Tutorial: How to create and debug a simple „Hello world“ application with Visual Studio 2005/2008.

Developing Software for PicoCOM-, PicoMOD-
and the NetDCU series

Version 1.00 Date: 2009-03-06

F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Fon: +49(0)711-123722-0
Fax: +49(0)711-123722-99

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2009-03-06	1.00	*	A	*	First version	DK

V	Version
A,M,R	Added, Modified, Removed
Au	Author: CZ, DK, HF, HK, MK

Contents

1 Native Code and managed Code 1
1.1 Installing the platforms' SDK 1

2 Creating Visual Studio Projects for smart devices 2

3 Debugging an application from Visual Studio 5

4 Developing a simple C console program 7

5 Developing a simple C++ program with MFC 8

6 Developing a simple C# application 10

7 Writing to serial port from C++ and from C# 11

8 Appendix 12
Listings 12
Figures 12

1 Native Code and managed Code

Application developing for Windows Embedded CE based modules from F&S is made from Visual Studio 2005 (at least Standard Edition) or Visual Studio 2008 (at least Professional Edition). You have the choice to either develop your application in native code (C or C++) or from managed code (C# or VB.NET). All F&S platforms provides Windows Embedded CE kernel images with support for either Compact Framework 2.0 or Compact Framework 3.5.

Note:

When you develop your application in managed code for Compact Framework 3.5 you need to use Visual Studio 2008 Professional.

1.1 Installing the platforms' SDK

To develop applications with C or C++ (native code) you need to install the respective SDK for your F&S platform.

You can download the SDK from: http://www.fs-net.de/download/bin/.
For the PicoCOM series you can download the SDK from: http://www.picocom.de/download.

Simple follow the installation wizard of the <SDK name>.msi file. The SDK gets installed to `C:\Programs\Windows CE Tools\<WINCE_VERSION>`.

Note:

When installing a SDK into Visual Studio 2008 please deselect the documentation from the Custom installation menue.

2 Creating Visual Studio Projects for smart devices

To develop applications for the PicoCOM-, PicoMOD- or NetDCU- series you need Visual Studio 2005 Standard Edition or Visual Studio 2008 Professional Edition.

A. Create a new project:

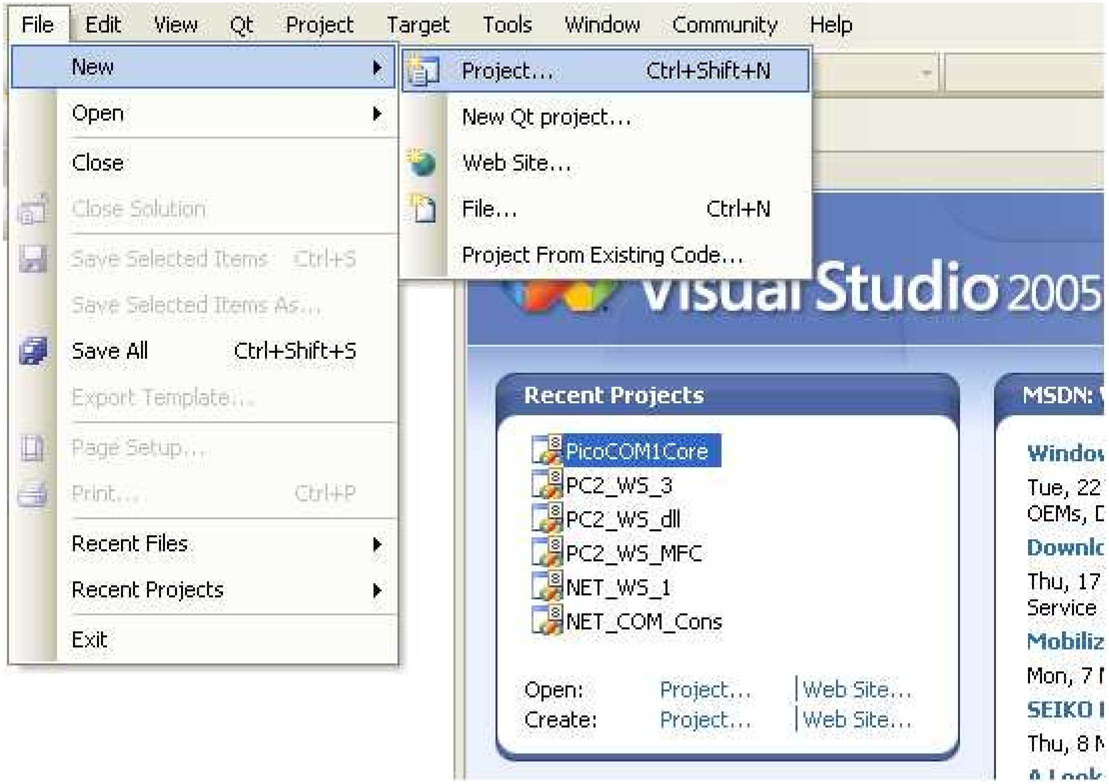
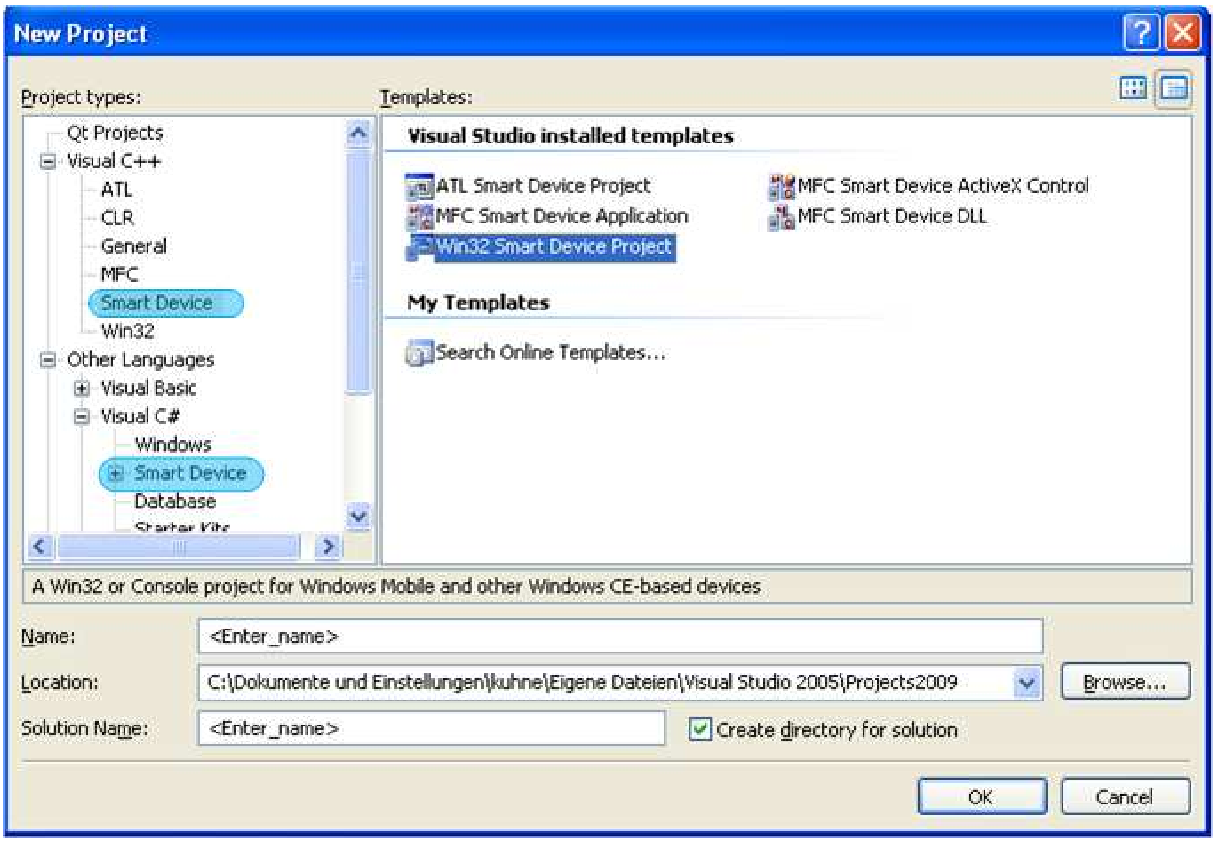


Figure 1: Creating a new project

B. Choose the project type:



A Win32 or Console project for Windows Mobile and other Windows CE-based devices

Name: <Enter_name>

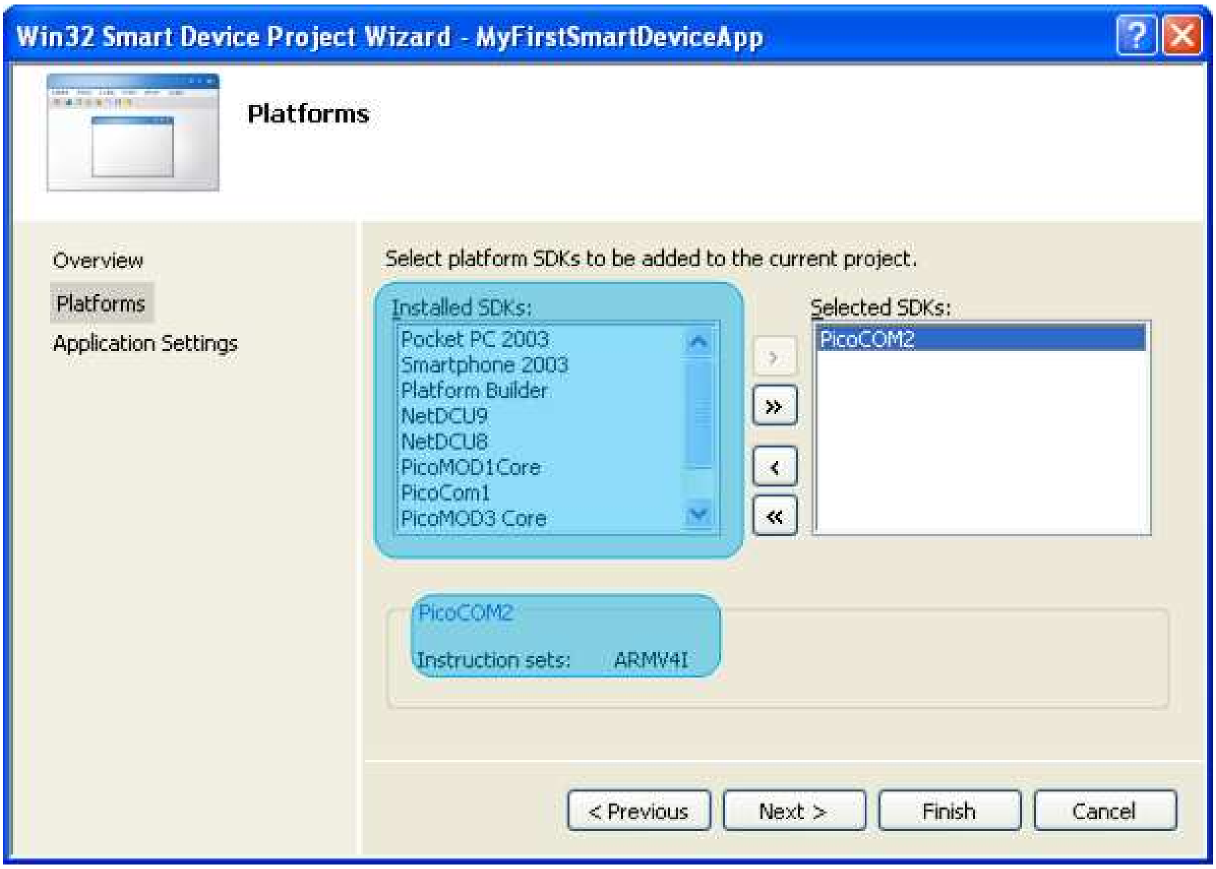
Location: C:\Dokumente und Einstellungen\kuhne\Eigene Dateien\Visual Studio 2005\Projects2009

Solution Name: <Enter_name>

☑ Create directory for solution

Figure 2: Choosing the project type

C. SDK selection (only for native code):



Win32 Smart Device Project Wizard - MyFirstSmartDeviceApp

Platforms

Select platform SDKs to be added to the current project.

Installed SDKs:

- Pocket PC 2003
- Smartphone 2003
- Platform Builder
- NetDCU9
- NetDCU8
- PicoMOD1Core
- PicoCom1
- PicoMOD3 Core

Selected SDKs:

- PicoCOM2

PicoCOM2

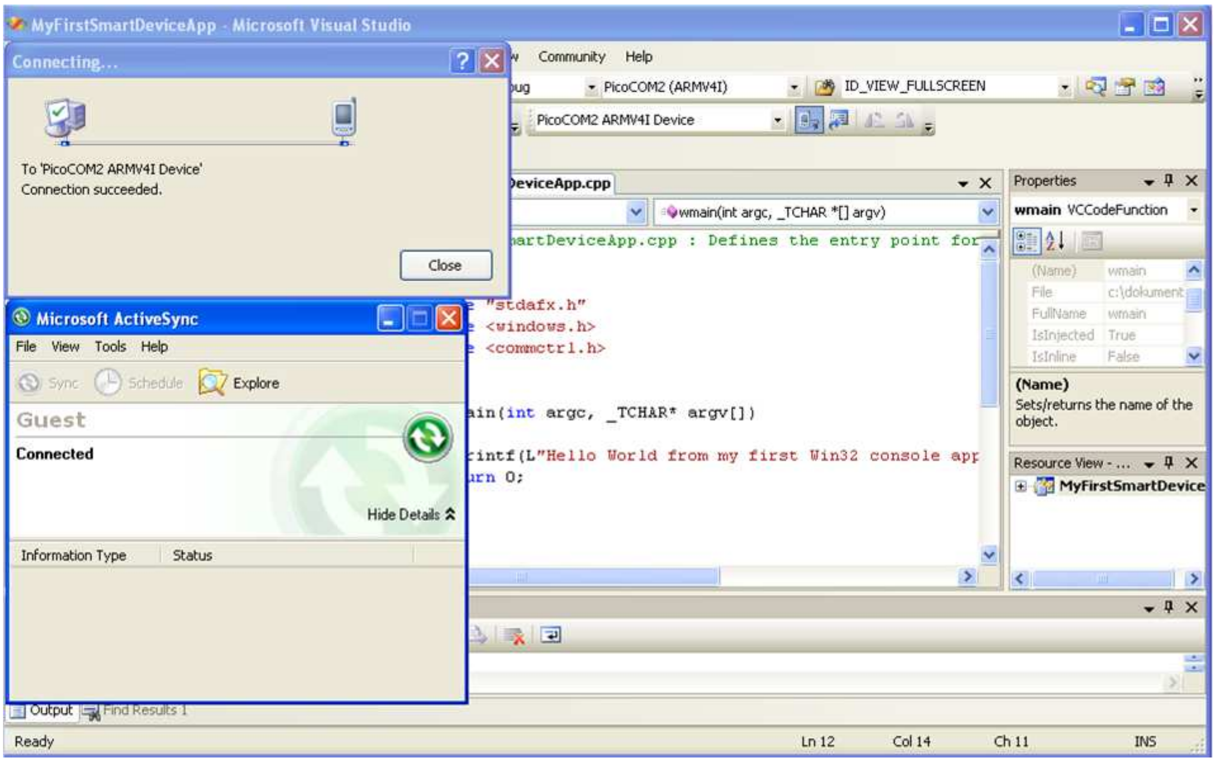
Instruction sets: ARMV4I

Figure 3: Selecting the respective SDK

D. Choose the project settings and finish the wizard (read the chapters 4 to 6 for your respective project type).

3 Debugging an application from Visual Studio

You need a ActiveSync connection to the device to be able to debug your application. Be sure you are connected before start debugging and deploying the application.



To 'PicoCOM2 ARMV4I Device'
Connection succeeded.

Connected

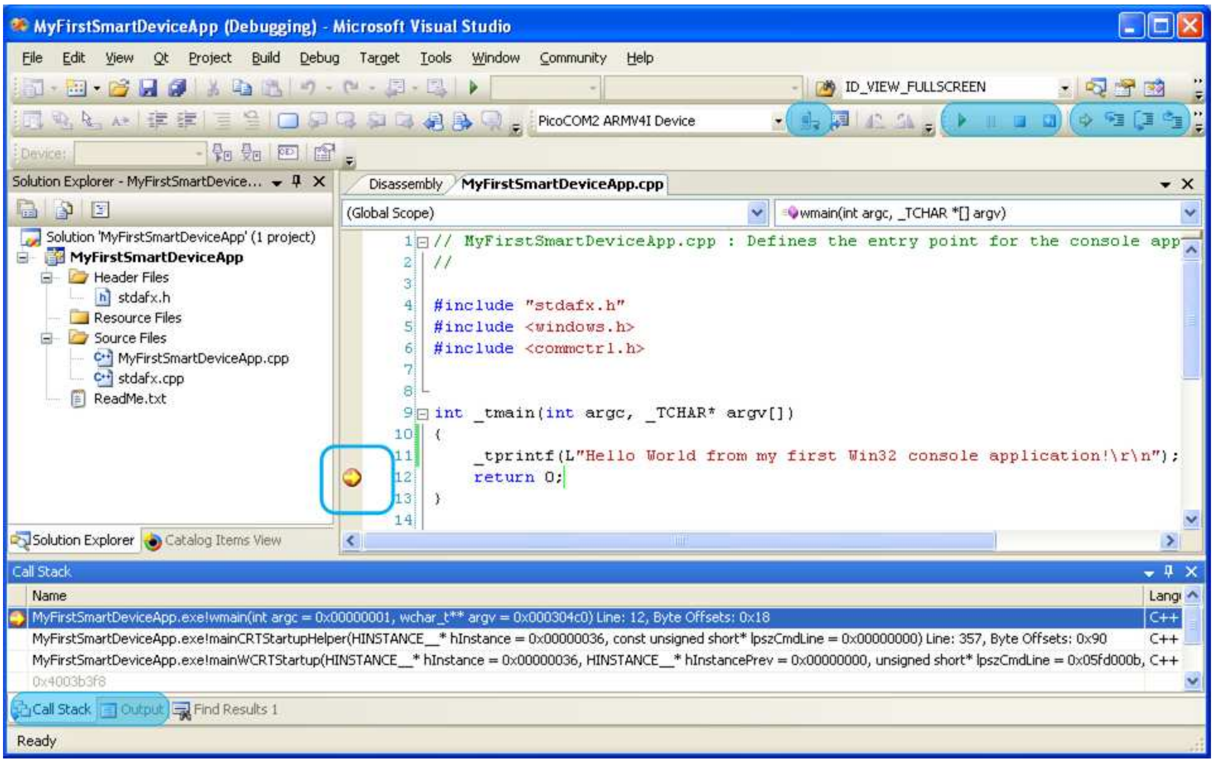
```
artDeviceApp.cpp : Defines the entry point for
"stdafx.h"
<windows.h>
<commctrl.h>

ain(int argc, _TCHAR* argv[])

intf(L"Hello World from my first Win32 console app
rn 0;
```

(Name)	wmain
File	c:\dokument
FullName	wmain
IsInjected	True
IsInline	False

(Name)
Sets/returns the name of the object.

Figure 4: Connect VS2005 to your platform

After you are connected with ActiveSync you can start debugging by pressing >>F5<<. You can set breakpoints and examine variables and the callstack, etc.


MyFirstSmartDeviceApp (Debugging) - Microsoft Visual Studio
File Edit View Qt Project Build Debug Target Tools Window Community Help
ID_VIEW_FULLSCREEN
PicoCOM2 ARMV4I Device
Device:
Solution Explorer - MyFirstSmartDevice...
Solution 'MyFirstSmartDeviceApp' (1 project)
MyFirstSmartDeviceApp
Header Files
stdafx.h
Resource Files
Source Files
MyFirstSmartDeviceApp.cpp
stdafx.cpp
ReadMe.txt
Solution Explorer
Catalog Items View
Disassembly
MyFirstSmartDeviceApp.cpp
(Global Scope)
wmain(int argc, _TCHAR *[] argv)
Call Stack
Output
Find Results 1
Ready

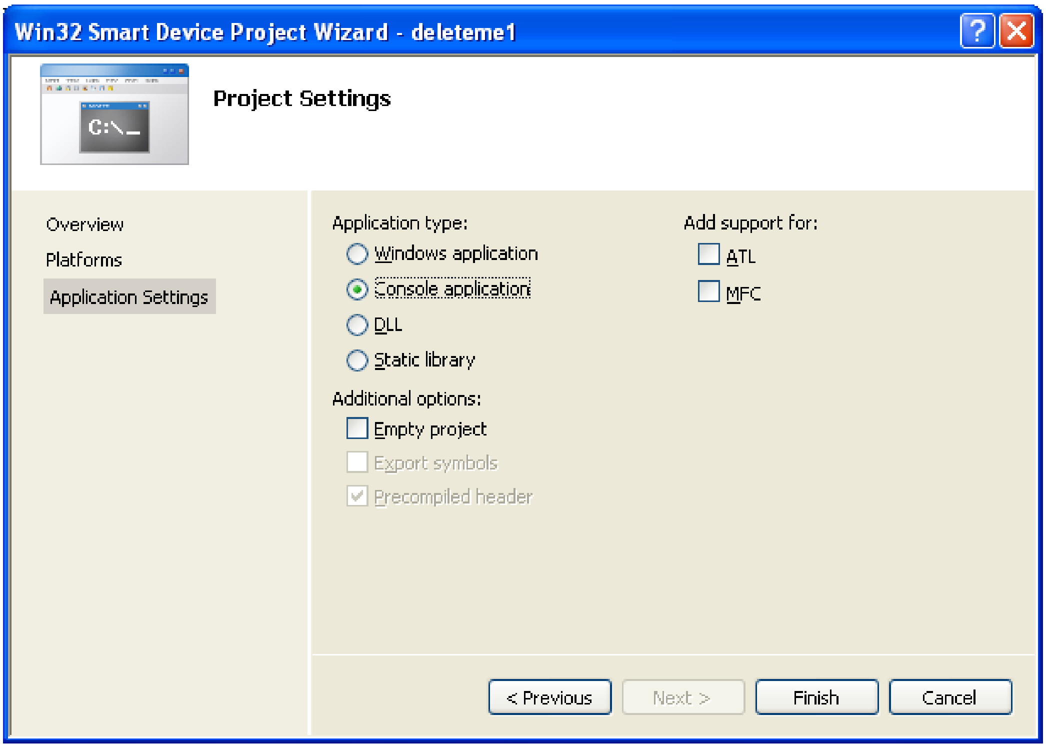
```
// MyFirstSmartDeviceApp.cpp : Defines the entry point for the console app
//

#include "stdafx.h"
#include <windows.h>
#include <commctrl.h>


int _tmain(int argc, _TCHAR* argv[])
{
    _tprintf(L"Hello World from my first Win32 console application!\r\n");
    return 0;
}

```

```
Name                                                                                                                                          Lang
MyFirstSmartDeviceApp.exe!wmain(int argc = 0x00000001, wchar_t** argv = 0x000304c0) Line: 12, Byte Offsets: 0x18                              C++
MyFirstSmartDeviceApp.exe!mainCRTStartupHelper(HINSTANCE__* hInstance = 0x00000036, const unsigned short* lpszCmdLine = 0x00000000) Line: 357, Byte Offsets: 0x90   C++
MyFirstSmartDeviceApp.exe!mainWCRTStartup(HINSTANCE__* hInstance = 0x00000036, HINSTANCE__* hInstancePrev = 0x00000000, unsigned short* lpszCmdLine = 0x05fd000b, C++
0x4003b3f8
```

Figure 5: Debugging in Visual Studio 2005

4 Developing a simple C console program

Choose >>Win32 Smart Device Project<<.



Win32 Smart Device Project Wizard - deleteme1

Project Settings

Application type:

- ○ Windows application
- ● Console application
- ○ DLL
- ○ Static library

Additional options:

- ☐ Empty project
- ☐ Export symbols
- ☑ Precompiled header

Add support for:

- ☐ ATL
- ☐ MFC

Figure 6: Creating a Win32 console application

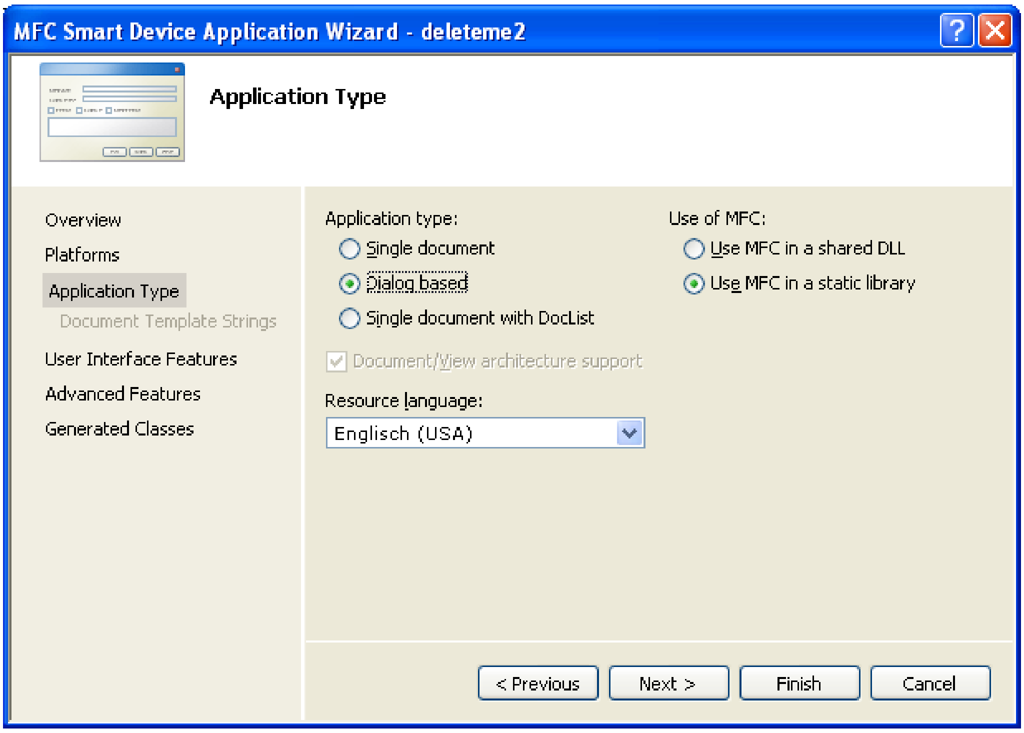
Demo application: Create Thread to print the current time to a console

```
#include "stdafx.h"
#include <windows.h>
#include <commctrl.h>

/*
 *      Simple Win32 Thread to print the system time and the
 *      local time to a console
 */
DWORD WINAPI printTheTime(LPVOID lParam)
{
    SYSTEMTIME st, lt;
    GetSystemTime(&st);
    GetLocalTime(&lt);
    printf("The system time is: %02d:%02d\n", st.wHour, st.wMinute);
    printf(" The local time is: %02d:%02d\n", lt.wHour, lt.wMinute);
    return 0;
}//printTheTime

/*
 *      Entry point
 */
int _tmain(int argc, _TCHAR* argv[])
{
        HANDLE hT = CreateThread(NULL,0,printTheTime,NULL,0,NULL);
        WaitForSingleObject(hT,INFINITE);
        return 0;
}//_tmain
```

Listing 1: Create Thread to print the current time to a console

5 Developing a simple C++ program with MFC

Choose >>MFC Smart Device Application<< and select the respective SDK.



MFC Smart Device Application Wizard - deleteme2

Application Type

Application type:

○ Single document

● Dialog based

○ Single document with DocList

☑ Document/View architecture support

Resource language: Englisch (USA)

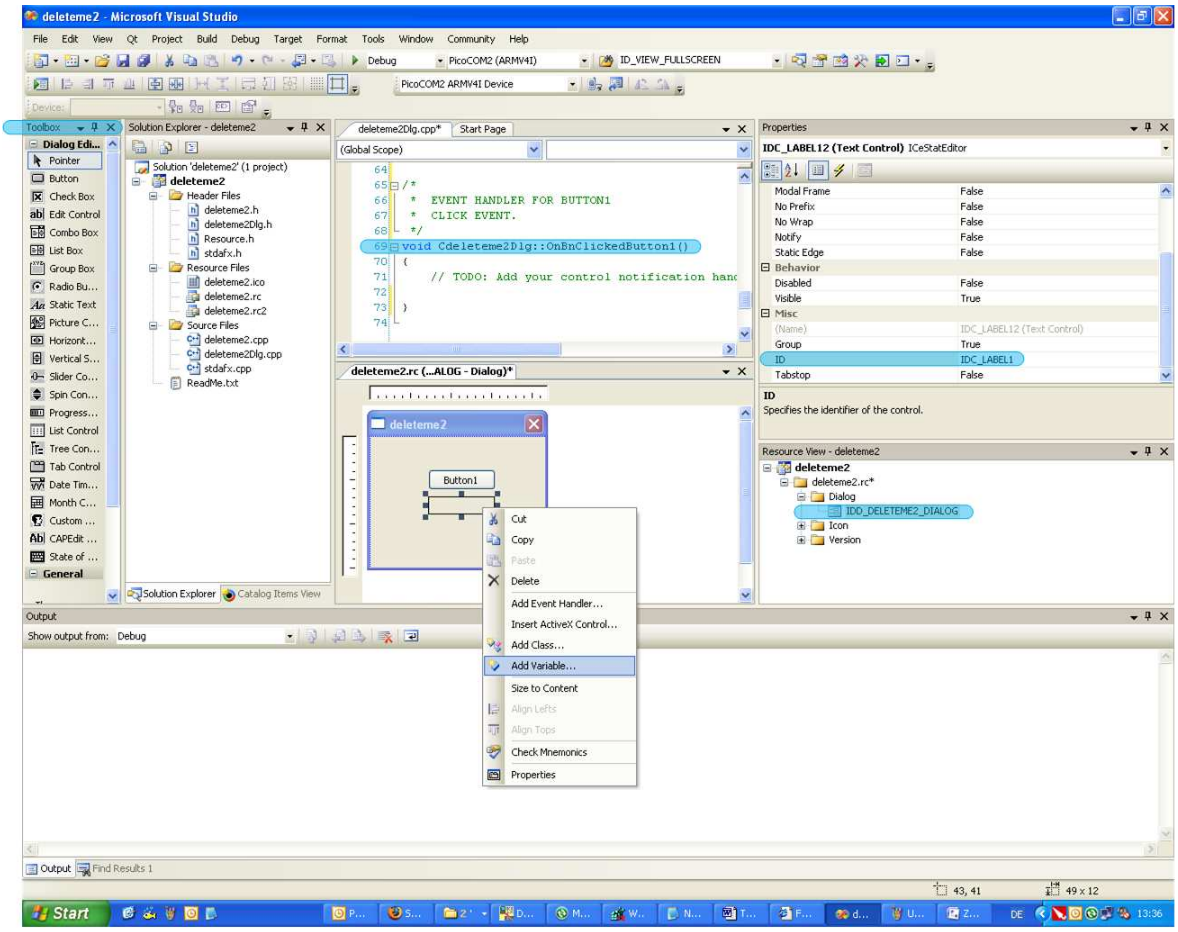
Use of MFC:

○ Use MFC in a shared DLL

● Use MFC in a static library

Figure 7: MFC Smart Device Application type selection

A. Using the Toolbox and the Ressource View



```
/*
 *  EVENT HANDLER FOR BUTTON1
 *  CLICK EVENT.
 */
void Cdeleteme2Dlg::OnBnClickedButton1()
{
    // TODO: Add your control notification hanc

}
```

Properties

IDC_LABEL12 (Text Control) ICeStatEditor

Modal Frame	False
No Prefix	False
No Wrap	False
Notify	False
Static Edge	False
Behavior	
Disabled	False
Visible	True
Misc	
(Name)	IDC_LABEL12 (Text Control)
Group	True
ID	IDC_LABEL1
Tabstop	False

ID

Specifies the identifier of the control.

Figure 8: MFC: Using Toolbox and Ressource View

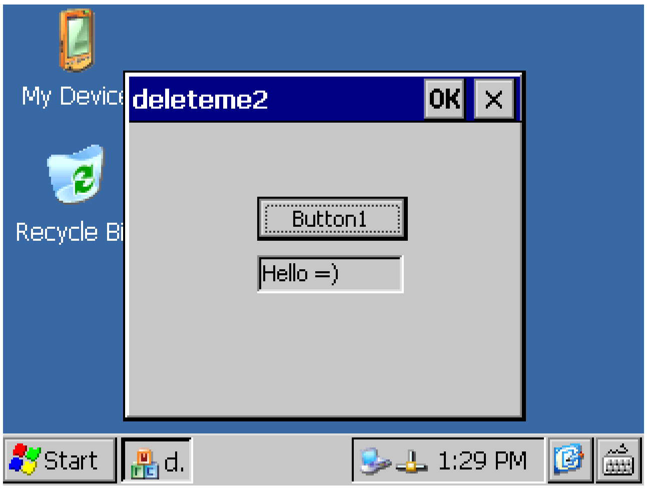
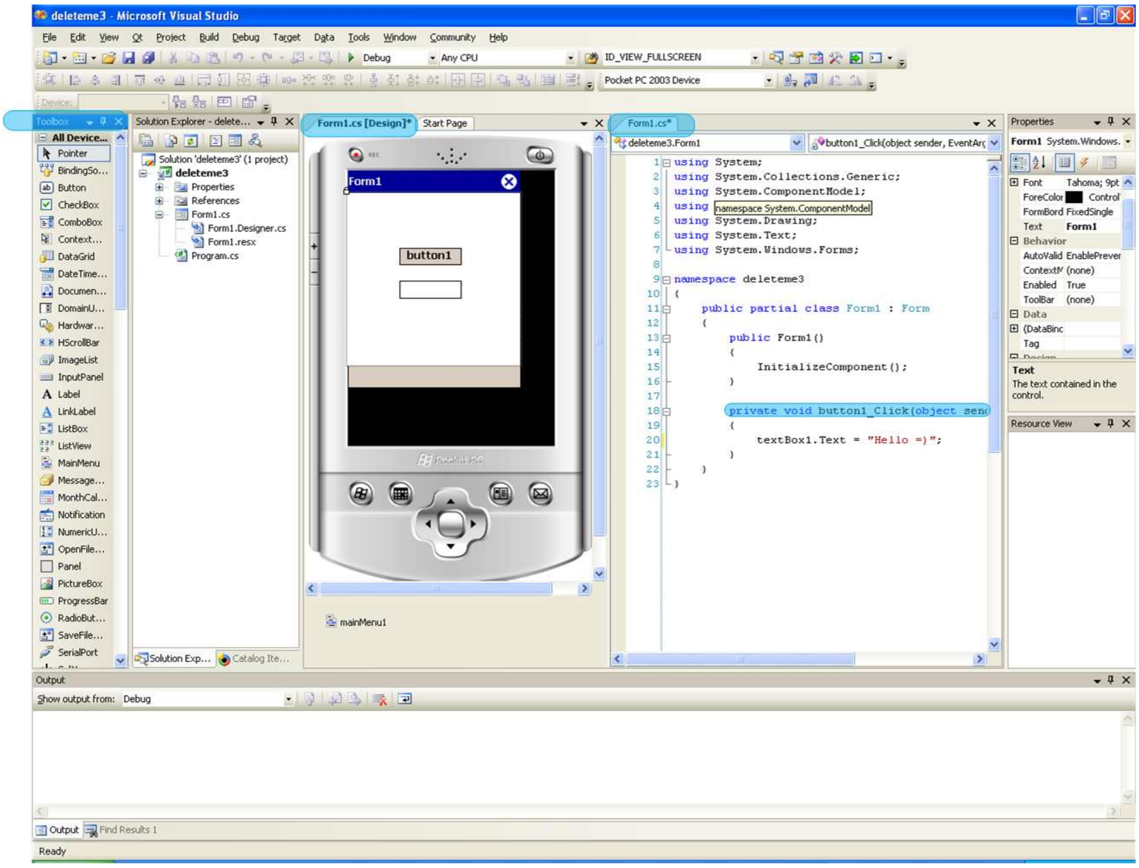


Figure 9: Screenshot: Windows Embedded CE MFC smart device app

6 Developing a simple C# application

Choose >>Other Languages<< → >>Visual C#<< → >>Smart Device<< → >>Device Application<<



deleteme3.Form1 — button1_Click(object sender, EventArg

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using namespace System.ComponentModel
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace deleteme3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object send
        {
            textBox1.Text = "Hello =)";
        }
    }
}
```

Figure 10: C#: Develop an application

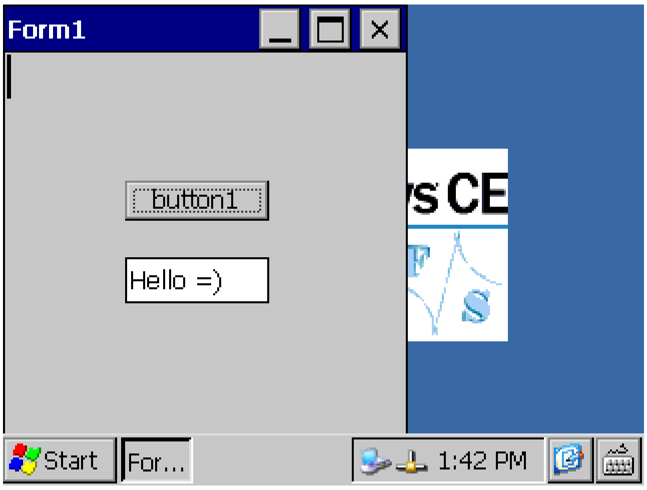


Figure 11: Screenshot: Windows Embedded CE C# smart device app

7 Writing to serial port from C++ and from C#

A. From C / C++:

```
#include "stdafx.h"
#include <windows.h>
#include <commctrl.h>
int _tmain(int argc, _TCHAR* argv[])
{
  HANDLE hCom;
  hCom = CreateFile(_T("COM2:"),GENERIC_WRITE|GENERIC_READ,
            0,NULL, OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL, NULL );
  if(INVALID_HANDLE_VALUE == hCom)
  {
    RETAILMSG(1,(L"can not open com port\r\n"));
    _tprintf(L"can not open com port\r\n");
    return FALSE;
  }
  /* Get / Set CommState */
  DCB cDcb;
  cDcb.DCBlength = sizeof( cDcb );
  GetCommState(hCom,&cDcb);
  cDcb.ByteSize = 8;
  cDcb.Parity = NOPARITY;
  cDcb.StopBits = ONESTOPBIT;
  cDcb.BaudRate = CBR_38400;
  cDcb.fOutxCtsFlow = 0;
  cDcb.EvtChar = 0x0A;
  SetCommState(hCom,&cDcb);
  /* Write to port */
  char *szMessage = "Hello World from PicoCOM2...";
  char *szMess2 = "\r\n:-)";
  DWORD dwLen = strlen(szMessage);
  DWORD dwLen2 = strlen(szMess2);
  DWORD dwNumR = 0;
  WriteFile(hCom,(LPVOID) szMessage,dwLen,&dwNumR,NULL);
  Sleep(3000);
  WriteFile(hCom,(LPVOID) szMess2,dwLen2,&dwNumR,NULL);
  CloseHandle(hCom);
return 0;
}//_tmain
```

Listing 2: Writing to serial port from C / C++

B. From C#:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO.Ports;
using System.Threading;
namespace NET_COM_Cons
{
    class Program
    {
        static void Main(string[] args)
        {
            SerialPort com2 = new SerialPort("COM2", 38400
            , Parity.None, 8, StopBits.One);
            com2.Open();
            com2.Write("Hello World from PicoCOM2 (.NET)");
            Thread.Sleep(3000);
            com2.Write("\r\n:-)");
            com2.Close();
        }
    }
}  }
```

Listing 3: Writing to serial port from C#

8 Appendix

Listings

Listing 1: Create Thread to print the current time to a console 7
Listing 2: Writing to serial port from C / C++ 11
Listing 3: Writing to serial port from C# 11

Figures

Figure 1: Creating a new project 2
Figure 2: Choosing the project type 3
Figure 3: Selecting the respective SDK 4
Figure 4: Connect VS2005 to your platform 5
Figure 5: Debugging in Visual Studio 2005 6
Figure 6: Creating a Win32 console application 7
Figure 7: MFC Smart Device Application type selection 8
Figure 8: MFC: Using Toolbox and Ressource View 9
Figure 9: Screenshot: Windows Embedded CE MFC smart device app 9
Figure 10: C#: Develop an application 10
Figure 11: Screenshot: Windows Embedded CE C# smart device app 10