LULEÅ TEKNISKA UNIVERSITET

2001:24 HIP

EXAMENSARBETE

TEMS Automatic Field Support Tool

Gunilla Lestander och Joanna Sjödin

HÖGSKOLEINGENJÖRSPROGRAMMET
DATATEKNIK

Institutionen i Skellefteå

2001: 24 HIP · ISSN: 1404 - 5494 · ISRN:LTU - HIP - EX -- 01/24 --SE

ERICSSON

Preface

This thesis represents our ten week long Bachelor's project at Ericsson Erisoft in Ursviken at section N/TA.

This project is the final assignment to get our bachelor's degree in computer engineering at University of Technology, the institution in Skellefteå.

We would like to give a thank you note to everyone at N/TA that helped us during this project.

We also thank the teachers at Luleå University of Technology, Skellefteå Campus, for good teaching, and the staff at "Studentexpen" for good service and happy faces during these three years.

Ursviken 2001-06-01

Gunilla Lestander
Joanna Sjödin

ERICSSON

Purpose

The purpose of this report is to complete the written part of our exam project in Computer science. First in our report we have made an short overview of the system involved in our project. Then follows a short description of our exam project to explain our task. We have also tried to give a more accurate description of our course of action. The functionality of our application, which is the result of our project, is shown with both images and text. As a conclusion to our report a discussion, problem and reference section was made.

ERICSSON

Abstract

TEMS Automatic is a system for analysing the signal quality and accessibility in the wireless network. It consists of among other things of a number of mobile test units (MTU) placed in vehicles. Ericsson Erisoft has experienced need for a small mobile field tool to facilitate error detection and maintenance of the MTUs.

In our exam project we have developed an application suitable for an iPAQ out on the field. The application reads trace messages sent from the MTU and have an option to save the messages for later analysis.

ERICSSON

TABLE OF CONTENT

ERICSSON

ERICSSON

Introduction

Background

TEMS Automatic is a system for analysing the signal quality and accessibility in the wireless network. TEMS system consists, among other things, of a number of Mobile Test Units (MTU) placed in vehicles on the move to enable collection of measurement data in realistic conditions. The MTU continuously sends trace messages in which it is possible to follow the course of events from the MTU.

Ericsson Erisoft has experienced that MTUs have been returned for restoration, but detected no errors during the analysis at Erisoft. For that reason a better diagnostic tool is needed to rectify the MTU on the field, and this is the purpose of our degree project.

Task

Develop a program suited for an iPAQ to satisfy the need of a better diagnostic tool. The application shall display the essential parts of the trace messages sent from the MTU during the drive.

The user shall be able to choose whether to save/quit saving the trace messages to file or just monitoring them on the screen. Saved trace files are supposed to be analysed on a stationary PC.

Thesis

This thesis was carried out according to Ericsson's project model PROPS. A project specification was written after we had understood our task and made some research. The project specification contained the purpose and background of the thesis and a time schedule, including milestones and tollgates to help us keep on track.

ERICSSON

Tutorial

Stina Öhlund, N/TA section, at Ericsson Erisoft, was designated to be our tutor. The work was performed during ten weeks at Ericsson Erisoft AB in Ursviken.

Methods

Hardware

To carry out this project we needed an iPAQ, an MTU and a serial cable to connect the two through the communication ports. The handheld piece supplied to us was a Compaq iPAQ H3630.

Software

The Operating System in iPAQ is Windows Powered PocketPC.
Our application is written in C++ in the eMbedded Visual C++ 3.0 environment in Windows NT 4.0.

To enable transfer of our applications executable file from the PC to the iPAQ, Microsoft Active Sync was used. Active Sync synchronises the information on the PC with the information on the iPAQ, and updates the iPAQ with the most recent information.

Our task was to develop a program suited for an iPAQ, where the user easily could monitor the essential parts of the trace messages sent from the MTU.

The trace messages has a length of about 80 characters. The iPAQ display capability is limited to 38 characters in a row; therefore the first part of the messages is not displayed. The user is able to double-click on a row to see the complete trace message. To minimise use of memory only the last 50 trace messages is visible on the display.

ERICSSON

The functionality save/quit saving the trace messages to file is also implemented. The entire trace messages are saved, and the saved file is supposed to be analysed on a stationary PC.

MTU

The Mobile Test Units are placed in vehicles on the move in the operator's and the competitors' network area. This is done to enable collection of measurement data in realistic conditions, with respect to geography, movement, radio environment, etc.

The MTU is implemented as an integrated unit with a built-in computer, a test mobile station, a positioning system (GPS receiver) and data communication equipment.

Each MTU acts as a simulated subscriber. It initiates and receives calls at certain times and in certain places, as prescribed in the measurement orders sent from the administration centre. The MTU stores all the measurements as logfiles, which later are sent to the central server over the GSM data channel.

The MTU continuously sends trace messages. These messages are the same as the data stored in the logfiles, but they are sent in readable text format, unlike the logfile data.
The trace messages are mostly used for error handling and detecting.

ERICSSON

iPAQ H3630 – features

Power supply	950 mAh Lithium Polymer, rechargeable in docking cradle or with AC Adapter, battery life up to 12 hours
Display	Colour (4096 colours (12 bit) touch-sensitive reflective thin film transistor (TFT) liquid crystal display (LCD)
Communication port	with USB/Serial connectivity
Infrared port	115 Kbps
Memory expansion	Through Expansion Pack
Operating system	Windows powered Pocket PC
Processor	206-MHz Intel StrongARM SA-1110 32-bit RISC Processor
RAM	32-MB SDRAM
ROM	16-MB Flash ROM Memory
Dimensions (HxWxD)	5.11 x 3.28 x 0.62 in/12.99 x 8.33 x 1.57 cm 130 x 83.5 tapering to 77.5 x 15.9 mm
Resolution (WxH)	240 x 320
Weight	6.3 oz/178.6 g (including battery)

ERICSSON

Application

System Requirements

To be able to use the application, the MTU and the iPAQ need to be connected with serial cable.

If the handheld device is used without external power, its strongly recommended that the automatic “turn off device if not used for x min” is disabled, otherwise a possible loss of data may occur. External power for using the iPAQ while driving is supported by an auto adapter.

The iPAQ H3630 has a limited memory space of 32 MB. If large files are to be saved, it is recommended to add a Flash card.

Analysis

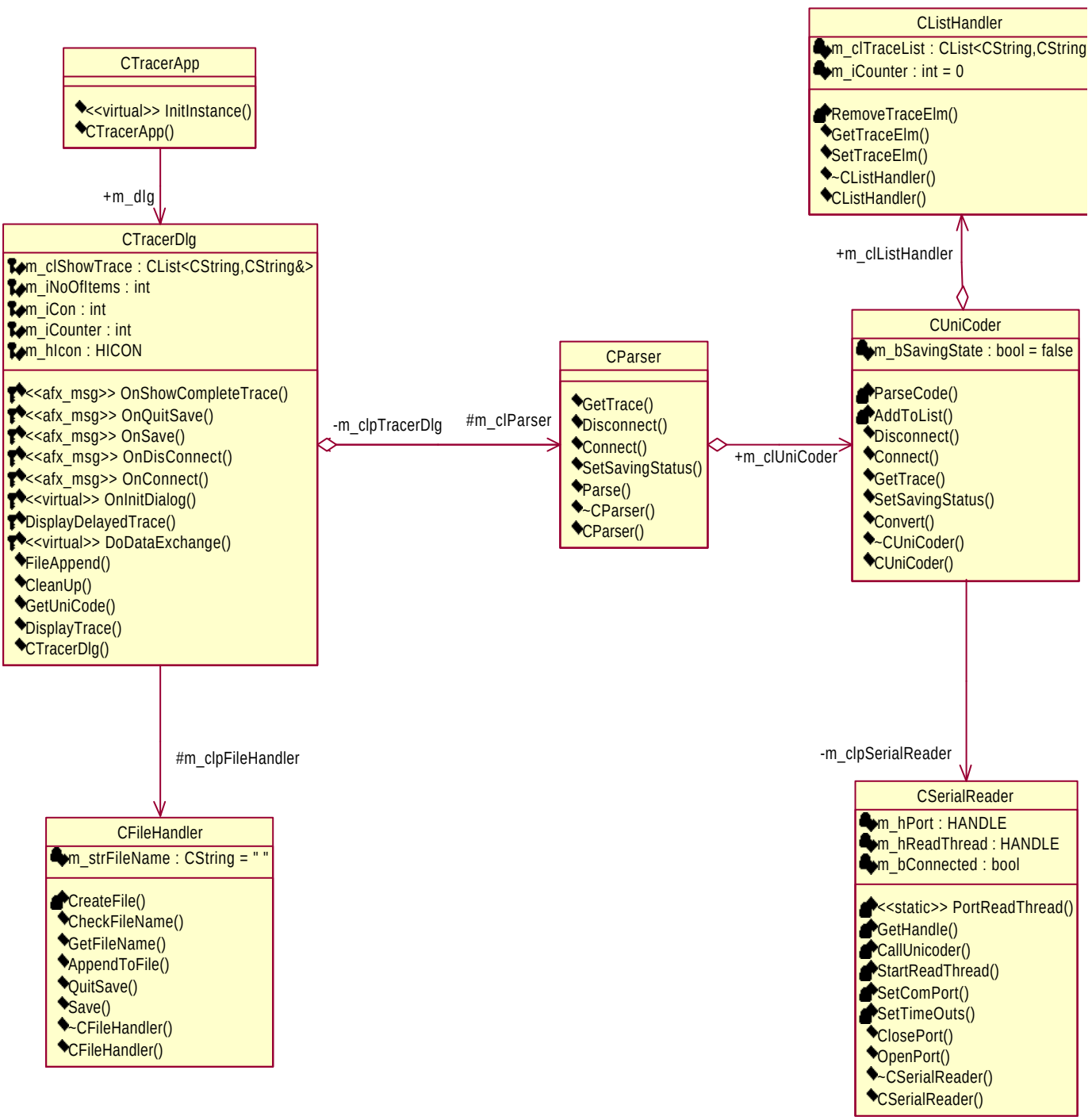
Our analysis was made during the first two weeks. During this research phase information was mostly found through international search tools on the Internet. We also found a lot of help regarding our task in the eMbedded Visual C++ Guide and Microsoft Foundation Class Library for Windows CE. The information we found served as a very good basis to the design phase.

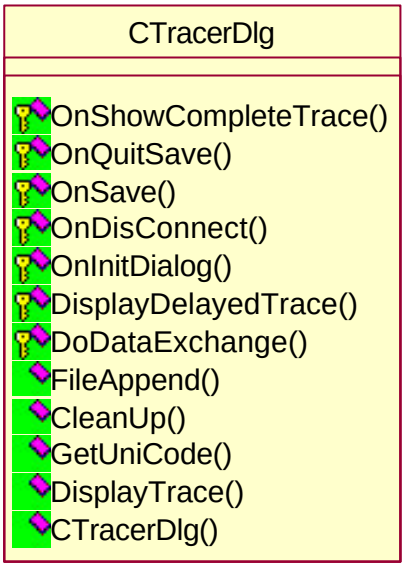
Design

During the design phase we used the Rational Rose RealTime to create our classes and class diagram, with UML notation.

Se next page for our class diagram drawing.

ERICSSON

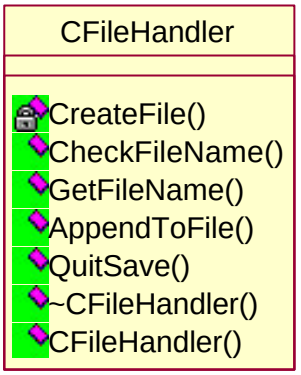

CTracerApp
<<virtual>> InitInstance()
CTracerApp()
+m_dlg
CTracerDlg
m_clShowTrace : CList<CString,CString&>
m_iNoOfItems : int
m_iCon : int
m_iCounter : int
m_hIcon : HICON
<<afx_msg>> OnShowCompleteTrace()
<<afx_msg>> OnQuitSave()
<<afx_msg>> OnSave()
<<afx_msg>> OnDisConnect()
<<afx_msg>> OnConnect()
<<virtual>> OnInitDialog()
DisplayDelayedTrace()
<<virtual>> DoDataExchange()
FileAppend()
CleanUp()
GetUniCode()
DisplayTrace()
CTracerDlg()
-m_clpTracerDlg
#m_clParser
CParser
GetTrace()
Disconnect()
Connect()
SetSavingStatus()
Parse()
~CParser()
CParser()
+m_clUniCoder
CListHandler
m_clTraceList : CList<CString,CString
m_iCounter : int = 0
RemoveTraceElm()
GetTraceElm()
SetTraceElm()
~CListHandler()
CListHandler()
+m_clListHandler
CUniCoder
m_bSavingState : bool = false
ParseCode()
AddToList()
Disconnect()
Connect()
GetTrace()
SetSavingStatus()
Convert()
~CUniCoder()
CUniCoder()
#m_clpFileHandler
CFileHandler
m_strFileName : CString = " "
CreateFile()
CheckFileName()
GetFileName()
AppendToFile()
QuitSave()
Save()
~CFileHandler()
CFileHandler()
-m_clpSerialReader
CSerialReader
m_hPort : HANDLE
m_hReadThread : HANDLE
m_bConnected : bool
<<static>> PortReadThread()
GetHandle()
CallUnicoder()
StartReadThread()
SetComPort()
SetTimeOuts()
ClosePort()
OpenPort()
~CSerialReader()
CSerialReader()



CTracerDlg

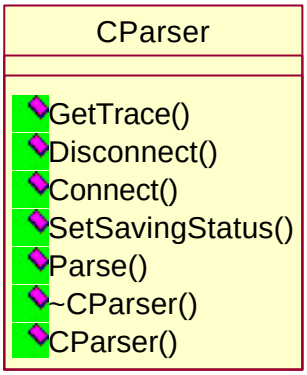
Responsibility: This class displays the functionality offered to the user. It captures input from the user and acts accordingly to the chosen option. If the option is to save messages to file it creates the class CFileHandler, and if the chosen option is to quit save, it deletes CFileHandler. CTracerDlg also manage the display of trace messages.

If the user interface is shut down with the OK button, this class make sure that the closing procedure is done properly.



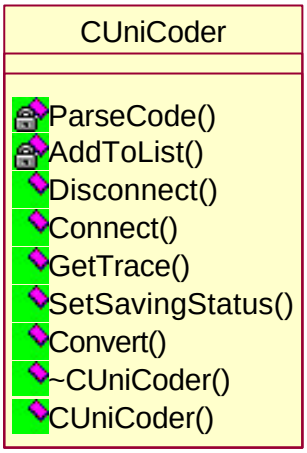
CFileHandler

Responsibility: This class creates a file with a chosen filename, and appends the trace messages received. It stores the current filename to display to the user if the save option is selected when saving is already activated.



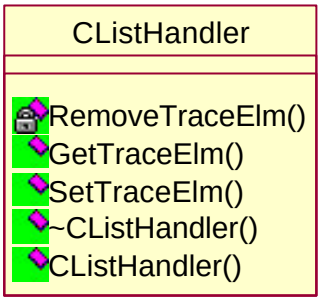
CParser

Responsibility: CParser receives strings in Unicode format from CUniCoder. If saving to file is activated, it passes the strings to CFileHandler. Before it passes the strings to CTracerDlg to be displayed, the strings are given the appropriate length by removing the first 39 characters.



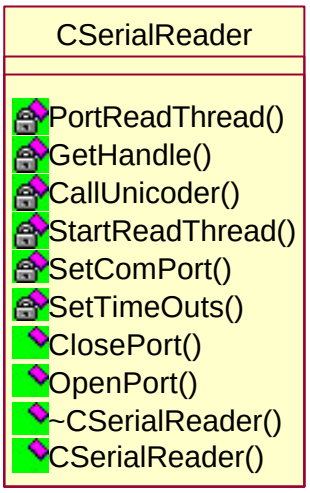
CUniCoder

Responsibility: To receive ASCII strings from the CSerialReader. If saving to file is activated, it passes the strings to CFileHandler. The ASCII strings are converted into Unicode strings, and then passed to CListHandler and CParser.



CListHandler

Responsibility: To store the unparsed, last 50 messages in a list. If the user double tap on a row at the display, CListHandler presents the complete string on the display.



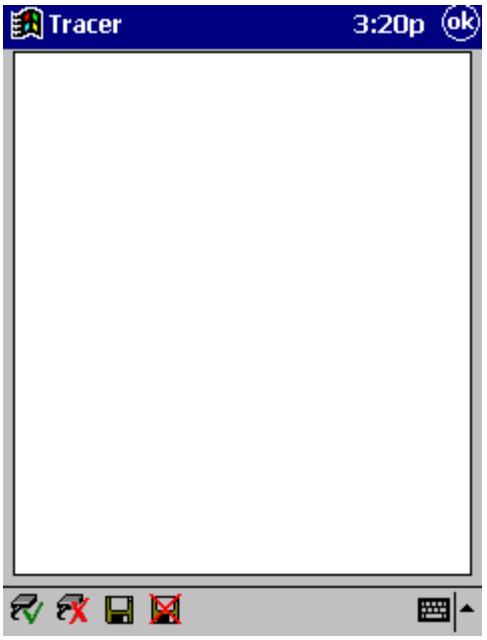
CSerialReader

Responsibility: Configure the port and start a reading thread. The reading thread reads one character at the time until it discovers Linefeed and Carriagereturn The characters read from the port are passed on as strings to CUniCoder. When the application is shut down it is incumbent upon CSerialReader to close the port and stop the reading thread.

ERICSSON

Functionality

The application is very simple and easy to use. At start-up this main window faces the user.

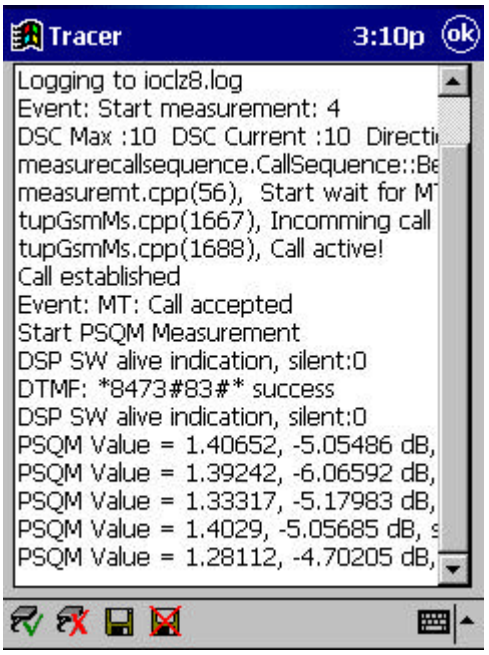


	**Connect**	Open port and connect to the MTU
	**Disconnect**	Close port and file if it is not closed
	**Save**	Start saving trace message to a selected
	**Quit Save**	Close the open file and stop saving

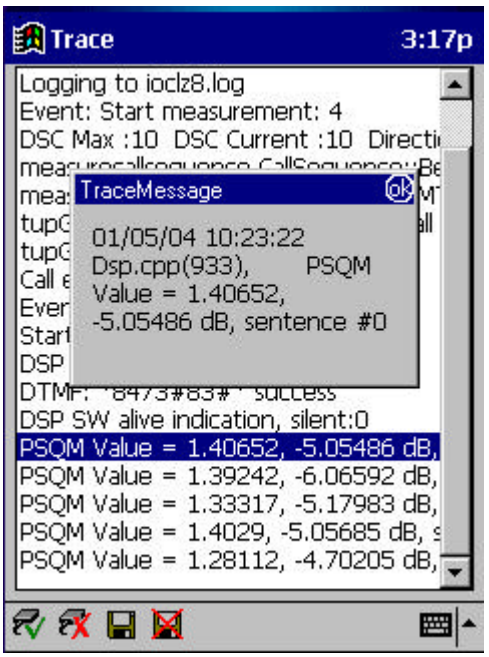
ERICSSON

Connect

The first thing the user has to do is to press the Connect button; this command opens the serial port. This can be done either before or after connecting the MTU and the iPAQ. Then the trace messages starts to scroll on the display.



A trace message sent from the MTU consists of about 80 character, but the iPAQ only display 38. If the user wishes to see the complete message it is possible to double-tap on wanted row, this causes the complete message to be displayed in a separate “

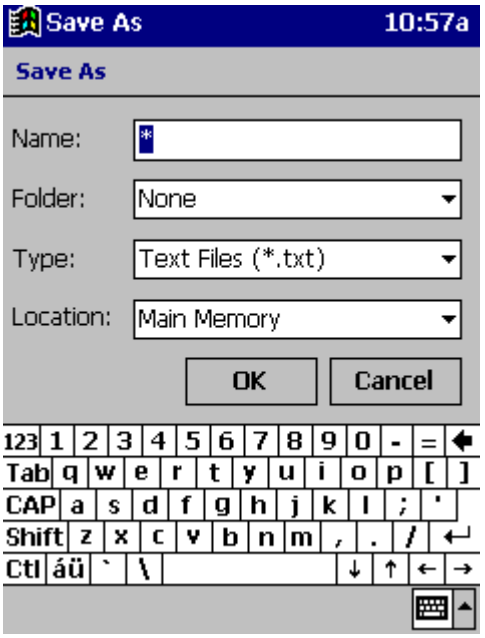


If trace messages are read from port while the “TraceMessage” window is active they will not be displayed. They are saved in a temporary list and displayed after “TraceMessage” window is closed.

ERICSSON

Save file

During the display of trace messages it is up to the user to select if or when to save to file. When the save button is pressed a new dialog is shown. Here the user can choose in which directory the file shall be saved, and what to call it. For file extension .txt is predefined. It is only possible to save one file at the same time.



ERICSSON

Quit save

When the user wishes to stop save trace messages, he/she has to press Quit Save button or Disconnect, or escape the program through the OK button.

Disconnect

To stop the application from reading the trace messages sent from the MTU, the Disconnect button is pressed or the program is terminated through the OK button.

Analysis of saved file

To read a file the user may transfer the file from the iPAQ to a stationary PC. Using Microsoft Active Sync easily does this. It is recommended to open the file in WordPad for analysis.

Limitation

To minimise use of memory the user is only able to double-click on the last 50 trace messages for displaying the complete trace message.

The trace message “calculated size” will never be displayed, but it will be saved to file if that functionality is activated.

ERICSSON

Discussion

The progress of our Bachelor project has followed the stated time schedule,
thanks to everyone who helped and applied us whit what ever we needed.

The essential thing we started with was the serial port. We had to figure out how to configure the port, because reading the port was the thing our application was all about. Another thing was the conversion from multibyte character too wide character, which was needed to display readable text on the iPAQ's display.

Eventually we succeeded to read from port and trace messages started to display. The messages were converted to Unicode and we parsed them to the proper length. What we did not know was if we lost anything from the serial port. Therefore we asked for a special cable with two serial connections so we could compare our saved file on the iPAQ with the HyperTerminal printout. The printouts showed that we did not loose any character from the port.

Another thing we been busy with are the user interface. We got the opportunity to design it as we thought was best. We decided to use a dialog-based application with buttons. This because it was the one we knew best.

The buttons we made in the beginning were placed in the already small display we had to our disposal. The best was if we could use the command bar in the bottom of the iPAQ display, and show our own buttons there. This made us try the single document application instead, but with poor result. At last we figured out how to use the dialog-based application and get a hold of the command bar.

The scroll bar had a tendency to flicker up and down during test, due to deleting and adding items in the dialog listbox. Another probably cause was that the items (fake trace messages) were added with a tap on a button. This, however, does not seem to be a problem when the application is reading, adding and deleting real trace messages from the port.

Problem

One problem we met was the lack of debugging possibility through the emulator; this had to do with a mismatch between emulator and the COM port. When we in the beginning were debugging our application, we got an INVALID_HANDLE_VALUE when we tried to open the port for reading. The invalid handle indicated that the port was not opened, but it was. When we discovered this we had to download a new version of the application as soon as we had made one small change in the code, to know whether we could read something from the port or not.

However, the debugging problem was solved during the last two weeks at Ericsson Erisoft. It turned out that if we removed the ActiveSync process, there was no problem debugging the port.

ERICSSON

References

Literatures

Richter, J.(1999). *Programming Applications for Microsoft Windows Fourth Edition.* Washington, Redmond: Microsoft Press. ISBN 1-57231-996-8

Makofsky, S, et al.(1999). *Sams Teach Yourself Windows â CE Programming in 24 Hours.* USA. ISBN 0-672-31658-7

The Web

www.msdn.com

“MSDN online libraries”

www.ericsson.com/tems/gsm/automatic-gsm.shtml

“Ericsson external web”