminal and the base station in order them to get work together. By default the data scrambling –feature is disabled but if needed it can be enabled with the RF600 configuration software (Piccopla).

The RF650 Bluetooth radio is secure enough to transfer confidential data.

2.2 Checklist for good quality backend software developing

- Keep response times as fast as possible
- Ignore the requests from unidentified hand terminals
- Always answer to requests (except those from the unidentified hand terminals)
- Validate the data from the hand terminal and responce to that
- Develop application as multithread type.
- Do not resend the UI-control if the hand terminal has it already.
- Use a different radio channel than in adjacent companies.
- Do not overrun the send buffer. The maximum size of transmitted data to the hand terminal is 255 bytes. If the send buffer is full the PLServer UI-elements creation functions return FALSE. The RF6XX protocol commands DATA cell sizes are listed in Appendix B.
- Estimate the maximum number of hand terminals and transactions / hand terminal in worst case scenario. This can occur in e.g. stock taking situation and there is a danger that radio link or backend software chokes. In the RF600 backend system max 3 transactions per second can be achieved if the response time is below 50ms in the backend software.
- Test the system as well as possible before the installing it to the customer. E.g. the software works with one or two hand terminals, but how does it work in a 10 hand terminal environment?

- Estimate the number of base stations needed. If the operating area of the hand terminals can not be covered with one base station, the number of base stations needs to be increased. Also the base stations have to be connected to the LAN with the Device server.

2.3 Character map of the hand terminal

The character map of the hand terminals is shown in ASCII table below.

If no desired characters found, please contact NordicID.

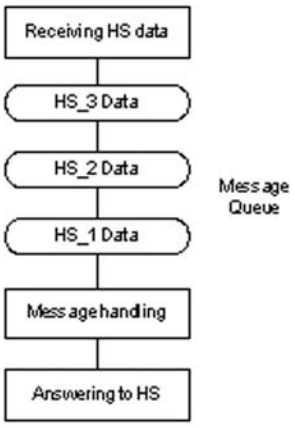
2.4 Single- or multi threaded application?

In the RF600 system there can be dozens of hand terminals operating simultaneously with the backend software. When the number of hand terminals increase, the backend software has to be able to handle them all. The hand terminal users may simultaneously send data to the backend software.

2.4.1 Single thread application

The single thread application the messages are handled in a queue. If the handling of the message takes seconds, all other hand terminals are not able to receive any information from the backend software at the time.

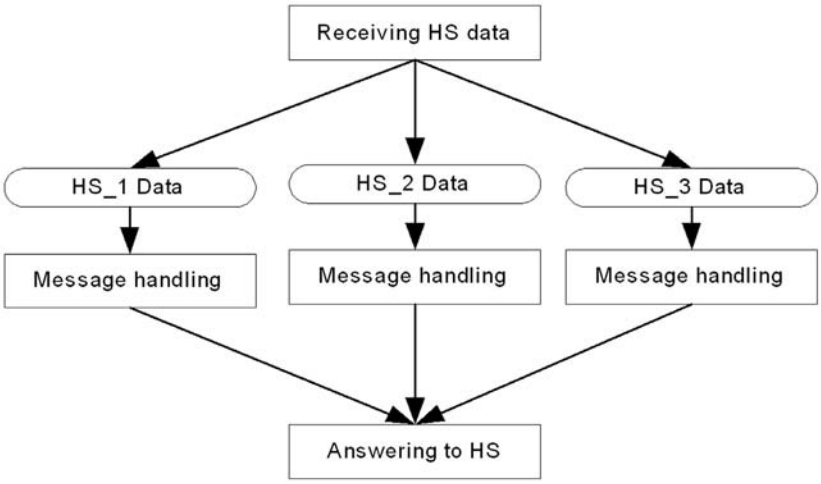
The single threaded application can be used in the system where the hand terminals are not at very intensive use. Fast data handling can prevent the delays in hand terminal.



2.4.2 Multi threaded application

In the multi threaded application the messages are handled simultaneously at separate processes. The response is sent after the message is handled regardless the other processes states.

The multithreaded application is recommended to use if there are several hand terminals in the system and the use of them is intensive.
Visual Basic does not offer appropriate tools for multithreaded applications. All multithreaded



examples in this document are made with Visual C++.

2.5 Drivers

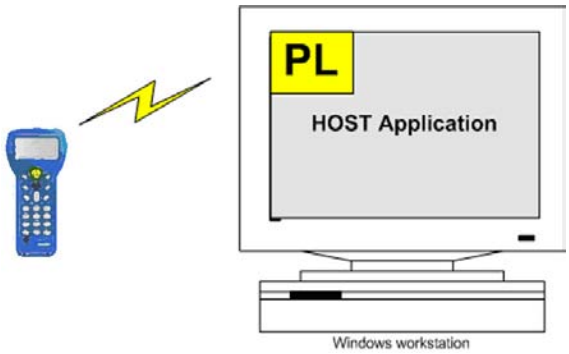
The driver software preprocesses the data from the hand terminals. The base station has already checked the CRC and the origin of the data before relaying it to the software driver.

Functions of the driver:

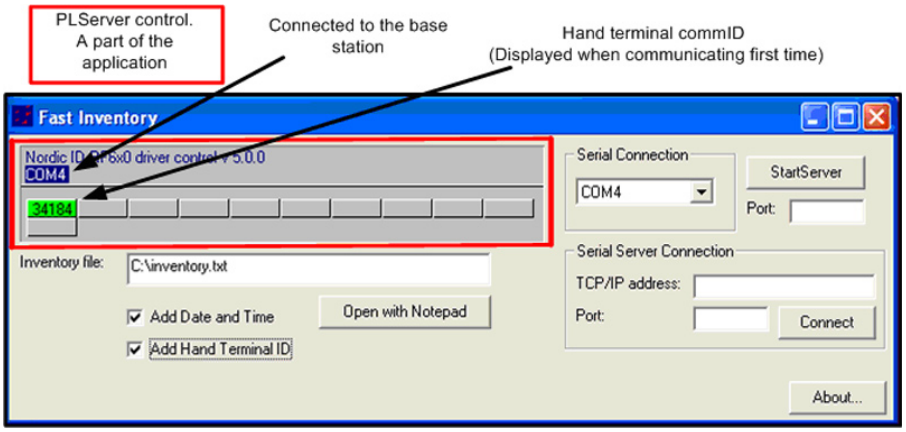
- Relays the data to the application
- Resends the data
- Blocks similar consecutive messages to the application.
- In a multiple base station system driver sends the response from the host thru the best received signal strength base station.

2.5.1 PLServer

Nordic ID provides the driver for the Windows environment. The PLServer is a Windows ActiveX control (PLServer.ocx), which controls the data communication between the base station and the HOST application.



Example of the application where PLServer control is used



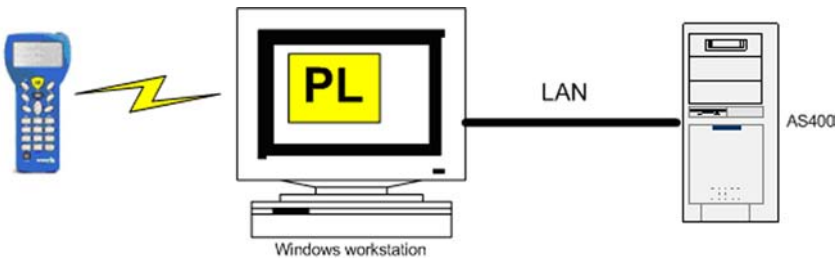
The PLServer makes HOST system software development easy by doing all the dirty work for you. The use of PLServer provides a lot of functionality and code that you don't have to write or debug. All you have to do is to figure out the entry points and how to use them.

Note: Latest PLServer version can be downloaded from:
http://www.nordicid.com

2.5.2 Drivers for non-windows environment

If the backend software is executed in the non-windows environments (UNIX, Linux, AS400) the driver software must be developed to the environment also (see RF-family communication protocol, Appendix B).

However it is possible to create a backend system where the business logic is executed in non-windows environment e.g. AS400 or UNIX. In this case the PLServer runs in the windows-based computer and communicates in a network with an actual backend system.



The advantage of this procedure is that there is no need for developing drivers to the non-windows operating system. So the actual application developing time is faster.

See also Chapter: RF6XX Application Router from this document.

This document describes the application development to the Windows OS with PLServer ActiveX component.

3 Building development environment

The ”RF-family host programming tutorial” cd includes example programs in Visual basic and Visual C++. All examples use the PLServer ActiveX component.

3.1 System requirements

System requirements for the application developing:

- RF-family hand terminal (RF600 or RF650)
- RF600 Base Station / Bluetooth dongle (RF650)
- PC with Windows OS (W2k or XP)
- Serial port connection for the RF600 Base Station
- Visual Studio .NET 2003 or later (optional)
- ”RF-Family host programming tutorial” CD
- RF-family hand terminal (RF600 or RF650)
- RF600 base station or Bluetooth dongle (RF650)
- Windows PC (w2k or XP)
- Physical Serial port (RS232) for the RF600 base station
- Visual Studio .NET 2003 or later (optional)
- ”RF-Family host programming tutorial” CD

3.2 Sample program installation

Insert “RF-Family host programming tutorial” CD in to the disk drive and wait until setup starts up.

Default directory for the sample-files:
Program files→NordicID→RF-family→Samples

The example software can be launched at:
Start→ Programs→NordicID→RF-family→Samples

Setup reqisters the PLServer automatically.

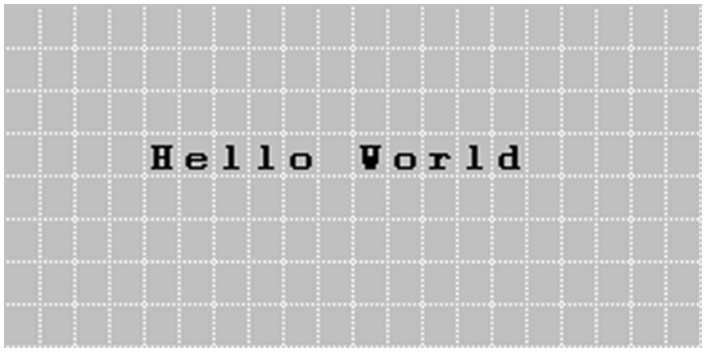
3.3 Installing hardware

Follow the instructions with the hand terminal and base station to install the hardware.

3.4 Test your hardware

Follow the steps below to ensure the communication between the host and the hand terminal:

1. Start ”Hello world” application from Start→Programs→NordicID→RF-family→Samples
2. Select correct COM port for the RF600 base station.
3. Press OK- or any F-key from the hand terminal.
4. “Hello World” text should appear in to the hand terminal screen.
5. “Hello World” text disappears when pressing any key from the hand terminal.



Possible reasons for failure:

- No batteries in the hand terminal
- No serial cable connected between the base station and the PC COM port.
- No power in the base station
- The Base station and the hand terminal are in different channel
- The base station has been connected to wrong COM port
- The COM port is already used by another device. (Create file error.. in PLServer screen.

4 My first host application

Follow instructions in this chapter to create “Hello World” –type application with Visual C++.

4.1 Creating”Hello World” Visual C++ project

1. Start Visual Studio .NET 2003
2. Select from File menu New →Project…
3. Select project type as MFC and template: MFC application
4. Set Name and Location and press OK
5. MFC application wizard starts
6. Set application type: Dialog based. Press “Finnish” button.
7. Open Main dialog from the Resource view.
8. Right mouse click over dialog and select Insert ActiveX control..
9. Select PLServer control from the list.
10. Place PLServer control to proper place in to the dialog.
11. Right mouse click over PLServer and select “Add variable..” .
12. Set PLServer control variable like: m_pl
13. Right mouse click over PLServer and select “Add Event Handler..”
14. At Event handler wizard, select “DataArrived”.
15. Go to edit DataArrived handler and add two lines of code:

```
m_pl.PopMessage(id,64,”Hello World”);
m_pl.Send(id,-1);
```

16. Add button to dialog
17. Set button caption “Connect to COM port”
18. Goto button click handler and add one line of code:

```
m_pl.Connect(4);
// This Connects to COM4.
// Choose correct COM number for your base station
```

19. Compile executable and run it.
20. Press “Connect to COM port” button.
21. Press OK or and F-key from the hand terminal.
22. “Hello World” PopMessage appears to the hand terminal screen.

HelloWorld esimerkki koodit löytyvät Samples hakemistosta:

- HelloWorld_VC (Visual C++)
- HelloWorld_VB (Visual Basic)

5 Handling hand terminal messages

When user starts a transaction by sending a data to the host, the PLServer sends a “DataArrived”-event to the backend software. The application handles the data and sends a response. The response can be a new form or some supplementary information to the excising form in the hand terminal. The backend software has almost unlimited possibilities choosing which data is sent to the hand terminal.

Note: The backend software has always to send a response to the hand terminal. Except if the hand terminal is not authorized. Then the response should not be submitted! (see chapter “Host application security”)

5.1 DataArrived event

PLServer triggers this event when receiving data from the hand terminal. Host application handles received data and sends an answer back to the hand terminal with Send() -method.

5.1.1 DataArrived (event)

PLServer launches this event when receiving data from the hand terminal. Host application handles received data and sends an answer back to the hand terminal with Send( ) method.

While DataArrived event is handling hand terminal data, PLServer cannot launch any other DataArrived event at this time. Therefore, it's recommended that Host application creates a separate thread to handle hand terminal data and release DataArrived handling as fast as possible. Multithread application is not necessary application gives fast response and only few hand terminals are used at same time.

(event) DataArrived(long id, short frameid)	
Parameters (given by PLServer)	
id	Hand terminal commID
frameid	The number of form defined by the user. If the frameid is -1, the data comes from the initial screen or from a function key. This value is defined in **Send**( ) method.
See also	
Send( ) method	

Note: During the DataArrived –event is handled any other data cannot be handled at the same event. So it is highly recommended to free the DataArrived –handler as quick as possible.

5.1.2 Starting the data handler thread

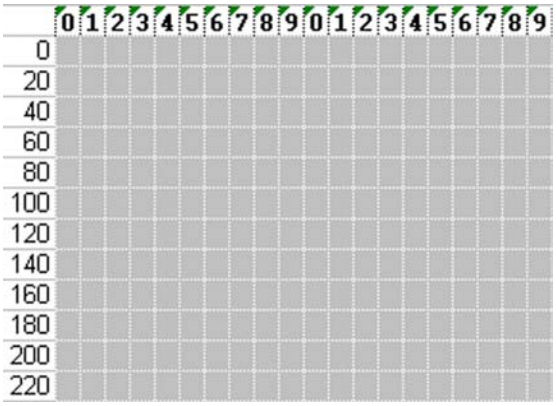
In the multi threaded applications the data handling –thread is launched in DataArriver –handler. Only commID and frameID parameters are passed to the DataHandled –thread. Then the DataArrived –handler is exited immediately. This action ensures that data can be simultaneously processed and received from the hand terminal.

```
//DataArrived event. Start data handling thread and return immediately.
void CFormsDlg::DataArrivedPlserverctrl1(long id, short frameid)
{
	//Create thread
	m_DataHandlerThread=(DataHandlerThread*)AfxBeginThread(RUNTIME_CLASS(
	DataHandlerThread), THREAD_PRIORITY_NORMAL,0,CREATE_SUSPENDED);

	//This windows object is needed in handling thread
	m_DataHandlerThread->SetOwner(this);

	//Passing data to handling thread
	m_DataHandlerThread->DataToProcess(id,frameid);
	//Run it.
	m_DataHandlerThread->ResumeThread();
}
```

Example of starting the Data Handler -thread

5.2 Forms

5.2.1 Hand terminal display area



The Hand terminal has a virtual display page of 12 x 20 characters. The actual display size) is 8 x 20 characters, so that only one third of the virtual page can be viewed at one time.

All UI-elements are addressed by positions. Position 0 is upper left corner and position 239 is down right corner.

5.2.2 Application forms

Usually the backend software has several interfaces for the user. So the software has to know what kind of form is currently used in the hand terminal. The PLServer method SetFormID can be used to identify every form before it is sent to the hand terminal.

The FormID is saved to the memory of the hand terminal. In every transaction when the user sends data to the host, this FormID is sent also. The GetFormID is used to find out what kind of form user has in the hand terminal.

5.2.3 Designing forms

Before the writing the code of the backend software, there has to be made some planning for the forms. FormPlanner.xls –excel file is helpful tool for that. That tool can be also used to create “screenshots” for the user manual of the backend software.

5.2.4 User Interface elements

The backend software can use several elements to create forms:

User Interface elements are:

- Text - Text can be created with TextEx –method.
- Input field - These fields may be filled by using the keyboard or the laser scanner. Fieds are underlined. Input field is active when cursor is visible: 

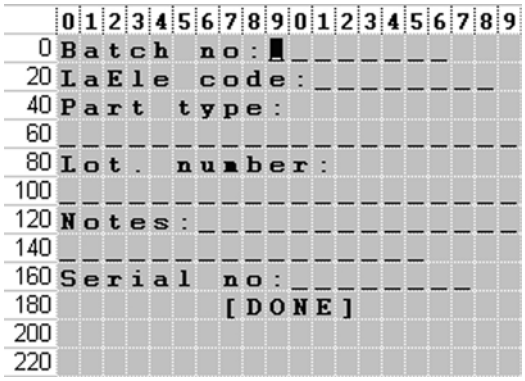
  Some input fields can only be filled by keyboard. See more information from NewField and NewFieldEx method.
- Button - When active, user can only press OK button to send information to the host.

Active button is completely black. Inactive button may look like a plain text, so it would be useful to add extra characters to beginning and end like < > or [ ] in order to separate texts and buttons. Button method creates Button element to the form.

The UI-elements are displayed on the form until one of the following events occurs:

- User resets device (batteries out and back in or keys:SHIFT+DEL)
- Host sends ClearForm method
- Host sends ClearCmd method

Set of UI-elements in the form



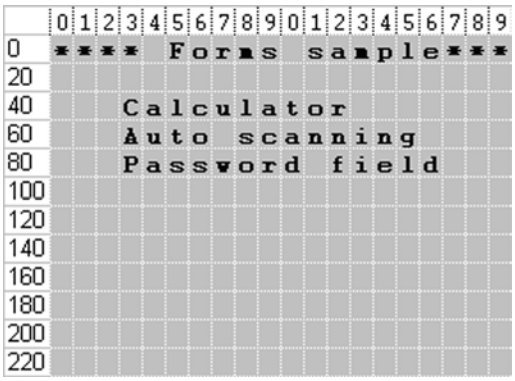
There is always one active UI-element on the hand terminal. The user can activate the desired UI-element by pressing the arrow buttons or the active UI-element can be set from the backend software.

Tip: There is no need to create the form if it is already at the hand terminal. This action reduces the radio traffic and simplifies the backend software.

5.2.5 Main menu

First screen in the hand terminal is usually the main menu which is displayed after the hand terminal connects for the first time to the backend software.

Main menu of Forms sample application



The main menu creation function:

```
void DataHandlerThread::SendMainMenu()
{
	//Let's make sure that screen is empty from texts and fields
	m_dlg->m_pl.ClearForm(id);

	m_dlg->m_pl.TextEx(id,0,"*** Forms sample ***");
	//Create menu buttons
	m_dlg->m_pl.Button(id,CALCBUT_POS,"Calculator");
	m_dlg->m_pl.Button(id,AUTOSCANBUT_POS,"Auto scanning");
	m_dlg->m_pl.Button(id,PASSWORDBUT_POS,"Password field");
	//Set form id
	m_dlg->m_pl.SetFormID(id,MAIN_MENU_FORM);
	//Send to HS immediately
	m_dlg->m_pl.Send(id,MAIN_MENU_FORM);
	//Set form identification to frameid also
}
```

Position defines the Button –UI element displacement on the form. MAIN_MENU_FORM is the unique ID-number which is saved to the hand terminal with the SetFormID-method. Following constants are found in PLConst.h –file.

```
//Main menu constants
#define	MAIN_MENU_FORM	1
#define	CALCBUT_POS	43
#define	AUTOSCANBUT_POS	63
#define	PASSWORDBUT_POS	83
```

5.2.6 Handling data from Main menu form

The user of the hand terminal starts the transaction by choosing one of the input fields in the Main menu with the arrow buttons and confirms the selection with the OK-key. From the data received the backend software has to identify the Button –pressed.

```
void DataHandlerThread::DataFromMainMenu()
{
	//we know now that one of the Main menu items has been selected
	//Let's finds out which one using IsData method.

	if(m_dlg->m_pl.IsData(id,CALCBUT_POS))
		SendCalculatorForm();

	if(m_dlg->m_pl.IsData(id,AUTOSCANBUT_POS))
		SendAutoscanningForm();

	if(m_dlg->m_pl.IsData(id,PASSWORDBUT_POS))
		SendPasswordFieldForm();


	return;
}
```

In the transaction the contents of the ALL input fields are not necessary sent to the backend software. In the Main menu –example application there are 3 locked input fields where the user cannot type text with the keyboard. The only thing that the user can do is to activate the input field by choosing it and pressing the OK-key to send the content of the particular input field to the backend software.

IsData –function of the PLServer can be used to check which button is pressed. IsData returns TRUE, if the data is from the particular input field.

BOOL IsData (**long** id, **short** pos)	
Parameters	
id	Hand terminal commID number
pos	Position of the input field (0-239)
Return value	
True	Data coming from pos field.
False	No data coming from pos field.
See also:	
GetData	

5.2.7 The branching of the program

Because of the integrity of the backend software it is recommended that the creation and the data handling of the form are coded in the separate functions.

Functions from the Forms -example

```
SendMainMenuForm( ); //Form creation
DataFromMainMenu( ); //Data handling

SendCalculatorForm();
DataFromCalculatorForm();

SendAutoscanningForm( );
DataFromAutoscanningForm( );

SendPasswordFieldForm( ) ;
DataFromPasswordFieldForm( );
```

See the source code of the Forms –example at Run() –function in the DataHandlerThread.cpp.

5.2.8 Clearing the UI elements

The UI-elements on the form are displayed until the device is resetted or the ClearForm –method is called. Every time the new form is created in the hand terminals display, it is recommended to call ClearForm –method.

If only a part of the UI-element in the form has to be removed, ClearCmd method can be used. With ClearCmd –method it is able to remove text only and/or input fields or those content.

6 Input fields

The maximum number of input field in the form is 20. The length of the input field is can be 1 – 63 characters. If there is no text in the input field, in other words it is empty, it displays as underline characters (_ _ _ _ _ _) in the hand terminal.

Empty input field can be completed either typing with the keyboard or scanning a barcode with a barcode scanner. User can activate the desired input field or button by using the arrow buttons to move the cursor on it.

All the input from the keyboard or the scanner will be displayed in the active input field. Exception to this rule is the NewFieldEx –methods READER_DEFAULT feature. By using this feature it is possible to display characters to the non-active input field.

The input fields can be defined so that it can not be filled with the barcode reader.

6.1 Creating an input field

Input fields to the form are created with NewField and NewFieldEx –methods. The behavior of the input field can be set in the parameters of these methods.

BOOL NewField (**long** id, **short** pos, **short** field_len, **short** style)

Parameters		
id	Hand terminal commID number	
pos	Start position of the input field. (0 - 239)	
field_len	Field length	
style	Style bits of the input field.	
Bit 0	SND_ENTER	Field is sent to HOST by pressing the OK key
Bit 1	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, if the field itself gave the SND_ALL command.
Bit 2	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the OK key is pressed.
Bit 3	FLD_LOCK	Field is locked. Field cannot be written to (used for “button” style fields.)
Bit 4	FLD_LINE	Field is underlined. _ _ _ _ _ _ _ _ _
Bit 5	FLD_READER	Field can be filled with laser- or external scanner data.
Bit 6	FLD_ACKCLR	Field is cleared if ACK is received.
Bit 7	FLD_ACTIVE	Field is active (it has a cursor).
SND_ENTER and FLD_READER combinations:		
SND_ENTER	FLD_READER	FUNCTION
0	**0**	Field is not sent by pressing the OK key, and cannot be read with a laser scanner.
0	**1**	Field is not sent by pressing the OK key, but it can be filled from the keyboard and with a laser scanner.
1	**0**	Field is sent by pressing the OK key, but cannot be filled with a laser scanner.
1	**1**	Field is sent by pressing the OK key or by reading with a laser scanner.
Return value		
True **False**	Command added to the send buffer successfully Send buffer is full	

NewFieldEx

- Defines a new Input field in to the hand terminal screen.
- NewFieldEx method is same as NewField ( ) but style bit's OVR and READER_DEFAULT are replaced bit's FLD_LINE and FLD_ACKCLR.
- Field is underlined automatically.
- A maximum of 20 input fields can be defined for one form.

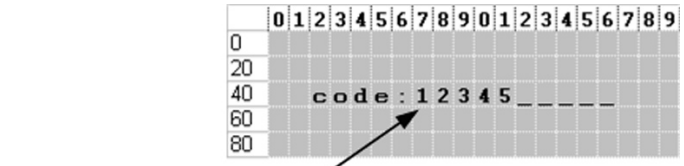
BOOL NewFieldEx (**long** id, **short** pos, **short** field_len, **short** style)

Parameters		
id	Hand terminal commID number	
pos	Start position of the input field. (0 - 239)	
field_len	Field length	
style	Style bits of the input field.	
Bit 0	SND_ENTER	Field is sent to HOST by pressing the OK key
Bit 1	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, if the field itself gave the SND_ALL command.
Bit 2	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the OK key is pressed.
Bit 3	Reserved	Reserved set as "0"
Bit 4	OVR	Overwrite mode. When OVR is set, and the field becomes active, the cursor moves to the starting position of the field. When the cursor is in the starting position, the previous text will be overwritten by typing.
Bit 5	FLD_READER	Field can be filled with laser- or external scanner data.
Bit 6	READER_DEFAULT	If another field is active and it has not FLD_READER set, laser scanner is activated and data goes to this field automatically. Only one READER_ DEFAULT field can be defined per form. This bit has no effect if FLD_READER bit is not set.
Bit 7	FLD_ACTIVE	Field is active (it has a cursor).
SND_ENTER and FLD_READER combinations:		
SND_ENTER	FLD_READER	FUNCTION
0	**0**	Field is not sent by pressing the OK key, and cannot be read with a laser scanner.
0	**1**	Field is not sent by pressing the OK key, but it can be filled from the keyboard and with a laser scanner.
1	**0**	Field is sent by pressing the OK key, but cannot be filled with a laser scanner.
1	**1**	Field is sent by pressing the OK key or by reading with a laser scanner.
Return value		
True	Command added to the send buffer successfully	
False	Send buffer is full	

6.2 Input field reading to the application

The GetData –method returns a string from the defined input field. However the IsData -method should be used to check if the hand terminal has just sent the data from that particular input field. The IsData –method is no need to use if it is already known that the only data the hand terminal can send is from that particular input field.

In this example GetData returns the string of the input field in position 47.




```
CString codeData;

If(m_pl.IsData(id,47)==TRUE)
{
    codeData= m_pl.GetData(id,47);

    //codeData=="12345"
}
```

6.2.1 The validation of the input field contents

The user input to the input field can be any character in the characters set. So the validation of the input data is very important.

The input field cannot validate the data input. E.g. if the input has to be between 1 and 100, the backend software has to do the validation of the data. If the data is not within the range, the Popmessage can be sent to the hand terminal.

The data validation example:

```
Int value;
CString txt;

If(m_pl.IsData(id,47)==TRUE);
{
      txt=m_pl.GetData(id,47);
      value=atoi(txt,2);
      if(value < 1 || value > 100) //Wrong range
      {
            m_pl.PopMessage(id,21,"Type between 1-100");
            m_pl.FldClear(id,47,FLD_CLR);
            m_pl.Send(id,-1);

            return; //Don't go further

      }

      //DATA ok. Continue data handling normally
```

6.3 Sending the text to the input field

The application can send text to the input field by using FldTxt –method if the input field is already created with NewField or NewFieldEx –functions. The input field creation function can be sent at the same time than text input –method. If the input field contains text it will be overwritten.

```
#define POS_1     25
#define LENGTH    10
.
//Create field
m_pl.NewFieldEx(id,POS_1,LENGTH,SND_ENTER);
//Fill it
m_pl.FldTxt(id,POS_1,"1234");
.
.
m_pl.Send(id,-1)

RESULT:
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0
20           1 2 3 4 _ _ _ _ _ _
40
.
.
```

BOOL FldTxt (**long** id, **short** pos, **LPCTSTR** txt)

Parameters	
id	Hand terminal commID number
pos	Position of the input field. (0 - 239)
txt	Text for the field.
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

6.3.1 Modifying the existing input field

The backend software can modify the parameters of the existing input field. The application should be designed so that the user of the hand terminal has to make as few keystrokes as possible. The FieldCmd –function can activate the certain input field so that the user has no need to use the arrow keys for navigation.

With the FieldCmd –function following commands are available:

- Remove the input field
- Clear the input field
- Lock the input field
- Activate the input field

Note: There can be only one FieldCmd –command with FLD_ACTIVE –flag on at the transmission to the hand terminal. This is because of in the form there can be only one UI element active at the time.

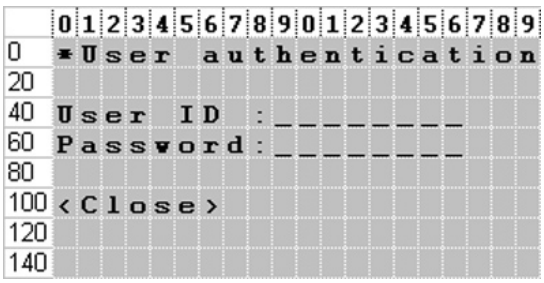
See the using of FieldCmd –method at the Forms sample –source code.

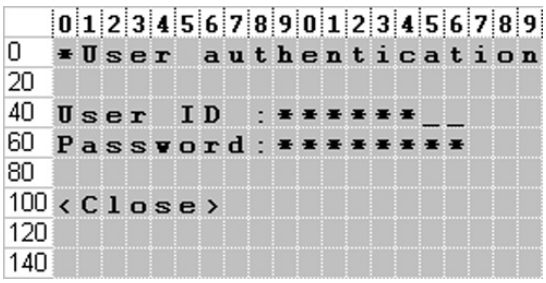
BOOL FieldCmd(long id, **short** pos, **short** fcmd)			
Parameters			
id	Hand terminal commID number		
pos	Position of the input filed		
fcmd	**Bit 0**	FLD_ REMOVE	Removes the field
	Bit 1	FLD_CLEAR	Clears the field (from locked fields also)
	Bit 2	Reserved	
	Bit 3	FLD_LOCK	Locks the field (cannot be written to)
	Bit 4	Reserved	
	Bit 5	Reserved	
	Bit 6	Reserved	
	Bit 7	FLD_ACTIVE	Field is set to active.
Return value			
True	Command added to the send buffer successfully		
False	Send buffer is full		

6.3.2 Password style input fields

It is recommended to authorise the user of the hand terminal before the access is granted to the system.

Below is an example form from the Forms -sample application. The input fields where the user inputs the user ID and the password are shown to the user as *-characters.





The backend software handles the content of the input fields as normally. SpecialCmd –method is used to set max. 3 input fields to ”Password style” input fields.

See at the Forms-sample source code and at definition of SpecialCmd –method how to create “Password style” input fields.

Note: Input fields are created normally with the NewField- or NewFieldEx –methods and after that they are set to “Password style” input fields by using the SpecialCmd –method.

7 PopMessage

The PopMessage –method is used to send informative text to the hand terminal screen without destroying the UI-elements on the form.

All UI-elements on the hand terminal screen are hidden when the PopMessage is displayed. The PopMessage disappears by pressing any key of the keyboard and the previous UI-elements appears on the display.

PopMessage is an efficient way to inform the user in various situations e.g. if the user input is not in the allowed range.

BOOL PopMessage (**long** id, **short** pos, **LPCTSTR** txt)

Parameters	
id	Hand terminal commID number
pos	Start position of the button. (0 - 79)
txt	Text for the screen. (May include VT and CR characters.)
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

8 Sounds

The backend software can use Bell- and Beep –methods to play sound on a hand terminal. Most common situations where sounds are used is to signal a warning or a failure to the user. It is recommended that PopMessage is used in parallel with sounds for detailed info.

The Bell –method is used to create a short sound. With the Beep –method it is able to play a short sequence of sounds with different pitch and duration.

9 Sending a message to hand terminal

The hand terminal is not able to receive data from the backend software if the user of the hand terminal hasn't started a transaction. This feature saves the battery. Usually the transaction begins with pressing the OK- or F- keys. The radio of the hand terminal stays on until the response from the backend software.

However it is possible to set the hand terminal so that in the defined time intervals it keeps asking if the backend software wants to send data to the hand terminal.

9.1 Application example

The hand terminals are used in the store – warehouse environment. The clerk in the store counts the items in the shelves and sends an order to the warehouse if needed. The clerk scans the barcode from the item, inputs a quantity and sends an order to the backend software.

The hand terminal in the warehouse is configured to ask data from the backend software. So next time the hand terminal in the warehouse asks the data from the application by using the WHAT -command, the backend software relays the order to the hand terminal. The hand terminal receives the order and signals the user with a beeping sound to notify of an incoming request. In the display

9.1.1 What method

What –method is used to configure the hand terminal to send the “What” –query at intervals (1 s – 99 s) to the backend software. The What-query is a ”Is there any messages for me?” –like question to the backend software.

When the hand terminal sends “What” signal, the PLServer triggers What –event instead of DataArrived. It is recommended (but not necessary) that the application sends a response to the hand terminal .

9.1.2 What

The hand terminal sends WHAT command (18h) to HOST every time between specified delay seconds.

BOOL What (**long** id, **short** delay)	
Parameters	
id	Hand terminal commID number
delay	0 – 99 seconds If DELAY is 0, The hand terminal stops sending “WHAT” commands
Return value	
True **False**	Command added to the send buffer successfully Send buffer is full
See also	
“What” event	

Hand terminal send WHAT command to HOST every time between specified ***delay*** parameter.

The hand terminal stops sending “WHAT” when the hand terminal is reset or *delay* parameter is 0.

Example, host command What (id,5); set HS to send “ what” code between 5 second. When Host gives What (id,0), HS will stop sending “what” code.

9.1.3 Hand terminal behavior when “What” is sent.

Hand terminal waits answer from host after WHAT code is sent. Wait time is same as “Reception time limit” parameter in hand terminal. Default is 1 sec.

If no answer from host within Reception time limit, WHAT code resends are made as many times specified at “Resending time” parameter. Default 3 times.

If all resends are used and still no answer from host, HS will not generate transaction failure beeps and “F” sign. Then “WHAT” code is sent to the host next time until x seconds is expired.

9.1.4 Using “What” method instead of “Receiver” method

Some cases RECEIVER mode has been used to keep radio open and listening messages from host. However, problem of using this method is higher battery consumption and possibility of data loss. Also host application developing might be more complicated.

Using “What” is more reliable and simple than Receiver mode so it’s highly recommended to use What instead of Receiver mode

Note: What-method can operate in two different ways in the hand terminal. The old style is that the query is sent only once after a period of time. The new style is to send the queries continuously at intervals until the backend software stops the process with What(0) -command.

Behavior of the ”What” is chosen from the menu in the hand terminal:

Settings→ ”What” Behavior
(0 = New style)
(1 = Old style)

10 Base station connections

The base station is needed for communication between the hand terminal and the backend software. The base station transmits the signals to and from the hand terminals and relays them to the backend system.

10.1 Serial port connection

The base station connects to the backend software thru the RS232-port. Default settings for the RS232-port are 19200,8,n,1 and it is connected to the COM-port in PC with the special cable from the Nordic ID.

The backend software needs to open and connect to the COM-port with the PLServers connect -method.

The application is connected to PC serial port by using the Connect -method.

BOOL Connect(short com_port)**:**	
Parameters:	
<com_port>	COM port number of the HOST PC-
Example:	

Connect(1) //Connect to COM 1
Connect(0) //Disconnect	

Return value:	
True	if connected successfully
False	if port not exist or port is already in use.

The opened COM-port can be seen in the PLServer control window.

COM-port opened successfully.

Unsuccessful opening of the COM-port.

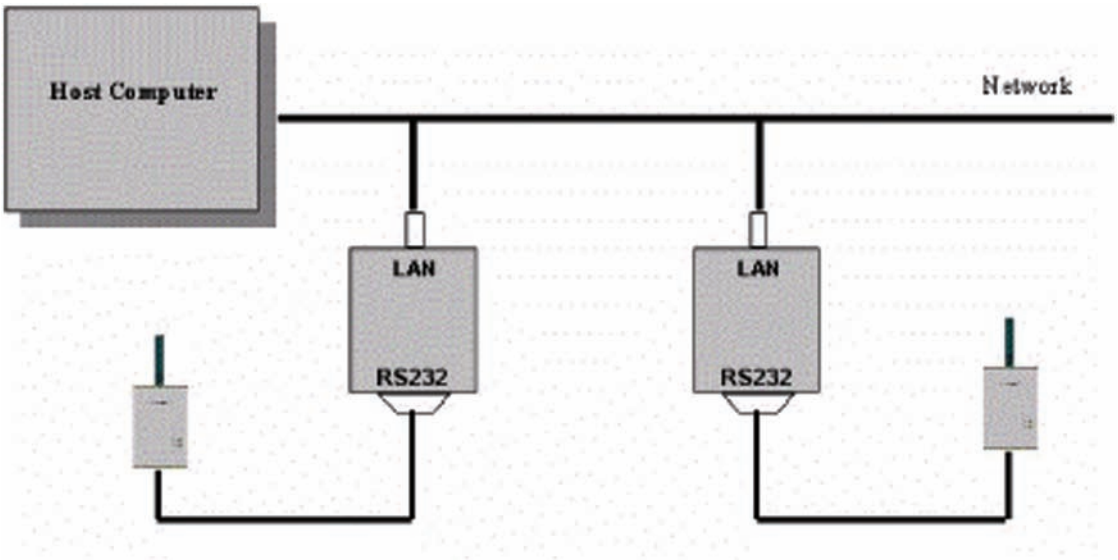
10.2 Connecting several base stations

If the working area cannot be covered with only one base station, more base stations can be connected to the backend system by using the Local Area Network (LAN).

10.2.1 Device servers

Devices that transform the serial-communication to TCP/IP –communication are called Device servers.

There are several device server brands and manufacturers and those should also work with the RF600 base station. Usually those are server type of devices which means that backend software creates the TCP/IP connection to the device. In some cases the device server may work as client-type, which mean that they make the connection to the backend software.



The backend software creates the TCP/IP connection by using the PLServer –method.

This method creates a network session between device server and PLServer. Up to 255 server connections can be made simultaneously. This method starts connection procedure and returns immediately.

If network adapter is not available, PLServer tries to reconnect in to it automatically between 10 seconds until DisconnectSerialServer has been called or application closes.

short ConnectToSerialServer(LPCTSTR addr, short port)**:**

Parameters	
addr	String value of TCP/IP address of the device server.
port	Port number of the network adapter (short value)

Example

retval = ConnectToSerialServer(“192.168.0.1”, 7001);
if (retval == -1) <Cannot connect to serial server>
else <connection succesful. Save retval as a session number>

Return value

Return value indicates the session number.
If -1, All 255 servers has been used.

See also

DisconnectSerialServer
SerialServerMessage (event)

The successful connection to the Device server is indicated with a red text shown below:

The number after the ”Serial Servers:” –text indicates how many PLServers are connected to the application.

Usually the TCP/IP connections are created when the application starts up. The PLServer automatically closes the connection when the application shuts down.

The PLServer can act like a server and wait for connections from clients. When the Device server has configured to act like a client, it has to be configured also to contact to the PLServer.

10.2.2 StartServer

Starts to listen client connections to the specified TCP/IP port

BOOL StartServer(short port)**:**

Parameters:	
<port>	Port number where Clients connects

Example:

```
StartServer( 1024)
//Starting server and allow connections to 1024
```

Return value:	
True	If server started succesfully
False	If the server cannot be started

See also

StopServer

Green clients –text is shown in the PLServer window, when the StarServer –method is called. The number after the Clients: states the number of clients connected.

11 Receiving and Transmitting RAW data

PLServer contains the functions of receiving and sending raw data over the TCP/IP network. The raw data can be used with third party hardware e.g. network printer or GPS-receiver.

First thing is to create TCP/IP connection by using the ConnectToSerialServer –method. This function returns the device ID.

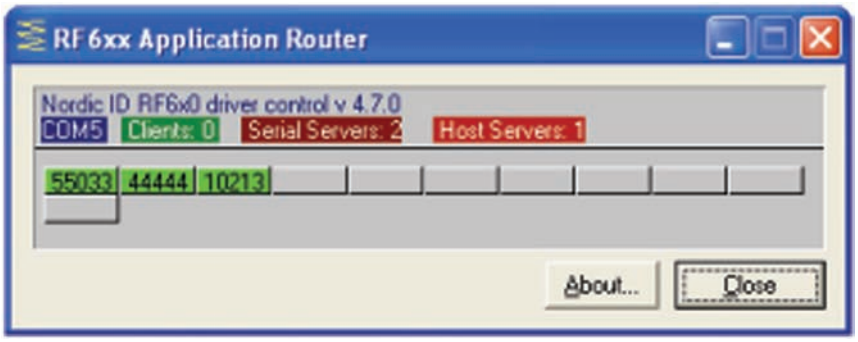
Example: Creating TCP/IP –connection to receive GPS-data.

The PLServer RawDataArrived -event (with parameters connID, received data and data length) is triggered when data is sent by the third party hardware.

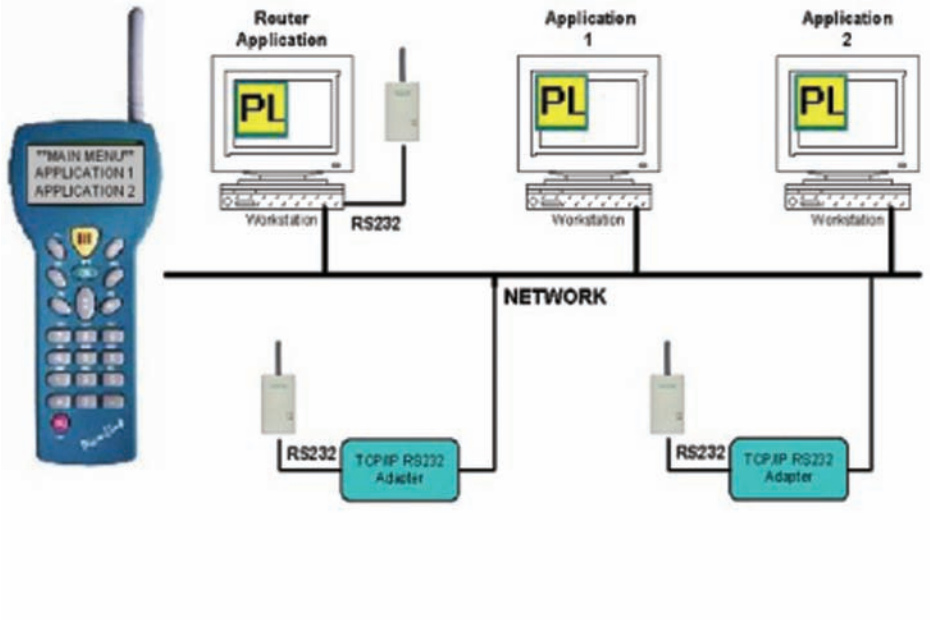
SendRawData –method can be used to sent raw data to third party device.

12 RF6xx Application Router

The RF6xx Application Router is a Windows application which handles several functions that system integrator doesn't have to implement in the host application. It helps system developers who don't want to use PLServer in their business application. Usually, it is quite hard programming job for the non PLServer developer to implement roaming, resend handling, CRC checking and other functionalities that RF6xxAppRouter (PLServer) already has.



In some cases, there is need to use several separate host applications but still use same base stations. The user of the hand terminal can select host application from the menu. The Application Router and the remote host applications are connected together by using a TCP/IP connection.



12.1 The RF6xx Application Router features

- Handling the base stations and the remote host connections.
- Filters received messages that only one message is passed to the remote host.
- Controlling the hand terminal access rights. Possible to define that the specific hand terminal can use only the specific host applications.
- Main menu of the Application Router can be opened by pressing defined (hot) key. (Like: “F10”)
- Up to six (6) remote host applications can be defined.
- Notifies the hand terminal user when the remote host application is not available.
- Supports the RF600 and RF650Direct hand terminals
- connList.txt Connection list file for defining the base station and the remote host connections.
- appList.txt Application list file for specifying remote host applications and the hand terminal access rights.

12.2 System operation

The RF6xxAppRouter program can be set to start automatically when PC is started. While starting, the RF6xxAppRouter reads connList.txt file and opens the connections of the base station and the remote hosts. After connections, the RF6xxAppRouter opens appList.txt file which specifies remote host applications and hand terminal access rights. The appList.txt consist also the text for the hand terminal display when main menu is opening and text when remote host is not available.

When the hand terminal user connects first time to the AppRouter, main menu of the available host applications will be appear in to the hand terminal screen. The hand terminal user can select the application to use. When selected, the hand terminal goes to the initial state. Next sending with the hand terminal will be routed to the remote host. The AppRouter will remember routing path as long as the user presses “main menu” key (Like: “F10”).

Note: When making changes to the connList.txt and appList.txt, RF6xxAppRouter program must be restarted.

12.3 Format of connList.txt

12.3.1 Serial Port Connection

Open the serial port connection for the RF600 base station. Only one serial port connection at the time is possible.

[COM:<port number>]

Example:
COM:1 //Opens COM1

12.3.2 Start server

Listen client TCP/IP connections to the specific port.

[SERVER:<port to listen>]

Example:
SERVER:1200

12.3.3 Client connection.

Create TCP/IP connection to the specific address and port.

[CLIENT:<tcp/ip_addr or name:port]

Example:

CLIENT:194.100.186.39:10101
CLIENT:DemoPC:7001

12.3.4 HOST application connections

Create connection to the remote host. The remote host must be in TCP/IP server mode for listening incoming connections to the specific port.

[HOST:<tcp/ip addr or name:port:<App name>]

NOTE: App name must be same than specified in appList.txt APPLICATIONS section

Example:

HOST:127.0.0.1:500:PLAPP
HOST:DemoPC:1100:FAST_INVENTORY

12.4 Format of appList.txt

12.4.1 List of Remote host applications

[APPLICATIONS:<App_1_name>:<App_2_name>:<App_3_name>:<App_4_name>...]

- App_x_name must be same than name when creating remote host connection in connList.txt
- Up to 6 applications can be routed.
- Application names are printed in the main menu where the user can select.

Example:

APPLICATIONS:PLAPP:FAST_INVENTORY

12.4.2 Hand terminal access right list

[HS_ID:<ID nro>:<App number>]

- Hand terminal ID which only have access to the system.
- If no HS_ID entries, all HS have access to all applications.
- App number specifies access right to available applications. In same order than in APPLICATION section.
- If HS entries but no correct HS_ID in the list, then no response from AppRouter.

Example:

HS_ID:12345:1 *This HS can only use app1 (PLAPP)*
HS_ID:44444:12 *This HS can use app1 and app2 (PLAPP and FAST_INVENTORY)*
HS_ID:23022:2 *This HS can use only app2 (FAST_INVENTORY)*

12.4.3 Main menu item text

[MAIN_MENU_STRING:<"STRING">]

- When RF6xxAppRouter receives this string, the main menu of the applications will be opened.
- <STRING> can be any which comes from the hand terminal initial screen or from F-key as a plain string.

Example:

MAIN_MENU_STRING:F10
//When HS user sends F10, the main menu will be opened

12.4.4 Main menu header text

MAIN_MENU_HEADER:<"Header string">

- This string will be displayed in the main menu form as a header text. (Max 20 char)

Example:

MAIN_MENU_HEADER:* RF600 Demos *

12.4.5 Host down notification text

HOST_DOWN_TEXT:<"Host down text">

- Host down text will be displayed as a PopMessage in the hand terminal screen when connection to the specific host has been lost.

Example:

HOST_DOWN_TEXT: Host down! Try again later

13 Host application samples on the CD

The product manual CD contains the demo applications with source codes. The applications demonstrate different functionalities of the PLServer. The examples are made with VB6.0, VC++ 6.0, VB NET 2003 or VC++ 2003.

The source codes with working solutions are offered to study for the application developer.

Most of the applications offer three alternatives for connecting to the base station.

Choose:

- Serial Connection, if the base station is connected to the COM-port of the PC
- StartServer, if the base station is connected to the DeviceServer that is configured to client-mode. Type the port number and press "StartServer" –button. There has to be TCP/IP –address and -port of the PLServer computer configured in the Device server.
- Serial Server Connection, if the Device server is configured to host. Type the TCP/IP –address and –port in the Serial Server Connection –dialogue. This method allows to connect several Device server.

13.1 Forms sample

In the main menu of the Forms sample –application the user can choose the desired function by using the arrow keys. This main menu looks like typical main menu of the backend software.

Forms sample is a multithreaded application and it is created with Visual C++. The basic structure of this program can be used when creating the actual backend software.

13.1.1 Calculator

This is simple two input calculator. The user input value1 and value2 with the keyboard and then chooses the operand below the values. The result is shown after the “=” –character.

This demonstrates different types of input fields. Value1 and Value2 input fields does not send data when the user presses the ok-key.

When the user chooses the operand by activating it and pressing the ok-key, the input text of the Value1 and Value 2 input fields are sent to the backend software.

The operation buttons (+,-,x and /) are created by using the NewFieldEx and FldTxt –methods with SND_ALL flag ON. This is because of the content of the Value1 and Value2 –input fields has to be sent simultaneously with the operand to the host.

13.1.2 Auto Scanning

The backend software can ”press” the button in the hand terminal. By using the SpecialCmd –method it is able to define the key which is ”pressed” immediately after the hand terminal has received response from the backend software.

The example application is meant for user to read barcodes continuously and quick. When the application receives the code it immediately sends command which activates the barcode reader again. The user doesn’t need to press scan-key to activate scanner.

The READER_DEFAULT flag of the NewFieldEx -method allows activating the barcode scanner even if the input field is not active. E.g. if the “Start” button in the application is active, the reader activates with the scan-key.

After the hand terminal has read the barcode, it is sent to the application. The application reply activates the barcode scanner without a user entry.

If the barcode scanner is held in front of the barcode, it is read continuously until the barcode is removed or the user presses any key.

The number of successful scans is also displayed.

Note: If more than one hand terminal is using the same form simultaneously, the displayed number is sum of the all hand terminals scans.

13.1.3 User authorization form

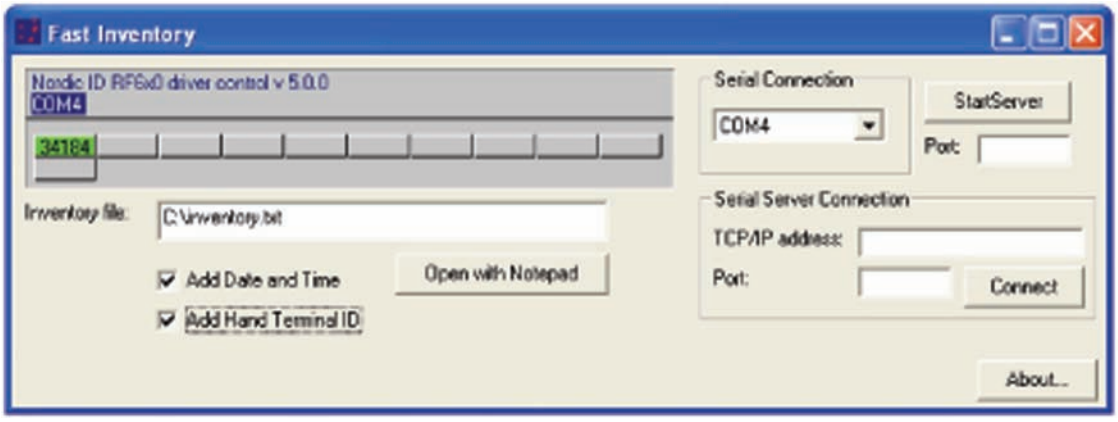
This form is example of input fields that are defined with SpecialCmd –method. The characters of the password are displayed as * -characters in the hand terminal display.

After the UserID and Password –fields are sent to the host, it replies by using the PopMessage and displays the actual content of the input fields, clears the input fields and sets the UserID –field active by using the FieldCmd –method.

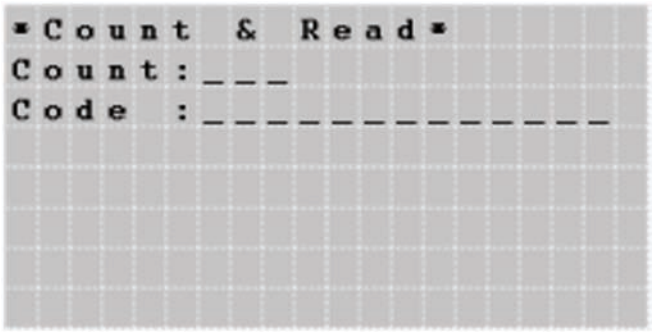
13.2 FastInventory

The Fast Inventory is a simple application for collecting data to the specific text file using the RF6xx hand terminal.

Screenshot of Fast Inventory Application:



When the hand terminal connects first time to the FastInventory program, following screen appears in to the hand terminal screen:



The hand terminal user can type count and scan code of product: After scanning the code, data will be transmitted to the host and save to the specific text file. Fields are separated by semicolon (;) for example: (code; count; date; hand terminal ID)

023942874102; 15; 20.9.1999 9:07:55; 3564

Item fields are separated by semicolon which is easy to import to Excel, Access, etc.

This information at the input fields is sent to the backend software after successful barcode reading or OK-key entry when Code-field is active.

READER_DEFAULT –flag is set to the Code –field. This means that the information of the barcode scanning is displayed to the Code –field even if it is not active.

The FastInventory example application is made with Visual Basic 6.0.

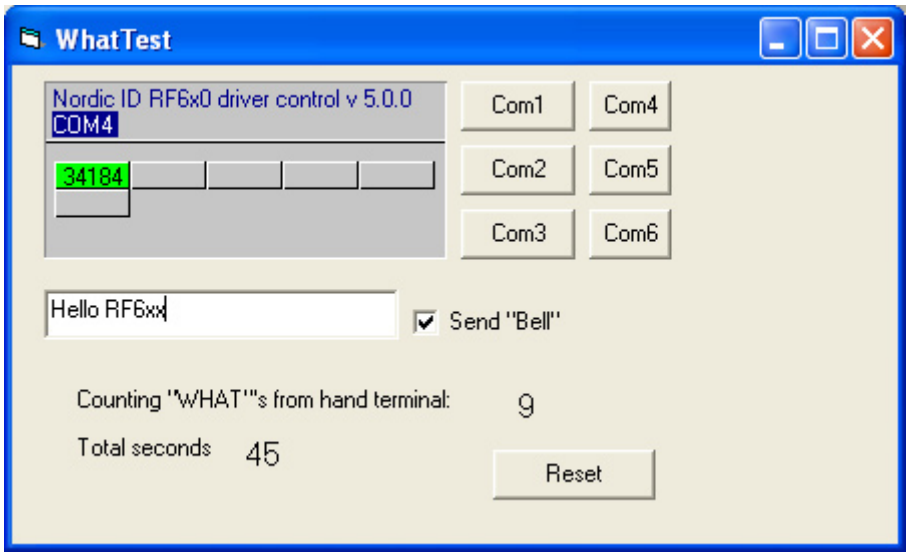
13.3 WhatTest

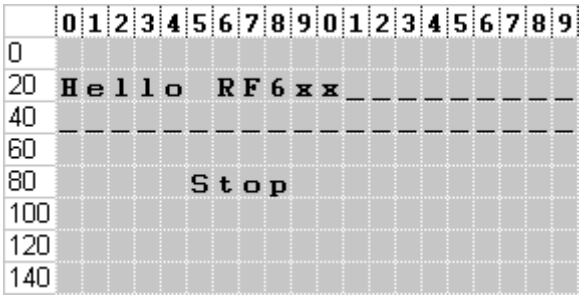
This is a simple What –method example. By using the What –method the hand terminal is configured to send What –event in 5 second intervals. The response is the content of the edit –box and is sent to the input field on the hand terminals display.

The hand terminal enters to the sleep mode after 30 seconds of inactivity. However it keeps on sending the What –event as in normal mode.

If the Send Bell is checked at the display of the hand terminal, it signals with a beep and turns on the display while on sleep.

Note: Set the ”What behavior” to 0 (New style) at the settings menu of the hand terminal.

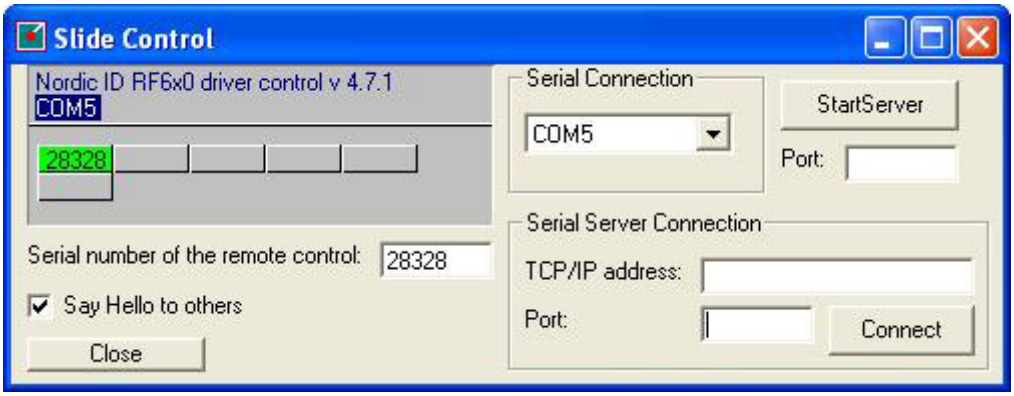


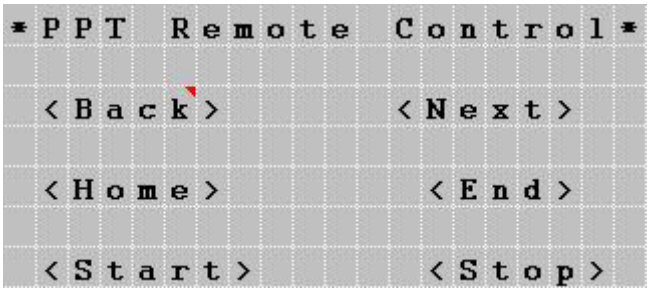


WhatTest is a Visual Basic 6.0 application.

13.4 SlideControl

The Slide control application makes hand terminal to work as a remote control of the power point presentation program.





Only one hand terminal can control the PPT slides at the time.
The hand terminal serial number must be in the “Serial number of the remote control:” edit box.

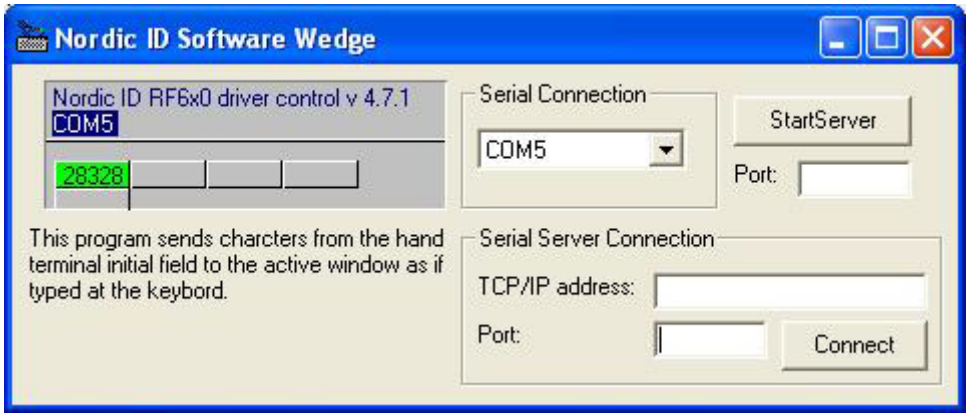
Note: Slide control program generates key press events to the operating system.
Therefore, all keystrokes from hand terminal (Back, next...) app goes always to the currently active application.

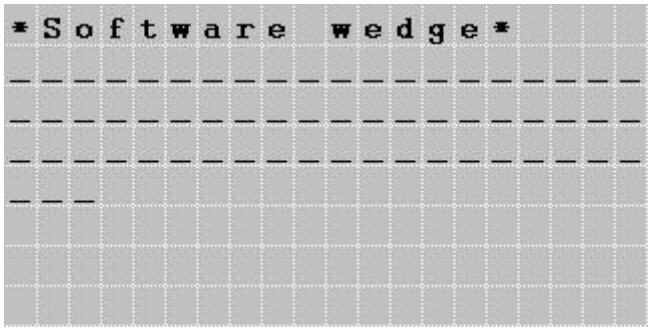
Setting up Slide Control:

1. Open Slide Control program
2. Connect to the Base station
3. Type hand terminal serial number to “Serial number of the remote control” edit box.
4. Minimize Slide control
5. Activate PowerPoint program
6. Start using Slide Control with the hand terminal.

13.5 Software wedge

Software wedge program receives data from the hand terminal and sends characters to the active window as if typed at the keyboard. Up to 63 characters can be typed or read with scanner to the input field of the hand terminal.





Data will be sent to the host after scanning barcode or pressing OK key.
Line feed are added automatically to the end of the scanned code.

Setting up Software Wedge:

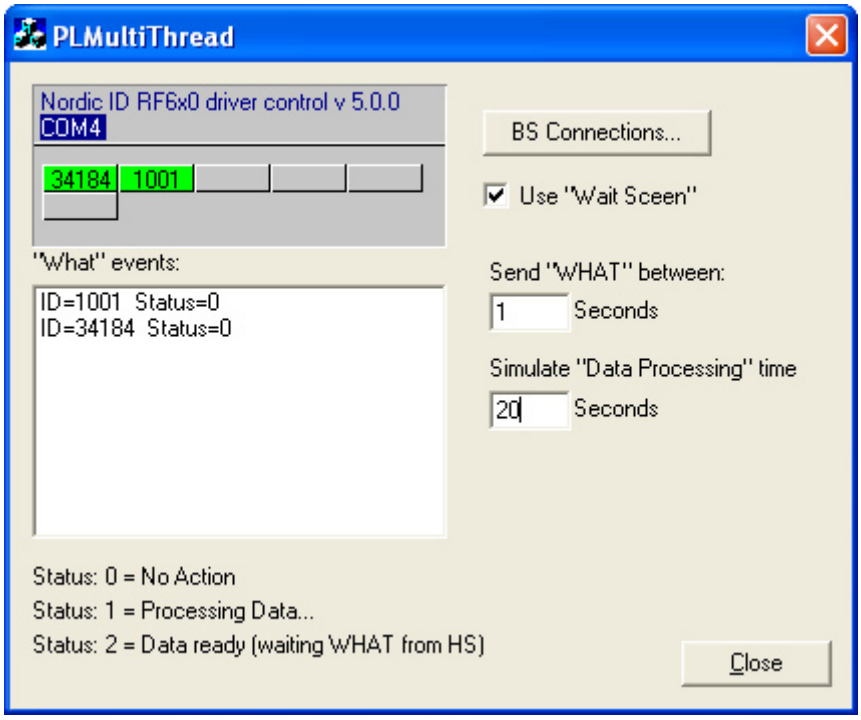
1. Open Software Wedge program
2. Connect to the Base station
3. Minimize Software Wedge
4. Open and activate any program which wanted to receive keystrokes. (word, excel, text editor...)
5. Start using Software wedge with the hand terminal.

13.6 PLMultiThread

The PLMultithread is a multithreaded application that is made with Visual C++ 6.0. It demonstrates the long data handling times and the use of the What –method.

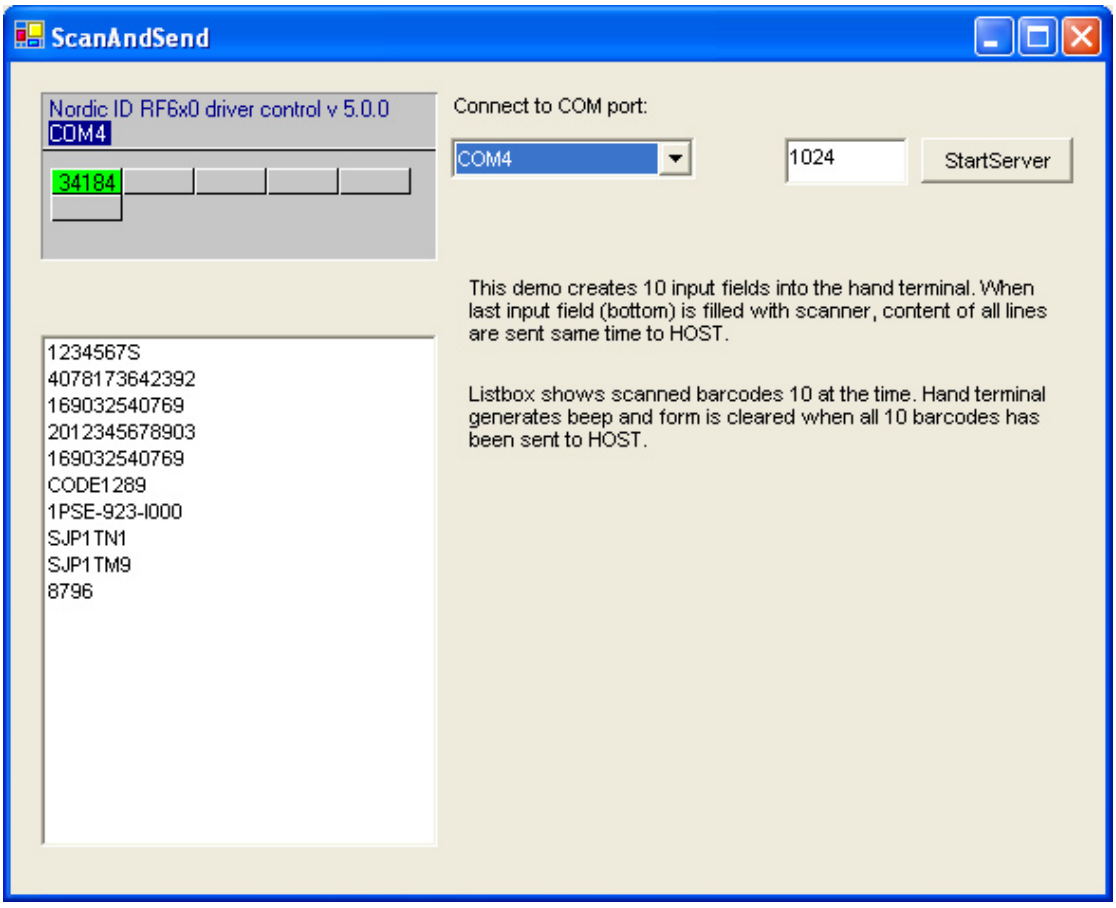
Note: Set the ”What behavior” as New style

The handling of the PLServers DataArrived- and What -events are at the PLMultiThreadDlg.cpp –file. The actual data from the hand terminal is handled in the other thread (DataProcessing.cpp).



13.7 ScanAndSend

This demo software creates 10 input fields in the hand terminal display. The data at the input fields is sent to the backend software after the input to the last input field. This sends all the data in the input fields to the backend software which displays it on the Listbox of the ScanAndSend –application. After receiving the data the backend software clears the hand terminal screen and sets the view to the top of the form by using the SetView(0)-method.



ScanAndSend is programmed by using Visual Basic.NET

14 APPENDIX A PLServer methods and events

The Nordic ID RF6xx driver control is a 32-bit Windows ActiveX control (PLServer.ocx), which controls the data communication between the base station and the HOST application.

PLServer makes HOST system software development easy because it does all the dirty work for you. The use of PLServer provides a lot of functionality and code that you don't have to write or debug. All you have to do is to figure out the entry points and how to use them.

You can use PLServer control in applications that are developed with Visual Basic , Visual C++, Excel, Access , FoxPro, Delphi and many more products who support ActiveX controls.

Note: There are some undocumented methods that are not useful for software developers. It is recommended to use only methods documented in this manual.

14.1 Base station connection methods

14.1.1 Connect

Serial port connection for the RF600 base station.

BOOL Connect(short com_port)**:**	
Parameters:	
<com_port>	COM port number of the HOST PC-
Example:	

Connect(1) //Connect to COM 1
Connect(0) //Disconnect	

Return value:	
True	if connected successfully
False	if port not exist or port is already in use.

14.1.2 StartServer

Starts to listen client connections to the specified TCP/IP port

BOOL StartServer(short port)**:**
Parameters:
<port> Port number where Clients connects
Example:
StartServer(1024) //Starting server and allow connections to 1024
Return value:
True If server started succesfully **False** If the server cannot be started
See also
StopServer

14.1.3 StopServer

Disconnect all client connections.
This is done automatically when host application closed.

void StopServer()**:**

14.1.4 ConnectToSerialServer

Create a network session between device server and PLServer. Up to 255 server connections can be made simultaneously. This method starts connection procedure and returns immediately. If network adapter is not available, PLServer tries to reconnect in to it automatically between 10 seconds until DisconnectSerialServer has been called or application closes.

short ConnectToSerialServer(LPCTSTR addr, short port)**:**

Parameters

addr	String value of TCP/IP address of the device server.
port	Port number of the network adapter (short value)

Example

retval = ConnectToSerialServer(“192.168.0.1”, 7001);
if (retval == -1) <Cannot connect to serial server>
else <connection succesful. Save retval as a session number>

Return value

Return value indicates the session number.
If -1, All 255 servers have been used.

See also

DisconnectSerialServer
SerialServerMessage (event)

14.1.5 DisconnectSerialServer

Disconnect the network adapter session.
When application closes, all TCP/IP connections are closed automatically.

void DisconnectSerialServer (**short** ss_id)

Parameters

ss_id	Session number returned by **ConnectToSerialServer** method

See also

ConnectToSerialServer
SerialServerMessage (event)

14.1.6 GetSerialSvrStatus

Return current status of the connection.

short GetSerialSvrStatus (**short** connID)	
Parameters	
connID	Session number returned by **ConnectToSerialServer** method
Return value	
0	Not in use
1	Trying connect
2	Connected
See also	
ConnectToSerialServer	
SerialServerMessage (event)	

14.1.7 SerialServerMessage (event)

PLServer launches SerialServerMessage event which includes text information about the specific session.
PLServer uses Windows sockets and therefore *msg* might contain error codes of windows sockets. See Appendix C of Windows socket error codes.

(event) SerialServerMessage(short ss_id, LPCTSTR msg)**:**	
Parameters: (given by PLServer)	
ss_id	Session number of the connection. Returned by **ConnectToSerialServer** method
msg	String buffer which contains the text description about the message.
Return value	
Return value indicates the session number. If -1, All 255 servers has been used.	
See also	
ConnectToSerialServer ( )	
DisconnectSerialServer( )	

14.2 User Interface elements

14.2.1 Text and TextEx

Define a text string to specific position in to the hand terminal screen.
Note: Text and TextEx methods are same. Use TextEx instead of Text in VB.NET applications.

BOOL TextEx (**long** id, **short** pos, **LPCTSTR** txt)

Parameters	
id	Hand terminal commID number
pos	Start position of the text. (0 - 239)
txt	Text string to be displayed.
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

18.2.2 Button

Create a “button” style field. The Button method will create a locked field filled with text. Field text will be sent to the HOST by pressing the OK key. This method is suitable for menu creation.

SND_ALL and NO_SEND functions are disabled. (Please see **NewField**() method) To create a button field with SND_ALL and NO_SEND functions, use the **NewField**() and **FldTxt**() methods instead of the **Button**() method.

BOOL Button (**long** id, **short** pos, **LPCTSTR** txt)

Parameters	
id	Hand terminal commID number
pos	Start position of the button. (0 - 239)
txt	Text for the button.
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.2.3 NewField

Define a new Input field in to the hand terminal screen.
A maximum of 20 input fields can be defined for one form.

BOOL NewField (**long** id, **short** pos, **short** field_len, **short** style)

Parameters		
id	Hand terminal commID number	
pos	Start position of the input field. (0 - 239)	
field_len	Field length	
style	Style bits of the input field.	
Bit 0	SND_ENTER	Field is sent to HOST by pressing the OK key
Bit 1	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, if the field itself gave the SND_ALL command.
Bit 2	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the OK key is pressed.
Bit 3	FLD_LOCK	Field is locked. Field cannot be written to (used for “button” style fields.)
Bit 4	FLD_LINE	Field is underlined. _ _ _ _ _ _ _ _ _
Bit 5	FLD_READER	Field can be filled with laser- or external scanner data.
Bit 6	FLD_ACKCLR	Field is cleared if ACK is received.
Bit 7	FLD_ACTIVE	Field is active (it has a cursor).

SND_ENTER and FLD_READER combinations:		
SND_ENTER	FLD_READER	FUNCTION
0	**0**	Field is not sent by pressing the OK key, and cannot be read with a laser scanner.
0	**1**	Field is not sent by pressing the OK key, but it can be filled from the keyboard and with a laser scanner.
1	**0**	Field is sent by pressing the OK key, but cannot be filled with a laser scanner.
1	**1**	Field is sent by pressing the OK key or by reading with a laser scanner.

Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.2.4 NewFieldEx

Define a new Input field in to the hand terminal screen. NewFieldEx method is same as NewField() but style bit's OVR and READER_DEFAULT are replaced bit's FLD_LINE and FLD_ACKCLR. Field is underlined automatically. A maximum of 20 input fields can be defined for one form.

BOOL NewFieldEx (**long** id, **short** pos, **short** field_len, **short** style)

Parameters		
id	Hand terminal commID number	
pos	Start position of the input field. (0 - 239)	
field_len	Field length	
style	Style bits of the input field.	
Bit 0	SND_ENTER	Field is sent to HOST by pressing the OK key
Bit 1	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, if the field itself gave the SND_ALL command.
Bit 2	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the OK key is pressed.
Bit 3	Reserved	Reserved set as "0"
Bit 4	OVR	Overwrite mode. When OVR is set, and the field becomes active, the cursor moves to the starting position of the field. When the cursor is in the starting position, the previous text will be overwritten by typing.
Bit 5	FLD_READER	Field can be filled with laser- or external scanner data.
Bit 6	READER_DEFAULT	If another field is active and it has not FLD_READER set, laser scanner is activated and data goes to this field automatically. Only one READER_DEFAULT field can be defined per form. This bit has no effect if FLD_READER bit is not set.
Bit 7	FLD_ACTIVE	Field is active (it has a cursor).
SND_ENTER and FLD_READER combinations:		
SND_ENTER	FLD_READER	FUNCTION
0	**0**	Field is not sent by pressing the OK key, and cannot be read with a laser scanner.
0	**1**	Field is not sent by pressing the OK key, but it can be filled from the keyboard and with a laser scanner.
1	**0**	Field is sent by pressing the OK key, but cannot be filled with a laser scanner.
1	**1**	Field is sent by pressing the OK key or by reading with a laser scanner.
Return value		
True	Command added to the send buffer successfully	
False	Send buffer is full	

14.2.5 PopMessage

Define a message which is displayed in the hand terminal screen as long as any key is pressed. Only one PopMessage method call can be included in the send buffer.

BOOL PopMessage (**long** id, **short** pos, **LPCTSTR** txt)

Parameters	
id	Hand terminal commID number
pos	Start position of the button. (0 - 79)
txt	Text for the screen. (May include VT and CR characters.)
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.2.6 Ack

This method adds the Ack command to the send buffer. Ack method can be used at NewField command. (Please see style bits).

BOOL Ack (**long** id)

Parameters	
id	Hand terminal commID number
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full
See also:	
NewField	

14.2.7 SpecialCmd

SpecialCmd defines special functionalities to the hand terminal.

BOOL SpecialCmd (**long** id, **short** type, **long** data)	
Parameters	
id	Hand terminal commID number
Type	
Bit 0	If this bit is set, the hand terminal will emulate the one key press. The key is defined in parameter data **SpecialCmd(12000, 0x1 , 0x06)** emulates a BAR button press.
Bit 1	If this bit is set, the hand terminal sends keystrokes directly to the HOST. The HOST application can define the keys with parameter data that sends the keystrokes to the HOST. Example: UP and DOWN keys will send keystokes to the HOST: **SpecialCmd(12000, 0x2 , 0x600)** This function is valid as long as user resets the device or HOST send **SpecialCmd** where Bit 1 is not set. When the user presses the key which is defined to send keystroke, the hand terminal sends SPC_MSG to the HOST and data can be read by **GetSpecialData** method. Appropriate key value is returned. (see list below). After that the hand terminal will wait a response from the HOST.
Bit 2	Password style for input field. Three input fields can be defined as a password style field. The Hand terminal user can only see '*' asterix chars when typing to the input field. Example: Input fields in position 20 (hex 14) and 40 (hex 28) will be defined as a password style field. **SpecialCmd(12000,0x4,0x00FF1428)**
Bit 3	Reserved
Bit 4	Reserved
Bit 5	Reserved
Bit 6	Reserved
Bit 7	Reserved

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	
Type	Emulates the key press	Hand terminal sends keystrokes to the HOST	Password style input fields					
Data	Key value (decimal) F1 = 1 F2 = 2 F3 = 3 F4 = 4 F5 = 5 BAR = 6 DEL = 7 DOT = 8 MINUS = 9 UP = 10 DOWN = 11 OK = 12 0 = 13 1 = 14 2 = 15 3 = 16 4 = 17 5 = 18 6 = 19 7 = 20 8 = 21 9 = 22	Key (return value) See bits below:	Positions of thepassword input fields. 255 = not defined					
Bit 0		F1 (1)	First Password input field position. 0xff = not defined					
Bit 1		F2 (2)						
Bit 2		F3 (3)						
Bit 3		F4 (4)						
Bit 4		F5 (5)						
Bit 5		BAR (6)						
Bit 6		DEL (7)						
Bit 7		DOT (8)						

Bit 8		MINUS (9)	Second Password input field position 0xff = not defined	
Bit 9		UP (10)		
Bit 10		DOWN (11)		
Bit 11		OK (12)		
Bit 12		0 (13)		
Bit 13		1 (14)		
Bit 14		2 (15)		
Bit 15		3 (16)		
Bit 16		4 (17)	Third Password input field position. 0xff = not defined	
Bit 17		5 (18)		
Bit 18		6 (19)		
Bit 19		7 (20)		
Bit 20		8 (21)		
Bit 21		9 (22)		
Bit 22		x		
Bit 23		x		
Bit 24		x		
Bit 25		x		
Bit 26		x		
Bit 27		x		
Bit 28		x		
Bit 29		x		
Bit 30		x		
Bit 31		x		
x = not used				

Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

See also

GetSpecialData

14.2.8 GetSpecialData

This method reads special data sent by the hand terminal.

short GetSpecialData (**long** id, **short** type)	
Parameters	
id	Hand terminal commID number
type	Type of the special data.
Return value	
Value (0- 255) from the hand terminal from specific type (operation). -1 if there is no data received from specific type.	
See also:	
SpecialCmd	

14.3 Form and UI-element commands

14.3.1 ClearForm

Clears all existing texts and fields from the display and sets view to row 0.

BOOL ClearForm (**long** id)	
Parameters	
id	Hand terminal commID number
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.3.2 ClearCmd

Partially clears the display of any text or field

BOOL ClearCmd(long id, **short** cmd, **short** start, **short** stop)

Parameters			
id	Hand terminal commID number		
cmd	**Bit 0**	CLEAR_TXT	Removes text from <**start**> to <**stop**>
	Bit 1	CLEAR_FLD	Removes fields from <**start**> to <**stop**>
	Bit 2	CLEAR_FLDDATA	Clears the field data from <**start**> to <**stop**>
	Bit 3-7	Reserved	
start	start position of the display (0-239)		
stop	stop position of the display (0-239)		
Return value			
True	Command added to the send buffer successfully		
False	Send buffer is full		

14.3.3 FieldCmd

Changing function of existing input filed

BOOL FieldCmd(long id, **short** pos, **short** fcmd)

Parameters			
id	Hand terminal commID number		
pos	Position of the input filed		
fcmd	**Bit 0**	FLD_ REMOVE	Removes the field
	Bit 1	FLD_CLEAR	Clears the field (from locked fields also)
	Bit 2	Reserved	
	Bit 3	FLD_LOCK	Locks the field (cannot be written to)
	Bit 4	Reserved	
	Bit 5	Reserved	
	Bit 6	Reserved	
	Bit 7	FLD_ACTIVE	Field is set to active.
Return value			
True	Command added to the send buffer successfully		
False	Send buffer is full		

14.3.4 FldTxt

Fill (presets) the field with the text. If the length of txt is more than the length of the field, the excess txt will be corrupted. If the field is already filled with text, it will be replaced with new text.

BOOL FldTxt (**long** id, **short** pos, **LPCTSTR** txt)

Parameters	
id	Hand terminal commID number
pos	Position of the input field. (0 - 239)
txt	Text for the field.
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.3.5 SetView

Sets the display view to be started from specific line. This method may need to use in situations where hand terminal screen has been scrolled downwards because only 8 lines are visible in the display at the time and virtual form size is 12 lines.

BOOL SetView (**long** id, **short** row)

Parameters	
id	Hand terminal commID number
row	View line number (0-3). If this value is greater than 3, then this method will be ignored by the hand terminal.
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.3.6 SetFormID

This method defines a specific ID number which describes the form. This method is useful if the HOST application uses several different forms on the hand terminal.
Up to 65535 different form ID:s can be defined.

The formID value is stored in the Hand Terminal memory. The Hand Terminal sends this value every time to the HOST.

If formID is defined to 0 or Hand Terminal is resettled, then Hand Terminal doesn't send formID value to the HOST.

Other technique for identifying different forms is frameid which is defined in Send method. DataArrived event passes frameid value to the application. Advantage of using SetFormID instead of frameid is that Function key press actions can be connected to the specific form.

In the HOST application, this method can be called just before Send method.

FORM_ID is not sent by the Hand Terminal if:

- FORM_ID is defined to 0
- Hand Terminal is resettled

BOOL SetFormID (**long** id, **short** formID)	
Parameters	
id	Hand terminal commID number
formID	User defined value 0 - 65535. 0 = not used
Return value	
True **False**	Command added to the send buffer successfully Send buffer is full
See also	
GetformID	

14.3.7 GetFormID

If form ID has been defined by SetFormID method and after the data has sent from the hand terminal, the HOST application can read received form ID using this method and then application can determine the actions of data.
In the HOST application, this method can be called in handle of DataArrived event.

FORM_ID is not sent by the Hand Terminal if:

- FORM_ID is defined to 0
- Hand Terminal is resettled

short GetFormID (**long** id)	
Parameters	
id	Hand terminal commID number
formID	User defined value 0 - 65535. 0 = not used
Return value	
The number of the form defined by **SetFormID** method.	
See also	
SetformID	

14.3.8 What

The hand terminal sends WHAT command (18h) to HOST every time between specified delay seconds.

BOOL What (**long** id, **short** delay)	
Parameters	
id	Hand terminal commID number
delay	0 – 99 seconds If DELAY is 0, The hand terminal stops sending “WHAT” commands
Return value	
True **False**	Command added to the send buffer successfully Send buffer is full
See also	
“What” event	

"What" functionality:
(HS=Hand Set)
When the host gives "WHAT" command to the HS:

HS will send "What" code every time between specified time.
For example, host command What (5); set HS to send "WHAT" code between 5 second. "What" event fires in PLServer every time. When Host gives What (0), then HS will stop sending "what" code.

Hand terminal behavior when WHAT code is sent.

Hand terminal waits answer from host after WHAT code is sent. Wait time is same as "Reception time limit" parameter. Default 1 sec.

If no answer from host within Reception time limit, WHAT code resends are made as many times specified at "Resending time" parameter. Default 3 times.

If all resends are used and still no answer from host, HS will not generate transaction failure beeps and "F" sign. Then "WHAT" code is sent to the host next time until x seconds is expired.

Note:
The What -method in Nordic ID RF-seriex can be set to work in two different ways. The "old style" RF6xx käsipäätteissä on What metodille kaksi eri toiminta tapaa. Aikaisemmin What metodi toimi siten että käsipääte lähetti What:n vain kerran määrätynajan kuluessa. Nyt uudella toimintatavalla, käsipääte lähetää What:n aina määrätynajan välein kunnes tausta pysäyttää What(0) komennolla.

Käsipäätteistä voidaan valita toiminta tapa sisäisestä valkiosta:
The behaviour of "What" can be set from the build-in menu of the hand-terminal:
Settings→ "What" Behavior

(0 = New style)
(1 = Old style)

14.3.9 What (event)

"What" event is fired when hand terminal sends "What" code.

(event) What(long id)

Parameters (given by PLServer)	
id	Hand terminal commID

See also
What ( ) method

14.3.10 Send

This method will send the defined commands to the Hand Terminal.

BOOL Send (long id, short frameID)

Parameters	
id	Hand terminal commID number
frameID	The number of form defined by the user. If frameid is -1 , the previously defined frameid is valid.
Return value	
True	Data are beginning to send to the hand terminal.
False	Data is already under sending.

14.3.11 DataArrived (event)

PLServer launches this event when receiving data from the hand terminal. Host application handles received data and sends an answer back to the hand terminal with Send( ) method.

When DataArrived event is handling hand terminal data, PLServer cannot launch any other DataArrived event at this time.

Therefore, it's recommended that Host application creates a separate thread to handle hand terminal data and release DataArrived handling as fast as possible. Multithread application is not necessary application gives fast response and only few hand terminals are used at same time.

(event) DataArrived(long id, short frameid)	
Parameters (given by PLServer)	
id	Hand terminal commID
frameid	The number of form defined by the user. If the frameid is -1, the data comes from the initial screen or from a function key. This value is defined in **Send**( ) method.
See also	
Send( ) method	

14.3.12 DataFromHsField (event)

This event offers a string from a specific Hand Terminal, from a specific form and from a specific position. DataFromHsField is launched as many times as there is input field contents received from the hand terminal.

After all DataFromHsField handles, DataArrived event is launched.

Note that this event is launched only if SetNotify (TRUE) is called.

Note: For the integrity of the backend system it is recommended to use only DataArriver –event for data handling.

(event) DataFromHsField(long id, short frameid, short pos, LPCTSTR data, short data_len)	
Parameters (given by PLServer)	
id	Hand terminal commID
frameid	The number of form defined by the user. If the frameid is -1, the data comes from the initial screen or from a function key. This value is defined in **Send**( ) method.
Pos	Position of the input field. (0 - 239)
data	Text string from the input field
data_len	Length of the string in bytes
See also	
SetNotify( )	
(event) DataArrived( )	

14.3.13 SetNotify

SetNotify allows DataFromHsField event to launch when hand terminal sends an input field data.

void SetNotify (**BOOL** bNewValue)

Parameters	
bNewValue	TRUE allows DataFromHsField to launch.

See also

(event) DataFromHsField( )

14.4 Sound methods

14.4.1 Bell

Generates a single beep. When hand terminal receives message from host it will beep as default. Adding Bell method will give extra beep.

BOOL Bell (**long** id)

Parameters	
id	Hand terminal commID number

See also

Beep

Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.4.2 Beep

Generate a specific length of beep and delay.

BOOL Beep (**long** id, **short** length, **short** delay)

Parameters	
id	Hand terminal commID number
length	Beeper is ON 10 x length milliseconds. (eg. value 100 = 1 seconds)
delay	Beeper is OFF 10 x delay milliseconds.

Example

```
“S.O.S tone”
Beep(12000, 5, 10); //Beep 50ms ON and 100 ms OFF
Beep(12000, 5, 10); // “S”
Beep(12000, 5, 30);
Beep(12000, 15, 10); // 150ms ON and 100ms OFF
Beep(12000, 15, 10); // “O”
Beep(12000, 15, 30);
Beep(12000, 5, 10);
Beep(12000, 5, 10); // “S”
Beep(12000, 5, 10);
```

Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

14.5 Data receiving

14.5.1 GetData

Return a string from the specific input field position (0-239)

BSTR GetData (**long** id, **short** pos)	
Parameters	
id	Hand terminal commID number
pos	Position of the input field (0-239)
Return value	
Basic String. A pointer to a null-terminated Unicode character array that is preceded by a 4-byte length field.	
See also:	
IsData	

14.5.2 IsData

This method will return TRUE if data is coming from ***pos***.

BOOL IsData (**long** id, **short** pos)	
Parameters	
id	Hand terminal commID number
pos	Position of the input field (0-239)
Return value	
True	Data coming from pos field.
False	No data coming from pos field.
See also:	
GetData	

14.5.3 GetExtraID

This method returns Extra ID of the Hand Terminal. Extra ID value can be defined in to the hand terminals

short GetExtraID (**long** id)

Parameters

id	Hand terminal commID number

Return value

Extra ID number (0 - 255)

14.5.4 GetMessageNumber

This method returns Message Number of the Hand Terminal.

short GetMessageNumber (**long** id)

Parameters

id	Hand terminal commID number

Return value

Message number (0 - 15)

14.5.5 GetRSSI

This method returns received signal strength of message.

short GetRSSI (**long** id)

Parameters

id	Hand terminal commID number

Return value

RSSI level of the last received message. (70 - 140)

14.5.6 GetLastFrameID

This method returns frame id from the last received message.

short GetLastFrameID (**long** id)

Parameters	
id	Hand terminal commID number
Return value	
This method returns last sended frameid [Send() method parameter 2].	

14.5.7 GetReceiveBuffer

This method allows HOST application to get raw data frame sent by hand terminal.

short GetReceiveBuffer (**long** id, **short** FAR* data)

Parameters	
id	Hand terminal commID number
data	Data frame from the hand terminal
Return value	
Length of the *data* in bytes.	
See also:	
RF6xx communication protocol	

14.5.8 GetBatteryLevel

This method allows HOST application to monitor battery level of the hand terminal. Example, the HOST application can generate the warning messages to the hand terminal if battery level is too low.

short GetBatteryLevel(**long** id)

Parameters	
id	Hand terminal commID number
data	Data frame from the hand terminal
Return value	
Battery level in %. (8 fixed levels: 0,10,20,40,50,70,80,100)	

14.5.9 GetCRCValue

Return CRC value of the last received message.

long GetCRCValue(long id)
Parameters
id | Hand terminal commID number
Return value
Long value of CRC (0-65535)

14.5.10 GetHsIdString

Return the hand terminal commID value as a string format. Can be used for indicating is hand terminal communicated or not since application startup.

BSTR GetHsIdString(long id)
Parameters
id | Hand terminal commID number
Return value
String value of commID. Empty string if HS is not communicated with the application.

14.5.11 GetSourceIPAddr

Return the IP address of the network source which has received last message from the hand terminal. String format like: “172.16.32.17”

BSTR GetSourceIPAddr(long id)
Parameters
id | Hand terminal commID number
Return value
String value of source IP address

14.6 Receiver mode

14.6.1 Receiver

This command makes the hand terminal to listen transmissions continuously. The RECEIVER function can be set ON/OFF by keyboard as well (SHIFT+F1).

After a hand terminal has received the message and the HOST has sent it independently, and when RECEIVER is ON, the hand terminal answers to the HOST by RECEIVER_ACK command (1Ch). In practice, the commands to the hand terminal are PopMessages or Bells. Sending other commands can disturb the already existing works in the hand terminals. This is not however prevented, so the HOST application has a responsibility for sending the commands.

When the RECEIVER is ON, PopMessages will be acknowledged as read by a DEL key. This prevents the accidentally key press before reading the PopMessage.

PLServer is not able to know if the RECEIVER is switched ON/OFF by the user in the hand terminal.

Note: WaitSerial and What methods cannot be used same time with this method.

Note:
Problem of this method is higher battery consumption and possibility of data loss. Also host application developing might be quite complicate.
Consider to use “What” method instead of “Receive” method.

BOOL Receiver (**long** id, **short** mode)	
Parameters	
id	Hand terminal commID number
mode	
0	RECEIVER mode OFF
1	Receiver cycle of listening messages from the HOST. Radio is 1 sec on and 2 sec off to save battery power. **RECEIVER_ACK is sent** to the HOST after receiving message. When sending messages with **SendMessage** method then parameter *resendTimes* should be **> 0.**
2	Receiver cycle of listening messages from the HOST. Radio is 2 sec on and 1 sec off to save battery power. **RECEIVER_ ACK is not sent** to the HOST after receiving message. When sending messages with **SendMessage** method then parameter *resendTimes* should be **0.**
3	Listening **Continuously** messages from HOST. **RECEIVER_ ACK is sent** to the HOST after receiving message. When sending messages with **SendMessage** method then parameter *resendTimes* should be **> 0.**
4	Listening **Continuously** messages from HOST. **RECEIVER_ ACK is not** sent to the HOST after receiving message. When sending messages with **SendMessage** method then parameter *resendTimes* should be **0.**
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full
See also:	
"What", IsReceiver, SendMsg,MessageReceiverNotFound (event)	

14.6.2 IsReceiver

Determines if RECEIVER mode is ON or OFF in the hand terminal

BOOL IsReceiver(**long** id)	
Parameters	
id	Hand terminal commID number
Return value	
True	Receiver mode has been set on the hand terminal.
False	Not Receiver mode set
See also	
Receiver ( )	
SendMsg( )	
MessageReceiverNotFound( )	

14.6.3 SendMsg

Similar to Send method. Application can sends messages to the specific hand terminal where receiver mode is ON.

BOOL SendMsg(**long** receiverID, **long** senderID, **short** frameID, **short** resendTimes)

Parameters	
receiverID	Hand terminal commID, which receives the messages.
senderID	User defined ID, which describes the sender.
frameID	The number of form defined by the user. Id frameID =-1, the previously defined frameID
resendTimes	Defines how many ResendTimes cycles the message is sent to the hand terminal if no answer is received. If the PLServer cannot get RECEIVER_ACK from the hand terminal, and all the resendTimes have been used, a **MessageReceiverNotFound** event is launched. The cycle length is about 3 second. For example, if resendTimes is two and the hand terminal has not responded within 2*3 sec = 6sec, *MessageReceiverNotFound* event is launced.
Return value	
True	Returns always True

See also

Receiver ( )
IsReceiver( )
MessageReceiverNotFound( )

14.6.4 WaitReceiverAck

After message has been sent to the hand terminal using SendMsg, application can use WaitReceiverAck method to check is hand terminal received message or not.

Waiting time is max 2 second. This method returns immediately if ack received. Receiver mode 1 or 3 must be used.

BOOL WaitReceiverAck(**long** id)

Parameters	
id	Hand terminal commID number
Return value	
True	Ack received from the hand terminal
False	Ack not received

See also

SendMsg( )
MessageReceiverNotFound( )

14.6.5 MessageReceiverNotFound (event)

When sending messages with SendMessage method, PLServe waits RECEIVER_ACK from the hand terminal. If the RECEIVER_ACK does not occur within resendTimes*3 seconds, this event is launched.

(event) MessageReceiverNotFound(long receiverID, long senderID)

Parameters (given by PLServer)	
receiverID	Hand terminal commID, which does not response
senderID	Message sender ID defined by SendMsg method.

See also

Receiver ( )
IsReceiver( )
SendMsg( )

14.7 The hand terminal serial port methods

14.7.1 DataToSerial

Send text string to the serial port of the hand terminal.
Portable printer, card reader or any external device can be connected to the serial port of hand terminals.

Serial settings are fixed to: 19200, n, 8, 1

BOOL DataToSerial (long id, LPCTSTR txt)

Parameters	
id	Hand terminal commID number
txt	String to the hand terminal serial port.
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full

See also

WaitSerial ( )
GetSerialData( )
BinaryToSerial

14.7.2 BinaryToSerial

Send raw binary data to the serial port of the hand terminal.
Portable printer, card reader or any external device can be connected to the serial port of hand terminals.

Serial settings are fixed to: 19200, n, 8, 1

BOOL BinaryToSerial (**long** id, **short FAR*** buffer, **short** length)

Parameters	
id	Hand terminal commID number
buffer	Binary data (values 0-255)
length	Length of the data in bytes.
Return value	
True	Command added to the send buffer successfully.
False	Send buffer is full
See also	

WaitSerial ( )
DataToSerial( )
GetSerialData( )

14.7.3 WaitSerial

Open the serial port of the hand terminal for specific time. This command can be used to receive data from the external device such as card reader. Serial port of the hand terminal cannot be open all the time because current consumption then is higher.

Any data from the serial port goes directly to the active input field of the hand terminal. It depends on the style bits of the input field whether the data is sent to the HOST or not after reading from the serial port.

If delay value is 255, then the hand terminal sends serial port data directly to the HOST. The HOST can read data with GetSerialData( ) method. By default, the hand terminal keeps serial port open 30 seconds after receiving this command where delay value is 255.

Serial settings are fixed to: 19200, n, 8, 1

BOOL WaitSerial (**long** id, **int** delay)	
Parameters	
id	Hand terminal commID number
delay	time to keep serial port open (1-99) seconds 255 = Data goes directly to the host. Otherwise to the active input field
Return value	
True	Command added to the send buffer successfully
False	Send buffer is full
See also	
DataToSerial ( ) BinaryToSerial( ) GetSerialData( )	

14.7.4 GetSerialData

If the hand terminal serial port has been opened with WaitSerial method and delay parameter is 255, the hand terminal sends data from serial port directly to the HOST. The HOST can read received data to its buffer with this method.

Serial settings are fixed to: 19200, n, 8, 1

short GetSerialData (**long** id, **short FAR*** serdata)	
Parameters	
id	Hand terminal commID number
serdata	Data from the hand terminal serial port.
Return value	
length of the *serdata* in bytes	
See also	
WaitSerial ( ) DataToSerial( ) BinaryToSerial( )	

14.8 Raw data methods

14.8.1 SendRawData

Send raw binary data to the TCP/IP connection session specified by ConnectToSerialServer method.

short SendRawData (**long** connID, **short FAR*** rawData, **short** length)	
Parameters	
connID	Session number of the connection. (returned by **ConnectToSerialServer** method)
rawData	Binary data (values 0-255)
length	Length of the data in bytes.
Return value	
0	Sending failed. (connection not established)
1	Sending successful
See also	
RawDataArrived (event)	

14.8.2 RawDataArrived (event)

PLServer launches this event when receiving raw data from the TCP/IP connection.

(event) RawDataArrived(**long** connID, **short FAR*** data, **short** length)	
Parameters (given by PLServer)	
connID	Session number of the connection. (returned by **ConnectToSerialServer** method)
data	Raw Data from the TCP/IP connection
length	Length of the received data
See also	
SendRawData( ) method	

14.9 General methods

14.9.1 DataIn

The data of the base station can be delivered to the driver control by this method.
When the data from the base station have been delivered to the driver control using the method, the data to the base station will be offered by **DataOut** event.

void DataIn (**short FAR*** data, **short** data_len, **short** source_addr)

Parameters	
data	Data from the base station
data_len	data length
source_addr	User defined data source address
See also	
DataOut( )	

14.9.2 DataOut (event)

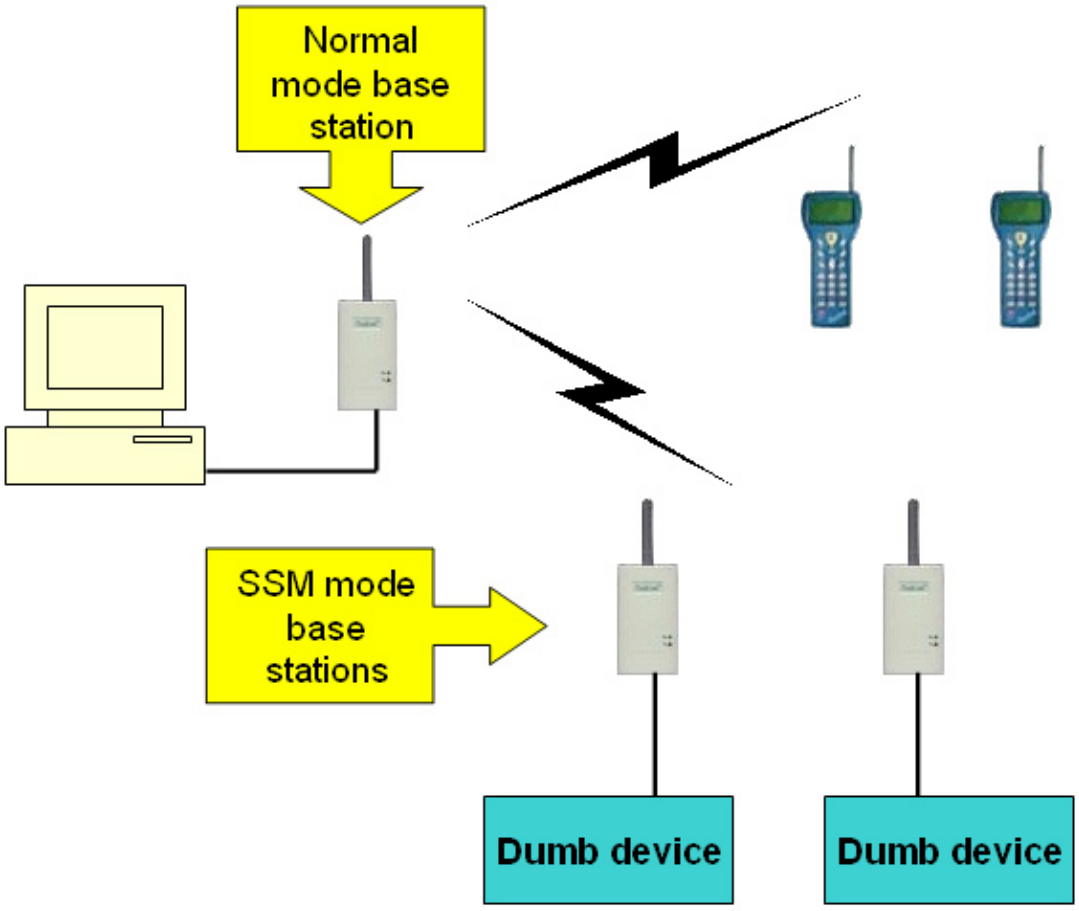
When the data from the basestation have been delivered to the driver control using the DataIn method, the data to the basestation will be offered by this event.

event DataOut (**short FAR*** data, **short** data_len, **short** dest_addr)

Parameters	
data	data to the base station
data_len	data length
source_addr	Destination address. Defined in **DataIn** method (source_addr).
See also	
DataIn( )	

14.10 Sub Station Mode methods (SSM)

SSM is the new operating mode to D05BS/2 type base stations. Firmware version 2.0 and later supports SSM mode. SSM functionality of the base station type D05BS/2 allows for asynchronous transfer of arbitrary data wirelessly between the host and the any device which is provided with the RS232 serial interface.



14.10.1 SendSSMData

This method will send data to SSM mode base station and waits acknowledgement from the SSM base station. Function wait acks from SSM 250*retries milliseconds.
The function returns when data ack received from SSM or the 250*retries milliseconds interval elapses.

BOOL SendSSMData (**long** ssmID, **short** msgNum, **short FAR*** data, **short** length, **short** retries)

Parameters	
ssmID	SSM base station ID number.
msgNum	SSM device requires different message numbers for each data. SSM device passes data to it's serialport only if *msgNumber* is changed.
data	data to the SSM base station. Allows byte values from 0-255
length	length of data buffer in bytes
retries	if no ack from SSM within 250 ms, then resend is made (max. *retries* times).
Return value	
TRUE	Data is sent to the SSM base station successfully.
FALSE	No acknowledgement from SSM base station.
See also	
WaitSSMData( )	
SendSSMAck( )	
event DataFromSSM( )	
SendSSMASCIIData ( )	

14.10.2 SendSSMASCIIData

This method works same as SendSSMData except data is in string format. It sends string to SSM mode base station and waits acknowledgement from the SSM base station. Function waits ack from SSM 250*retries milliseconds. The function returns when data ack received from SSM or the 250*retries milliseconds interval elapses.

BOOL SendSSMASCIIData (**long** ssmID, **short** msgNum, **LPCTSTR** dataString, **short** length, **short** retries)

Parameters	
ssmID	SSM base station ID number.
msgNum	SSM device requires different message numbers for each data. SSM device passes data to its serial port only if *msgNumber* is changed.
dataString	String to the SSM base station. Allows byte values from 32-255
length	length of data buffer in bytes
retries	If no ack from SSM within 250 ms, then resend is made (max. *retries* times).
Return value	
TRUE	Data is sent to the SSM base station successfully.
FALSE	No acknowledgement from SSM base station.
See also	
WaitSSMData( )	
SendSSMAck( )	
event DataFromSSM( )	
SendSSMData ( )	

14.10.3 WaitSSMData

This method waits data from SSM device at specific time
The function returns when data is received from SSM or the *ms_waitTime* interval elapses.
Note: After receiving data from SSM, host application is responsible to send acknoledgement to the SSM base station with SendSSMAck( ) method.

short WaitSSMData (**long** ssmID, **short FAR*** ssmData, **short** msWaitTime)

Parameters	
ssmID	SSM base station ID number.
ssmData	Data from SSM device.
msWaitTime	Time how long function waits data from the SSM device in milliseconds.
Return value	
-1	NO data from SSM base station (*msWaitTime* interval has elapsed).
>=0	Received length of *ssmData* in bytes.

See also
SendSSMAck( )
event DataFromSSM( )
SendSSMASCIIData ( )
SendSSMData ( )

14.10.4 SendSSMAck

Send an acknowledgement to the SSM base station.
When the SSM base station has sent a data to the host, it waits for an acknowledgement.

It is the host program's responsibility to send this ack after receiving data.
If the SSM base station doesn't receive ack, it will resend the frame as many times as defined in the SSM base station (typically 3)

void SendSSMAck (**long** ssmID)

Parameters	
ssmID	SSM base station ID number.

See also
WaitSSMData( )
event DataFromSSM( )
SendSSMASCIIData ( )
SendSSMData ( )

14.10.5 DataFromSSM (event)

PLServer launches this event when receiving raw data (byte values 0-255) from the SSM base station.

Note: After receiving data from SSM, host application is responsible to send acknowledgement to the SSM base station with SendSSMAck( ) method.

event DataFromSSM (**long** ssmID, **short FAR*** data, **short** length)

Parameters	
ssmID	data to the base station
data	Data from the SSM base station.
length	Length of the *data* in bytes.

See also
DataFromSSMASCII (event)

14.10.6 DataFromSSMASCII (event)

PLServer launches this event when receiving text string from the SSM base station. DataFromSSM is launched before DataFromSSMASCII.

Note: After receiving data from SSM, host application is responsible to send acknowledgement to the SSM base station with SendSSMAck( ) method.

event DataFromSSMASCII (**long** ssmID, **LPCTSTR** dataString, **short** length)

Parameters	
ssmID	data to the base station
dataString	String from the SSM base station. (ASCII values 32-255)
length	Length of the *data* in bytes.

See also
DataFromSSM (event)

14.11 Colors of PLServer “ID box”

- RED DATA_IN Hand Terminal is sent data to the PLServer. DataArrived event is launched.
- YELLOW READY_TO_SEND (hardly seen) HOST application has processed hand terminal data and ’Send’ method is called. Then answer to the hand terminal is ready to send.
- GREEN SENT. Answer to the hand terminal has been sent.
- BLUE WAIT_WHAT Host application has built commands to the hand terminal and Send method is called. If any hand terminal data is not in processed at the time, PLServer will send this answer when hand terminal sends “WHAT” command.
- WHAT_IN Hand terminal is sent WHAT command but any answer to the hand terminal is not ready. WHAT event is launched.
- WHITE with coloured sides. Indicates that hand terminal has a RECEIVER mode ON.

Installing PLServer and Redisributable DLLs

The setup program must install the necessary redistributable DLL files in the Windows system directory. If any of the DLLs are already present on the user’s machine, the setup program should compare their versions with the versions you are installing. Reinstall a file only if its version number is higher than the file already installed.

Before PLServer can be used, it must be register to Windows registration database. You may want your setup program to register the PLServer when it is installed. RegSvr32.exe can be used to register PLServer.ocx.

PLServer uses Microsoft Foundation Classes (MFC) version 7.1.

15 APPENDIX B RF6xx Communication protocol

15.1 Message Frame Structure

The communication between the RF6xx hand terminal and the HOST unit uses following message structure:

Direction: Hand terminal to HOST

SOH	LENGTH	PREFIX_A	PREFIX_B	ID_H	ID_L	ID_X	DATA	CRC_H	CRC_L

Direction: HOST to hand terminal

SOH	LENGTH	PREFIX_A	ID_H	ID_L	DATA	CRC_H	CRC_L

SOH	Start Of Header (ASCII 01H).
LENGTH	The number of bytes in the **DATA** field
PREFIX_A	**bit 0** “1” indicates that the message is repeated by a repeater **bit 1** “ 0 “ indicates the message direction RF600 -> Host. “ 1 “ indicates the message direction Host -> RF600. **bit 2** “0”=RECEIVER ON “1”=RECEIVER OFF (direction RF600->Host) **bit 3** “0”=Normal msg. transaction has started from the hand terminal. “1”=HOST has sent message independently. (direction HOST->RF600) **bit 4-7** RSSI - level. This value indicates the field strength of the signal received at the Base Station
PREFIX_B	(Hand terminal >> Host only) **bit** 0-3 Message number (0-15). This is incremented by 1 after a successful transaction. **bit** 4-6 Battery Level (since version 4.0) **bit** 7 Reserved
ID_H	High byte of the Hand terminal serial number.
ID_L	Low byte of the Hand terminal serial number.
ID_X	Extra ID. A byte that can be set by the user.
DATA	This field includes the actual data and the commands. ASCII values from **0E**h to **1F**h are reserved for the commands.
CRC_H	High byte of the Checksum.
CRC_L	Low byte of the Checksum.

15.1.1 Calculating CRC

The 16-bit CRC-CCITT Checksum is calculated from ID_H to the last databyte. See C-source example function as follows:

```
unsigned short crc;
crc = 0;
calc_crc( ID_H );
calc_crc( ID_L );
.
.
calc_crc( <last databyte>);
CRC_H = (unsigned char)(crc>>8);
CRC_L = (unsigned char)crc;
.
.
void calc_crc( unsigned char character )
{
   unsigned short crc_tmp;
   crc_tmp = (crc ^ character) & 0x0F;
   crc = (crc >> 4) ^ (crc_tmp * 4225);
   crc_tmp = (crc ^ (character >> 4)) & 0x0F;
   crc = (crc >> 4) ^ (crc_tmp * 4225);
}
```

15.1.2 Hand terminal Display

The RF600 Hand terminal has a virtual display page of 12 x 20 characters. The actual display size is 8 x 20 characters, so that only one third of the virtual page can be viewed at one time.

15.1.3 Initial Display Prompt

An initial prompt will always be displayed if no fields have been defined. This will usually occur when the Hand terminal is switched on or when the RAM is cleared. The initial display uses a user writeable header and a 18 character length input field.

15.1.4 Text output on the Hand terminal Display

The Hand terminal fills its display buffer with the data from the received frame. When no cursor control commands are given (SET_CURSOR), the string will be displayed starting with position 0. The SET_CURSOR command moves the starting point to the position given in the command. The size of the Virtual Display cannot be exceeded. Carriage Return (0Dh) gives a line feed and moves the cursor to the start of the next line, TAB (09h) moves the cursor forward 4 positions.

15.1.5 Commands HOST --> Hand terminal

Command Table

HEX	SYMBOL	BYTES	SYNTAX	DESCRIPTION
06	ACK	1	06h	Acknowledgement from the Host
07	BELL	1	07h	Generates a beep
09	VT	1	09h	Tabulator, moves the cursor 4 positions
0A	SPC_CMD	5	0Ah	Special command for hand terminal
0D	CR	1	0Dh	Carriage Return + Line Feed
0E	SET_CURSOR	2	0Eh | POS	Cursor Control
0F	NEW_FIELD	4	0Fh | POS | L | STYLE	Definition of the Input Field
10	BELL_EX	>=2	10h | L | duration | delay..	Generates a beep sequence with specfic duration of beep and duration of delay
11	FIELD_CMD	3	11h | POS | FCMD	Defines the parameters of the Input Field
12	CLEAR_CMD	4	12h | CCMD | POS_START | POS_STOP	Partially clears the Display.
13	SET_VIEW	2	13h | ROW	Sets the Viewing Window
14	POP_MSG	>2	14h | L | txt...	Puts the message in the Display

16	FLD_TXT	>=2	16h | POS | L | txt..	Fills (presets) the field with the text.
17	BUTTON	>=3	17h | POS | L | txt..	Definition of the “button” style input field
18	SEND_WHAT	2	18h | DELAY	Send “WHAT” command to HOST when DELAY has expired.
19	DATA_TO_SERIAL	>=2	19 | L | txt	Sends txt to the serial port of the hand terminal
1A	WAIT_SERIAL	2	1A | DELAY	Waits data from the serial port at specific time
1B	NEW_FIELD_EX	4	1Bh | POS | L | STYLE	Definition of the Input Field
1C	RECEIVER	2	1Ch | mode	Receiver mode on/off
1D	FORM_ID	3	1Dh | FID_HI | FID_LO	User defined ID of the form

15.1.6 Order of Execution OF Commands

The Hand terminal executes commands from the Host in the following order:

1.	POP_MSG	(only 1 allowed per frame)
2.	CLEAR_CMD	
3.	ACK	
4.	Text printing and BELLs	
5.	BUTTON	
6.	FIELD_CMD	
7.	NEW_FIELD	
8.	NEW_FIELD_EX	
9.	FLD_TXT	
10.	SET_VIEW	only 1 allowed per frame
11.	SEND_WHAT	only 1 allowed per frame
12.	WAIT_SERIAL	only 1 allowed per frame
13.	DATA_TO_SERIAL	only 1 allowed per frame
14.	RECEIVER	only 1 allowed per frame
16.	FORM_ID	only 1 allowed per frame
15.	SPC_CMD	only 1 allowed per frame

15.1.7 SET_CURSOR

Description: Cursor control (2 bytes)

Syntax: **0Eh | POS**

POS Cursor position (0 - 239).

15.1.8 NEW_FIELD

Description: Defines a new Input field (4 bytes)

Syntax: **0Fh | POS | L | STYLE**

POS Starting point of the field (0-239).
L Field length (1 - 63).
STYLE Field functions.

STYLE bits can be set or cleared to enable or disable the function.

Bit 0	SND_ENTER	Field is sent to HOST by pressing the OK key.
Bit 1	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, If the field itself gave the SND_ALL command.
Bit 2	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the OK key is pressed.
Bit 3	FLD_LOCK	Field is locked. Field cannot be written to (used for “Button” style fields.)
Bit 4	FLD_LINE	Field is underlined ._ _ _ _ _ _ _ _ _ _ _
Bit 5	FLD_READER	Field can be filled with laser- or external scanner data.
Bit 6	FLD_CLR	Field is cleared if ACK is received .
Bit 7	FLD_ACTIVE	Field is active (it has a cursor).

SND_ENTER and FLD_READER combinations:

SND_ENTER	FLD_READER	FUNCTION
0	**0**	Field is not sent by pressing the OK key, and cannot be read with a laser scanner.
0	**1**	Field is not sent by pressing the OK key, but it can be filled from the keyboard and with a laser scanner.
1	**0**	Field is sent by pressing the OK key, but cannot be filled with a laser scanner.
1	**1**	Field is sent by pressing the OK key or by reading with a laser scanner.

A maximum of 20 input fields can be defined for one form.

15.1.9 NEW_FIELD_EX

Description: Defines a new Input field (4 bytes)

Syntax: **1Bh | POS | L | STYLE**

POS Starting point of the field (0-239).
L Field length (1 - 63).
STYLE Field functions.

STYLE bits can be set or cleared to enable or disable the function.

Bit 0	SND_ENTER	Field is sent to HOST by pressing the OK key.
Bit 1	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, If the field itself gave the SND_ALL command.
Bit 2	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the OK key is pressed.
Bit 3	Reserved	Reserved
Bit 4	OVR	Overwrite mode. When OVR is set, and the field becomes active, the cursor moves to the starting position of the field. When the cursor is in the starting position, the previous text will be overwritten by typing.
Bit 5	FLD_READER	Field can be filled with laser- or external scanner data.
Bit 6	READER_ DEFAULT	If another field is active and it has not FLD_READER set, laser scanner is activated and data goes to this field automatically. Only one READER_DEFAULT field can be defined per form. This bit has no effect if FLD_READER bit is not set.
Bit 7	FLD_ACTIVE	Field is active (it has a cursor).

SND_ENTER and FLD_READER combinations:

SND_ENTER	FLD_READER	FUNCTION
0	**0**	Field is not sent by pressing the OK key, and cannot be read with a laser scanner.
0	**1**	Field is not sent by pressing the OK key, but it can be filled from the keyboard and with a laser scanner.
1	**0**	Field is sent by pressing the OK key, but cannot be filled with a laser scanner.
1	**1**	Field is sent by pressing the OK key or by reading with a laser scanner.

NEW_FIELD_EX is same as NEW_FIELD command but style bit's OVR and READER_DEFAULT are replaced bit's _LINE and FLD_CLR.

Field is underlined automatically. A maximum of 20 input fields can be defined for one form.

15.1.10 FIELD_CMD

Description: Definition of the field function.

Syntax: **11h | POS | FCMD**

POS Field position (0-239).
FCMD Commands of the field

FCMD:

Bit 0	FLD_REMOVE	Removes the field.
Bit 1	FLD_CLEAR	Clears the field (also locked fields).
Bit 2	*****	Reserved.
Bit 3	FLD_LOCK	Locks the field (cannot be written to).
Bit 4	*****	Reserved.
Bit 5	*****	Reserved.
Bit 6	*****	Reserved.
Bit 7	FLD_ACTIVE	Field is set to active.

15.1.11 FLD_TXT

Description: Fills (presets) the field with the text. (3 +TXT bytes)
If the length of TXT is more than the length of field, the excess TXT will be corrupted. If field is already filled with text, it will be replaced with the new text.

Syntax **16h | POS | L | TXT**

POS Field POS.
L Length of text
TXT Text for the field.

15.1.12 BUTTON

Description: Creates a “button” style field. (3 +TXT bits)

The BUTTON command will create a locked field filled with text. Field text (TXT) will be sent to the HOST by pressing the OK key.

Syntax **17h | POS | L | TXT**

POS Button position (0-239).
L Length of button.
TXT Text for a button.

This command is suitable for menu creation.
SND_ALL and NO_SEND functions are disabled.

To create a button field with SND_ALL and NO_SEND functions, use the NEW_FIELD and FLD_TXT commands instead of the BUTTON command.

15.1.13 CLEAR_CMD

Description: Partially clears the display of any text or field

Syntax **12h | CCMD | POS_START | POS_STOP**

CCMD Clearing flags

CCMD:

Bit 0	CLEAR_TXT	Removes text from POS_START to POS_STOP.
Bit 1	CLEAR_FLD	Removes fields from POS_START to POS_STOP.
Bit 2	CLEAR_FLDDATA	Clears the field data from POS_START to POS_STOP.
Bit 3-7	*****	Reserved.

15.1.14 SEND_WHAT

Description: Hand terminal send WHAT command (18h) to HOST when DELAY has expired.

Syntax **18h | DELAY**

DELAY 1-99 seconds.

15.1.15 SET_VIEW

Description: Sets the Hand terminal display view. Generally the display view will be set automatically when fields are edited.

Syntax **13h | ROW**

ROW	Top most line number of the row (0-3). If this value is greater than 3, this command will be corrupted.

15.1.16 POPMSG

Description: This command defines a text string which is displayed as long as any key is pressed.

Syntax **14h | L | TXT**

L	Length of TXT (Text length+cursor controls).
TXT	Printable ASCII characters. (may include SET_CURSOR , VT and CR)

The text will be displayed from position 0. The cursor can be set from 0-79. Text over this area will be corrupted. Only one POPMSG command can be included in the frame.

15.1.17 BELL

Description: Generates a beep

Syntax **07h**

15.1.18 BELL_EX

Description: Generates a beep sequence with specfic duration of beep and duration of delay

Syntax: **10h | L | BEEP_ON_DURATION | BEEP_OFF_DURATION**

L	Length of ON/OFF durations in bytes
BEEP_ON_DURATION	Beeper is ON 10 x BEEP_ON_DURATION milliseconds. (eg. value 100 = 1 seconds)
BEEP_OFF_DURATION	Beeper is OFF 10 x BEEP_OFF_DURATION milliseconds.

Example of “S.O.S” tone:
10h | 18 | 5 | 10 | 5 | 10 | 5 | 30 | 15 | 10 | 15 | 10 | 15 | 30 | 5 | 10 | 5 | 10 | 5 | 10

15.1.19 DATA_TO_SERIAL

Description: Sends DATA to the serial port of the hand terminal.

Portable printer, card reader or any external device can be connected to the serial port of hand terminals and thus are able to receive the DATA sent by HOST.

Serial settings are fixed to: 19200,n,8,1

Syntax **19h | L | DATA**

L DATA length
DATA Any ASCII characters. (0-255)

15.1.20 WAIT_SERIAL

Description: Open the serial port of the hand terminal for specific time.

This command can be used to receive data from the external device such as card reader. Serial port of the hand terminal cannot be open all the time because current consumption then is higher.
Any data from the serial port goes directly to the active input field of the hand terminal. It depends on the style bits of the input field whether the data is sent to the HOST or not after reading from the serial port.

Serial settings are fixed to: 19200,n,8,1

Syntax **1Ah | DELAY**

DELAY 1-99 seconds

New function since version 4.0
When DELAY parameter is 255, hand terminal will send all received serial data to the HOST directly. (Command: 11h | L | DATA)

Serial port of the hand terminal will stay open 30 seconds.

Note: RECEIVER command cannot be used same time with this command.

15.1.21 RECEIVER

Description: This command makes the hand terminal to listen transmissions continuously. The RECEIVER function can be set ON/OFF by keyboard as well (SHIFT+F1).

After a hand terminal has received the message and the HOST has sent it independently, and when RECEIVER is ON, the hand terminal answers to the HOST by RECEIVER_ACK command (1Ch).

In practise, the commands to the hand terminal are PopMessages or Bells. Sending other commands can disturb the already existing works in the hand terminals. This is not however prevented, so the HOST application has a responsibility for sending the commands.

When the RECEIVER is ON, PopMessages will be acknowledged as read by a DEL key. This prevents the accidentally key press before reading the PopMessage.

Syntax **1Ch | mode**

mode:

0	RECEIVER mode OFF
1	Receiver cycle of listening messages from the HOST. Radio is 1 sec on and 2 sec off to save battery power. **RECEIVER_ACK is sent** to the HOST after receiving message.
2	Receiver cycle of listening messages from the HOST. Radio is 2 sec on and 1 sec off to save battery power. **RECEIVER_ACK is not sent** to the HOST after receiving message.
3	Listening **Continuously** messages from HOST. **RECEIVER_ACK is sent** to the HOST after receiving message.
4	Listening **Continuously** messages from HOST. **RECEIVER_ACK is not** sent to the HOST after receiving message.

Note: WAIT_SERIAL command cannot be used same time with this command.

15.1.22 FORM_ID

Description: This command defines a specific ID number which describes the form. This command is useful if the HOST application uses several different forms on the hand terminal. After the data has sent from the hand terminal, the HOST application receives the ID number of the form and then application can determine the actions of data. Upto 65535 different form ID:s can be defined.

The FORMID value is stored in the Hand Terminal memory. The Hand Terminal sends this value everytime to the HOST. If FORMID is defined to 0 or Hand Terminal is resetted, then Hand Terminal does not send FORMID value to the HOST.

Syntax **1D | FID_HI | FID_LO**

FID_HI	High byte of form ID
FID_LO	Low byte of form ID

FORM_ID is not sent by hand terminal if:

- FORM_ID is not defined
- FID_HI = 0 and FID_LO = 0
- Hand Terminal is resetted

15.1.23 SPC_CMD

Description: SPC_CMD defines special functionalities to the hand terminal.

Syntax **0A | FUNC | D1 | D2 | D3**

FUNC	
Bit 0	If this bit is set, the hand terminal will emulate the one key press. The key is defined in parameter D1 Example: 0A | 01 | 06 | 0 | 0 BAR emulates a BAR button press.
Bit 1	If this bit is set, the hand terminal sends keystrokes directly to the HOST. The HOST application can define the keys with parameter D1-D3 that sends the keystrokes to the HOST. Example: UP and DOWN keys will send keystokes to the HOST (hex values): **0A | 02 | 0 | 6 | 0** This function is valid as long as user resets the device or HOST send SPC_CMD where Bit 1 is not set. When the user presses the key which is defined to send keystroke, the hand terminal sends SPC_MSG to the HOST **(0A | 01 | KEYSTROKE)**. After that the hand terminal will wait a response from the HOST.
Bit 2	Password style for input field. D1,D2,D3 will define a position of the password input field. The Handterminal user can only see '*' asterix chars when typing to the input field.
Bit 3	Reserved
Bit 4	Reserved
Bit 5	Reserved
Bit 6	Reserved
Bit 7	Reserved

FUNC	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7			
	Emulates the key press	Hand terminal sends keystrokes to the HOST	Password style input fields								
D1	Key value (decimal) F1 = 1 F2 = 2 F3 = 3 F4 = 4 F5 = 5 BAR = 6 DEL = 7 DOT = 8 MINUS = 9 UP = 10 DOWN = 11 OK = 12 0 = 13 1 = 14 2 = 15 3 = 16 4 = 17 5 = 18 6 = 19 7 = 20 8 = 21 9 = 22	Key (return value)	Position of password input field. 255 = not defined								
Bit 0		F1 (1)									
Bit 1		F2 (2)									
Bit 2		F3 (3)									
Bit 3		F4 (4)									
Bit 4		F5 (5)									
Bit 5		BAR (6)									
Bit 6		DEL (7)									
Bit 7		DOT (8)									
D2	x		Position of password input field. 255 = not defined								

Bit 0	x	MINUS (9)									
Bit 1	x	UP (10)									
Bit 2	x	DOWN (11)									
Bit 3	x	OK (12)									
Bit 4	x	0 (13)									
Bit 5	x	1 (14)									
Bit 6	x	2 (15)									
Bit 7	x	3 (16)									
D3	x		Position of password input field. 255 = not defined								
Bit 0	x	4 (17)									
Bit 1	x	5 (18)									
Bit 2	x	6 (19)									
Bit 3	x	7 (20)									
Bit 4	x	8 (21)									
Bit 5	x	9 (22)									
Bit 6	x	x									
Bit 7	x	x									
x = not used											

15.1.24 Commands Hand terminal --> HOST

Command Table

HEX	SYMBOL	BYTES	SYNTAX	DESCRIPTION
10	FIELD_DATA	3+L	10h | POS | L | DATA	Description: Data from the Hand terminal. Indicates the position (address) of the data sent from the fields. Syntax: **10h | POS | L | DATA** (3+L bytes) **POS** Field POS (0 - 239). **L** Number of bytes in the **DATA** part. **DATA** Field data. (ASCII text) Text, sent from the Hand terminals initial screen or from F-keys do not use this command.
18	WHAT	1	18h	The Hand terminal sends this command when DELAY has expired in SEND_WHAT command.
1C	RECEIVER_ACK	1	1Ch	Hand terminal has received message from the HOST which is sended independently by the HOST.
1D	FORM_ID	3	1Dh | FID_HI | FID_LO	If FORM_ID is defined, Hand Terminal sends it always at beginning of the DATA block. (first 3 bytes in data block)
0B	SPC_MSG	3	0Ah | TYPE | D1	Special message from hand terminal. Hand terminal sends this message if specific function is defined earlier with SPC_CMD: If **TYPE** = 1 : Keystroke from hand terminal. **D1** contains keystroke number **Note:** The hand terminal requires response.

1F	SSM_DATA	2+L	1Fh | L | DATA	When the SSM base station receives data from the device attached to its serial port, it encapsulates the data into an SSM_ DATA command and sends it to the host. L Number of bytes in the **DATA** part. If the attached device tries to send a string that is longer than the maximum frame size (about 250 bytes), it is split to two or more frames. When the SSM base station has sent a frame to the host, it waits for an acknowledgement. It is the host program's responsibility to send this acknowledgement; unlike the SSM base station, the host base station will not automatically send an acknowledgement. An empty frame serves as an acknowledgement for the SSM base station, ie. a frame that has length 0 and no data part: SOH | 0 | PREFIX_A | ID_H | ID_L | CRC_H | CRC_L If the SSM base station receives no acknowledgement, it will resend the frame. The host program should use the message number field in the message prefix A byte to distinguish between new frames and possible superfluous resends.
11	SERIAL_DATA	2+L	11h | L | DATA	Data from serial port of the hand terminal. When HOST has defined WAIT_SERIAL command and delay parameter is 255, hand terminal sends serial port data directly to the HOST with this command. **L** Length of DATA **DATA** data from serial port **Note:** The hand terminal requires response.

16 APPENDIX C RF600 SSM mode protocol

16.1 Purpose of this document

This presentation documents the RF600 communication protocol extensions that support the substation modem (SSM) mode of the base station. It is assumed that the reader is already familiar with the other aspects of the communication protocol (see the document 'RF600 Communication Protocol').

16.2 Overview

The SSM mode of the RF600 base station allows for asynchronous transfer of arbitrary data between the host and the SSM base station. For the host base station, the SSM base station looks like a hand terminal and is treated identically. The data transmitted to the SSM base station is stripped of the RF600 protocol frame and sent to the serial port as a stream of raw bytes. When the device attached to the SSM station sends data to the SSM station, the data is encapsulated into a protocol frame and sent to the host system. The device connected to the SSM base station need not understand anything about the communication protocol; the SSM base station acts as a mediator between the device and the RF600 wireless network. Due to the asymmetric nature of parties involved in the data transfer, the procedure to send data from the host to the SSM base station is different from the procedure used for the other direction. Both are described in detail separately below. The traffic to the SSM base station uses the host to hand terminal protocol, whereas communication from the SSM base station to the host station uses the hand terminal to host protocol.

16.3 The power-on configuration

In SSM mode the base station supports arbitrary serial port configurations (between transfer rates of 1200 to 19200 bps). However, to configure the base station with the Piccopla program, the serial port must in a known configuration. Therefore, when the base station that has been set to function in the SSM mode is powered, it sets the serial port into 19200,8,n,1 mode and waits a certain period for configuration commands. If no commands are received before the period (set to 10 seconds as a factory setting) expires, the base station initializes its serial port to the user-configured mode. During the configuration period both lights on the base station will be lit. When the base station goes to normal operation and is ready to receive data, the transmit light will go out.

16.4 Sending data from the host to the SSM base station

The SSM base stations are addressed by their serial numbers in the same way hand terminals are. The frame sent to the SSM base station should contain a frame number in the first byte of the data part, and immediately after that the data as a sequence of raw bytes. Note that there is no special command character before the actual data part; everything (after the frame number) sent to an SSM base station is assumed to be data. When the SSM base station receives the frame, it strips the header, frame number and the checksum from it and passes only the actual data to the serial port, in the exact order and format given in the frame.

(Header)	Frame number (1 byte)	Data (1 or more bytes)	(Checksum)

When the SSM base station receives a frame (that is not itself an acknowledgement; see below), it sends back an acknowledgement frame. This frame contains only the RECEIVER_ACK character (1Ch). It is the host program's responsibility to wait for the acknowledgement and manage resends, if no acknowledgement is received.

(Header)	1Ch	(Checksum)

In case the acknowledgement sent by the SSM base station is lost or the SSM base station receives a duplicate data frame for some other reason, the frame number byte is used to distinguish between resends of a previous frame and completely new frames. Therefore, it should be different for every new frame, but remain the same on every resend. It is the host programs responsibility to ensure that this convention is followed. Note that there is no inherent meaning to the frame numbers; as long as the rules laid out above are followed, any convenient numbering convention may be adopted (for example, sequential frame numbering or simply numbers alternating between one and zero).

16.5 Sending data from the SSM base station to the host

When the SSM base station receives data from the device attached to its serial port, it encapsulates the data into an SSM_DATA command and sends it to the host. The SSM_DATA has the following format:

1Fh	Len	Data

The Len field indicates the number of bytes in the data part. If the attached device tries to send a string that is longer than the maximum frame size (about 250 bytes), it is split to two or more frames.

When the SSM base station has sent a frame to the host, it waits for an acknowledgement. It is the host program's responsibility to send this acknowledgement; unlike the SSM base station, the host base station will not automatically send an acknowledgement. An empty frame serves as an acknowledgement for the SSM base station, i.e. a frame that has length 0 and no data part:

SOH	0	PREFIX_A	ID_H	ID_L	CRC_H	CRC_L

If the SSM base station receives no acknowledgement, it will resend the frame. The host program should use the message number field in the message PREFIX_A byte to distinguish between new frames and possible superfluous resends.