Digi ConnectCore 9C/Wi-9C for Windows Embedded CE 6.0 User's Guide

90000849_B

Digi International Inc.

11001 Bren Road East

Minnetonka, MN 55343 (USA)

☎ +1 877 912-3444 or +1 952 912-3444

http://www.digi.com

Contents

1. Concepts

Developing applications for embedded systems differs from developing them for a desktop computer. Embedded-system applications involve several more elements than the applications themselves, such as the operating system and any necessary customization of it, the hardware drivers, the file system, and other elements. This topic introduces the software elements of an embedded system and the development environment needed to create them.

This topic provides an overview of the software elements of an embedded system and the development environment needed to create them.

1.1. Windows Embedded CE concepts

Embedded systems are ubiquitous. These dedicated small computers are present in communications systems, transportation, manufacturing, detection systems, and many machines that make our lives easier.

Windows Embedded CE is a componentized operating system designed to power small-footprint devices and help get them to market fast. It provides a wide variety of technology components and pre-existing templates for quickly building hard real-time commercial and consumer electronics devices.

1.1.1. Cross-compilation

Whenever code is generated for an embedded target on a development system with a different microprocessor architecture, a cross-development environment is needed. A cross-development compiler is one that executes in the development system, for example, an x86 PC, but generates code that executes in a different processor, for example, if the target is ARM.

Windows Embedded CE provides the cross-development toolchain for ARM architectures, including the compiler, linker, assembler, and libraries needed to generate software for the supported platforms.

1.1.2. Boot loader

A boot loader is a small piece of software that executes soon after a computer powers up. On a desktop PC, the boot loader resides on the master boot record (MBR) of the hard drive and is executed after the PC’s Basic Input Output System (BIOS) performs system initialization. The boot loader passes system information to the kernel (for instance, which hard drive partition to mount as root) and then executes the kernel.

In an embedded system, the boot loader’s role is more complicated because these systems do not have a BIOS to perform the initial system configuration. While the low-level initialization of the microprocessor, memory controllers, and other board-specific hardware varies from board to board and CPU to CPU, the initialization must be performed before a kernel image can execute.

At a minimum, a boot loader for an embedded system performs these functions:

- Initializing the hardware, especially the memory controller
- Providing boot parameters for the operating system image
- Starting the operating system image

Most boot loaders also provide convenient features that simplify development and update of the firmware, including:

- Reading and writing arbitrary memory locations
- Uploading new binary images to the board's RAM over a serial line or Ethernet
- Copying binary images from RAM to the flash memory

1.1.3. Kernel

The kernel is the fundamental part of an operating system. It is responsible for managing the resources and the communication between hardware and software components.

The kernel offers hardware abstraction to the applications and provides secure access to the system memory. It also includes an interrupt handler, which handles all requests or completed I/O operations.

1.1.4. File system

Operating systems rely on a hierarchical set of files and directories. The top of the hierarchical file tree is the file system, which contains the files and directories critical for system operation and programs for booting the system.

1.1.5. OS design

In Windows Embedded CE, the kernel and the file system form a whole. This unique element is called “OS design”. Note that this documentation sometimes refers to the OS design simply as “the kernel”.

1.1.6. Applications

Software applications are programs that use the capabilities and resources of a computer to do tasks. Applications use hardware devices by communicating with device drivers, which are part of the kernel.

1.1.7. Board Support Package (BSP)

In Windows Embedded CE, a BSP is a collection of files, drivers, OEM Adaptation Layers, and hardware abstraction layers (HALs) that have been created for a specific hardware platform. The BSP reduces the time to market phase for software, leveraging Microsoft Windows Embedded CE 6.0 running on Digi modules.

1.1.8. Software Development Kit (SDK)

An SDK is a collection of objects and methods that allow programmatic access to compiled and/or proprietary software. An application that is based on a certain SDK runs on any device that contains that SDK's components.

1.1.9. Projects and Solutions

In Windows Embedded CE, a *project* is a folder that contains all the software components for specific functionality. A project, for example, can be an application or a kernel for a given platform.

Solutions are containers of several related projects. For example, a solution can contain a kernel project and several applications projects.

1.2. Structure of Windows Embedded CE

The Windows Embedded CE software package on the CD contains all the necessary software components for developing applications with the Windows Embedded CE 6.0 hardware platform.New software is developed using Visual Studio 2005 for Digi embedded modules. Here is a review of the structure and software components of Windows Embedded CE.

1.2.1. Main directories

After installation, the Windows Embedded CE root folder (defined by an automatically defined variable **%_WINCEROOT%**) has this directory tree:

Directory	Description
%_WINCEROOT%\PLATFORM	Board specific modules; for example: DEVICEEMULATOR Samsung's emulator platform MAINSTONEIII Intel MainStone III platform CCX9C Digi ConnectCore 9C and Wi-9C platforms
%_WINCEROOT%\PLATFORM\COMMON\SRC\SOC\	CPU common functions, such as OAL, drivers, etc.
%_WINCEROOT%\OSDesigns	All project-specific parts; for example: MyCCX9C One ConnectCore 9C/Wi-9C based project.
%_WINCEROOT%\PUBLIC\COMMON	All common adjustments and drivers. Be careful with changes in this directory and its subfolders. A change in one of the sources in the common directory affects all platforms.

1.3. Platform Builder

Platform Builder is the environment for developing operating system designs based on the available BSPs. Platform Builder integrates into Microsoft Visual Studio 2005 as a plug-in.

1.4. License background

The BSP includes the full source code for the BSP and all the drivers related to the ConnectCore 9C/Wi-9C platforms.

The source code can be used freely on Windows Embedded CE images created to run on ConnectCore 9C/Wi-9C based products. **The use, modification or distribution of the source code on images created to run on other products is forbidden**.

For detailed information about licensing and royalties, read the License Agreements (**License_Agreements.rtf**) in the ConnectCore BSP for Windows Embedded CE 6.0 CD-ROM, or contact your sales representative.

For information about which Windows Embedded CE run-time license is required, see topic 14.7.

1.5. Conventions

This document uses these conventions, frames, and symbols to display information:

Convention	Use
`Style`	New terms and variables in commands, code, and other input.
`Style`	In examples, to show the contents of files, the output from commands. In text, the C code.
	Variables to be replaced with actual values are shown in italics.
Style	For menu items, dialogs, tabs, buttons, and other controls.
	In examples, to show the text that should be entered literally.
$	A prompt that indicates the action is performed in the host computer.
\>	A prompt that indicates the action is performed in the target device.
Menu name > option	A menu followed by one or more options; for example, **File > New**.

This manual also uses these frames and symbols:

A warning that helps to solve or to avoid common mistakes or problems.

A hint that contains useful information about a topic.

```
$   A host computer session.
    Bold text indicates what must be input.
```

```
\> A target session.
\> Bold text indicates what must be input.
```

1.6. Abbreviations

AES	Advanced Encryption Standard
AP	Access Point
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
BIOS	Basic Input Output System
CCMP	Counter Mode with Cipher Block Chaining Message Authentication Code Protocol
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DSSS	Direct-Sequence Spread Spectrum
EULA	End-User License Agreement
FPGA	Field-Programmable Gate Array
FTP	File Transfer Protocol
GPIO	General Purpose Input/Output
HAL	Hardware Adaptation Layer
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute for Electrical and Electronics Engineers
IOCTL	I/O Control
IP	Internet Protocol
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LSB	Less Significant Bit
MBR	Master Boot Record
MSB	Most Significant Bit
NVRAM	Non-volatile RAM
OAL	OEM Adaptation Layer
OEM	Original Equipment Manufacturer
OHCI	Open Host Controller Interface
OS	Operating System
PC	Personal Computer
PSK	Pre-Shared Key
RAM	Random Access Memory
RTC	Real-Time Clock
SPI	Serial Peripheral Interface
SSID	Service set identifier
TFTP	Trivial File Transfer Protocol

TKIP	Temporal Key Integrity Protocol
USB	Universal Serial Bus
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
WZCSAPI	Wireless Zero Config Service API
WZCSVC	Wireless Zero Config Service

2. Developing applications with Visual Studio 2005

This topic describes how to create, build, transfer, and debug Windows Embedded CE applications using Microsoft Visual Studio 2005 software. Application can also be developed using Visual C++, Visual Basic, or Visual C#.

2.1. Create the project

2.1.1. Visual Basic application

2.1.1.1. Create the Visual Basic project

This topic, creates a sample Hello World application in Visual Basic and Visual C#. These projects will be created within the bounds of one *solution* named **SampleSolution**.

1. In Visual Studio 2005 select **File > New > Project**.

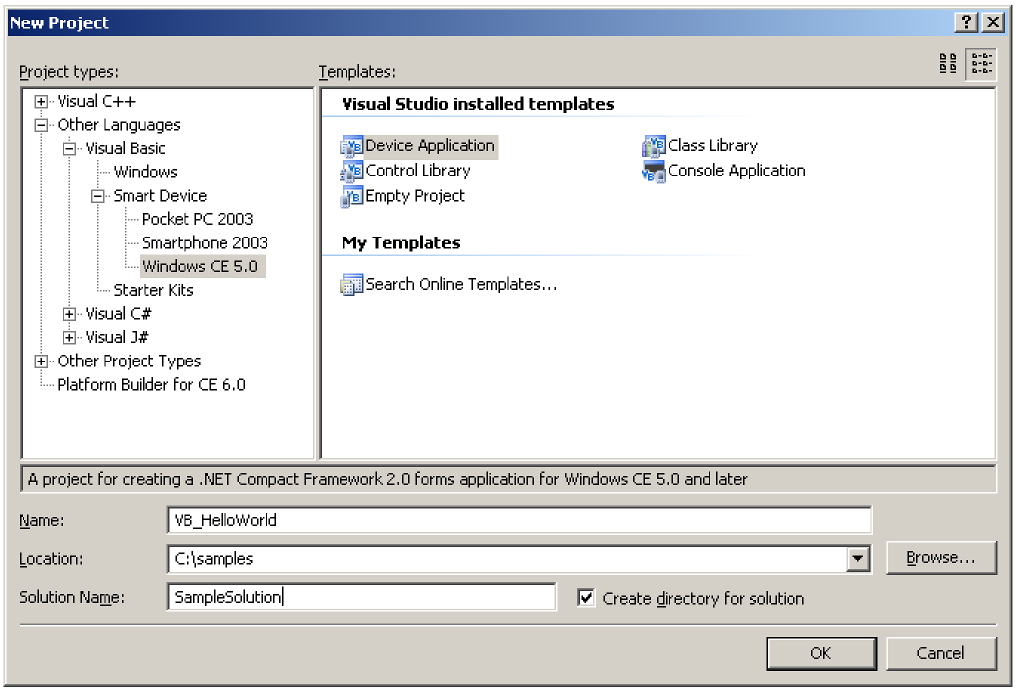
   The **New Project** dialog opens.

2. Under Project Types, expand **Other Languages > Visual Basic > Smart Device > Windows CE 5.0**

*Throughout this document, **Visual Studio** refers to Windows CE 5.0 instead of version 6.0, but the 5.0 selection is fully compatible with 6.0.*

3. From the **Templates** section, select **Device Application**.
4. At the bottom of the dialog, enter this information:
   - The name of the Visual Basic project (**VB_HelloWorld**).
   - The path to the location in which to store the solution.
   - The name of the solution (**SampleSolution**) in which to store this project.

   Then click **OK**. This creates a folder named **SampleSolution** in your location path with a subfolder named **VB_HelloWorld**, which contains the Visual Basic project with an empty form and some basic source files.


New Project

Project types:
- Visual C++
- Other Languages
  - Visual Basic
    - Windows
    - Smart Device
      - Pocket PC 2003
      - Smartphone 2003
      - Windows CE 5.0
    - Starter Kits
  - Visual C#
  - Visual J#
- Other Project Types
- Platform Builder for CE 6.0

Templates:

Visual Studio installed templates
- Device Application
- Control Library
- Empty Project
- Class Library
- Console Application

My Templates
- Search Online Templates...

A project for creating a .NET Compact Framework 2.0 forms application for Windows CE 5.0 and later

Name: VB_HelloWorld

Location: C:\samples

Solution Name: SampleSolution

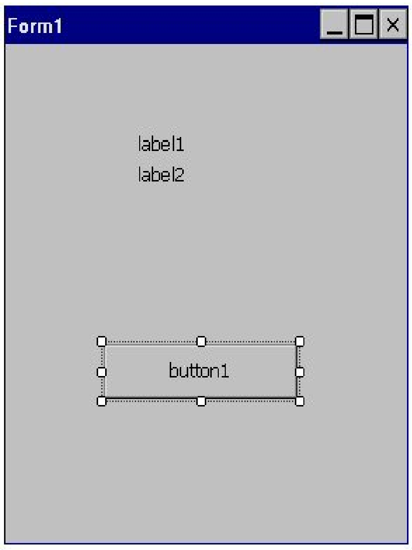
[✓] Create directory for solution

2.1.1.2. Generate the interface

Now add some content to the form.

1. To open the toolbox, select **View > Toolbox**.
2. Drag and drop a button and two labels into the form.

   The interface looks like this:



3. Right-click the button, select **Properties**, and change the text in the box to **Press me**. Leave the labels with their default text.

2.1.1.3. Generate the source code

Now put some code into the button's click method.

To open the button's click method source code, double-click the button on the form. Some code is displayed:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    End Sub
End Class
```

Label1 should display "**Hello World**" and **Label2** should display a counter's value that increases with each of the button. Add this code (in bold):

```
Public Class Form1
    Dim counter As Integer
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Label1.Text = "Hello World!"
        Label2.Text = counter.ToString
        counter = counter + 1
    End Sub
End Class
```

Save the file and close the editor.

2.1.1.4. Build the Visual Basic application

To build the sample application, select **Build > Build VB_HelloWorld**.

When the build finishes, the output window shows that the build was successful:

```
VB_HelloWorld                                                              ->
C:\samples\SampleSolution\VB_HelloWorld\bin\Debug\VB_HelloWorld.exe
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

This output also shows where the executable image has been placed; in this example, the location is:

C:\samples\SampleSolution\VB_HelloWorld\bin\Debug\VB_HelloWorld.exe

2.1.2. Visual C# application

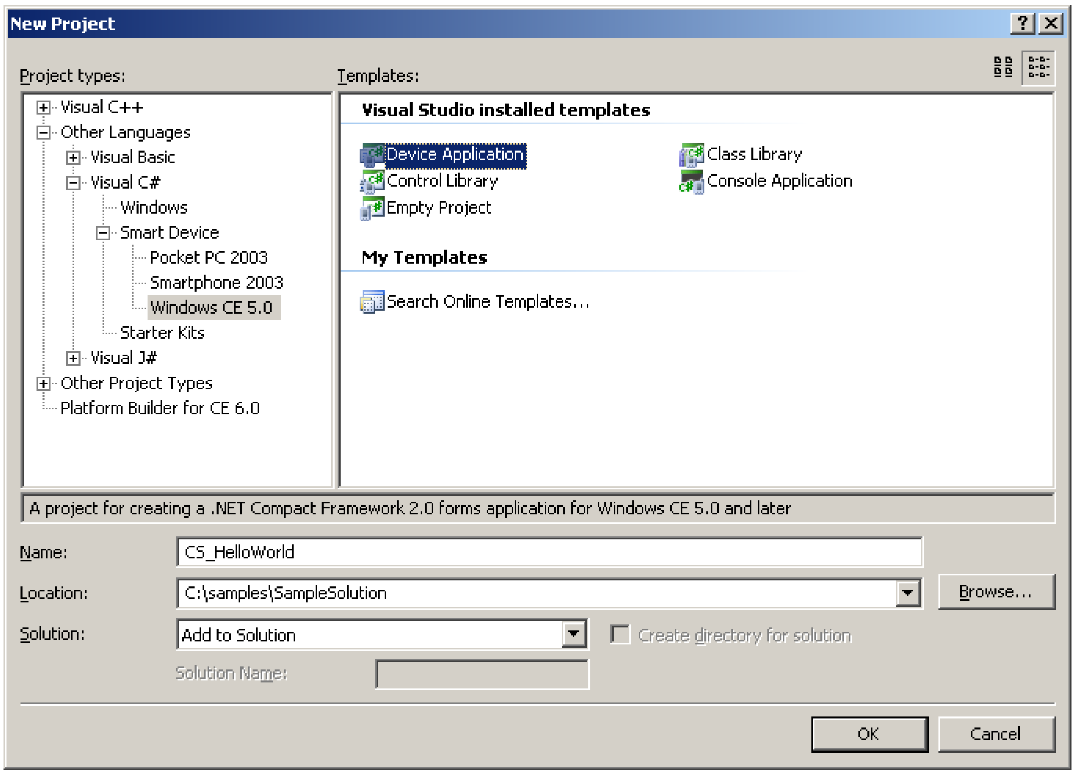
2.1.2.1. Create the C# project

1. Select **File > New > Project**.

   The New Project dialog opens.

2. Expand **Other Languages > Visual C# > Smart Device > Windows CE 5.0**.
3. In the Templates section, select **Device Application**.
4. Enter the name of the Visual C# project (**CS_HelloWorld**).
5. In the Solution combo box, to add the new project to the solution you previously created (**SampleSolution**), select **Add to solution**.

   This step populates the Location field with the path to **SampleSolution**.



A project for creating a .NET Compact Framework 2.0 forms application for Windows CE 5.0 and later

Name: CS_HelloWorld

Location: C:\samples\SampleSolution

Solution: Add to Solution

6. Click **OK**.

2.1.2.2. Generate the interface

Generate a similar interface to the one created in topic 2.1.1.2.

2.1.2.3. Generate the source code

Now put some code into the button's click method:

1. Double-click the button on the form to open its click method source code. This code is displayed:

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

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

        private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

2. **Label1** should display "**Hello World**" and **Label2** should display a counter's value that increases with each click of the button. Add this code (in bold):

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

namespace HelloWorld
{
    public partial class Form1 : Form
    {
        int counter = 1;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
           label1.Text = "Hello World!";
           label2.Text = counter.ToString ();
           counter++;
        }
    }
}
```

3. Save the file and close the editor.

2.1.2.4. Build the C# application

To build the sample application, select **Build > Build CS_HelloWorld**.

When the build finishes, the output window shows that the build was successful.

```
Compile complete -- 0 errors, 0 warnings
CS_HelloWorld                                                                   ->
C:\samples\SampleSolution\CS_HelloWorld\bin\Debug\CS_HelloWorld.exe
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

This output also shows where the executable image has been placed; in this example:

C:\samples\SampleSolution\CS_HelloWorld\bin\Debug\CS_HelloWorld.exe

2.1.3. Visual C++ application

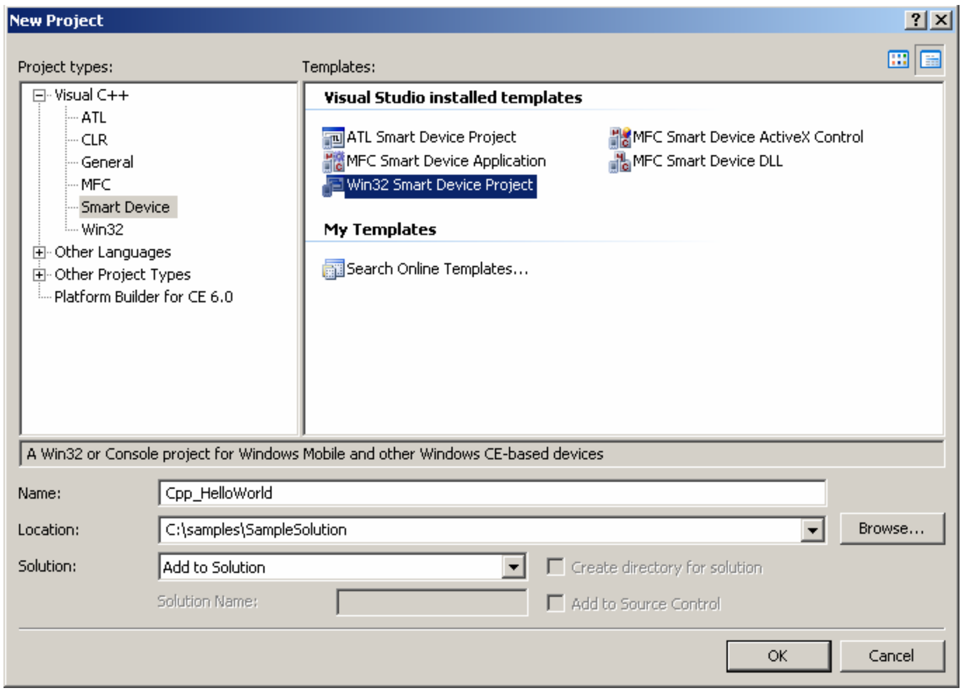
2.1.3.1. Create the C++ project

1. Select **File > New > Project**.

   The **New Project** dialog opens.

2. Expand **Visual C++ > Smart Device**.
3. In the **Templates** section, select **Win32 Smart Device Project**.
4. Enter the name of the Visual C++ project (**Cpp_HelloWorld**).
5. In the **Solution** combo box, to add the new project to the solution you previously created (**SampleSolution**), select **Add to solution**.

   This step populates the Location field with the path to **SampleSolution**.



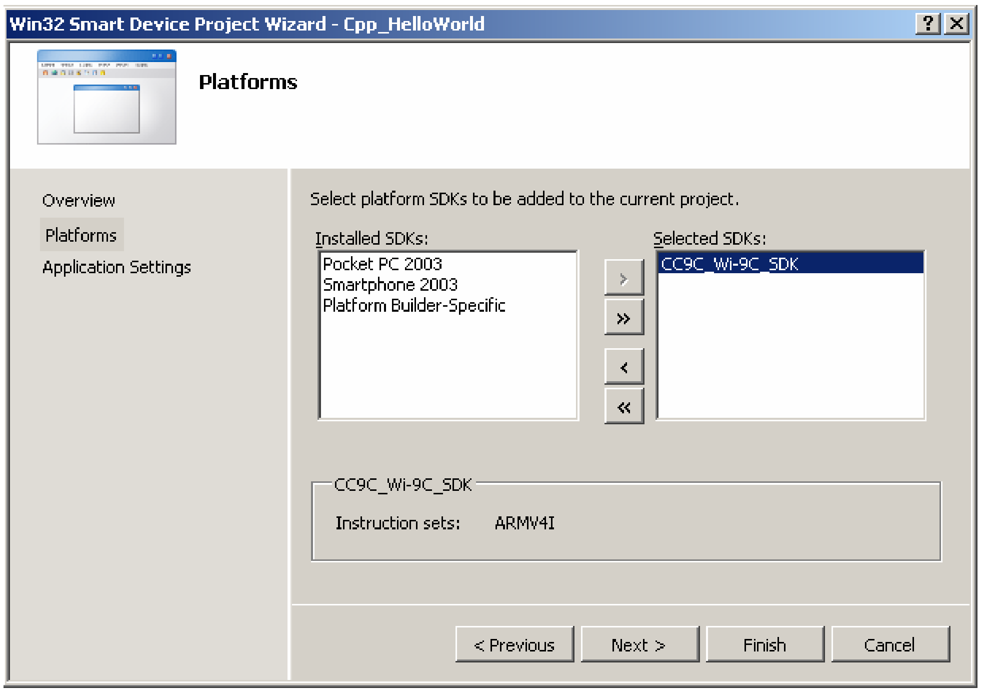
A Win32 or Console project for Windows Mobile and other Windows CE-based devices

Name: Cpp_HelloWorld

Location: C:\samples\SampleSolution

Solution: Add to Solution

6. Click **OK**.
7. In Platform window, select the platform SDK to be added to the project. Select Digi SDK (**CC9C_Wi-9C_SDK**) or the custom SDK created for your platform.


Win32 Smart Device Project Wizard - Cpp_HelloWorld

Platforms

Select platform SDKs to be added to the current project.

Installed SDKs:
Pocket PC 2003
Smartphone 2003
Platform Builder-Specific

Selected SDKs:
CC9C_Wi-9C_SDK

CC9C_Wi-9C_SDK
Instruction sets: ARMV4I

8. Click **Finish**.

2.1.3.2. Build the C++ application

To build the sample application, select **Build > Build Cpp_HelloWorld**.

When the build finishes, the output window shows that the build was successful.

```
Compile complete -- 0 errors, 0 warnings
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

In this example, the executable image has been placed in this location:

C:\samples\SampleSolution\Cpp_HelloWorld\CC9C_Wi-9C_SDK (ARMV4I)\Debug\Cpp_HelloWorld.exe

2.2. Build the solution

Although each project can be built separately, all the projects of a solution can be built as a whole.

To do so, go to **Build > Configuration Manager...** The **Configuration Manager** dialog lists all the projects of a solution. The projects that have the **Build** column checked for the active solution platform will be built when selecting **Build > Build solution**.

2.3. Deploy and debug applications

2.3.1. Device Transport configuration

To transfer and debug the applications, a *Device Transport configuration* must be established:

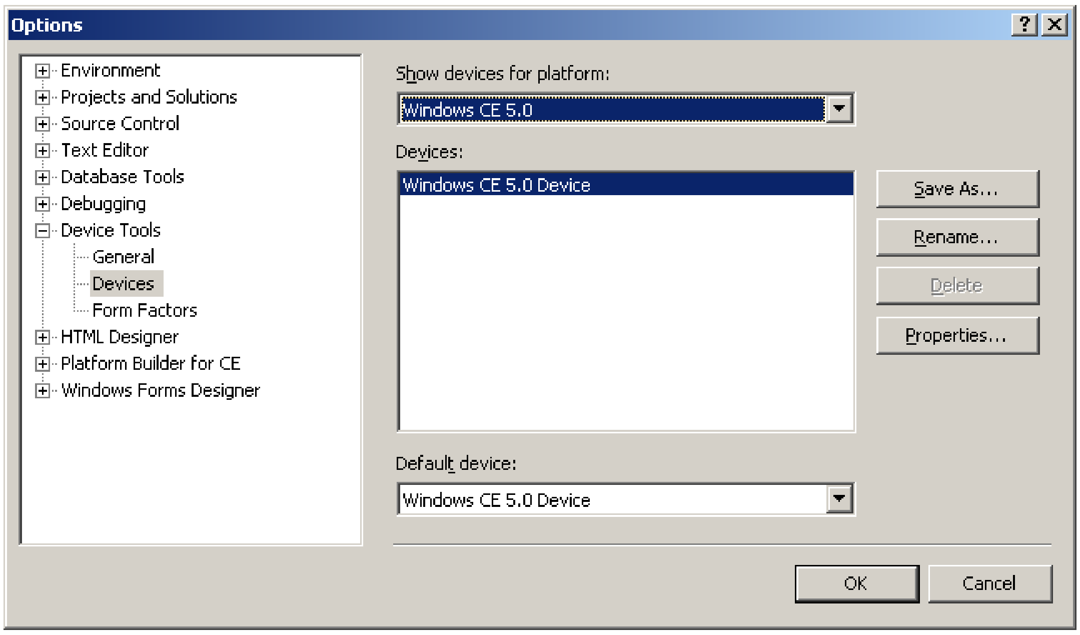
1. Select **Tools > Options**.

   The Options dialog opens.

2. Select **Device Tools > Devices**.

3. From the **Show devices for platform** pull-down menu, select **Windows CE 5.0**.

   In the Devices list box, select **Windows CE 5.0 Device**. Under Default device, select **Windows CE 5.0 Device**.



Show devices for platform: Windows CE 5.0

Devices: Windows CE 5.0 Device

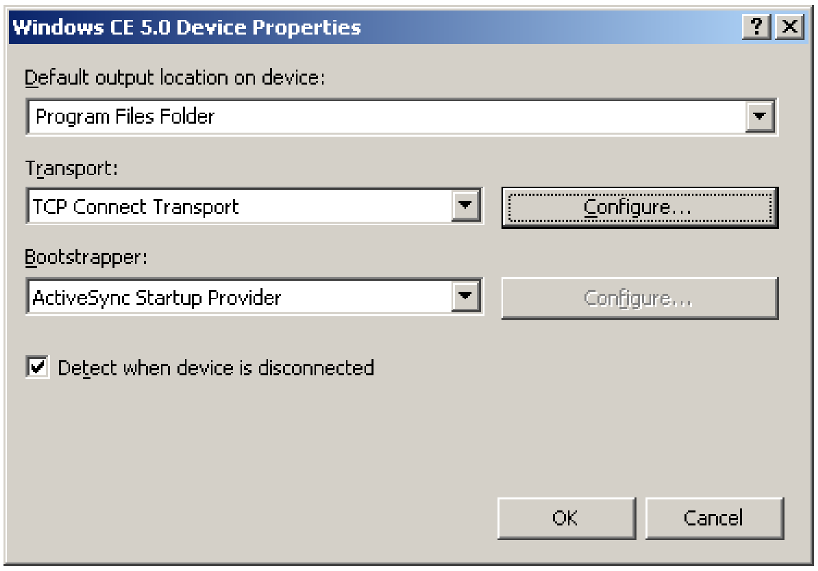
Default device: Windows CE 5.0 Device

Visual Studio refers to Windows CE 5.0 instead of version 6.0, but the 5.0 selection is fully compatible with 6.0.

4. Click **Properties**.

   The **Windows CE 5.0 Device Properties** dialog opens.

5. Select these values:
   - **Default output location**: Program Files Folder
   - **Transport**: TCP Connect Transport
   - **Bootstrapper**: ActiveSync Startup Provider



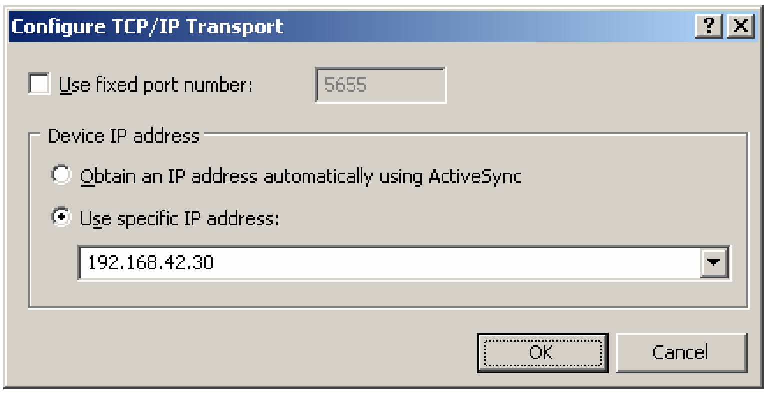
Default output location on device: Program Files Folder

Transport: TCP Connect Transport

Bootstrapper: ActiveSync Startup Provider

[✓] Detect when device is disconnected

6. Click the **Configure** button to the right of the **Transport** method pull-down menu to open the **Configure TCP/IP Transport** dialog.
7. Click **Use specific IP address**, enter the IP address of the target, and click **OK**.



[ ] Use fixed port number: 5655

Device IP address

○ Obtain an IP address automatically using ActiveSync

● Use specific IP address: 192.168.42.30

8. To accept the device transportation configuration, click **OK** in the rest of the open dialogs.

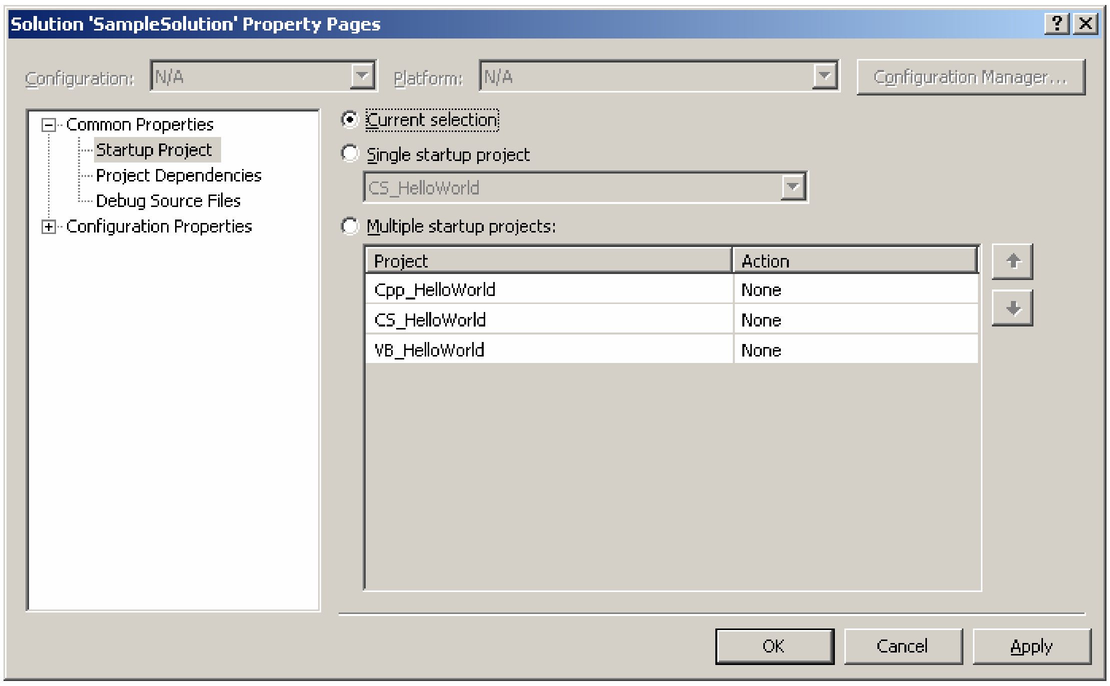
Visual Studio 2005 is now ready to transfer and debug any Windows Embedded CE application to the target board.

2.3.2. Set StartUp projects

By default, the **SampleSolution** project that has been created selects one of the projects (which appears in bold) as the one active for running and debugging.

To manually select a project to debug:

1. Right-click the **SampleSolution** item in the Solution Explorer and select **Set StartUp Projects.**
2. Select **Current Selection**, then click **OK**.


Solution 'SampleSolution' Property Pages

Configuration: N/A   Platform: N/A   Configuration Manager...

- Common Properties
  - Startup Project
  - Project Dependencies
  - Debug Source Files
- Configuration Properties

(●) Current selection
( ) Single startup project: CS_HelloWorld
( ) Multiple startup projects:

Project	Action
Cpp_HelloWorld	None
CS_HelloWorld	None
VB_HelloWorld	None

2.3.3. Connect to the device

Before applications can be debugged, two programs must be launched in the target to listen for debug connections:

- **conmanclient2.exe**
- **cmaccept.exe**

If the OS design was created using the ConnectCore 9C or Wi-9C template, these two applications are launched automatically. If the OS design was created using the Custom Device template, these applications must be included into the OS design. Topic 14.5 explains how to include and launch these applications.

To connect to the device:

1. In Visual Studio 2005, select **Tools > Connect to Device**.
2. Select **Windows CE 5.0 Device**, and click **Connect**.

   A dialog opens to show whether the connection was successful.
3. Close the dialog.

2.3.4. Start deploying and debugging the application

1. Select one of the application projects.
2. Select **Debug > Start debugging**.
3. Select **Windows CE 5.0 Device** again.

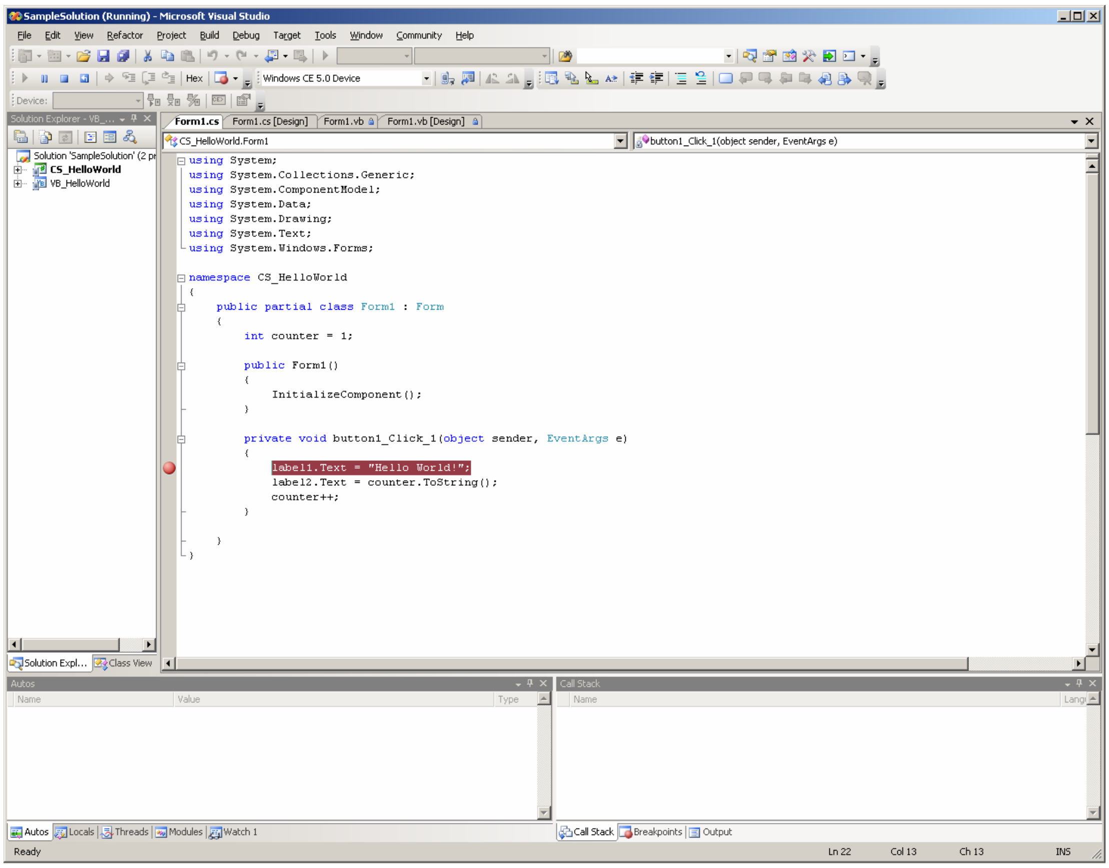
   To prevent the dialog from appearing again, deselect **Show me this dialog every time I deploy the application**.
4. Click **Deploy**. This transfers the application to the target and runs it for debugging. The Visual Studio perspective changes to Debug mode.

2.3.5. Add a breakpoint

To add a breakpoint, click the left border of the editor in any line of code. A red circle appears on the line that was clicked.

If problems occur accessing breakpoints, symbols may not have been loaded correctly. Stop debugging and try reloading the image.

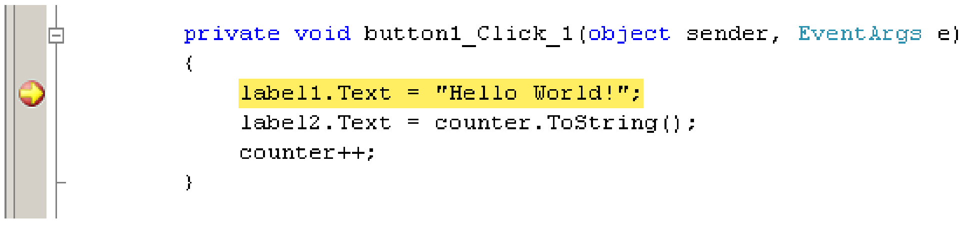
To remove a breakpoint, click again in the left border of the editor.



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

namespace CS_HelloWorld
{
    public partial class Form1 : Form
    {
        int counter = 1;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            label1.Text = "Hello World!";
            label2.Text = counter.ToString();
            counter++;
        }

    }
}
```

1. Add a breakpoint to any line within the button's click method.
2. Click the button on the application.

   The code stops at the breakpoint. The yellow arrow indicates the line where the program counter is.



```
private void button1_Click_1(object sender, EventArgs e)
{
    label1.Text = "Hello World!";
    label2.Text = counter.ToString();
    counter++;
}
```

2.3.6. Other debugging tools

There are other debugging tools used to watch variables, step over or into the code, modify variable values, view and edit memory positions. These and many other debugging options are accessible from the **Debug** menu.

2.4. Delete projects and solutions

To delete a project, right-click it and select **Remove** from the context menu.

To delete an entire solution, remove the solution folder manually.

3. Configure the Windows Embedded CE kernel

This topic describes how to create a Windows Embedded CE kernel for your hardware platform and how to customize the kernel. Customization allows for removing support for unneeded hardware of software services, resulting in a smaller image.

3.1. Create a new Platform Builder project

This topic creates a Platform Builder project in the sample solution created in topic 2.

Either use a standard template or customize one by individually selecting the components desired for the project. The components include communication services and networking; core OS services; file systems and data store; and so on.

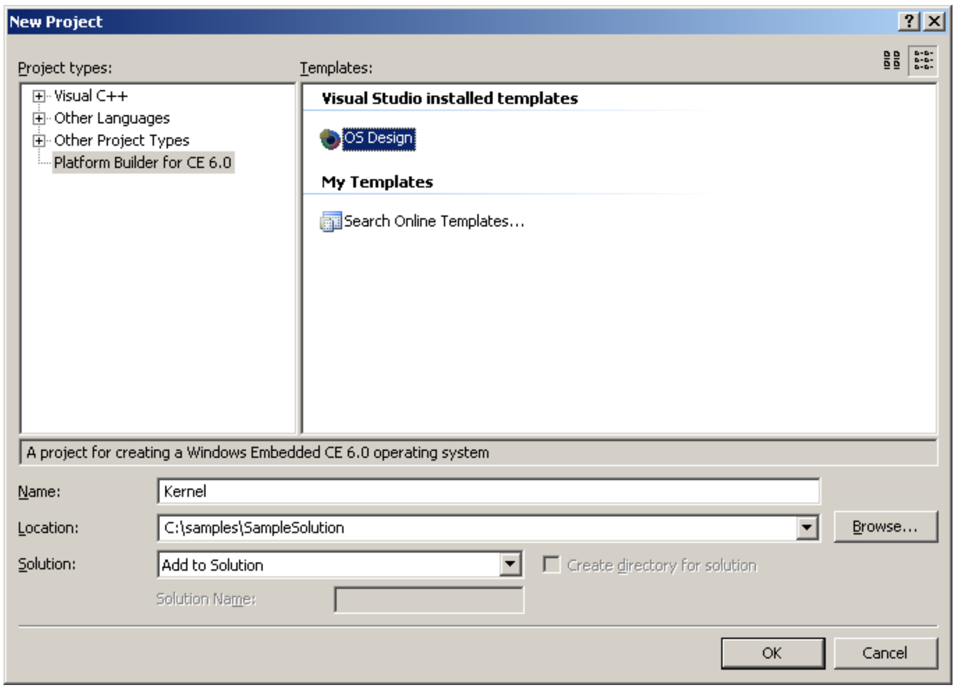
1. In Visual Studio 2005, select **File > New > Project**.

   The **New Project** dialog opens.

2. Do these steps:

   - Select **Platform Builder for CE**.
   - Select **OS Design** as the template.
   - Enter the name of the project; for example, **Kernel**.
   - In the **Solution** combo box, select **Add to Solution**.

     This step populates the **Location** field with the path to the currently open solution.


New Project

Project types:
- Visual C++
- Other Languages
- Other Project Types
- Platform Builder for CE 6.0

Templates:

Visual Studio installed templates
- OS Design

My Templates
- Search Online Templates...

A project for creating a Windows Embedded CE 6.0 operating system

Name: Kernel

Location: C:\samples\SampleSolution

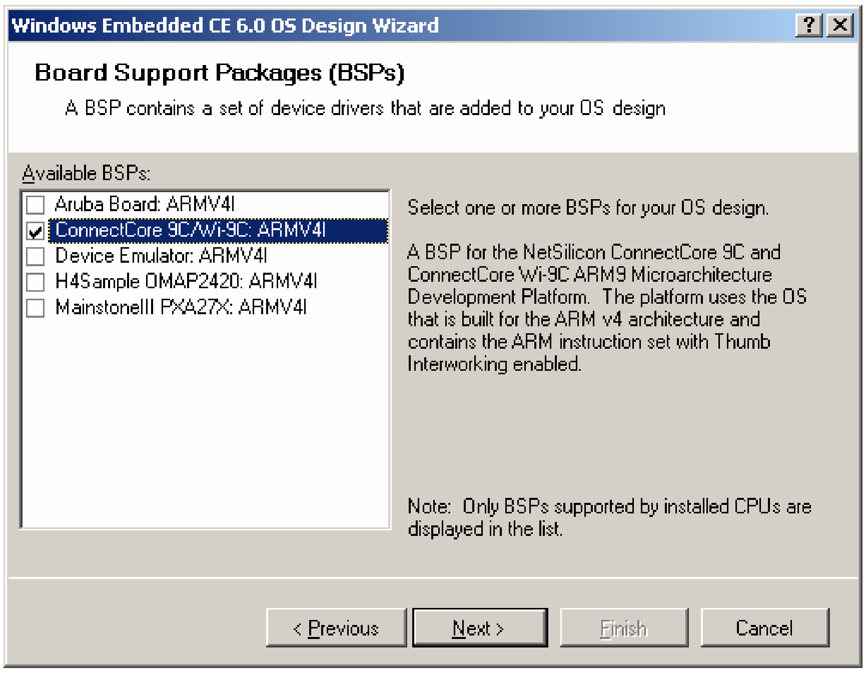
Solution: Add to Solution

Create directory for solution

Solution Name:

Then click **OK**.

3. From the available Board Support Packages, BSPs, select **ConnectCore 9C/Wi-9C: ARMV4I** and click **Next**.


Windows Embedded CE 6.0 OS Design Wizard

Board Support Packages (BSPs)

A BSP contains a set of device drivers that are added to your OS design

Available BSPs:

- [ ] Aruba Board: ARMV4I
- [✓] ConnectCore 9C/Wi-9C: ARMV4I
- [ ] Device Emulator: ARMV4I
- [ ] H4Sample OMAP2420: ARMV4I
- [ ] MainstoneIII PXA27X: ARMV4I

Select one or more BSPs for your OS design.

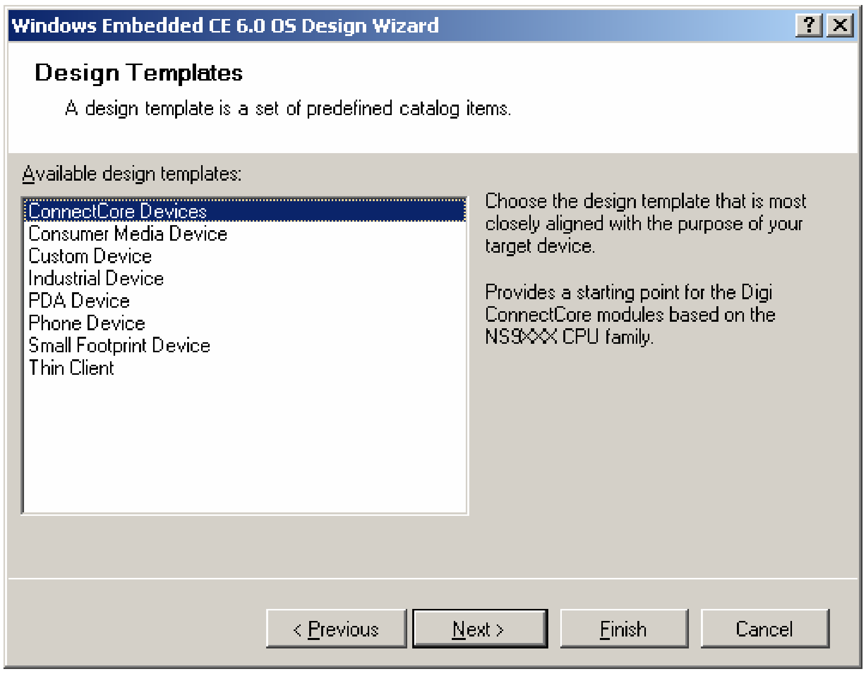
A BSP for the NetSilicon ConnectCore 9C and ConnectCore Wi-9C ARM9 Microarchitecture Development Platform. The platform uses the OS that is built for the ARM v4 architecture and contains the ARM instruction set with Thumb Interworking enabled.

Note: Only BSPs supported by installed CPUs are displayed in the list.

4. On the next wizard page, either select a design template or custom device configuration for the OS design:

   - To create a standard project for the module being used, select **ConnectCore Devices**.
   - To select every component individually, select **Custom Device** template project, and select the desired components. Click **Next**, then click **Finish**.

Components can be added and removed later. If additional components are needed because of dependencies, Platform Builder automatically resolves and includes them in the build.


Windows Embedded CE 6.0 OS Design Wizard

Design Templates

A design template is a set of predefined catalog items.

Available design templates:

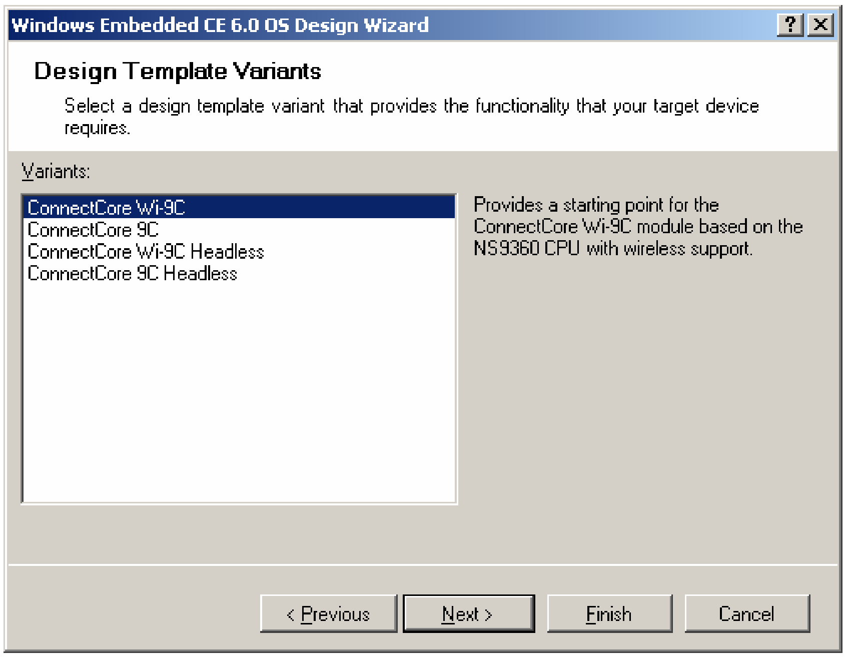
- ConnectCore Devices
- Consumer Media Device
- Custom Device
- Industrial Device
- PDA Device
- Phone Device
- Small Footprint Device
- Thin Client

Choose the design template that is most closely aligned with the purpose of your target device.

Provides a starting point for the Digi ConnectCore modules based on the NS9XXX CPU family.

5. On the next wizard page, select the template for the platform: either **ConnectCore 9C** or **ConnectCore Wi-9C** or their headless versions **ConnectCore 9C Headless** or **ConnectCore Wi-9C Headless**. Then click **Finish**.

Headless templates are designed for devices that have no user interface components or peripherals and must be accessed remotely over a network or serial connection. OS designs based in headless templates do not include graphics support or input devices such as a keyboard or mouse, resulting in a smaller image.


Windows Embedded CE 6.0 OS Design Wizard

Design Template Variants

Select a design template variant that provides the functionality that your target device requires.

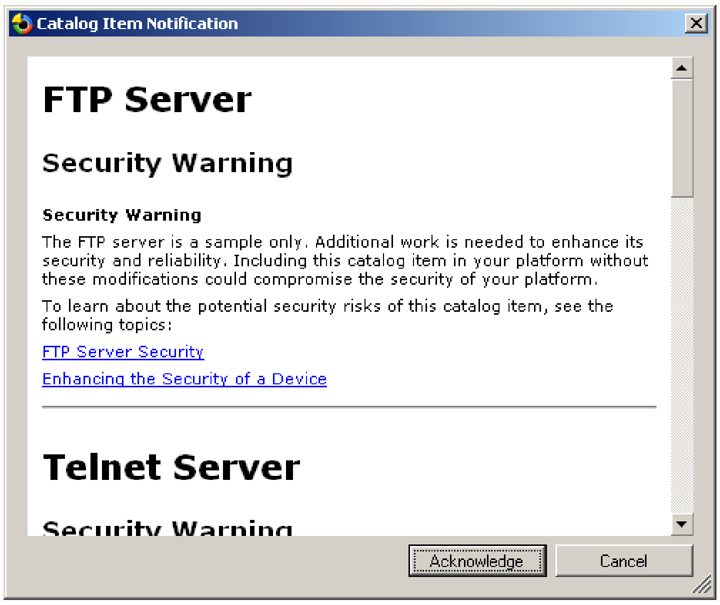
Variants:

- ConnectCore Wi-9C
- ConnectCore 9C
- ConnectCore Wi-9C Headless
- ConnectCore 9C Headless

Provides a starting point for the ConnectCore Wi-9C module based on the NS9360 CPU with wireless support.

6. After clicking **Finish**, a catalog item notification warning may be displayed. This warning is about security issues with the components that were selected. For example, if the FTP server

element is included, the notification warns that this is a possible security hole. Read this information carefully so you can solve any problems later. Then click **Acknowledge.**



FTP Server

Security Warning

Security Warning

The FTP server is a sample only. Additional work is needed to enhance its security and reliability. Including this catalog item in your platform without these modifications could compromise the security of your platform.

To learn about the potential security risks of this catalog item, see the following topics:

FTP Server Security

Enhancing the Security of a Device

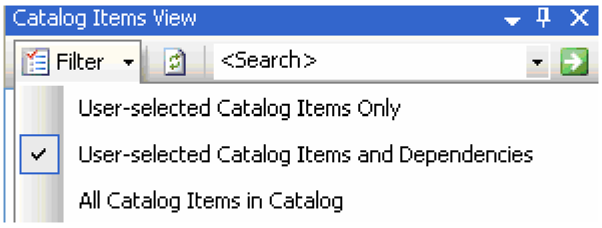
Telnet Server

Security Warning

3.2. Catalog view

The catalog view displays the complete list of components that can be added to a Windows CE kernel image. Each component is represented by a SYSGEN variable used during the build process to identify the system parts that need to be included into the image.

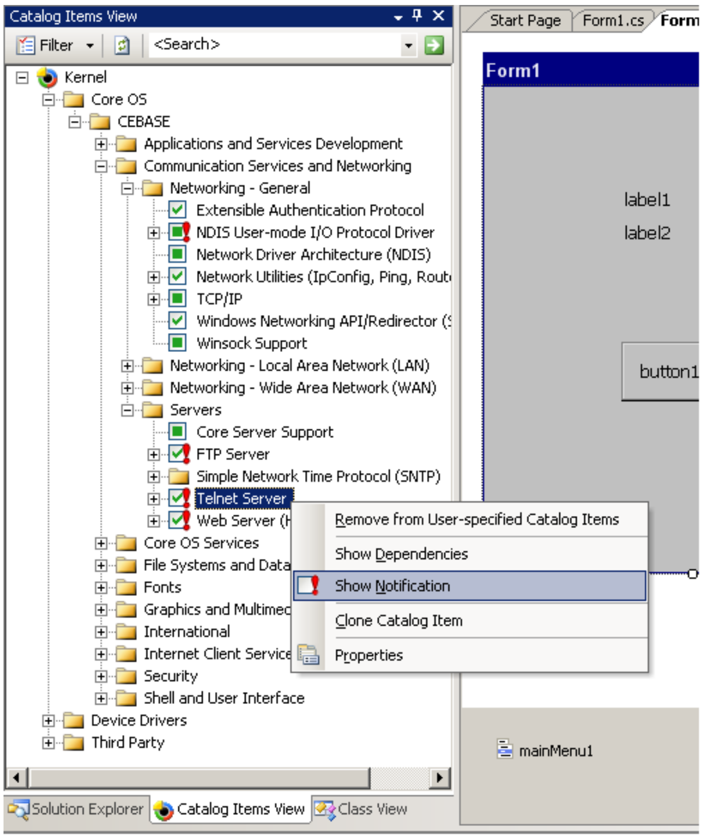
To view the catalog, select **View > Other Windows > Catalog Items View**. To see only the components that were selected, click **Filter** on the top of the view, and select **User-selected Catalog Items and Dependencies**.


Catalog Items View
Filter
<Search>
User-selected Catalog Items Only
User-selected Catalog Items and Dependencies
All Catalog Items in Catalog

Each component type has an icon, as shown in this table

Sign	Meaning
☑	Item selected by user (multiple choice)
✓	Item selected by user (one choice only)
🟩	Item automatically included due to dependencies with other included component
☒	Item excluded because of some incoherence between dependencies
❗	Item with an important warning notification

Items marked with a red exclamation mark have an important warning notification. These are the same security warnings displayed when creating the OS design. To display a notification, right-click the component and select **Show Notification**.



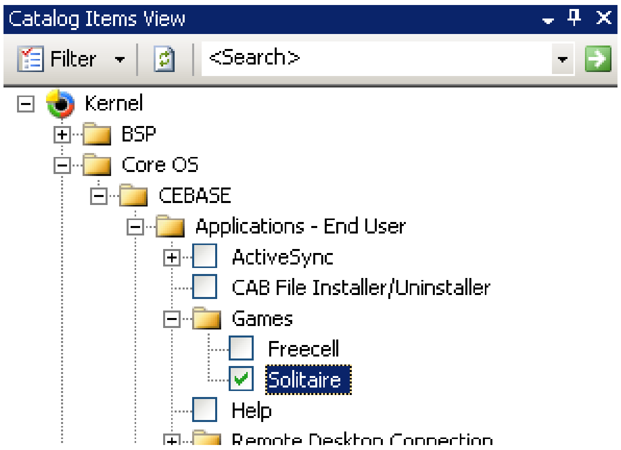
3.2.1. Include and remove project components

To include additional components in a project, display the complete Catalog (**Filter > All Catalog Items in Catalog**) and select components to include, To remove components from a project, deselect them in the **Catalog** view.

Including or removing components from the catalog requires a Sysgen of the Windows CE project.

For example, to include the Solitaire game:

1. Show all the items in the catalog.
2. Expand **Core OS > CEBASE > Applications - End User > Games**.
3. Click **Solitaire**.



3.3. OS design properties

Before compiling the kernel, it is important to understand the settings that can be changed in the OS design. Select **View > Solution Explorer**. Select the Kernel component, right-click and select **Properties**.

The **OS Design Property Pages** dialog displays and configures options for an entire OS design, and for individual configurations in the active OS design. By default, there are two configurations: Debug and Release (active by default).

The Debug configuration creates a kernel with debug information and services for enabling debug over Ethernet. The resultant image of this configuration is heavy (in size) and slow in execution (due to debug information messages)

The Release version doesn't include any debug information and its resultant image is light (in size) and quicker in execution.

This dialog configure many settings, such as the path to the OS build tree, the release directory for the OS design, the target file name for the debugger, the localization settings of the system, environment variables, and so on. To display online help for each group of settings, press F1 in each page.

3.3.1. Environment variables

The OS design has several *environment variables* used to configure the kernel and drivers. Environment variables are useful for quickly including/excluding components, files or modules during the creation of the final image rather than from the build process, as demonstrated later in this document. Most of these variables are set or cleared when selecting or deselecting components from the Catalog. Others are defined in the platform batch file in **%_WINCEROOT%\PLATFORM\PLATFORM\ccx9c.bat.** Here are some of the variables in the **ccx9c.bat** file:

Variable	Default value	Description
ENABLE_WATCH_DOG	1 (Release) 0 (Debug)	Enables the watchdog driver. Disabled in debug versions by default.
TFTPDIR	C:\tftproot	Folder exposed by the TFTP server. Images are copied here.
BSP_CONFIGOS	1	If set to 1, reconfigures the network settings of the target with the information extracted from U-Boot environment variables
BSP_LANNS9XXX	1 (Release) 0 (Debug)	If set to 1, activates the Ethernet driver for the NS9XXX processor. Disabled in debug versions by default.

Make sure the TFTPDIR environment variable matches the folder exported by the TFTP server. The U-Boot bootable Windows Embedded CE image is stored in this folder after building.

Environment variables added to the platform batch file apply to every project based in that platform. To add environment variables for one project only, use the **Environment** section of the **OS Design Property** pages.

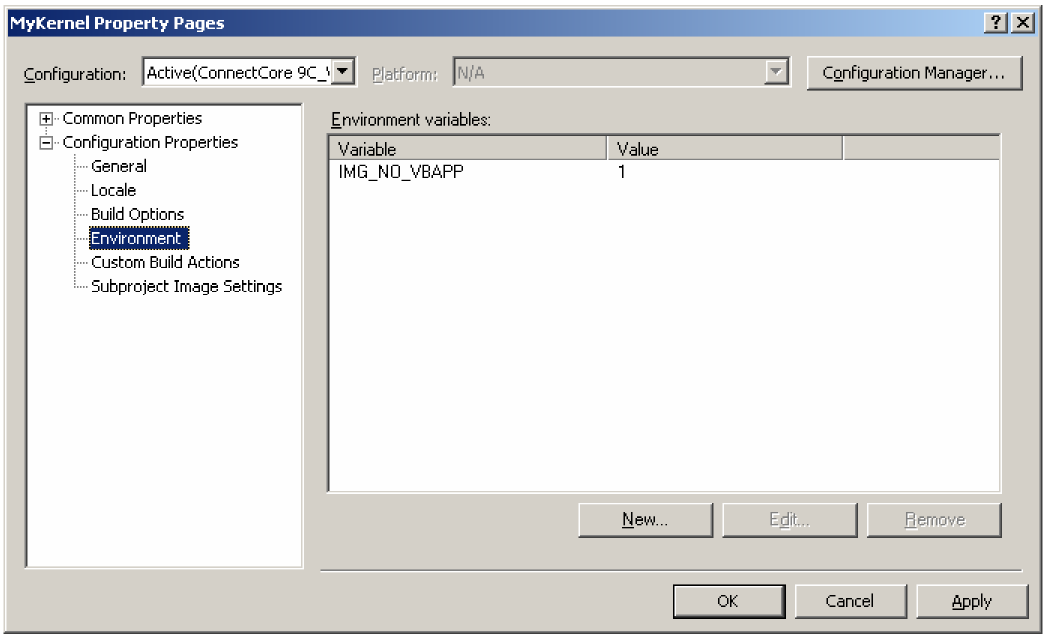
To add a pair of variables to the sample project:

1, In the **Solution Explorer**, right-click the **Kernel** component, and select **Properties**.

2. In the **Configuration** combo box select **All Configurations**.

3. Expand **Configuration Properties** and select **Environment**.

4. To add a new variable, click **New**. Add a variable with name **IMG_NO_VBAPP** and a value of **1**.



Environment variables:

Variable	Value
IMG_NO_VBAPP	1

3.4. Registry entries

As in any other Windows OS, important information for the kernel is stored in the Registry. The Registry of the kernel being created is generated during the build process by taking Registry entries in many different *.reg files.

The two most important files that form the Registry are:

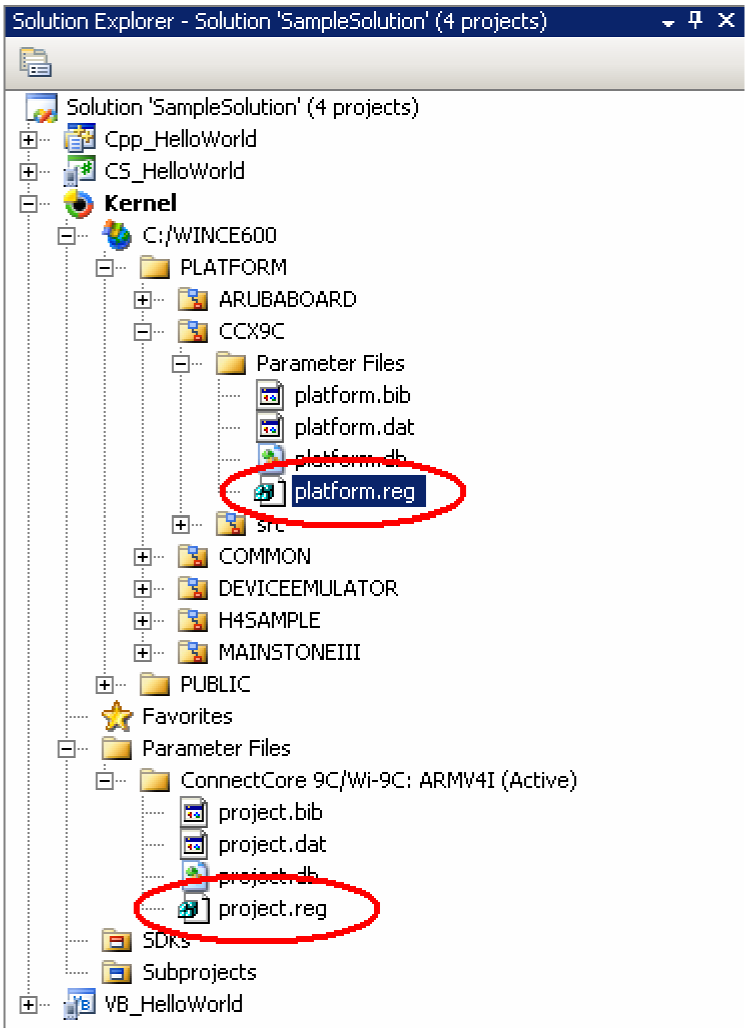
- **platform.reg**: contains Registry entries that affect all kernel projects based in the platform. This file is located in **%_WINCEROOT%\PLATFORM\*PLATFORM_NAME* \FILES**.
- **project.reg**: contains Registry entries that affect only the current project. This file is located in ***your_solution_path**kernel_name*\Wince600*platform_architecture*\OAK\files\**.

For the sample solution, these files are in these locations:

C:\WINCE600\PLATFORM\CCX9C\FILES\platform.reg

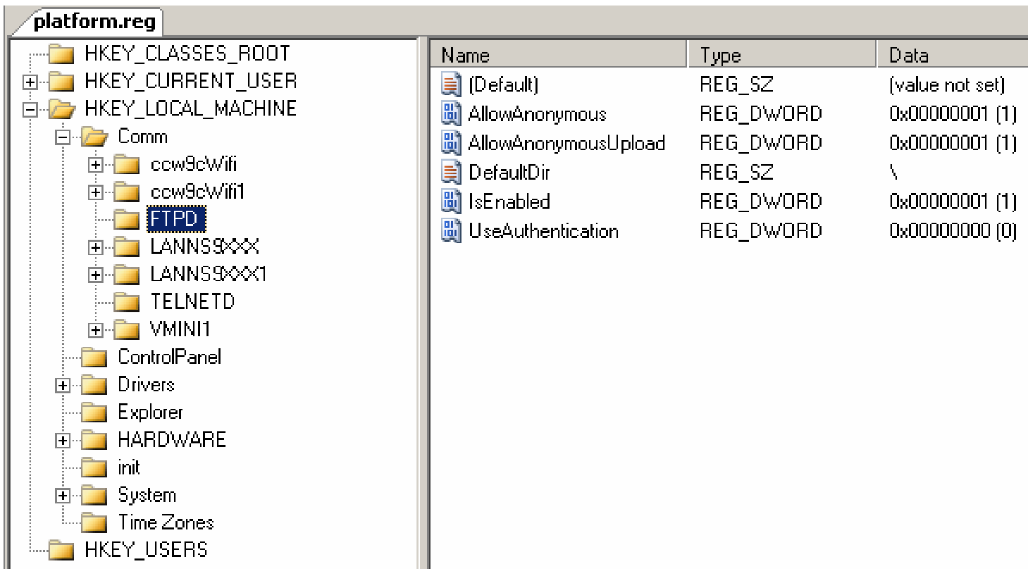
C:\samples\SampleSolution\Kernel\Wince600\CCX9C_ARMV4I\OAK\files\project.reg

These files can be browsed from the Solution Explorer.



```
Solution Explorer - Solution 'SampleSolution' (4 projects)
Solution 'SampleSolution' (4 projects)
+ Cpp_HelloWorld
+ CS_HelloWorld
- Kernel
  - C:/WINCE600
    - PLATFORM
      + ARUBABOARD
      - CCX9C
        - Parameter Files
            platform.bib
            platform.dat
            platform.db
            platform.reg
        + src
      + COMMON
      + DEVICEEMULATOR
      + H4SAMPLE
      + MAINSTONEIII
    + PUBLIC
    Favorites
  - Parameter Files
    - ConnectCore 9C/Wi-9C: ARMV4I (Active)
        project.bib
        project.dat
        project.db
        project.reg
    SDKs
    Subprojects
+ VB_HelloWorld
```

For example, some Registry information has been included for the FTP and the TELNET servers in **Platform.reg**. Double-click it to access the Registry information. Then expand **HKEY_LOCAL_MACHINE > Comm** and select **FTPD** and **TELNETD**.


platform.reg

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
  - Comm
    - ccw9cWifi
    - ccw9cWifi1
    - FTPD
    - LANNS9XXX
    - LANNS9XXX1
    - TELNETD
    - VMINI1
  - ControlPanel
  - Drivers
  - Explorer
  - HARDWARE
  - init
  - System
  - Time Zones
- HKEY_USERS

Name	Type	Data
(Default)	REG_SZ	(value not set)
AllowAnonymous	REG_DWORD	0x00000001 (1)
AllowAnonymousUpload	REG_DWORD	0x00000001 (1)
DefaultDir	REG_SZ	\
IsEnabled	REG_DWORD	0x00000001 (1)
UseAuthentication	REG_DWORD	0x00000000 (0)

These Registry keys provide complete access to the target using TELNET and FTP, for demonstration purposes. To preserve the security of targets, change these Registry keys by enabling the UseAuthentication key (enter a 1) and adding a list of allowed users.

For more information about security, read the topics *Telnet Server Security* and *FTP Server Security* of Windows CE on-line help.

3.5. Adjusting the memory layout

Depending on the size of the RAM memory of target module, it may be necessary to adjust the memory layout in the settings file of the OS design named **config.bib**. This is required for the modules with lower memories, because the default configurations are defined for modules with larger RAMs to include more features on the design templates.

To adjust the memory layout, edit the file **config.bib** located under:

%_WINCEROOT%\PLATFORM\CCX9C\FILES\

Then adjust the value of the macro **NKRAM_SIZE**. The **config.bib** file includes configuration examples for the different module variants. Comment/uncomment the macro value according to the RAM size of the used module.

3.6. File system

The OS design you have made creates a set of directories and files that form the file system of the target. The typical directory tree contains several files and folders, such as **Application Data**, **Documents and Settings**, or **Windows**.

All the files of the target file system are located in the **Windows** folder.

3.6.1. Include files and folders

When the OS design is built, all programs, Registry files, libraries, and so on are placed in a directory on the host named **Flat Release Directory**. The path of this directory is:

your_solution_path\kernel_name*\RelDir\*platform_architecture_configuration

In the sample solution, the files are in this location (release configuration version):

C:\samples\SampleSolution\Kernel\RelDir\ConnectCore_9C_Wi-9C_ARMV4I_Release\

The build process generates the contents of this directory and creates the final image with the files in it. For this reason the build tools must be instructed to copy any file(s) desired to be included to the flat release directory. This is done in the binary image builder (.bib) files. A binary image builder file defines which modules and files are included in a run-time image. The makeimg.exe file uses .bib files to determine how to load modules and files into the memory of a target device.

There are many binary image files, but the most important are these two:

- **platform.bib**: Changes in this file apply to every OS design based in the ConnectCore 9C/Wi-9C platforms.
- **project.bib**: Changes in this file apply only to the current project.

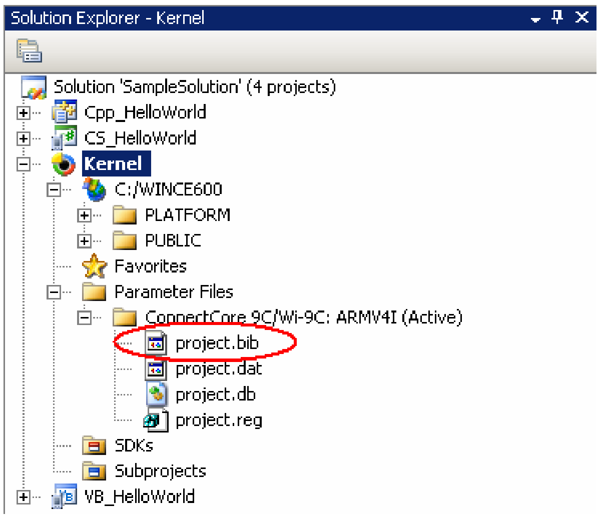
Because including files for the current project is usually desired, use **project.bib**, located at:

***your_solution_path\kernel_name*\Wince600\platform_architecture\OAK\files\**

In the example solution, the file is located at:

C:\samples\SampleSolution\Kernel\Wince600\CCX9C_ARMV4I\OAK\files\project.bib

The file can also be accessed from the Solution Explorer:



For example, to include the executable applications developed in topic 2, depending on the environment variables created in topic 3.3.1, edit **project.bib** and add these lines to the **FILES** section:

```
FILES
;   Name               Path                               Memory Type
;   --------------     -------------------------------    -----------
IF IMG_NO_VBAPP !
   vb_hello.exe      PathToTheFile\VB_HelloWorld.exe        NK
ENDIF

IF IMG_NO_CSAPP !
   cs_hello.exe      PathToTheFile\CS_HelloWorld.exe        NK
ENDIF

IF IMG_NO_CPPAPP !
   cpp_hello.exe      PathToTheFile\Cpp_HelloWorld.exe      NK
ENDIF
```

Substitute **PathToTheFile** with the path where the executable file is. n the example, the paths are **C:\samples\SampleSolution\CS_HelloWorld\bin\Debug\** , **C:\samples\SampleSolution\VB_HelloWorld\bin\Debug\** and **C:\samples\SampleSolution\Cpp_HelloWorld\CC9C_Wi-9C_SDK (ARMV4I)\Debug\**.

With these lines, unless a user-defined environment variable, **IMG_NO_*xx*APP**, exists, **makeimg.exe** includes the **xx_HelloWorld.exe** input file as the **xx_hello.exe** output file. It then loads the output file in a **MEMORY** region named **NK** as a normal file with no special attributes. For more information on the memory types, see the Visual Studio 2005 online help's topics *FILES Section* and *MODULES Section.*

In the example, because **IMG_NO_VBAPP** is set to 1, the Visual Basic application is not included in the image. But because **IMG_NO_CSAPP** and **IMG_NO_CPPAPP** are not defined, the C# application and the C++ application are included.

3.7. Launch an application after start-up

When the Windows Embedded CE kernel starts, several services are launched (such as the FTP, HTTP and Telnet servers), and the Windows Desktop is displayed.

If you want an application to launch automatically after start-up, you need to create a shortcut to the application in the Startup menu entry.

3.7.1. Create a shortcut to the application

Shortcuts to applications are plain text files with the path to the executable preceded by the number of characters of the shortcut path. To create a shortcut to the **CS_HelloWorld** application:

1. Go to the folder that contains the **CS_HelloWorld.exe** binary file.
2. Create a plain text file with the name **CS_HelloWorld.lnk**.
3. In the text file, write the path to the executable file in the target's file system (**\windows\cs_hello.exe**), preceded by the number of characters of the path and filename (**21**) and the hash symbol (**#**), like this:

```
21#\windows\cs_hello.exe
```

4. Save the file.

3.7.2. Add the shortcut to the OS design

Shortcuts, like normal files, must be added to the OS design to have them available in the target's file system:

1. Open the file **project.bib** as done in topic 3.6.1 and add an entry for the shortcut, under the entry of the program itself:

```
 FILES
 ;   Name               Path                               Memory Type
 ;   --------------     ------------------------------     -----------
IF IMG_NO_VBAPP !
    vb_hello.exe      PathToTheFile\VB_HelloWorld.exe        NK
ENDIF

IF IMG_NO_CSAPP !
    cs_hello.exe      PathToTheFile\CS_HelloWorld.exe        NK
    cs_hello.lnk      PathToTheFile\CS_HelloWorld.lnk        NK
ENDIF
```

2. Substitute *PathToTheFile* with the path where the executable file is (in the example **C:\samples\SampleSolution\CS_HelloWorld\bin\Debug\**

   This adds a file called **cs_hello.lnk** to the OS design, which is a copy of the shortcut **CS_HelloWorld.lnk** created in the previous topic.

3.7.3. Create the Startup entry

The shortcut file is placed in the *\windows* folder, like the rest of files of the target's file system. DAT files are used to place copies of files in a different place of the file system.

Normally, startup entries are added to the OS design **project.dat**, so that the changes affect only the current project.

1. In the Solution Explorer, expand **Kernel > Parameter Files > ConnectCore 9C/Wi-9C: ARMV4I (Active)**.
2. Double-click **project.dat** file.
3. Add this entry:

```
Directory("\Windows\Startup"):-File("cs_hello.lnk","\Windows\cs_hello.lnk")
```

   where:

   - The first entry (Directory) is the directory where the copy will be placed.
   - The second entry (File) contains the name for the copy (for example "cs_hello.lnk") and the path to the file to be copied (the shortcut file).

4. Save the **project.dat** file.

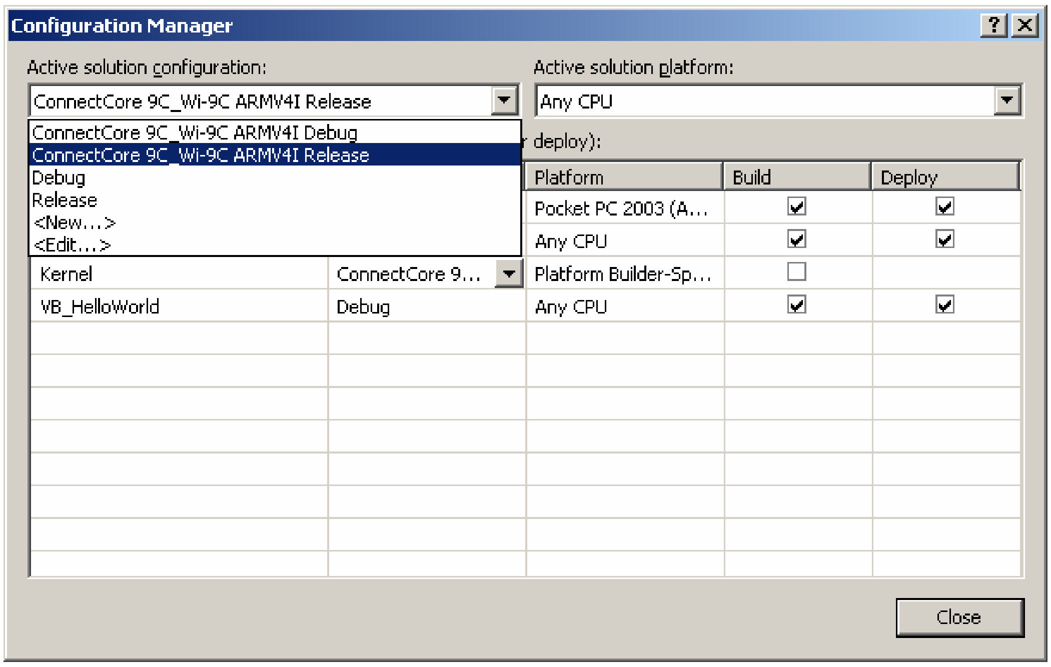
The .DAT files define the folder structure of an image. The filesystem parses information provided by .DAT files to create and populate the RAM directory structure.

4. Build the kernel

Now that the kernel is configured, it can be compiled.

4.1. Build the kernel (Release version)

1. Select **Build > Configuration Manager** and select the **Release** configuration. Then close the Configuration Manager.



Configuration Manager

Active solution configuration: ConnectCore 9C_Wi-9C ARMV4I Release

- ConnectCore 9C_Wi-9C ARMV4I Debug
- ConnectCore 9C_Wi-9C ARMV4I Release
- Debug
- Release
- <New...>
- <Edit...>

Active solution platform: Any CPU

deploy):

		Platform	Build	Deploy
		Pocket PC 2003 (A...	✓	✓
		Any CPU	✓	✓
Kernel	ConnectCore 9...	Platform Builder-Sp...	☐	
VB_HelloWorld	Debug	Any CPU	✓	✓

When the active solution platform is *Any CPU*, the Build column of the Kernel project is not checked. This means that it will not be built as part of the solution.

Because this configuration has no debug information, the resulting image is smaller and faster than a debug release. Use this configuration when debugging the kernel is unnecessary, either because the kernel sources have not been modified or a stable version of the kernel exists.

2. Select the **Kernel** component in the **Solution Explorer** view. Then select **Build > Build Kernel**. The first time, the build process can take between 20 and 40 minutes to complete, depending on the speed of the development computer. At the end, the Output view shows something like this:

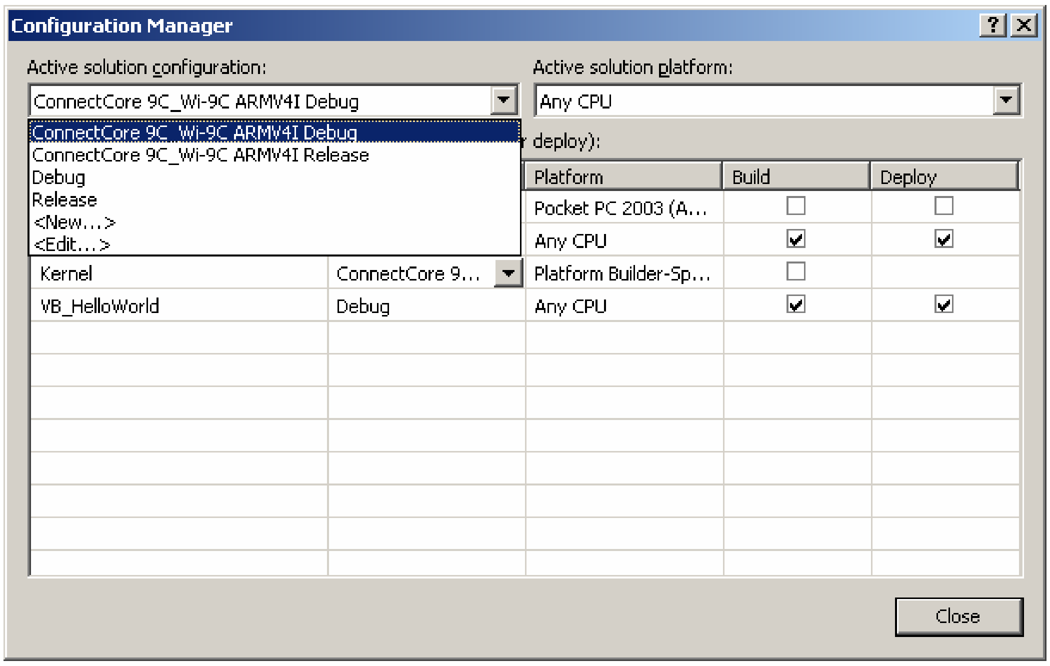
```
Directory of C:\samples\SampleSolution\Kernel\RelDir\CCX9C_ARMV4I_Release

16/01/2007  10:52        13.784.627 NK.bin
               1 File(s)     13.784.627 bytes
               0 Dir(s)  10.020.663.296 bytes free

BLDDEMO: Kernel build complete.

Kernel - 0 error(s), 29 warning(s)
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Some warnings may appear because Platform Builder postpones the importing of some Microsoft DLLs functions.

The build process generates an image called **NK.bin** which is placed into the Flat Release Directory, in the example:
C:\samples\SampleSolution\Kernel\RelDir\CCX9C_ARMV4I_Release.

4.2. Build the kernel (Debug version)

1. Select **Build > Configuration Manager.**
2. Under **Active solution configuration**, select the **Debug** configuration.



Configuration Manager

Active solution configuration: ConnectCore 9C_Wi-9C ARMV4I Debug

Active solution platform: Any CPU

ConnectCore 9C_Wi-9C ARMV4I Debug
ConnectCore 9C_Wi-9C ARMV4I Release
Debug
Release
<New...>
<Edit...>
deploy):
Close

		Platform	Build	Deploy
		Pocket PC 2003 (A...	☐	☐
		Any CPU	☑	☑
Kernel	ConnectCore 9...	Platform Builder-Sp...	☐	
VB_HelloWorld	Debug	Any CPU	☑	☑

Because this configuration has debug information, the resultant image will be bigger than the Release version. This version uses the Ethernet interface for debugging, and for this reason, it uses a standard driver for the Ethernet communication, instead of the NS9XXX driver. This configuration is useful when changing kernel sources, implementing custom drivers, or debugging the kernel step-by-step.

3. In the **Solution Explorer** view, select the **Kernel** component. Then select **Build > Build Kernel**. The first time, the build process can take between 20 and 40 minutes to complete, depending on the speed of the development computer. At the end, the **Output** view shows something like this:

```
Directory of C:\samples\SampleSolution\Kernel\RelDir\CCX9C_ARMV4I_Debug

16/01/2007  11:23        26.649.139 NK.bin
               1 File(s)     26.649.139 bytes
               0 Dir(s)   9.103.732.736 bytes free

BLDDEMO: Kernel build complete.

Kernel - 0 error(s), 29 warning(s)
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

The build process generates an image called **NK.bin** which has been placed into the Flat Release Directory, in the example:
C:\samples\SampleSolution\Kernel\RelDir\ConnectCore_9C_Wi-9C_ARMV4I_Debug.

5. Debug the kernel

Sometimes, modifying some kernel elements to adapt the OS to a specific hardware platform is desired. Other times, a new driver may need to be added into the kernel to support a hardware device. In these cases, the kernel must be tested and debugged to make sure that the final OS will be stable in the hardware platform.

Debugging the kernel requires building a debug version of it, as shown in topic 4.2.

5.1. About debugging over Ethernet

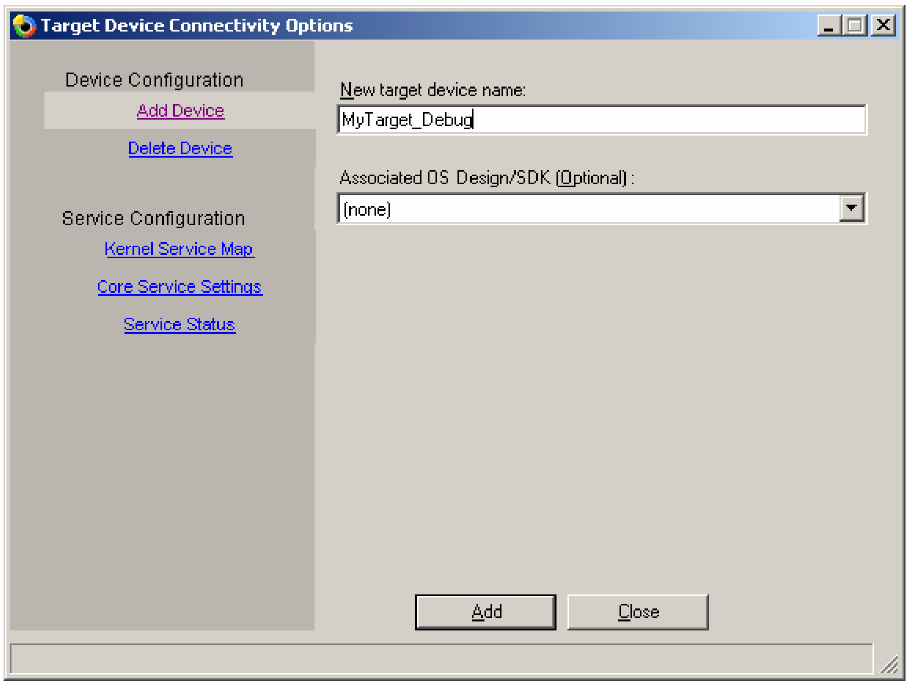
Debugging occurs over the Ethernet interface. Although the Ethernet interface is used for this purpose, it can share common network functionality using Vmini, a special network driver that handles both debug and standard frames.

For this reason, the behavior of Ethernet in the **Debug** version could differ from that of the **Release** version.

5.2. Establish the connection to the target

5.2.1. Target connectivity options

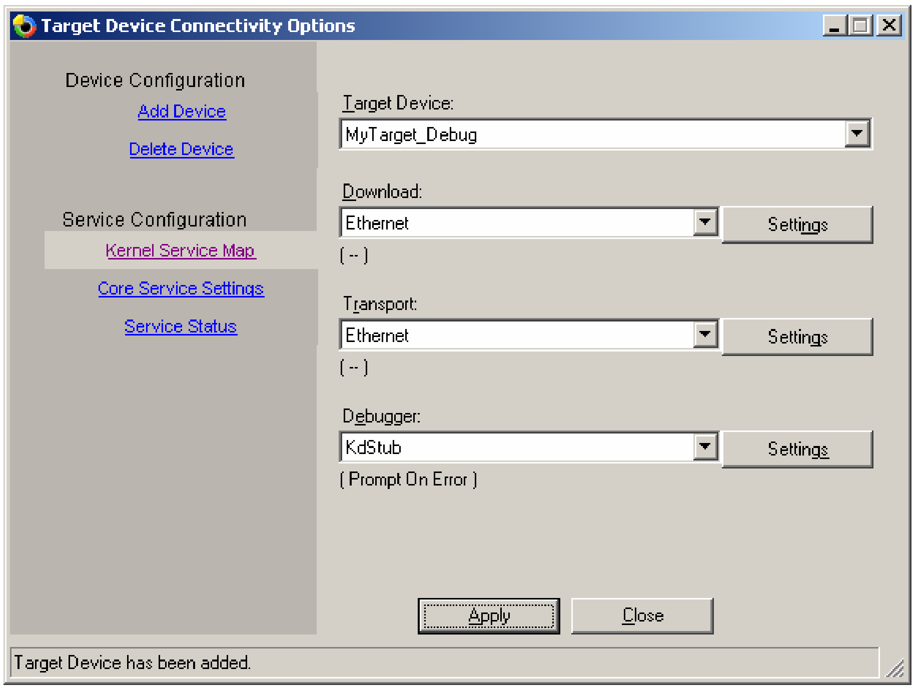
1. Select **Target > Connectivity options**, and click **Add Device** to add a connection profile to the target device.
2. In the **Name** field, enter the name of the profile; for example: **MyTarget_Debug**. Click **Add**.



New target device name: MyTarget_Debug

Associated OS Design/SDK (Optional) : (none)

The new profile is displayed, with predefined options for download, transport and debugging. Make sure these options are, respectively: **Ethernet** / **Ethernet** / **KdStub**.



Target Device: MyTarget_Debug

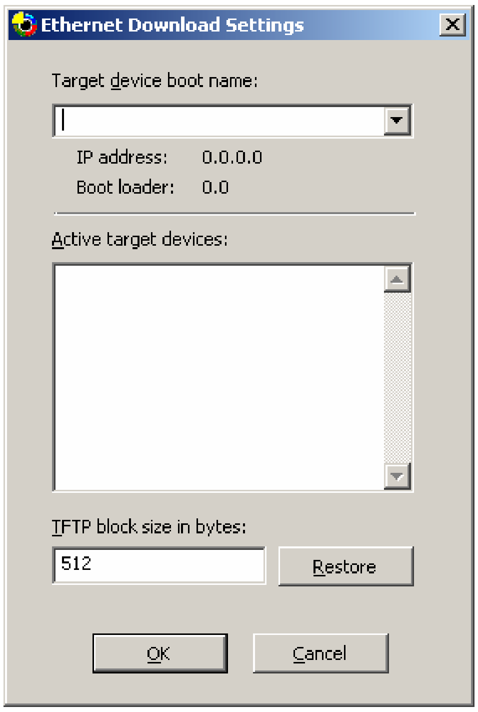
Download: Ethernet
(--)

Transport: Ethernet
(--)

Debugger: KdStub
(Prompt On Error)

Target Device has been added.

3. Click the **Settings** button next to the **Download** combo box. The **Ethernet Download Settings** dialog opens:



Target device boot name:

IP address: 0.0.0.0

Boot loader: 0.0

Active target devices:

TFTP block size in bytes: 512

Do not close this window. The target will appear here soon.

5.2.2. Send BOOTME messages from target

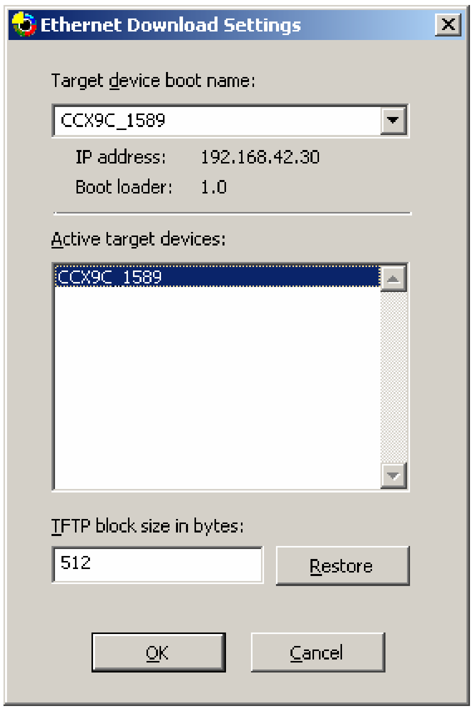
The U-Boot boot loader can send BOOTME messages through Ethernet to the Platform Builder interface. The Platform Builder catches these messages to establish the connection to the target.

1. Switch to the Serial Terminal program.
2. Power up the target board.
3. When boot loader messages are displayed, press any key to stop the auto boot process.
4. Enter this command:

```
#   dboot eboot
```

   This command runs the EBOOT program stored in flash, which sends Ethernet messages to the Platform Builder. The **Download Settings** window in Visual Studio shows that the target has been detected.

5. Select the target. The target's IP address is displayed.
6. Click **OK**.



Ethernet Download Settings

Target device boot name: CCX9C_1589

IP address: 192.168.42.30

Boot loader: 1.0

Active target devices: CCX9C_1589

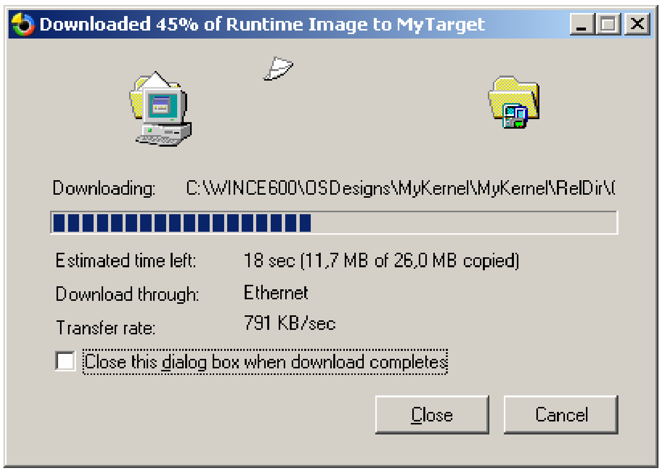
TFTP block size in bytes: 512

   The Target Device Connectivity options dialog is displayed.

7. Click **Apply** and then **Close**.

5.2.3. Attach the device

Select **Target > Attach Device**. This step catches the BOOTME messages from the target and establishes the connection. The kernel will begin to be downloaded.


Downloaded 45% of Runtime Image to MyTarget

Downloading: C:\WINCE600\OSDesigns\MyKernel\MyKernel\RelDir\(

Estimated time left: 18 sec (11,7 MB of 26,0 MB copied)

Download through: Ethernet

Transfer rate: 791 KB/sec

☐ Close this dialog box when download completes

The 'dboot eboot' command in U-Boot sends BOOTME messages for approximately 30 seconds. If the device is not attached within this period, the command cancels, and it must be launched again.

After downloading the kernel, the services CESH (Console Debug Shell tool) and CETerm (Target Messages) are initiated on the target. Windows Embedded CE starts without additional tools.

5.3. Debug the code

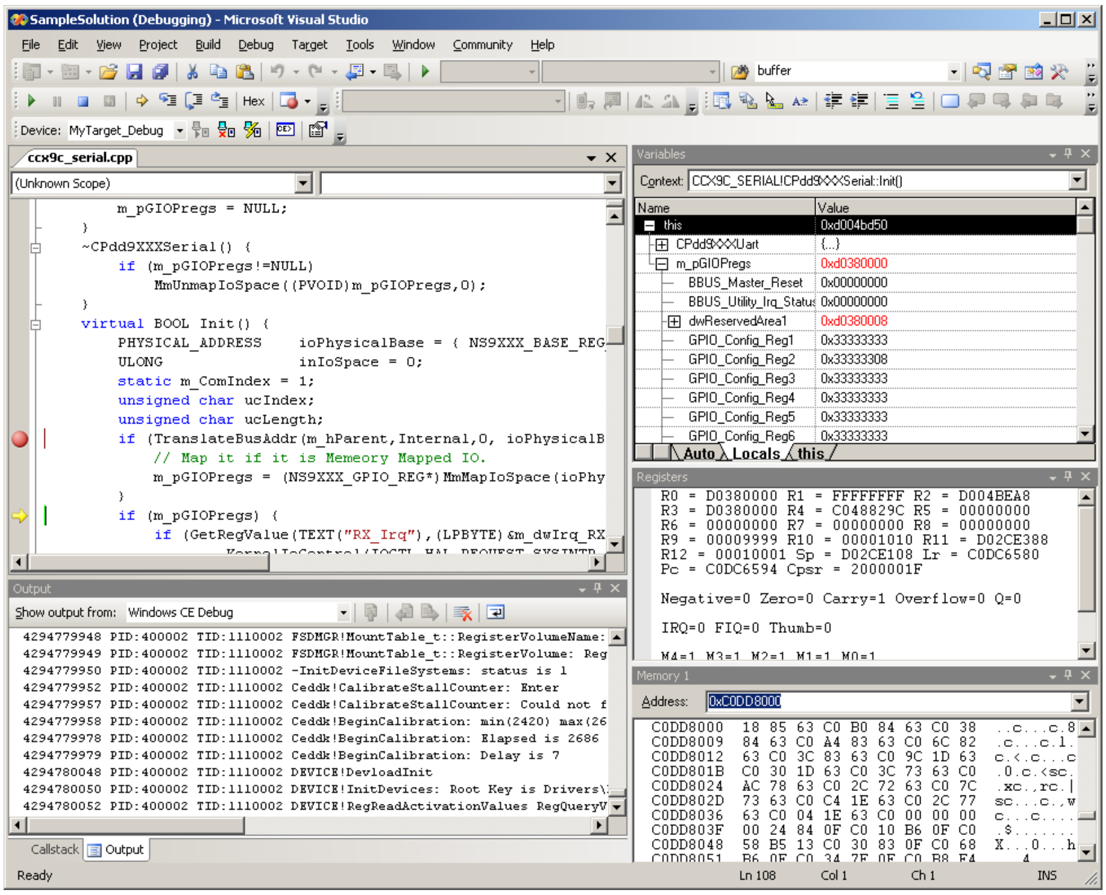
Now stop the target and debug the kernel. The debugging windows and menu items in the Platform Builder IDE allow review of processes, threads, and other target debugging information such as watch variables and memory dumps.

1. To stop the target, select **Debug > Break All**.

   When the target stops, a source file is opened where the system halted. If the source file doesn't exist (for example because it is internal code of Microsoft) the **Find executable** dialog may appear. Click **Cancel** to continue. If the source code cannot be opened, the **Disassembly** window opens with the assembler code.

2. If a breakpoint is set in a driver or kernel source file, the target stops when it reaches the breakpoint.

3. Select **Debug > Windows** and open the required debug views: **Variables**, **Call Stack**, **Registers**, **Memory**, etc.



```
        m_pGIOPregs = NULL;
    }
    ~CPdd9XXXSerial() {
        if (m_pGIOPregs!=NULL)
            MmUnmapIoSpace((PVOID)m_pGIOPregs,0);
    }
    virtual BOOL Init() {
        PHYSICAL_ADDRESS    ioPhysicalBase = { NS9XXX_BASE_REG
        ULONG               inIoSpace = 0;
        static m_ComIndex = 1;
        unsigned char ucIndex;
        unsigned char ucLength;
        if (TranslateBusAddr(m_hParent,Internal,0, ioPhysicalB
            // Map it if it is Memeory Mapped IO.
            m_pGIOPregs = (NS9XXX_GPIO_REG*)MmMapIoSpace(ioPhy
        }
        if (m_pGIOPregs) {
            if (GetRegValue(TEXT("RX_Irq"),(LPBYTE)&m_dwIrq_RX
```

Show output from: Windows CE Debug

```
4294779948 PID:400002 TID:1110002 FSDMGR!MountTable_t::RegisterVolumeName:
4294779949 PID:400002 TID:1110002 FSDMGR!MountTable_t::RegisterVolume: Reg
4294779950 PID:400002 TID:1110002 -InitDeviceFileSystems: status is 1
4294779952 PID:400002 TID:1110002 Ceddk!CalibrateStallCounter: Enter
4294779957 PID:400002 TID:1110002 Ceddk!CalibrateStallCounter: Could not f
4294779958 PID:400002 TID:1110002 Ceddk!BeginCalibration: min(2420) max(26
4294779978 PID:400002 TID:1110002 Ceddk!BeginCalibration: Elapsed is 2686
4294779979 PID:400002 TID:1110002 Ceddk!BeginCalibration: Delay is 7
4294780048 PID:400002 TID:1110002 DEVICE!DevloadInit
4294780050 PID:400002 TID:1110002 DEVICE!InitDevices: Root Key is Drivers\
4294780052 PID:400002 TID:1110002 DEVICE!RegReadActivationValues RegQueryV
```

Context: CCX9C_SERIAL!CPdd9XXXSerial::Init()

Name	Value
this	0xd004bd50
CPdd9XXXUart	{...}
m_pGIOPregs	0xd0380000
BBUS_Master_Reset	0x00000000
BBUS_Utility_Irq_Status	0x00000000
dwReservedArea1	0xd0380008
GPIO_Config_Reg1	0x33333333
GPIO_Config_Reg2	0x33333308
GPIO_Config_Reg3	0x33333333
GPIO_Config_Reg4	0x33333333
GPIO_Config_Reg5	0x33333333
GPIO_Config_Reg6	0x33333333

Auto Locals this

```
R0 = D0380000 R1 = FFFFFFFF R2 = D004BEA8
R3 = D0380000 R4 = C048829C R5 = 00000000
R6 = 00000000 R7 = 00000000 R8 = 00000000
R9 = 00009999 R10 = 00001010 R11 = D02CE388
R12 = 00010001 Sp = D02CE108 Lr = C0DC6580
Pc = C0DC6594 Cpsr = 2000001F

Negative=0 Zero=0 Carry=1 Overflow=0 Q=0

IRQ=0 FIQ=0 Thumb=0

M4=1 M3=1 M2=1 M1=1 M0=1
```

Address: 0xC0DD8000

```
C0DD8000  18 85 63 C0 B0 84 63 C0 38  ..c...c.8
C0DD8009  84 63 C0 A4 83 63 C0 6C 82  .c...c.l.
C0DD8012  63 C0 3C 83 63 C0 9C 1D 63  c.<.c...c
C0DD801B  C0 30 1D 63 C0 3C 73 63 C0  .0.c.<sc.
C0DD8024  AC 78 63 C0 2C 72 63 C0 7C  .xc.,rc.|
C0DD802D  73 63 C0 C4 1E 63 C0 2C 77  sc...c.,w
C0DD8036  63 C0 04 1E 63 C0 00 00 00  c...c....
C0DD803F  00 24 84 0F C0 10 B6 0F C0  .$.......
C0DD8048  58 B5 13 C0 30 83 0F C0 68  X...0...h
```

4. To step through the code and see the behavior of a driver or an application, press <**F10**>.

5. To resume execution, press <**F5**> (Go). Try other typical debugging actions such as modifying a variable, editing the memory, jumping to certain instructions, etc.

5.4. Remote Tools

Remote Tools allow monitoring a target from a desktop development environment, including editing the target's Registry, viewing running processes, taking screen captures of the target's screen, etc. Remote Tools are in **Target >Remote Tools**. They are executed through the debug connection opened for kernel debugging. They can also be executed in a Release version if the connection is established by other means, as seen in topic 6. Descriptions of the remote tools follow:

Remote Tool	Description
Call Profiler	Profiling and analysis tools within a graphical user interface (GUI) for identifying algorithmic bottlenecks in applications.
File Viewer	Displays a hierarchical view of the file system on a target device.
Heap Walker	Displays information about heap identifiers and flags for processes running on a target device.
Kernel Tracker	Provides a visual representation of OS and application events occurring on a target device.
Performance Monitor	Measures performance of a target device.
Process Viewer	Displays a list of processes and threads running on a target device.
Registry Editor	For viewing and managing the Registry for a target device.
Spy	Displays a list of windows opened on a target device and messages generated by those windows
System Information	Displays system settings and properties for a target device
Zoom-in	Displays an image from the screen of a target device

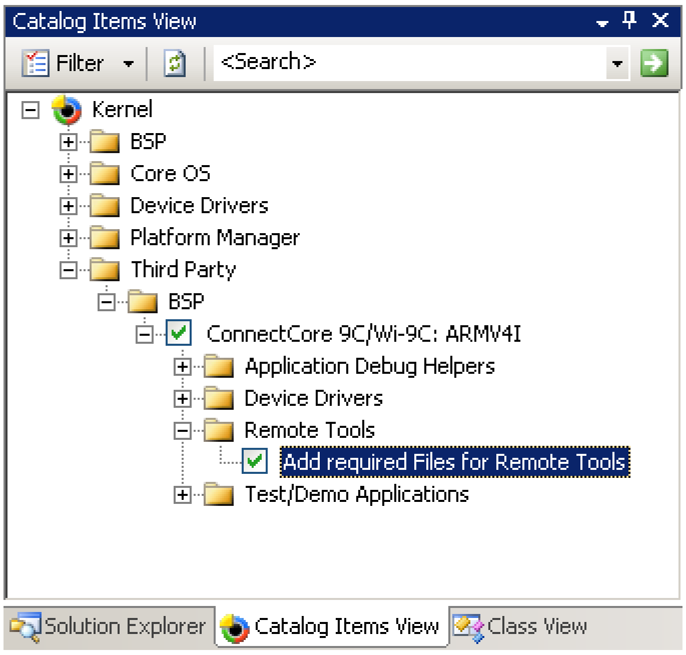
6. Connect to a Release kernel

6.1. Use Remote Tools in a Release configuration

6.1.1. Add the Remote Tools component

To use the Remote Tools in a Release version:

1. Add the **Remote Tools** component to the project. This component is in the **Catalog** view of the Platform Builder. Find the component under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Remote Tools** and check it to add it to the OS design.



Catalog Items View

Filter

<Search>

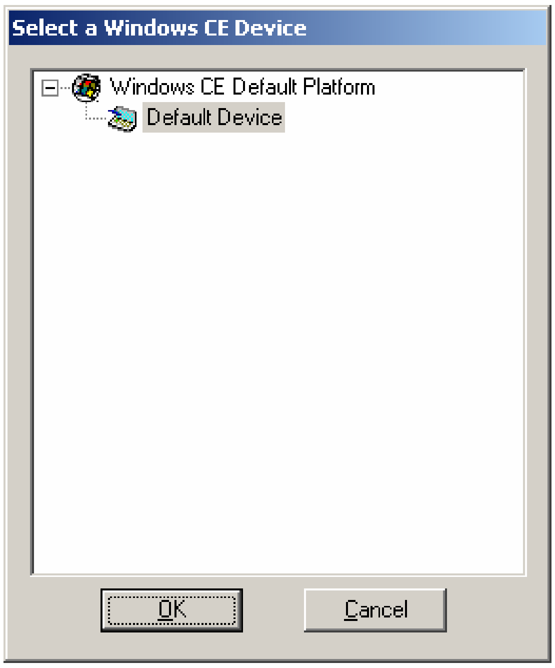
```
Kernel
  BSP
  Core OS
  Device Drivers
  Platform Manager
  Third Party
    BSP
      ConnectCore 9C/Wi-9C: ARMV4I
        Application Debug Helpers
        Device Drivers
        Remote Tools
          Add required Files for Remote Tools
        Test/Demo Applications
```

Solution Explorer | Catalog Items View | Class View

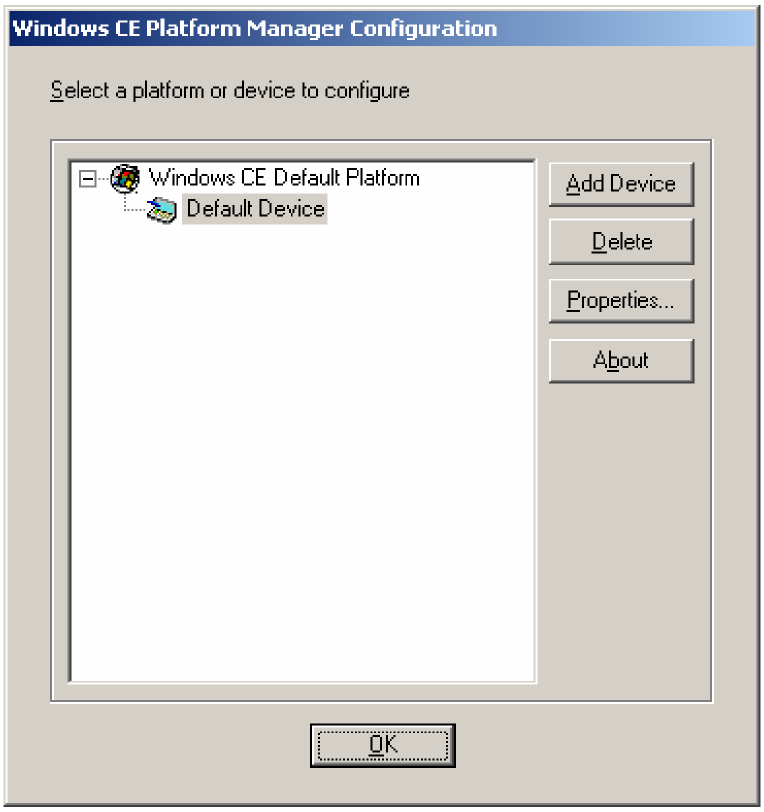
2. Build the **Release** version again.

6.1.2. Connect with Manual server

1. Select **Target > Remote Tools** and select the desired remote tool, for example, **Registry Editor**. The **Registry Editor** and a **Select Windows CE Device** dialog open:



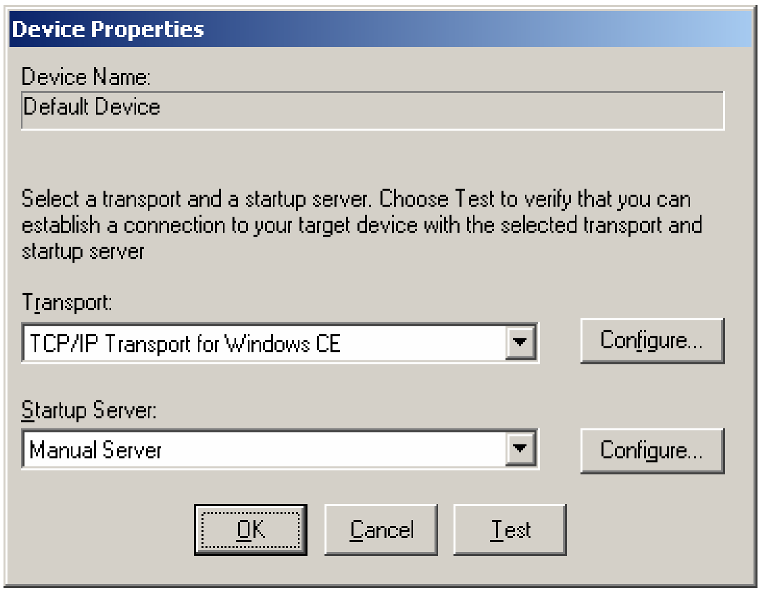
2. Click **Cancel** to close this dialog, because the connection cannot be established yet.
3. In the Windows CE Remote Registry Editor window, select **Connection > Configure Windows CE Platform Manager**. The **Windows CE Platform Manager Configuration** dialog opens:


Windows CE Platform Manager Configuration

Select a platform or device to configure

Windows CE Default Platform
Default Device

4. Select **Default Device** and click **Properties**.
5. In the dialog that opens, select:
   - **Transport**: TCP/IP Transport for Windows CE
   - **Startup Server**: Manual Server


Device Properties

Device Name: Default Device

Select a transport and a startup server. Choose Test to verify that you can establish a connection to your target device with the selected transport and startup server

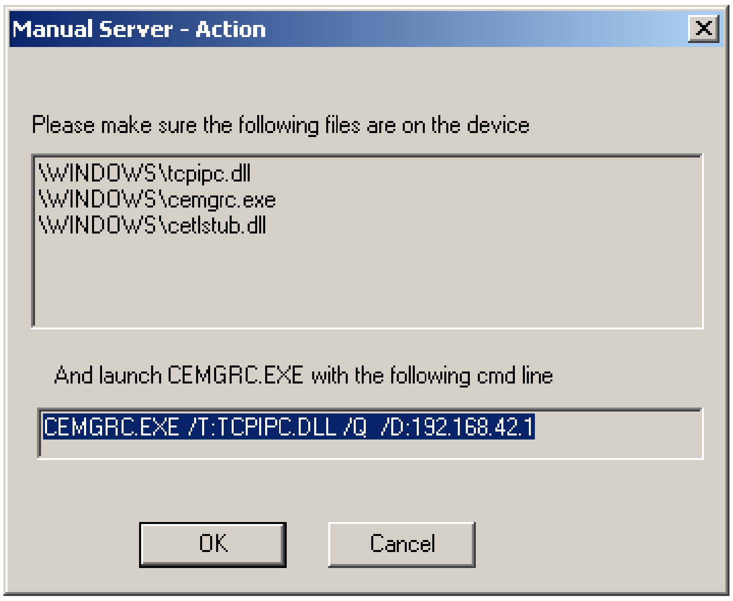
Transport: TCP/IP Transport for Windows CE

Startup Server: Manual Server

Then click **OK** in this and the previous dialog to accept the configuration.

If the development system has more than one Ethernet interface, verify that the correct interface is used for the connecting between the target and the host.

6. In the **Windows CE Remote Registry Editor** window, select **Connection > Add connection**. The Windows CE device selection dialog opens again.
7. This time select **Default Device** and click **OK**. The **Manual Server - Action** dialog opens:



Please make sure the following files are on the device

\WINDOWS\tcpipc.dll
\WINDOWS\cemgrc.exe
\WINDOWS\cetlstub.dll

And launch CEMGRC.EXE with the following cmd line

CEMGRC.EXE /T:TCPIPC.DLL /Q /D:192.168.42.1

The files that the dialog requests to be in the target are included by default in the kernel. If using a custom OS design, these files can be included in the OS design by including the **Platform Manager** element from the Catalog, under **Third Party > BSP > ConnectCore 9C/Wi-9C:ARMV4I > Platform Manager** and recompiling the OS design.

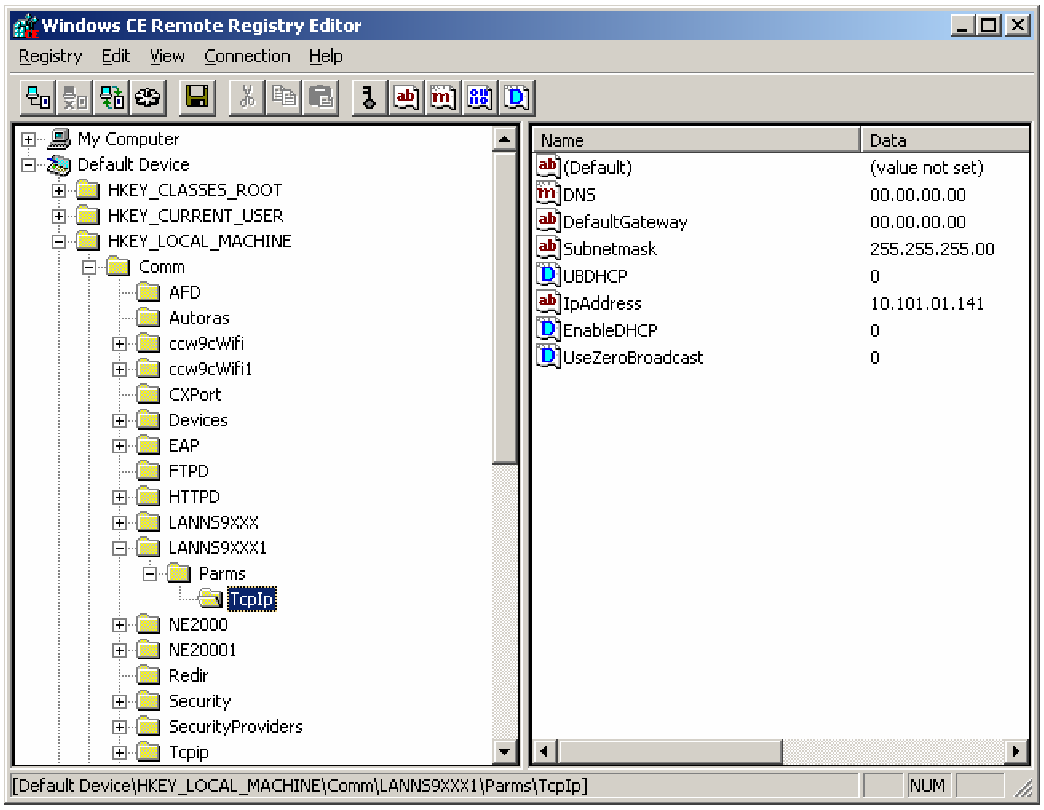
8. Select the **CEMGRC.EXE** command line and copy it to the Clipboard. Do not close this dialog yet.
9. Open a DOS box and connect to the target's IP using telnet:

```
C:\telnet 192.168.42.30
```

10. Paste the **CEMGRC.EXE** command line from the Clipboard and press **ENTER** to execute the command:

```
Welcome to the Windows CE Telnet Service on <target>

Pocket CMD v 6.00
\> CEMGRC.EXE /T:TCPIPC.DLL /Q  /D:192.168.42.1:4898
```

11. Return to the **Manual Server - Action** dialog and click **OK** to establish the connection. Now the Remote Registry Editor shows the target's Registry.


Windows CE Remote Registry Editor
Registry Edit View Connection Help

My Computer
Default Device
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
Comm
AFD
Autoras
ccw9cWifi
ccw9cWifi1
CXPort
Devices
EAP
FTPD
HTTPD
LANNS9XXX
LANNS9XXX1
Parms
TcpIp
NE2000
NE20001
Redir
Security
SecurityProviders
Tcpip

Name	Data
(Default)	(value not set)
DNS	00.00.00.00
DefaultGateway	00.00.00.00
Subnetmask	255.255.255.00
UBDHCP	0
IpAddress	10.101.01.141
EnableDHCP	0
UseZeroBroadcast	0

[Default Device\HKEY_LOCAL_MACHINE\Comm\LANNS9XXX1\Parms\TcpIp] NUM

After the connection has been established, the rest of Remote Tools (Zoom, System Information, and so on) can be opened directly. There is no need to repeat the previous steps.

7. Transfer the system to the target

Applications can be easily transferred to a running target, by means of network services like FTP. This topic shows how to transfer the kernel image for testing before updating the firmware in the flash memory.

7.1. TFTP server

The U-Boot boot loader running in the target board can write files to the flash memory of the module. A TFTP server is required to transport these files from the host computer to the target.

If a TFTP server is not available, Digi's TFTP server (**tftpd.exe**) can be used. This server can be installed from the ConnectCore BSP for Windows Embedded CE 6.0 CD-ROM; it is installed in **C:\tftproot** by default.

A shortcut to the TFTP server is located at **Start > Programs > Digi ConnectCore > TFTP Server**. Another shortcut is on the Desktop.

*The TFTP server exposes the folder where the program itself is located. For this reason, using the default path **C:\tftproot** is recommended.*

*If the TFTP server is installed in a different location, variable **TFTPDIR** must be changed accordingly in file **ccx9c.bat**, as seen in topic 3.3.1.*

7.2. Basic boot loader

To test the recently created kernel, it is useful to experiment with the boot loader commands. This topic explains the basic commands to download and boot a new system. For more information about U-Boot commands, see the *U-Boot Reference Manual*.

Power up the target board. When boot loader messages are displayed in the console, press a key to stop the auto boot process.

```
U-Boot 1.1.4 (Apr 20 2007 - 21:47:39) DUB-RevA
for Digi ConnectCore Wi-9C on Development Board

DRAM:  64 MB
NAND:  128 MiB
In:    serial
Out:   serial
Err:   serial
CPU:   NS9360 @ 154.828800MHz
Strap: 0x03
SPI ID:2007/02/21 V1_4, CC9C/CCW9C, SDRAM 64MByte, CL2, 7.8us, LE
FPGA:  wifi.ncd, 2007/01/25, 17:49:41, V2.01
Hit any key to stop autoboot:  0
CCW9C #
```

These U-Boot commands will be used:

Command	Description
printenv [variable]	Displays all (or one) current variables values.
setenv <variable> <value>	Sets a U-Boot variable with a given value.
saveenv	Saves all U-Boot variables into NVRAM memory for permanent storage.
dboot <os> <type> [partition]	Boots an image.

7.3. Environment variables

The U-Boot boot loader contains variables that configure its behavior. Some of these variables configure the network settings (IP address of the target and the host computer), as described in the *Building Your First Application* document.

7.4. Test the system

This section describes how to transfer the system to the target and boot it from RAM memory. This way the system can be tested without having to update the flash memory.

7.4.1. Transfer the system using Platform Builder

As shown in topic 5.2, a connection can be established between Visual Studio's Platform Builder and the target device for downloading a kernel. This is the normal way to transfer a Debug kernel. This method creates a permanent link between the target device and the development system.

7.4.2. Transfer the system by Ethernet

U-Boot can also download the binary image of the kernel and boot it from RAM.

When the kernel was created, a binary file named **wce-CCX9C** was automatically placed into the TFTP exposed folder if the TFTPDIR variable was set in **ccx9c.bat**, as seen in topic 3.3.1.

This wce-CCX9C image can be based on the Release version or on the Debug version, depending on the last configuration compiled.

Start the TFTP server. Then, from the U-Boot prompt, launch this command:

```
#   dboot wce tftp
```

The **wce-CCX9C** image is downloaded and booted by U-Boot. After a few seconds the Windows desktop is displayed and the **cs_hello.exe** application automatically starts. With this method, the system runs, but there is no link between the target device and the development computer.

7.4.3. Transfer the system by USB

U-Boot can also read the kernel from a USB flash disk. In this case, copy the kernel image **wce-CCX9C** to a USB flash disk formatted with FAT file system. Then, plug the USB to the target and power up the board.

Stop the U-Boot auto boot and tell U-Boot to read the kernel from the USB disk and boot it with this command:

```
#   dboot wce usb
```

The **wce-CCX9C** image is downloaded and booted by U-Boot. After a few seconds the Windows desktop is displayed, and the **cs_hello.exe** application automatically starts.

With this method, the system runs but no link exists between the target device and the development computer.

8. Update the flash memory

The kernel built in previous topics has been tested. This system was dependent on Ethernet/USB to download the kernel.

If the tested system behaves correctly meets requirements, consider writing it to the flash memory to permanently save the system.

There are several ways to update the flash memory, depending on what needs to be updated. Applications or simple files, for example, can be simply copied to a file system that resides in a flash partition.

Learning how to update flash memory requires an overview of its structure.

8.1. Structure of the flash

The flash memory is programmable non-volatile memory containing the whole operating system.

The flash memory is partitioned (logically divided) to contain the boot loader, the FPGA program, the Windows CE kernel, the EBOOT program, the persistent Registry, and some system configuration parameters.

The number, size, and position of these partitions can be modified as needed. This table shows the factory default partitioning structure:

Partition number	Name	Flash start address	Flash end address	Length	Description
0	U-Boot	0x00000000	0x000c0000	768 KiB	Stores the U-Boot boot loader image
1	NVRAM	0x000c0000	0x00140000	512 KiB	Stores permanent configuration parameters like the MAC address of the network interfaces, the serial number of the module, environment variables of U-Boot, and so on
2	FPGA	0x00140000	0x00240000	1 MiB	Stores the FPGA firmware
3	EBoot	0x00240000	0x00340000	1 MiB	Stores the EBOOT program for connecting with Platform Builder
4	Registry	0x00340000	0x00440000	1 MiB	Stores the Windows CE Registry
5	Kernel	0x00440000	0x01840000	20 MiB	Stores the Windows CE kernel image

For information about modifying the default flash partition table, see *"Using NVRAM"* in the *U-Boot Reference Manual*.

8.2. Update from a running Windows Embedded CE system

The ConnectCore 9C/Wi-9C BSP provides an application that can directly write image files to the flash memory. To see the syntax of this application, open a telnet session to the target and execute **update_flash –h**:

```
C:\> telnet 192.168.42.30
Welcome to the Windows CE Telnet Service on <target>

Pocket CMD v 6.00
\> update_flash -h
Application to read, write and list
the flash partitions. Revision 1.0
Copyright(c) 2007 Digi International Inc.

  Usage: update_flash <options>

  Where options are:
    -r               Reads the specified partition
    -w               Writes the specified partition
    -p <partition>   Selects partition (<partition> must be a number)
    -f <filename>    File name to write/read the partition data
    -l               Lists the partition table
    -h               Shows this help

  Examples:
  update_flash -r -p 3 -f image1.bin
  update_flash -w -p 4 -f image3.bin
\>
```

For updating the flash, the files first need to be transmitted to the target, for example using FTP.

The source code of **update_flash** is in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\Update_Flash**.

The installation procedure allows selecting where source code is installed. The default location is: ***%PROGRAM_FILES%\Digi\ConncectCore\ConnectCore 9C and Wi-9C\Apps\Source Code*** *If the source code was installed in another folder during the installation process, for example* ***my_folder****, the source code is in:* ***my_folder\Digi\ConncectCore\ConnectCore 9C and Wi-9C\Apps\Source Code***

8.3. Update from U-Boot

U-Boot also can write to flash memory. This way, even if the target is not running Windows Embedded CE, the flash memory can be reprogrammed.

To update flash memory in U-Boot, use the **update** command. This is the **update** command syntax:

```
#   help update
update partition source [file]
  - updates 'partition' via 'source'
    values for 'partition': uboot, linux, rootfs, userfs, eboot, wce
                            or any partition name
    values for 'source': tftp, usb
    values for 'file': the file to be used for updating
```

The **update** command gets the file either from either a USB flash disk or a TFTP exposed folder in the host, depending on the **source** parameter. For that, it uses the **file** given as parameter or, if no filename is provided, it uses the names stored in these U-Boot environment variables:

- Windows Embedded CE Kernel image filename: **wimg**
- U-Boot image filename: **uimg**
- EBOOT image filename: **eimg**

The default values for these variables correspond to the default image filenames generated during compilation of the system. If the image filenames were changed, provide the parameter **file** with the new name to the **update** command.

The U-Boot **update** command takes care of transferring the image file to RAM, erasing the flash sectors, and writing the new image.

There are some restrictions for updating large image files. See topic 14.6 for more information.

8.3.1. Update the kernel

For example, if the **wce-CCX9C** image is placed in the TFTP exposed folder on the development computer, the **update** command is:

```
#   update wce tftp
```

8.3.2. Update EBOOT

Normally, there is no need to update EBOOT. The original EBOOT program comes with Platform Builder embedded in Visual Studio and works fine.

The EBOOT program needs to be updated only if the flash memory or the EBOOT partition is accidentally erased.

In that case, copy the EBOOT program image (located in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Images\wince\*platform*\eboot-ccx9c,** where ***platform*** is substituted with the platform name) to the TFTP exposed folder, and run this command in U-Boot shell:

```
#   update eboot tftp
```

💡	*The installation procedure allows selecting where images are installed. The default location is:* ***%PROGRAM_FILES%\Digi\ConncectCore\ConnectCore 9C and Wi-9C\Images\*** *If the images were installed in another folder during the installation process, for example,* ***my_folder****, the images are in:* ***my_folder\Digi\ConncectCore\ConnectCore 9C and Wi-9C\Images\***

8.3.3. Update U-Boot

Updating the U-Boot boot loader is covered in topic 13.5.

9. SDK for the OS design

After creating a custom OS design, an SDK based on that OS design can be created for distribution to other developers. An SDK is a set of headers, libraries, connectivity files, run-time files, OS design extensions, and documentation that developers use to write applications for a specific OS design. The contents of an SDK allow developers to create and debug a C++ application on the run-time image built from the OS design.

The SDK is needed only for native code (C++). Managed code developed in C# or Visual Basic runs in any OS design that contains the .NET Compact Framework.

9.1. Included SDK

The ConnectCore BSP for Windows Embedded CE 6.0 CD-ROM includes an SDK for the factory default OS design that is running in the target. If this default kernel will be used and only applications will be developed, install this SDK, which supports all components of the default image.

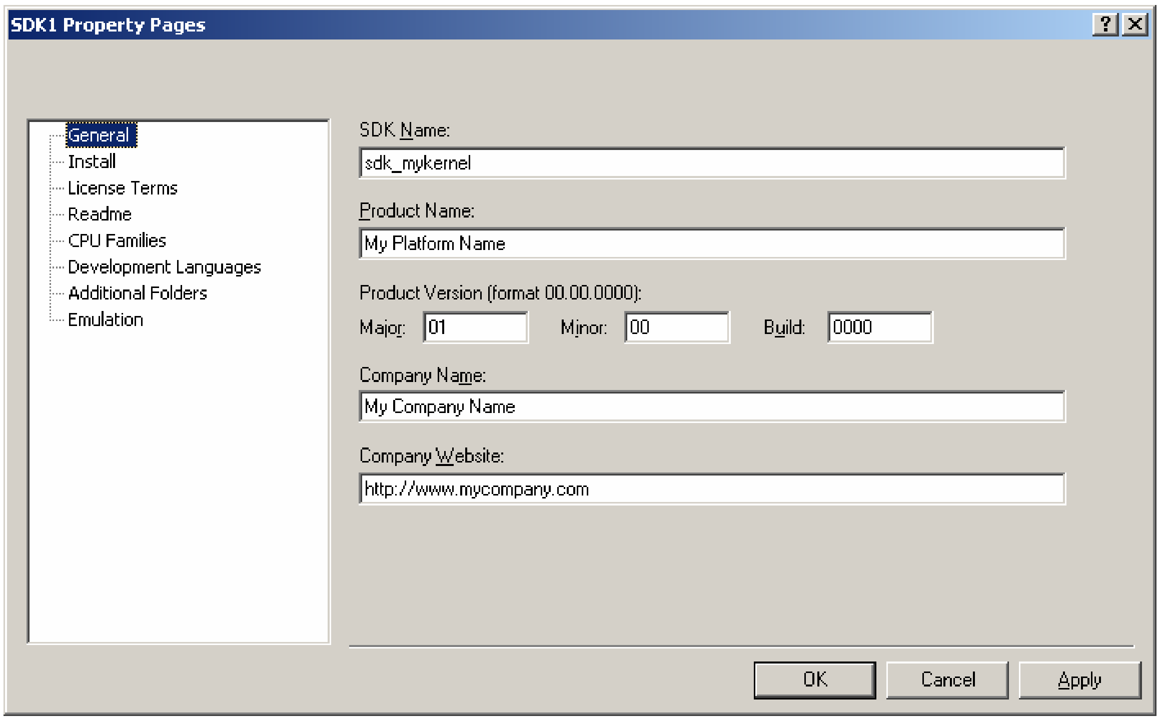
The included SDK is installed by default unless it is deselected during the installation process.

If a different OS design has been created with support for other drivers, the SDK must be created based on this kernel for distribution to C++ developers.

Platform Builder (embedded in Visual Studio 2005) can be used to develop an SDK based on a custom OS design for installation on another development workstation.

9.2. Create an SDK

1. In the **Solution Explorer**, select the **Kernel** component of the solution.
2. Select **Project > Add New SDK...**
3. In the dialog that opens, enter the SDK name, Product name, Company name, and product version:



SDK Name: sdk_mykernel

Product Name: My Platform Name

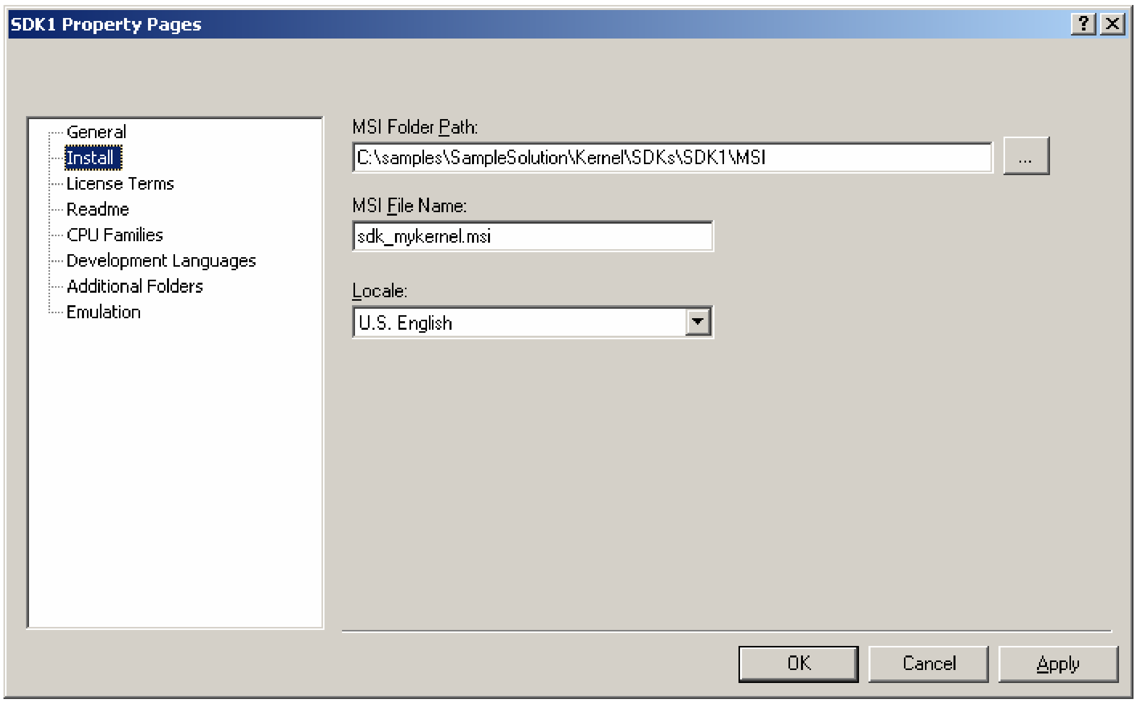
Product Version (format 00.00.0000):

Major: 01 Minor: 00 Build: 0000

Company Name: My Company Name

Company Website: http://www.mycompany.com

4. In the **Install** section, enter the name of the installer file, including the **.msi** extension.



MSI Folder Path: C:\samples\SampleSolution\Kernel\SDKs\SDK1\MSI

MSI File Name: sdk_mykernel.msi

Locale: U.S. English

5. In the remaining sections include a license file, a README text file with information about the SDK, additional folders and files, etc., as needed.
6. When finished, click **OK**.

9.3. Build the SDK

To build the SDK, select **Build > Build All SDKs.**

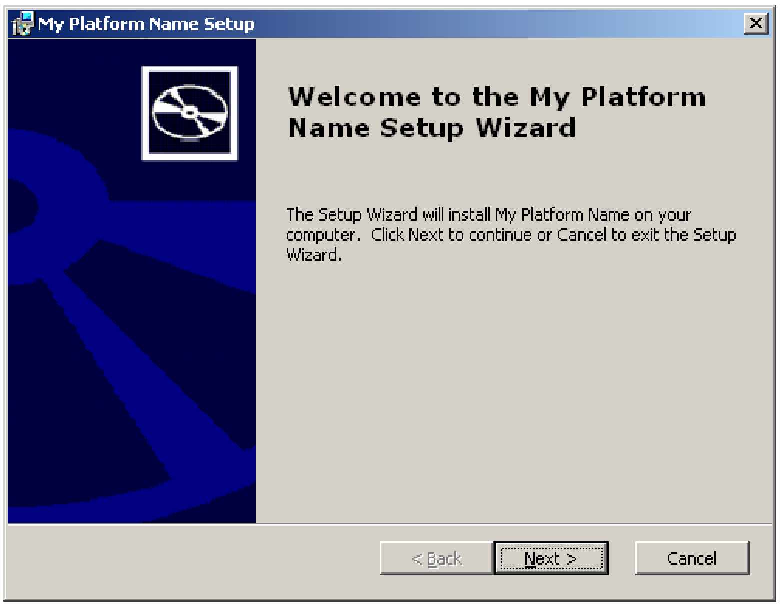
The SDK build process collects the headers and libraries associated with the modules and components in the OS design, additional files and folders, and text documents into a folder named **obj/**. It then compresses them all into an installable .msi file, and the file is placed at the path established during the creation of the SDK, which in this example is **C:\samples\SampleSolution\Kernel\SDKs\SDK1\MSI**. This is the file to distribute or install in the development PC for creating unmanaged C++ applications for the OS design.

9.4. Install the SDK

The Visual Studio Service Pack 1 must be installed first before installing the new created SDK. Otherwise unexpected behavior may occur, or the SDK may not be usable.

For information about obtaining Visual Studio 2005 Service Pack 1, visit http://go.microsoft.com/fwlink/?LinkID=70648

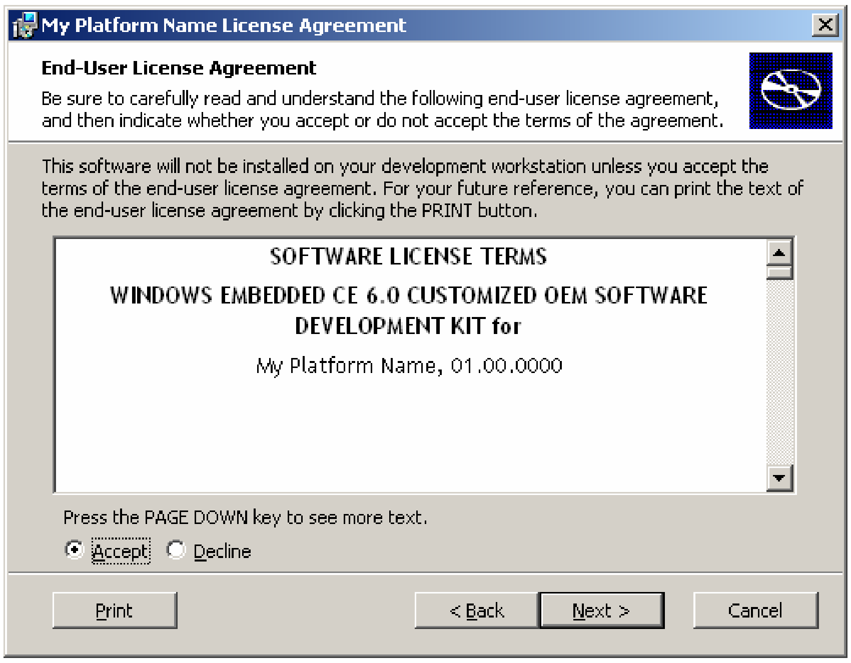
1. Double-click the SDK’s **.msi** file to install it.



Welcome to the My Platform Name Setup Wizard

The Setup Wizard will install My Platform Name on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

2. When the **End-User License Agreement** appears, click **Accept** and then click **Next**.



End-User License Agreement

Be sure to carefully read and understand the following end-user license agreement, and then indicate whether you accept or do not accept the terms of the agreement.

This software will not be installed on your development workstation unless you accept the terms of the end-user license agreement. For your future reference, you can print the text of the end-user license agreement by clicking the PRINT button.

SOFTWARE LICENSE TERMS

WINDOWS EMBEDDED CE 6.0 CUSTOMIZED OEM SOFTWARE DEVELOPMENT KIT for

My Platform Name, 01.00.0000

Press the PAGE DOWN key to see more text.

[✓ Accept] Decline

3. Enter your user and company names.
4. Select **Complete** to do a full installation. Then navigate to where the SDK should be installed–accept the default path.

After the SDK is installed, begin to develop Visual Studio 2005 native applications for the OS design. When changing the Kernel image by including or removing components, make a new SDK for secure application development.

10. Devices and Interfaces

This topic describes the devices and interfaces in the hardware platform, the hardware resources they use, and how to configure, enable disable, and manage them from user application space.

10.1. Table of devices and their hardware resources

This table shows each device/interface with its driver name and the hardware resources it uses:

Device	Driver	IRQ	GPIO	Physical Memory	Timer	Chip Select
GPIO	nx9xxx_gpio		0-72 (muxed)			
Ethernet	LANNS9xxx		50-64			
NAND	ccx9c_nfd			0x50xxxxxx		static 1
Serial A (UART)	ccx9c_serial		8,9 (10-15)[1]			
Serial B (UART)	ccx9c_serial		0,1 (2-7)[1]			
Serial C (UART)	ccx9c_serial		40,41 (42,43, 20-23)[1)]			
Serial D (UART)	ccx9c_serial		44,45 (46,47, 24-27)[1]			
Serial A (SPI mode)	ccx9c_spi		8,9,14,15			
Serial B (SPI mode)	ccx9c_spi		0,1,6,7			
Serial C (SPI mode)	ccx9c_spi		40,41,22,23			
Serial D (SPI mode)	ccx9c_spi		44,45,26,27			
Touch screen (SPI Port B)	Touch	Ext 3	0,1,6,7		2	
I^{2}C	ns9xxx_i2c		46,47			
I^{2}C I/O port PCA9554						
RTC						
USB	ccx9c_usb					
Wireless	ccw9cwifi		58,65-67	0x6xxxxxxx		static 2
Display	ns9xxx_disp		15,18-41			
Watchdog						

[1] Only when HW Handshaking is enabled

10.2. GPIO (General Purpose Input/Output) pins

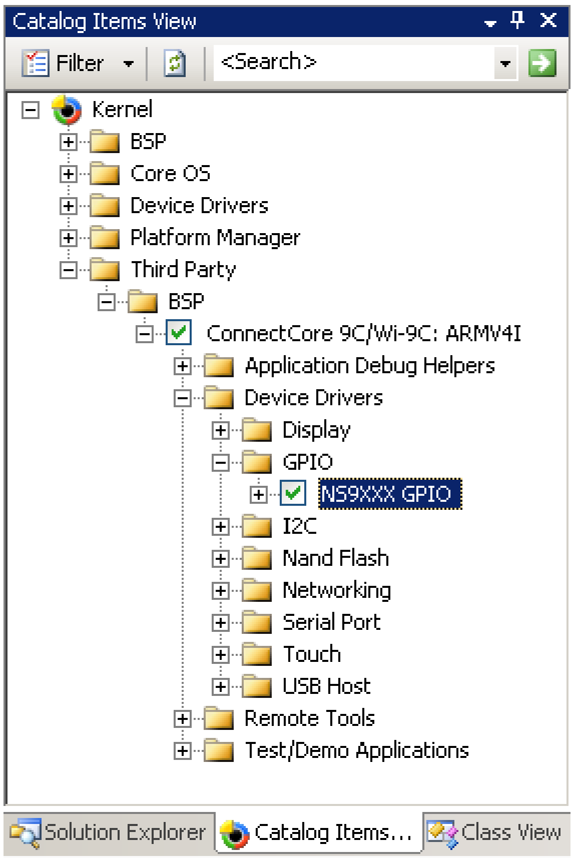
The NS9360 processor has 73 programmable GPIO pins (multiplexed with other functions). A custom driver has been developed for configuring and managing the GPIO pins. The sources are located in **%_WINCEROOT%\PLATFORM\COMMON\SRC\SOC\NS9XXX_DIGI_V1\Gpio\**.

10.2.1. Hardware resources used by the driver

Device	Driver	IRQ	GPIO	Physical memory	Timer	Chip select
GPIO	ns9xxx_gpio		0-72 (muxed)			

10.2.2. Enable the interface in the kernel

To include this device driver in the OS design, select it in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > GPIO**.



10.2.3. Manage the GPIOs from the user space

GPIOs can be managed from the applications using the standard Win32 API.

To use native functions from a C# application, the functions of some DLLs must be imported (see topic 14.3).

To use pointers and addresses in a C# application, the project must be configured for unsafe code (see topic 14.4).

10.2.3.1. Create the handle

To use the GPIOs, a device handle is needed. To create a handle, call the **CreateFile** function with the name of the device as first argument (**lpFileName**).

The GPIO device name is **PIO*n*:** where ***n*** is the index of the instance. Because there is only one instance of this device, the name is **PIO1:**.

This code opens a handle to the NS9360 internal GPIOs:

```
IntPtr hGPIO;

hGPIO = CreateFile("PIO1:", GENERIC_READ | GENERIC_WRITE,
                    0, 0, OPEN_EXISTING, 0, 0);
if( hGPIO == INVALID_HANDLE_VALUE)
{
    /* ERROR */
}
```

10.2.3.2. GPIO Message structure

The handle grants access to all GPIOs. To manage a specific GPIO pin, use the **GPIOMessage** structure. This structure is an argument to the functions of the API (**DeviceIoControl**, **ReadFile** and **WriteFile**). The **GPIOMessage** structure contains these fields:

```
public struct GPIOMessage
{
    public uint unPinNumber;
    public uint mode;
    public bool Block;
    public uint ulFlags;
    public uint unValue;
}
```

Type	Field	Description	Values
uint	unPinNumber	The pin number of the GPIO to use	0 to 72
uint	mode	GPIO working mode	GPIO_INPUT for input GPIO_OUTPUT for output GPIO_IRQ for interrupt
bool	Block	Define whether GPIO is blocked	False | true
uint	ulFlags	Configure the GPIO if working as IRQ	EXT_INTR_REG_BIT_LVEDG for Level sensitive IRQ (default is Edge sensitive) EXT_INTR_REG_BIT_PLTY for IRQ active low level or falling edge (default is active high level or rising edge)
uint	unValue	Value of the GPIO	0 | 1

For a native code example, the header files **pkfuncs.h**, **ns9xxx_ioctl.h,** and **ns9xxx_gpio_pdd.h** need to be included to the source files.

10.2.3.3. Device IO controls

The driver supports this IOTCL:

IOCTL	Description	Parameter
IOCTL_GPIO_CONFIG	Configures the behavior of the GPIO	GPIOMessage

10.2.3.4. Configure GPIO behavior

GPIO pins can be configured to have these behaviors:

- Input
- Output
- IRQ

This example code shows how to configure a GPIO as input, output or IRQ:

```
GPIOMessage GPIO;
uint uiBytesTransferred;

GPIO.unPinNumber = PIN_NUMBER; /* Set the GPIO number here */

/* Configure GPIO as INPUT */
GPIO.mode = GPIO_INPUT;
GPIO.Block = false;
if( !DeviceIoControl (hGPIO, IOCTL_GPIO_CONFIG, &GPIO,
                     sizeof(GPIOMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}

/* Configure GPIO as OUTPUT */
GPIO.mode = GPIO_OUTPUT;
GPIO.Block = false;
if( !DeviceIoControl (hGPIO, IOCTL_GPIO_CONFIG, &GPIO,
                     sizeof(GPIOMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}

/* Configure GPIO as IRQ low level sensitive */
GPIO.mode = GPIO_IRQ;
GPIO.Block = false;
GPIO.ulFlags = GPIO.ulFlags & EXT_INTR_REG_BIT_LVEDG & EXT_INTR_REG_BIT_PLTY;
if( !DeviceIoControl (hGPIO, IOCTL_GPIO_CONFIG, &GPIO,
                     sizeof(GPIOMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}
```

10.2.3.5. Read GPIO inputs

GPIOs working as inputs can be read using the **ReadFile** function, passing as arguments the handle and the **GPIOMessage** structure. This function fills the **unValue** field inside **GPIOMessage** structure, with the current value of the GPIO selected in **unPinNumber** field.

For reading an input value, the number of bytes to read (one) must be provided.

```
uint uiBytesToRead = 1;
uint uiBytesTransferred;
GPIOMessage GPIO;

/* Read an input value */
GPIO.unPinNumber = PIN_NUMBER; /* Set the GPIO number here */
ReadFile(hGPIO, &GPIOMEssage, uiBytesToRead, &uiBytesTransferred, 0)
if( uiBytesTranferred != uiBytesToRead )
{
    /* ERROR */
}
```

10.2.3.6. Set GPIO outputs

GPIOs working as outputs can be written using the **WriteFile** function, passing as argument the handle and the **GPIOMessages** structure. This function sets the GPIO selected in **unPinNumber** field to the value selected in **unValue** field.

For writing an output value, the number of bytes to write (one) must be provided:

```
uint uiBytesToWrite = 1;
uint uiBytesTransferred;
GPIOMessage GPIO;

/* Set an output to a certain value */
GPIO.unPinNumber = PIN_NUMBER; /* Set the GPIO number here */
GPIO.ulFlags = 0;
GPIO.unValue = VALUE; /* Set the value here */
WriteFile(hGPIO, &GPIOMEssage, uiBytesToWrite, &uiBytesTransferred, 0)
if( uiBytesTranferred !=  uiBytesToWrite)
{
    /* ERROR */
}
```

10.2.3.7. GPIO wait for IRQ on interrupt

GPIOs working as external interrupt can use the **DeviceIOControl** function, passing as arguments the handle and the GPIOMessages structure. The IOCTL **IOCTL_GPIO_WAIT_FOR_IRQ** waits until the condition defined for the GPIO IRQ occurs.

```
uint uiBytesTransferred;
GPIOMessage GPIO;

/* Configure GPIO to wait IRQ */
if(DeviceIoControl (hGPIO, IOCTL_GPIO_WAIT_FOR_IRQ, &GPIO,
                     sizeof(GPIOMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}
```

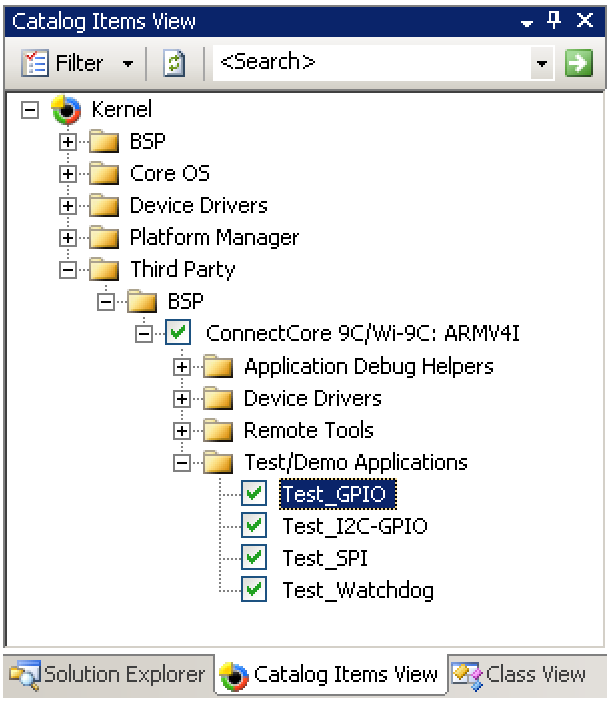
10.2.3.8. Close the handle

When done working with a GPIO, its handle must be closed to free the resources. This is done with the **CloseHandle** function:

```
CloseHandle( hGPIO );
```

10.2.3.9. Test application

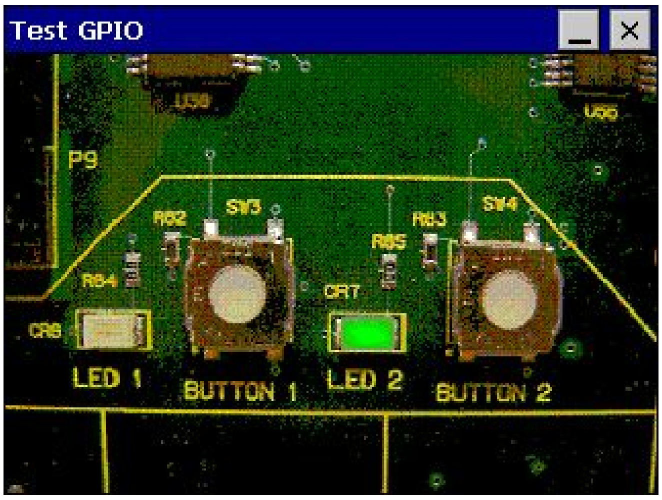
The ConnectCore 9C/Wi-9C BSP contains a test application of the GPIOS that uses two buttons and two LEDs of the development board. This application is included in the ConnectCore 9C and Wi-9C templates by default. The source code is in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\Test_GPIO**.



The **Test_GPIO** is a C# application that assigns **GPIO48** and **GPIO49** to **LED1** and **LED2** respectively, and **GPIO72** and **GPIO69** to **BUTTON1** and **BUTTON2** respectively. The GPIOs assigned to LEDs are configured as outputs and the ones assigned to buttons are configured as inputs.

The **Test_GPIO** is in the **Windows** folder (in the target) and it works like this: The buttons work like switches. When **BUTTON1** is pressed, **LED1** inverts its state; when **BUTTON2** is pressed, **LED2** inverts its state.

The application interface shows an image of the development board with the status of the buttons and LEDs.



10.3. Ethernet interface

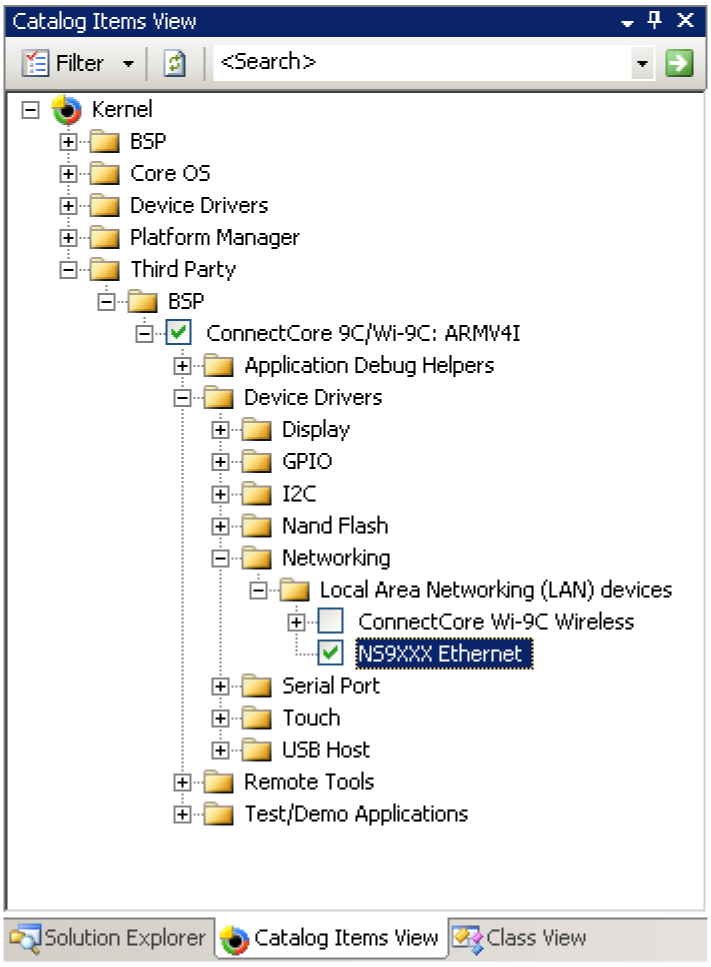
The NS9XX0 processor contains a high performance 10/100 Ethernet controller. This interface is extracted to the ConnectCore 9C/Wi-9C module in the form of an RJ45 network connector.

10.3.1. Hardware resources used by the driver

Interface	Driver	IRQ	GPIO	Physical Memory	Timer	Chip Select
Ethernet	LANNS9xxx		50-64			

10.3.2. Enable the Ethernet interface in the kernel

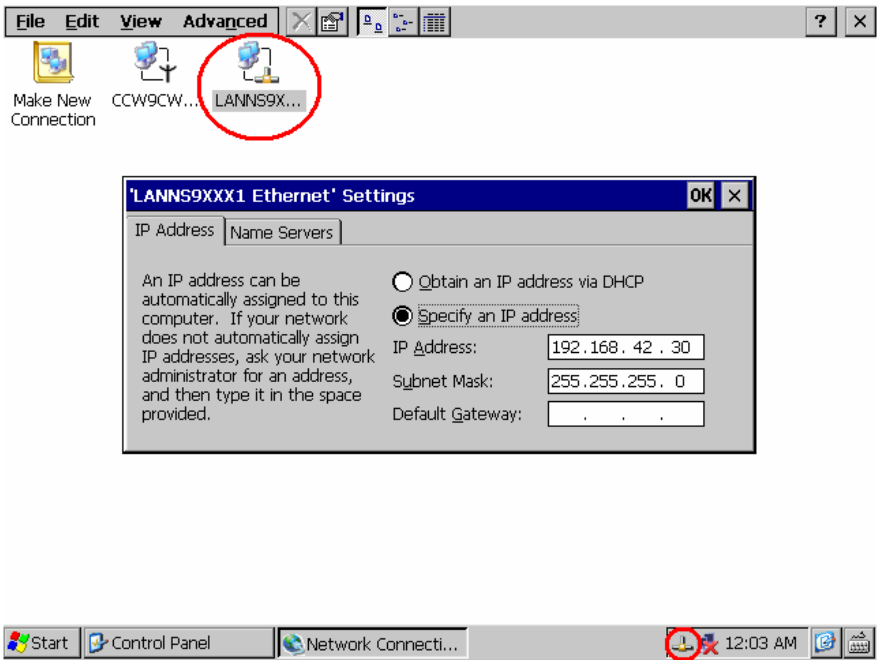
To include this device driver into the OS design, select it in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > Networking > Local Area Networking (LAN) devices**.



```
Kernel
  BSP
  Core OS
  Device Drivers
  Platform Manager
  Third Party
    BSP
      [✓] ConnectCore 9C/Wi-9C: ARMV4I
        Application Debug Helpers
        Device Drivers
          Display
          GPIO
          I2C
          Nand Flash
          Networking
            Local Area Networking (LAN) devices
              [ ] ConnectCore Wi-9C Wireless
              [✓] NS9XXX Ethernet
          Serial Port
          Touch
          USB Host
        Remote Tools
        Test/Demo Applications
```

10.3.3. The Ethernet interface in the system

The Ethernet interface can be accessed in Windows Embedded CE using the **Control Panel > Network and Dial-up connections > LANNS9XX01**, where settings like the IP can be modified. A network icon can also be seen in the taskbar (with a red X if a connection has not been established).



'LANNS9XXX1 Ethernet' Settings

An IP address can be automatically assigned to this computer. If your network does not automatically assign IP addresses, ask your network administrator for an address, and then type it in the space provided.

Obtain an IP address via DHCP [✓ Specify an IP address]

IP Address: 192.168. 42 . 30

Subnet Mask: 255.255.255. 0

Default Gateway: . . .

10.4. Wireless

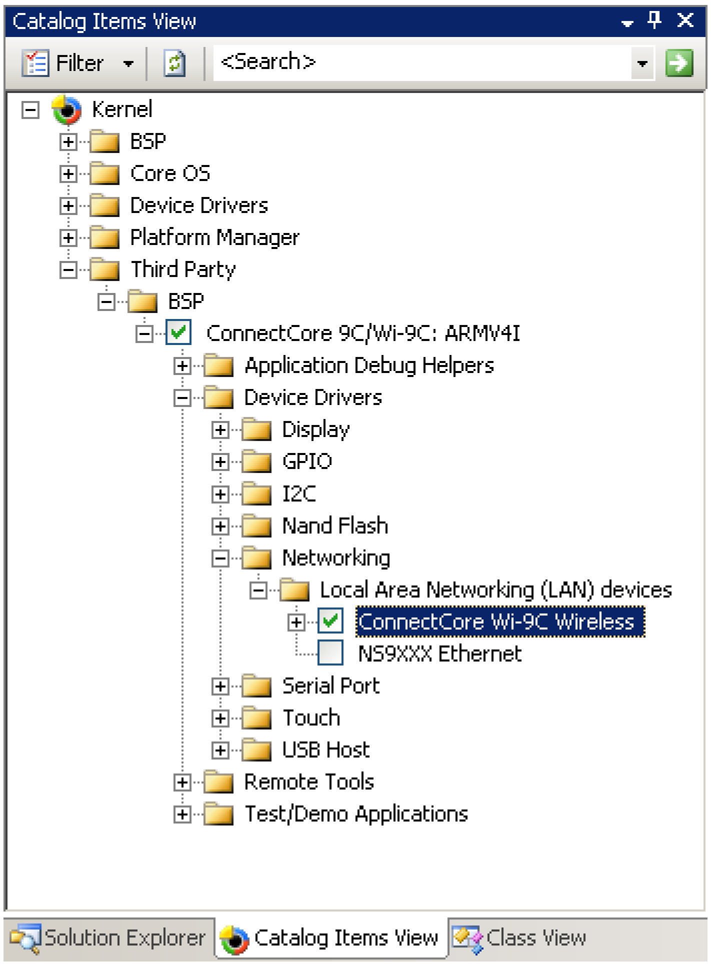
The ConnectCore Wi-9C module contains a Field-Programmable Gate Array (FPGA) which implements an IEEE 802. 11ab/g-compatible wireless network interface. Extensive information about the WLAN adapter is given in topic 11.

10.4.1. Hardware resources used by the driver

Interface	Driver	IRQ	GPIO	Physical memory	Timer	Chip Select
WLAN	ccw9cwifi		58,65-67	0x6xxxxxxx		static 2

10.4.2. Enable the Wireless interface in the kernel

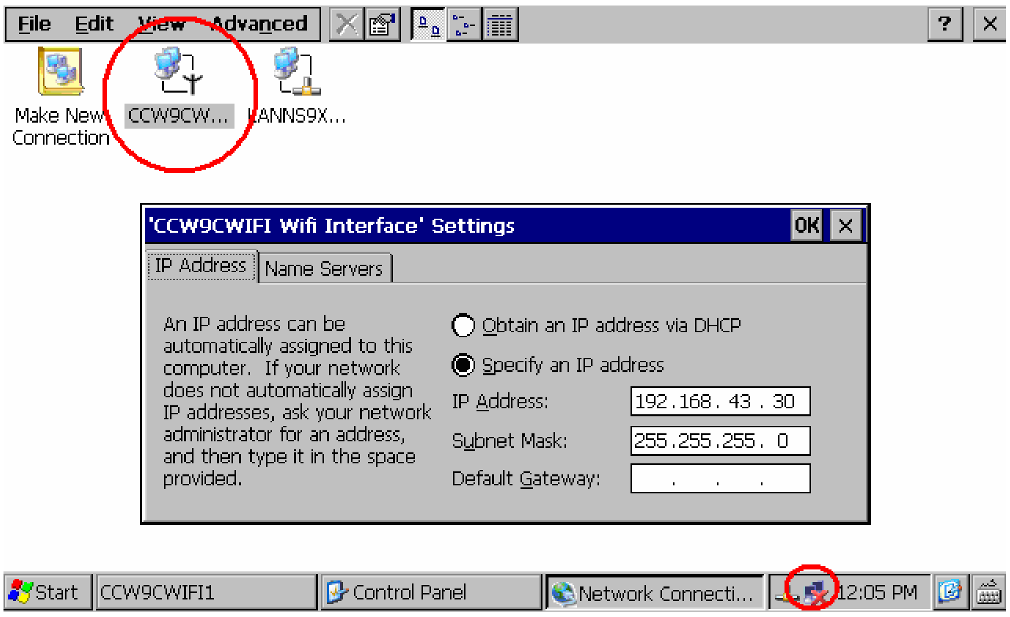
To include this device driver into the OS design, select it in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > Networking > Local Area Networking (LAN) devices**.



```
Catalog Items View
Filter   <Search>
⊟ Kernel
   ⊞ BSP
   ⊞ Core OS
   ⊞ Device Drivers
   ⊞ Platform Manager
   ⊟ Third Party
      ⊟ BSP
         ⊟ ✓ ConnectCore 9C/Wi-9C: ARMV4I
            ⊞ Application Debug Helpers
            ⊟ Device Drivers
               ⊞ Display
               ⊞ GPIO
               ⊞ I2C
               ⊞ Nand Flash
               ⊟ Networking
                  ⊟ Local Area Networking (LAN) devices
                     ⊞ ✓ ConnectCore Wi-9C Wireless
                       ☐ NS9XXX Ethernet
               ⊞ Serial Port
               ⊞ Touch
               ⊞ USB Host
            ⊞ Remote Tools
            ⊞ Test/Demo Applications
Solution Explorer | Catalog Items View | Class View
```

10.4.3. The wireless interface in the system

The Wireless interface can be accessed in Windows Embedded CE using the **Control Panel > Network and Dial-up connections > CCW9CWIFI1**, where settings like the IP can be modified. A network icon can also be seen in the taskbar.



'CCW9CWIFI Wifi Interface' Settings

IP Address | Name Servers

An IP address can be automatically assigned to this computer. If your network does not automatically assign IP addresses, ask your network administrator for an address, and then type it in the space provided.

Obtain an IP address via DHCP [✓ Specify an IP address]

IP Address: 192.168. 43 . 30

Subnet Mask: 255.255.255. 0

Default Gateway: . . .

10.5. Flash memory device

The ConnectCore 9C/Wi-9C modules contain a flash memory device for permanent storage of user data, the boot loader, the kernel, the wireless FPGA program, and the persistent NVRAM settings.

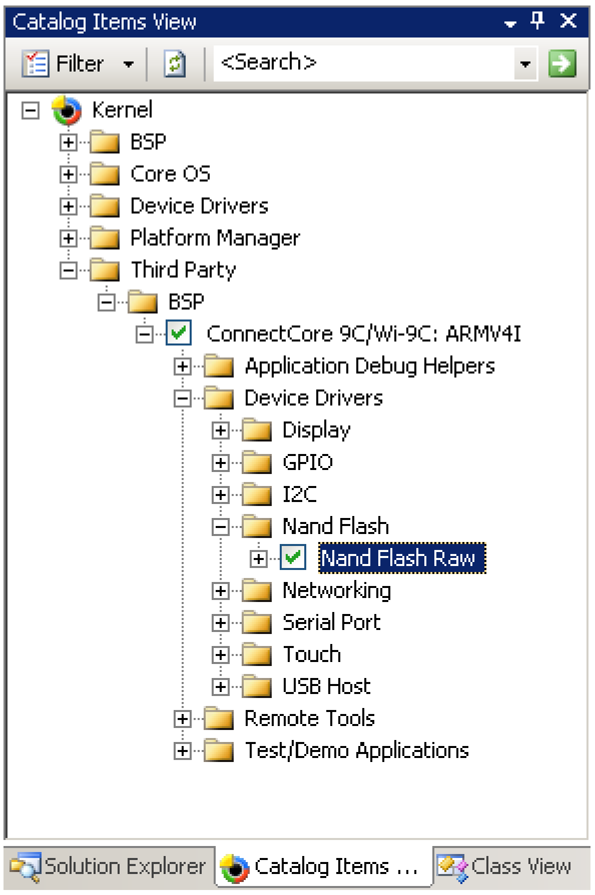
The memory chip is a NAND flash chip which is offered in different size configurations. A driver is implemented to be able to read and write to the flash.

10.5.1. Hardware resources used by the driver

Device	Driver	IRQ	GPIO	Physical memory	Timer	Chip Select
Flash	Ccx9c_nfd			0x50xxxxxx		static 1

10.5.2. Enable the device in the kernel

The **nandflash** driver grants raw access to the flash memory device by means of the standard stream interface (**CreateFile**, **ReadFile**, **WriteFile**, **DeviceIoControl**). The **regtool** application for example, explained in topic 12.1, uses this interface to save the Registry in a flash partition; for more information see topics 11.11.4 and 11.11.5. To include the **nandflash** driver into the OS design, select it in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > Nand Flash**.



10.6. Serial port device drivers

The NS9XX0 microprocessor contains four serial ports that can operate in UART or SPI master/slave modes. Two serial drivers control the internal serial ports: one for UART mode and another for SPI master mode.

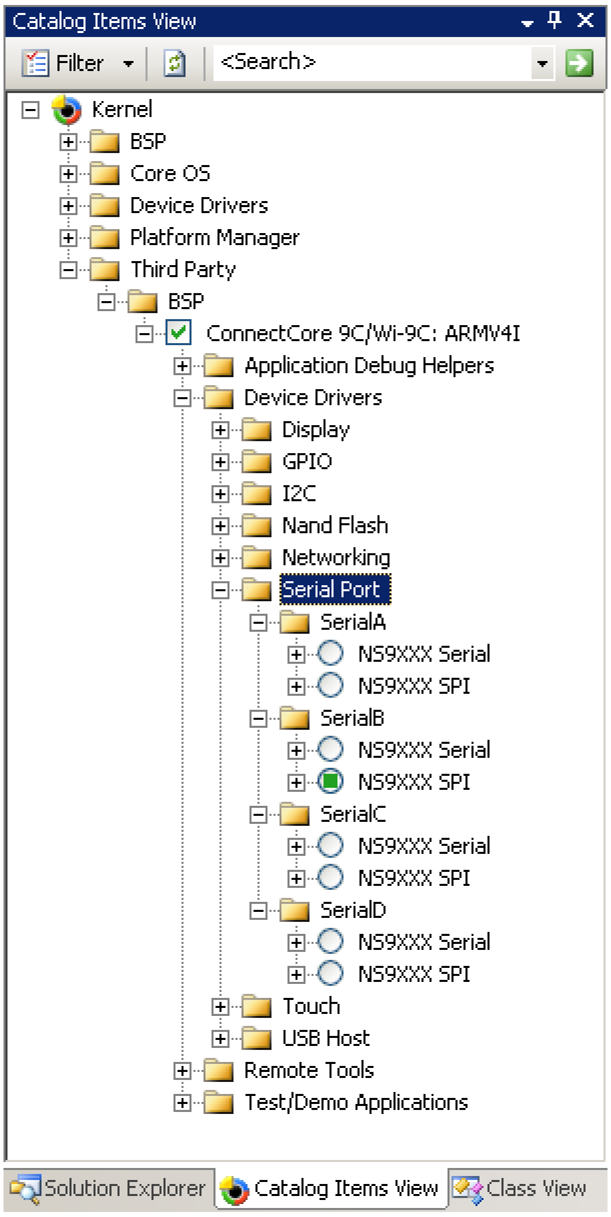
10.6.1. Hardware resources used by the driver

Device	Driver	IRQ	GPIO	Physical Memory	Timer	Chip Select
Serial A (UART mode)	ccx9c_serial	36,37	8,9 (10-15)[1]			
Serial B (UART mode)	ccx9c_serial	34,35	0,1 (2-7)[1]			
Serial C (UART mode)	ccx9c_serial	38,39	40,41 (42,43, 20-23)[1]			
Serial D (UART mode)	ccx9c_serial	40,41	44,45 (46,47, 24-27)[1]			
Serial A (SPI mode)	ccx9c_spi	60,61	8,9,14,15			
Serial B (SPI mode)	ccx9c_spi	58,59	0,1,6,7			
Serial C (SPI mode)	ccx9c_spi	62,63	40,41,22,23			
Serial D (SPI mode)	ccx9c_spi	64,65	44,45,26,27			

[1] Only when HW Handshaking is enabled (default)

10.6.2. Enable the serial ports in the kernel

Support in UART or SPI modes can be separately enabled for each of the four serial ports. To enable support for a serial port into the OS design, select the corresponding mode (UART or SPI) in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > Serial Port**.



```
Kernel
  BSP
  Core OS
  Device Drivers
  Platform Manager
  Third Party
    BSP
      ConnectCore 9C/Wi-9C: ARMV4I
        Application Debug Helpers
        Device Drivers
          Display
          GPIO
          I2C
          Nand Flash
          Networking
          Serial Port
            SerialA
              NS9XXX Serial
              NS9XXX SPI
            SerialB
              NS9XXX Serial
              NS9XXX SPI
            SerialC
              NS9XXX Serial
              NS9XXX SPI
            SerialD
              NS9XXX Serial
              NS9XXX SPI
          Touch
          USB Host
        Remote Tools
        Test/Demo Applications
```

If Port B is enabled in UART mode, the touch screen cannot be used (it uses port B in SPI mode).

Serial Ports C and D cannot be used in combination with a display driver (as they share GPIOs).

10.6.3. Identify the serial ports in the system

Windows Embedded CE enumerates the serial UART ports as **COM*n***, where ***n*** is a number from 1 to 4, depending on the number of ports enabled. For example, if only **portB** and **portD** are enabled, **portB** will be numbered as **COM1** and **portD** as **COM2**.

Windows Embedded CE enumerates the serial SPI ports as **SPI*n***, where ***n*** is a number from 1 to 4, depending on the number of ports enabled. For example, if only **portB** and **portD** are enabled, **portB** is enumerated as **SPI1** and **portD** as **SPI2**.

10.6.4. Manage the serial ports from the user space

Serial ports and SPI ports can be managed from the applications using the standard Win32 API. The API is identical, except the name for the serial port, which is **COM*x***, where ***x*** is the enumeration number of the port that should be used.

To use native functions from a C# application, some DLLs' functions must be imported (see topic 14.3).

To use pointers and addresses in a C# application, configure the project for unsafe code (see topic 14.4).

10.6.4.1. Create the handle (SPI)

To use an SPI port, a device handle is needed. This handle is created by calling the **CreateFile** function with the name of the device as first argument (**lpFileName**). The SPI device name is **SPI*n*:** where ***n*** is the index of the instance. For example, if only **portB** and **portD** are enabled, **portB** will be named **SPI1:** and **portD** as **SPI2:**.

This code opens a handle to the first instance of an SPI port:

```
IntPtr hSPI;

hSPI = CreateFile("SPI1:", GENERIC_READ | GENERIC_WRITE,
                    0, 0, OPEN_EXISTING, 0, 0);
if( hSPI == INVALID_HANDLE_VALUE)
{
    /* ERROR */
}
```

10.6.4.2. SPI Device IO controls

The driver supports these IOTCLs:

IOCTL	Description	Parameter
IOCTL_SERIAL_SET_CLOCK_MODE	Configures mode of the SPI	0 (Mode0), 1 (Mode1), 2 (Mode2), 3 (Mode3)
IOCTL_SERIAL_SET_CSPOL	Configures chip select polarity	0 (LOW) 1 (HIGH)
IOCTL_SERIAL_SET_BIT_ORDER	Configures bit order	0(MSB) 1 (LSB)

10.6.4.3. Configure SPI behavior

Serial ports working in SPI mode have some configurable parameters:

- **Mode**: Combination of clock polarity and clock phase (see the processor's hardware reference manual for more information about the four available modes).
- **Chip select polarity**: Active low | Active high.
- **Bit order**: MSB | LSB

This code configures the SPI behavior:

```
byte parameter;
uint uiBytesTransferred;

/* Configure mode */
parameter = MODE; /* Set the mode here */
if( !DeviceIoControl (hSPI, IOCTL_SERIAL_SET_CLOCK_MODE, &parameter,
                      sizeof(byte), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}

/* Configure Chip select polarity */
parameter = CS_POLARITY; /* Set the chip select polarity here */
if( !DeviceIoControl (hSPI, IOCTL_SERIAL_SET_CSPOL, &parameter,
                      sizeof(byte), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}

/* Configure bit order*/
parameter = BIT_ORDER; /* Set the bit order here */
if( !DeviceIoControl (hSPI, IOCTL_SERIAL_SET_BITORDER, &parameter,
                      sizeof(byte), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}
```

10.6.4.4. Read data (SPI)

The MISO (Master Input Slave Output) line supplies the output data from the slave to the input of the master. Data can be read using **ReadFile** function, passing as arguments the handle and a buffer where to place the data. The number of bytes to read also must be supplied.

```
uint uiBytesToRead;
uint uiBytesTransferred;
byte[] buffer;

fixed (byte* p = &buffer)
{
    if ( !ReadFile (hSPI, p, uiBytesToRead, &uiBytesTransferred, 0))
    {
        /* ERROR */
    }
}
```

10.6.4.5. Written data (SPI)

The MOSI (Master Output Slave Input) line supplies the output data from the master to the input of the slave. Data can be written using **WriteFile** function, passing the handle and the buffer with the data. The number of bytes to write must also be supplied.

```
uint uiBytesToWrite;
uint uiBytesTransferred;
byte[] buffer;

fixed (byte* p = &buffer)
{
    if ( !WriteFile (hSPI, p, uiBytesToWrite, &uiBytesTransferred, 0))
    {
        /* ERROR*/
    }
}
```

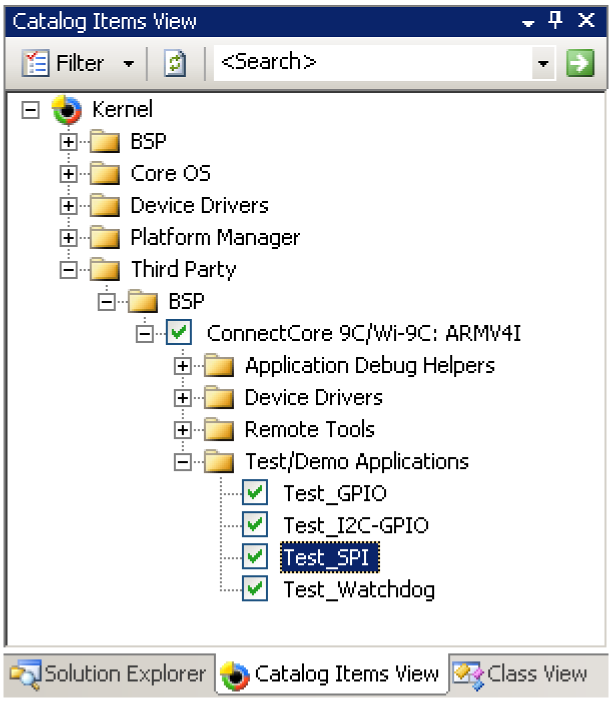
10.6.4.6. Close the handle (SPI)

When done working with an SPI port, close its handle to free the resources. This is done with the **CloseHandle** function:

```
CloseHandle( hSPI );
```

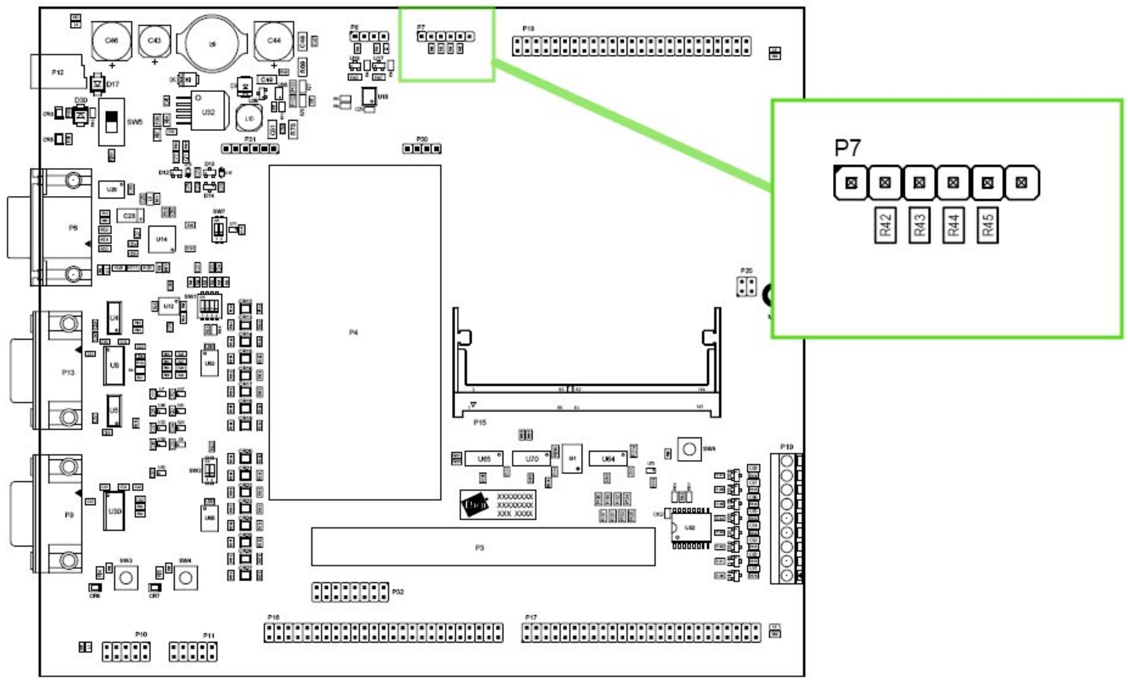
10.6.4.7. SPI port test application

The ConnectCore 9C/Wi-9C BSP includes a test application of the SPI ports. This application is included in the ConnectCore 9C and Wi-9C templates by default. The source code is in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\Test_SPI**.



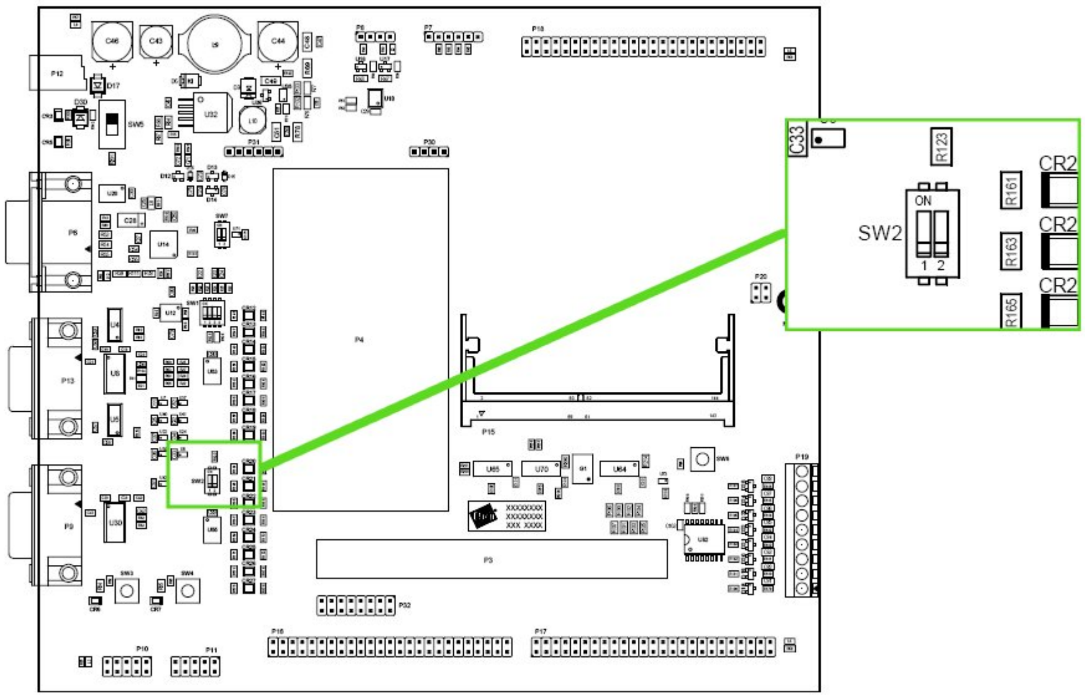
The **Test_SPI** is a C# application that works with the SPI driver. It is in the **Windows** folder. The application opens the SPI port instance selected on the **Port** dropdown list, which lists all the SPI ports available on the OS design. The port instances may vary depending on the ports included in the OS design. For example, if only two ports are available in the OS design, there will be two instances **SPI1** and **SPI2**, regardless of whether the available ports were **PORTA** and **PORTC**, **PORTC** and **PORTD**, or any other combination.

In the default kernel that is running in the target, the only SPI port available is serial port B, accessible at pin header **P7** on the development board.



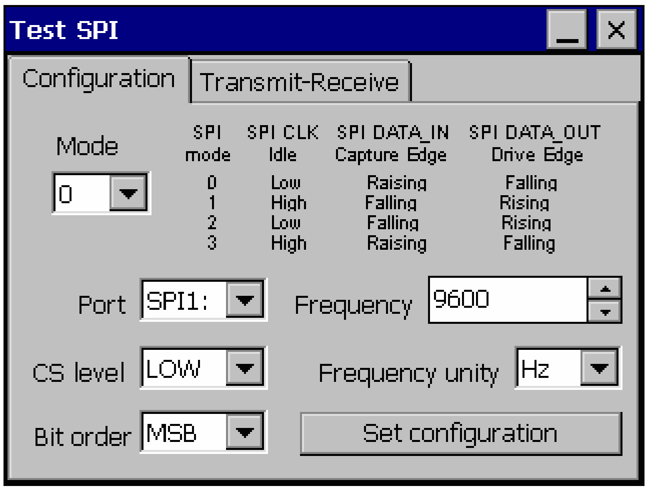
This test transmits and receives data using the same port. For this purpose, the lines **SPI_DOUT** and **SPI_DIN** of the SPI port must be interconnected. In the development board, these port B lines are accessible in connector **P7**, pins 2 and 3.

To use Serial Port B as an SPI, turn off microswitch **SW2.2** on the development board.



SW2	ON	OFF
SW2.2	UART mode	SPI mode

The **Test_SPI** application has two windows in a tab control. The first window contains some controls for modifying SPI parameters: mode, chip select polarity, bit order, and clock frequency. To apply the new settings, click **Set configuration**.



Test SPI

Configuration | Transmit-Receive

Mode: 0

SPI mode	SPI CLK Idle	SPI DATA_IN Capture Edge	SPI DATA_OUT Drive Edge
0	Low	Raising	Falling
1	High	Falling	Rising
2	Low	Falling	Rising
3	High	Raising	Falling

Port: SPI1:

Frequency: 9600

CS level: LOW

Frequency unity: Hz

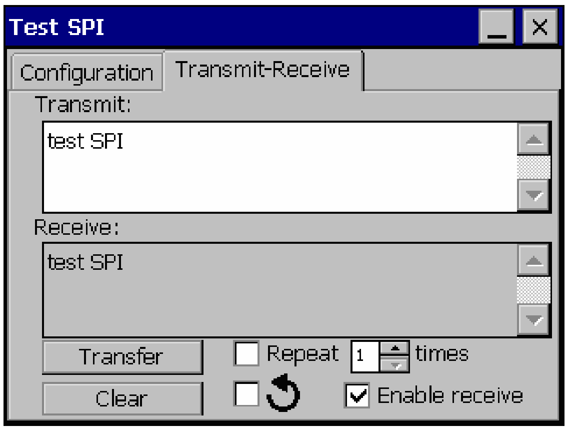
Bit order: MSB

Set configuration

The other window is the transmit-receive window, which contains two text boxes. In the top one, text to be transmitted can be typed. The lower one represents data received, in ASCII.

Additional controls in the window include:

- **Transfer** button: sends the transmit text over the port.
- **Clear** button: clears the transmit and receive text boxes.
- **Repeat** checkbox: repeats the transmission several times.
- **Infinite** checkbox: transmits the data in an infinite loop.
- **Enable receive** checkbox: enables reception of data.



If lines SPI_DOUT and SPI_DIN of the port are not interconnected, 0xFF bytes are received (they might look like ASCII symbols).

10.7. Touch screen

If an LCD Application Kit was purchased, the provided TFT LCD contains a touch screen sensor and an SPI touch screen controller (ADS 7846). The touch screen lines come together with the LCD lines in a single cable. Internally, the touch screen lines are connected to Serial Port B.

To use Serial Port B as a Serial Peripheral Interface (SPI), turn off microswitch **SW2.2** on the development board.

SW2	ON	OFF
SW2.2	UART mode	SPI mode

A driver implements support for the ADS 7846 touch screen controller.

10.7.1. Hardware resources used by the driver

Device	Driver	IRQ	GPIO	Physical memory	Timer	Chip Select
Touch screen ADS7846	touch	EXT 3	0,1,6,7			

10.7.2. Enable the touch screen device in the kernel

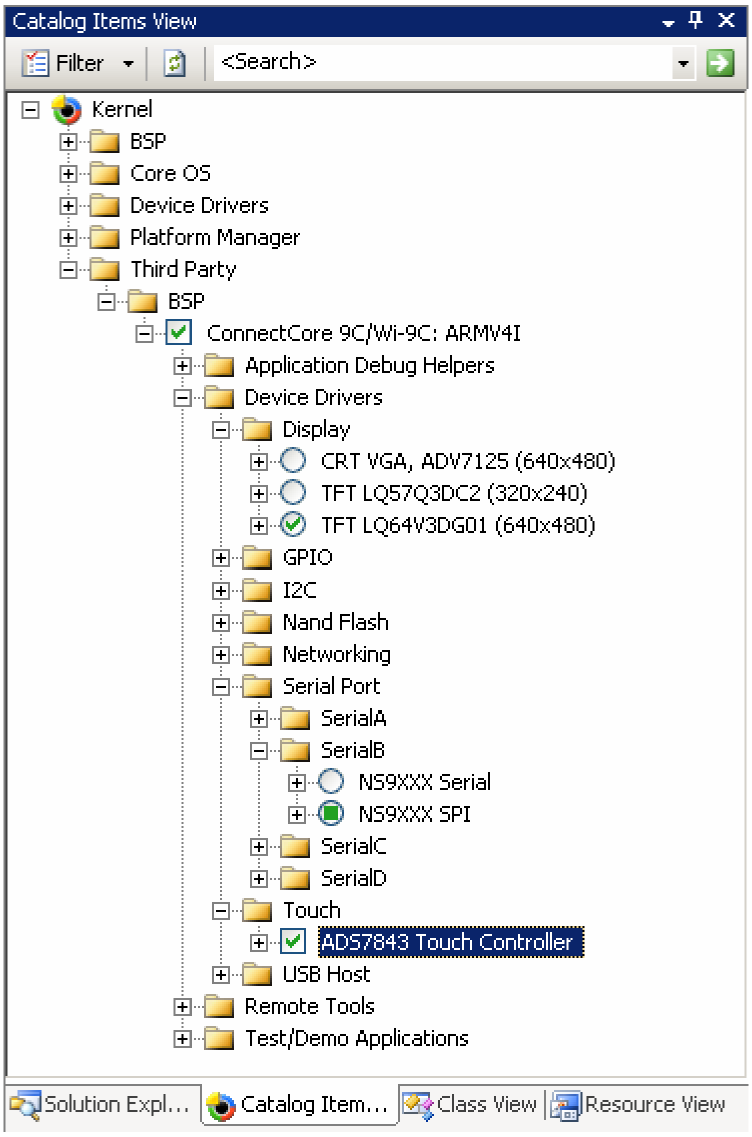
To include support for the touch screen controller in the OS design, select it in Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > Touch**.

To enable the touch screen device:

A TFT must be selected as display.

Serial port B must be selected in SPI mode.

If a device listed under Touch is marked with a red cross, review the SerialB component in the catalog to make that SerialB has been selected as SPI, not both SPI and Serial.



```
Catalog Items View
Filter    <Search>
Kernel
    BSP
    Core OS
    Device Drivers
    Platform Manager
    Third Party
        BSP
            ✓ ConnectCore 9C/Wi-9C: ARMV4I
                Application Debug Helpers
                Device Drivers
                    Display
                        ○ CRT VGA, ADV7125 (640x480)
                        ○ TFT LQ57Q3DC2 (320x240)
                        ✓ TFT LQ64V3DG01 (640x480)
                    GPIO
                    I2C
                    Nand Flash
                    Networking
                    Serial Port
                        SerialA
                        SerialB
                            ○ NS9XXX Serial
                            ● NS9XXX SPI
                        SerialC
                        SerialD
                    Touch
                        ✓ ADS7843 Touch Controller
                    USB Host
                Remote Tools
                Test/Demo Applications
Solution Expl...   Catalog Item...   Class View   Resource View
```

10.7.3. The touch screen interface in the system

Windows Embedded CE automatically recognizes the touch screen as an input device.

When support for the touch screen is included into the OS design, the **touchcal.exe** calibration application is automatically included too.

10.7.3.1. Calibrating the touch screen

The **touchcal.exe** application is launched automatically when the system starts, if no calibration information is found in the Registry. The application requests the user to press at certain places to calibrate the touch screen.

The source code is in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\TouchCal**.

Once calibrated, the information can optionally be saved in the Registry with the **regtool.exe** application (explained in topic 12.1).

10.8. USB host interface

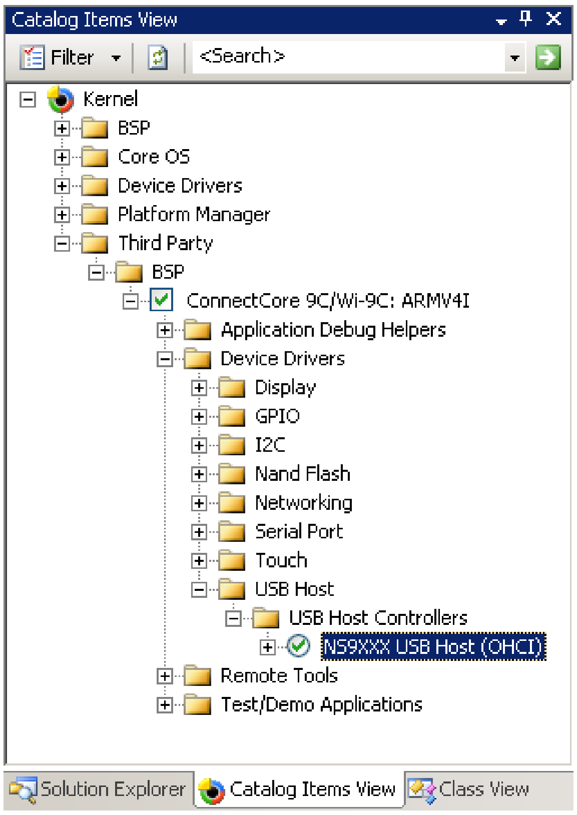
The NS9360 processor contains a USB 2.0 host interface that supports full-speed (12 Mbps) and low-speed (1.5 Mbps). The interface is extracted to the ConnectCore 9C/Wi-9C module in the form of two USB host ports.

10.8.1. Hardware resources used by the driver

Device	Driver	IRQ	GPIO	Physical Memory	Timer	Chip Select
USB Host	ccx9c_usb					

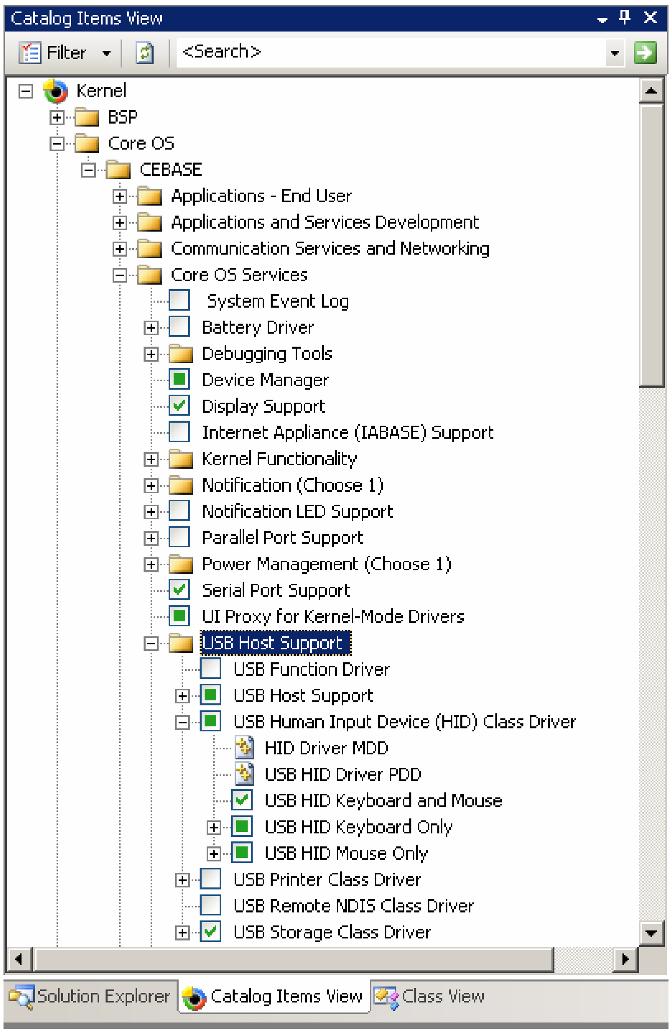
10.8.2. Enable the interface in the kernel

To include this device driver into the OS design, select it in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > USB Host > USB Host Controllers**.



The USB host (OHCI) driver supports only the interface. Specific USB devices are supported by USB class drivers, like Human Input Devices (HID) class driver (for mice and keyboards), the Storage class driver (for USB flash disks), and so on.

Support for USB class drivers can be included from the catalog view, under **Core OS > CEBASE > Core OS Services > USB Host Support**.



```
Kernel
  BSP
  Core OS
    CEBASE
      Applications - End User
      Applications and Services Development
      Communication Services and Networking
      Core OS Services
        System Event Log
        Battery Driver
        Debugging Tools
        Device Manager
        Display Support
        Internet Appliance (IABASE) Support
        Kernel Functionality
        Notification (Choose 1)
        Notification LED Support
        Parallel Port Support
        Power Management (Choose 1)
        Serial Port Support
        UI Proxy for Kernel-Mode Drivers
        USB Host Support
          USB Function Driver
          USB Host Support
          USB Human Input Device (HID) Class Driver
            HID Driver MDD
            USB HID Driver PDD
            USB HID Keyboard and Mouse
            USB HID Keyboard Only
            USB HID Mouse Only
          USB Printer Class Driver
          USB Remote NDIS Class Driver
          USB Storage Class Driver
```

10.8.3. USB devices in the system

USB host devices can be plugged directly to the module's USB connectors. If the corresponding class driver was included in the kernel, the device is automatically recognized by the system thus becoming ready to be used.

10.8.3.1. USB memory sticks

USB memory sticks formatted with FAT file system are recognized as storage units and are therefore populated with the name **Hard Disk n**. Windows Embedded CE also recognizes the different partitions on USB memory sticks, and populates each partition with a new storage unit entry.

This is how a USB memory stick with three partitions would look:

10.9. I^{2}C

The NS9XX0 processor contains an I^{2}C v.1.0 port, which can be configured in both master and slave modes. A custom driver has been developed for this interface in master mode.

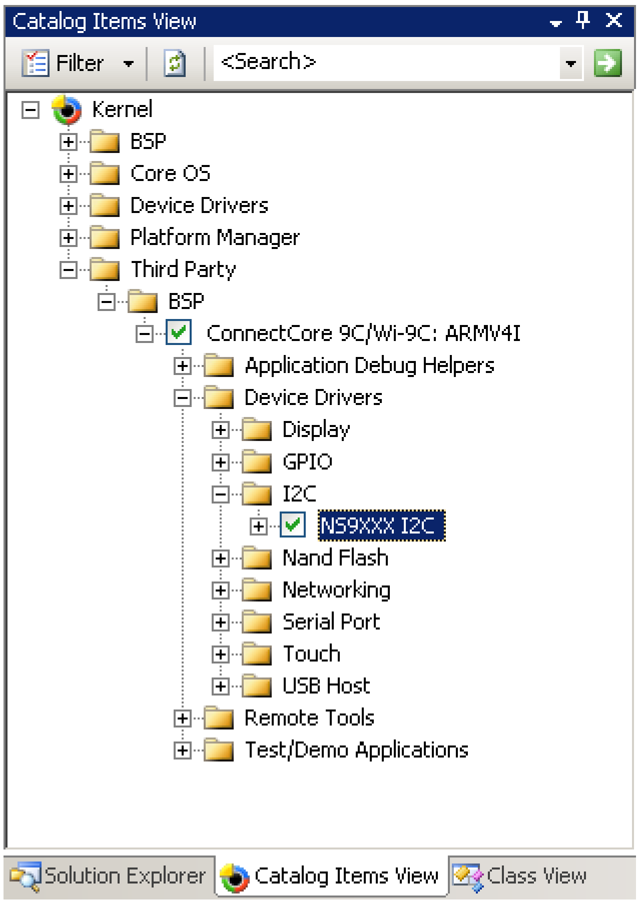
Additionally, the development board contains an I^{2}C 8-bit I/O device (Philips PCA9554). This device is managed by the I^{2}C interface.

10.9.1. Hardware resources used by the interface

Device	Driver	IRQ	GPIO	Physical Memory	Timer	Chip Select
I^{2}C	ns9xxx_i2c		46,47			
I^{2}C I/O port PCA9554						

10.9.2. Enable the interface in the kernel

To include this device driver into the OS design, select it in the Catalog view, under **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > I2C**.



10.9.3. Manage the interface from user space

I^{2}C devices can be managed from the applications using the standard Win32 API.

To use native functions from a C# application, the functions of some DLLs must be imported (see topic 14.3).

To use pointers and addresses in a C# application, the project must be configured for unsafe code (see topic 14.4).

10.9.3.1. Create the handle

To use the I^{2}C port a device handle is needed. This handle is created by calling the **CreateFile** function with the name of the device as first argument (*lpFileName*). The SPI device name is **SPI*n*:** where ***n*** is the index of the instance. Because there is only one port, the name is **I2C1**.

This code opens a handle to the I2C port:

```
IntPtr hGPIO;

hGPIO = CreateFile("I2C1:", GENERIC_READ | GENERIC_WRITE,
                    0, 0, OPEN_EXISTING, 0, 0);
if( hGPIO == INVALID_HANDLE_VALUE)
{
    /* ERROR*/
}
```

10.9.3.2. I2CMessage structure

The handle grants access to all I2C devices. To manage a specific I2C device the **I2CMessage** structure must be used. This structure is an argument to the functions of the API (**DeviceIoControl**, **ReadFile** and **WriteFile**). The structure contains these fields:

```
public struct I2CMessage
{
    public uint chip;
    public uint addr;
    public uint alen;
    public byte *buffer;
    public int count;
}
```

Type	Field	Description	Values
uint	chip	Address of the I^{2}C device	0 to 128 (LSB of address byte is used as read/write toggle in I^{2}C protocol)
uint	addr	Command byte to the I^{2}C device (can be an inner address or register)	0 to max uint.
uint	alen	Number of bytes of field **addr** (can be more than one if addr exceeds value 255)	1 to sizeof(uint)
byte *	buffer	Buffer with data to write and where data read are stored	
int	count	Size of **buffer** field	

10.9.3.3. Device IO controls

The driver supports these IOTCLs:

IOCTL	Description	Parameter
IOCTL_I2C_READ	Reads register value	I2CMessage
IOCTL_I2C_WRITE	Writes register value	I2CMessage

10.9.3.4. PCA9554 8-bit I/O port

This I^{2}C device has four inner registers (Input port, Output port, Polarity inversion and Configuration). The base address and the inner registers addresses are defined with constants in the application.

Constant	Description	Value
CFG_I2C_GPIO_EXP_ADDR	Base address of PCA9554 device	0x20
I2C_READ_INPUTS	Input port register address	0
I2C_RW_OUTPUTS	Output port register address	1
I2C_RW_POL_INVERT	Polarity inversion register address	2
I2C_RW_CONFIG	Configuration register address	3

The **buffer** field of the **I2CMessage** structure is used for referring to the specific GPIOs. This buffer will be only one byte, where the LSB refers to **I^{2}C-GPIO0** and the MSB to **I^{2}C-GPIO7**.

10.9.3.5. Configure PCA9554 I/O behavior

The eight I/Os of the PCA9554 device can be configured as inputs or outputs independently. The behavior is configured using **IOCTL_I2C_WRITE** to write to the **I2C_RW_CONFIG** register. Setting a bit to 1 configures that pin as input. Clearing the bit configures it as output.

This example code configures **I^{2}C-GPIO0** to **I^{2}C-GPIO3** as inputs and the rest as outputs:

```
I2CMessage I2C;
uint uiBytesTransferred;
byte buffer;

/* Configure I2C-GPIO0 to I2C-GPIO3 as INPUTS and
             I2C-GPIO4 to I2C-GPIO7 as OUTPUTS */
buffer= 0x0F;
I2C.buffer = &buffer;
I2C.addr = I2C_RW_CONFIG;
if( !DeviceIoControl (hI2C, IOCTL_I2C_WRITE, &I2C,
                     sizeof(GPIOMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}
```

10.9.3.6. Read the inputs

I^{2}C-GPIOs working as inputs can be read using the **IOCTL_I2C_READ**, passing as argument the handle and the I2CMessage structure. The value of the Input Port Register is returned in the **buffer** field of the I2CMessage structure:

```
I2CMessage I2C;
uint uiBytesTransferred;

I2C.addr = I2C_READ_INPUT;
if( !DeviceIoControl (hI2C, IOCTL_I2C_READ, &I2C,
                     sizeof(I2CMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}
```

10.9.3.7. Set the outputs

I^{2}C-GPIOs working as outputs can be written using the **IOCTL_I2C_WRITE**, passing as argument the handle and the I2CMessages structure. Before calling the **DeviceIoControl** function, the Output Port Register value must be set in the **buffer** field:

```
I2CMessage I2C;
byte data;
uint uiBytesTransferred;

data = OUTPUT_VALUE;
I2C.buffer = &data;
I2C.addr = I2C_RW_OUTPUT;
if( !DeviceIoControl (hI2C, IOCTL_I2C_WRITE, &I2C,
                     sizeof(I2CMessage), 0, 0, &uiBytesTransferred, 0))
{
    /* ERROR */
}
```

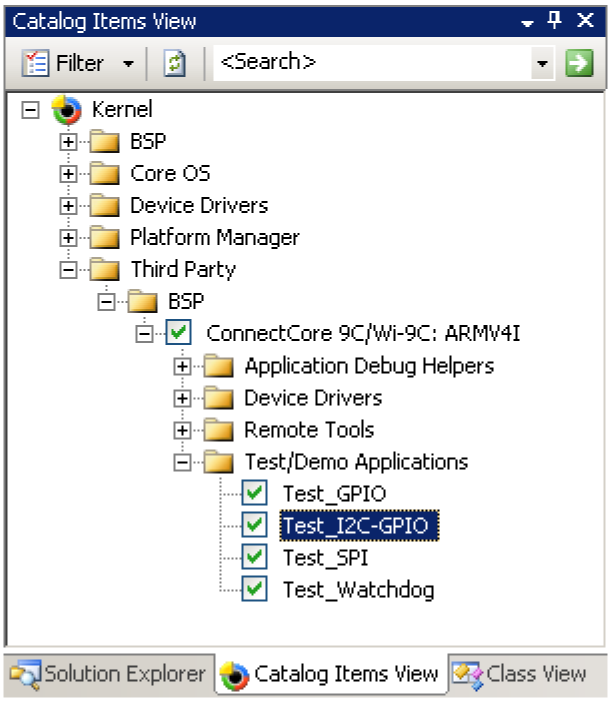
10.9.3.8. Close the handle

When done working with the I^{2}C port, its handle must be closed to free the resources. This is done with the **CloseHandle** function:

```
CloseHandle( hI2C );
```

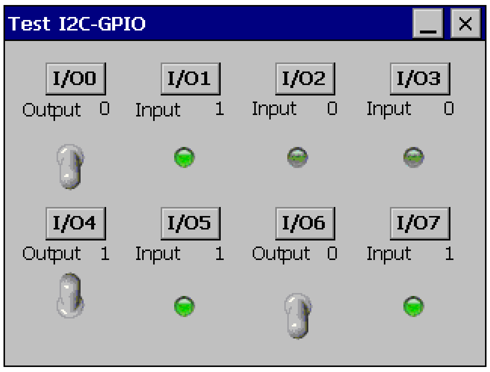
10.9.3.9. I2C test application

The ConnectCore 9C/Wi-9C BSP includes a test application of the I2C-GPIOS, which uses the eight general purpose input-output working of PCA9554 device. This application is included in the ConnectCore 9C and Wi-9C templates by default. The source code is in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\Test_I2C-GPIO**.

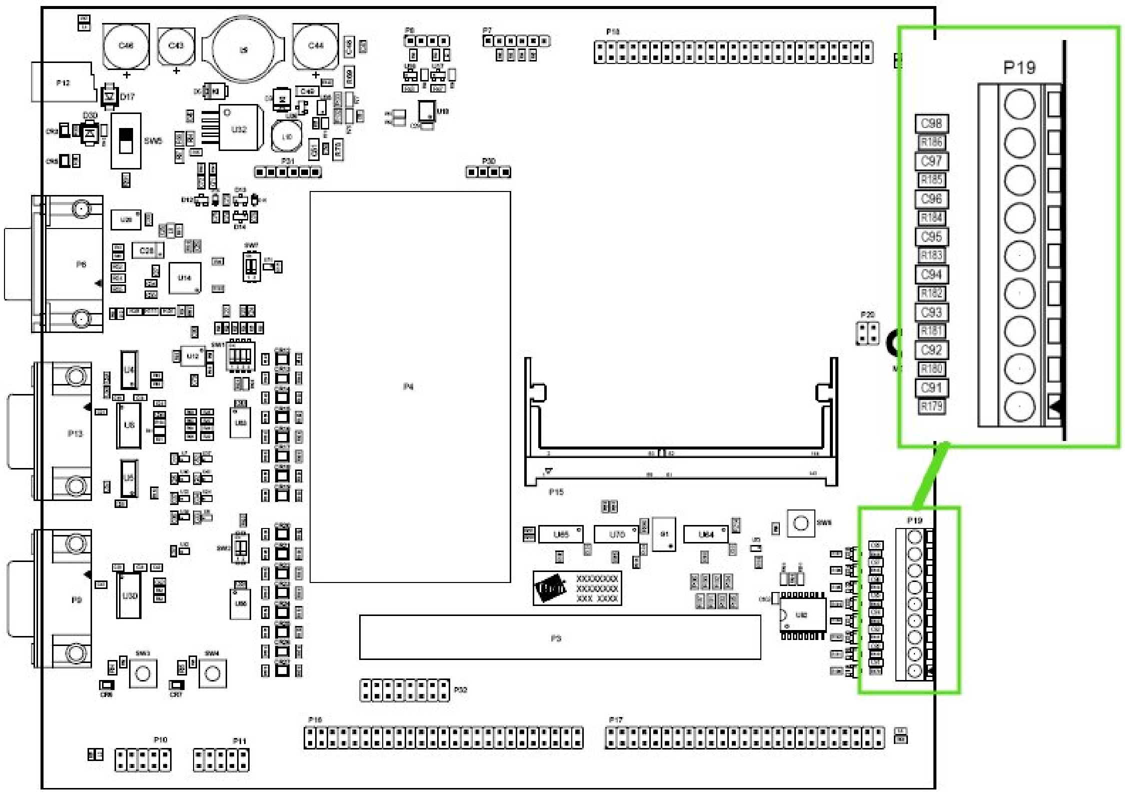


The **Test_I2C-GPIO** is a C# application that works with some of the I^{2}C registers (input port, output port and configuration). It is in the **Windows** folder.

The application contains eight buttons with labels going from **I/O0** to **I/O7**. Each button allows changing the behavior of the pin between input or output. Depending on their behavior, the I/Os are represented as a switch (output) or as a LED (input). A label under each button also tells whether the I/O is an input or an output, and its value.



The I/Os are accessible in connector **P19** in the development board.



To change the status of an output, click the switch drawing in the **Test_I2C-GPIO** application. To change the value of an input, force it by hardware to ground (0V) or to Vcc (5V).

10.10. RTC

The NS9XXX processor contains a real time clock module that tracks the time of the day to an accuracy of 10 milliseconds and provides calendar functionality that tracks day, month, and year.

Support for the RTC is provided by an OAL kernel layer.

Because the RTC is internal to the processor, no battery maintains the date/time. This device is mainly intended for triggering alarms and scheduled jobs.

10.10.1. Manage the device from user space

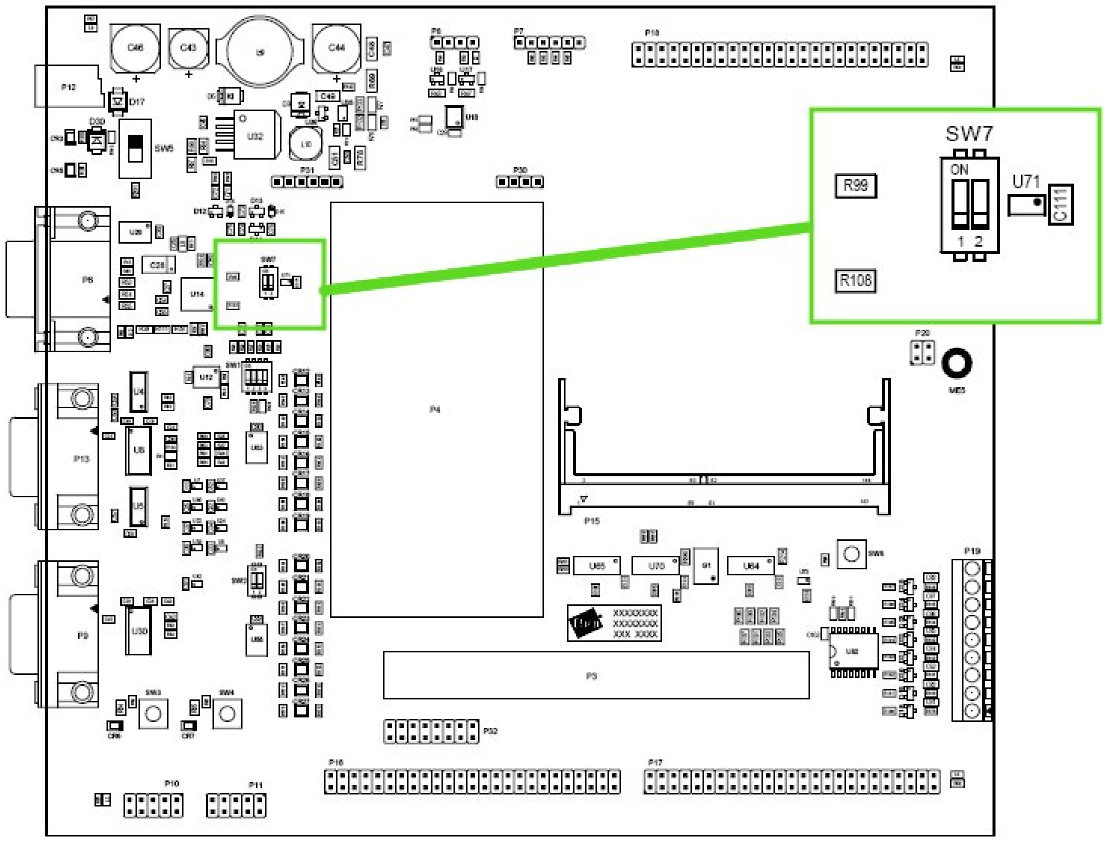
To manage the RTC, use the standard Time Functions of the Win32 API. Search for "Time Functions" in the Windows Embedded CE online help.

10.11. Video

Two options are available for video/graphics support using the ConnectCore 9C/Wi-9C modules: TFT LCD display or VGA monitor.

The NS9360 microprocessor contains a flexible LCD controller for TFT LCD displays. The development board contains a VGA DAC (external to the module) that converts the digital signal to analog, for VGA monitors.

Only one video configuration can be used at a time on the development board: either TFT LCD display or VGA monitor. Micro switch SW7 on the development board configures this setting.



SW7	ON	OFF
SW7.1	VGA disabled	VGA enabled
SW7.2	not used	

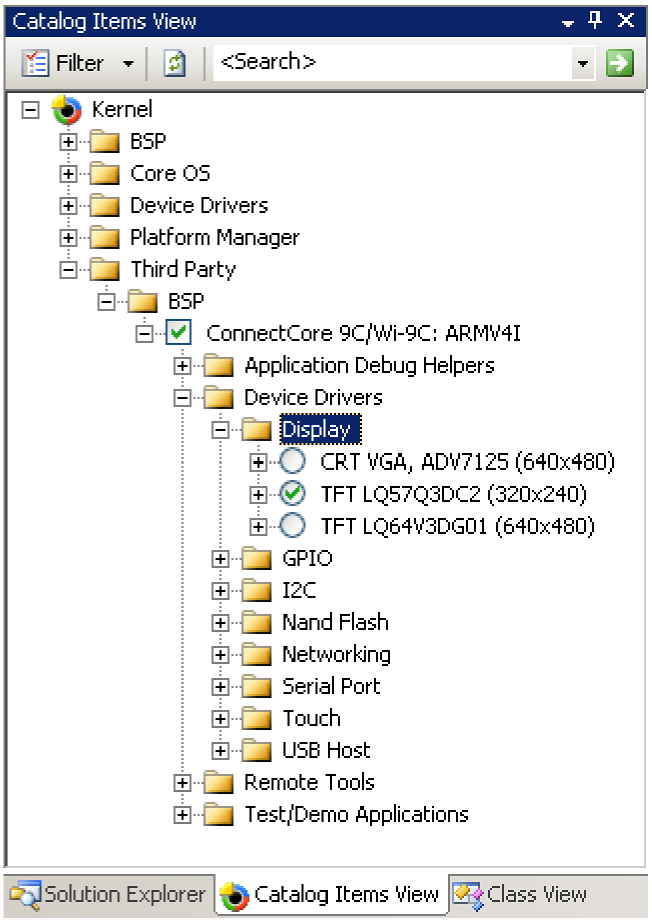
10.11.1. Hardware resources used by the driver

Device	Driver	IRQ	GPIO	Physical memory	Timer	Chip Select
Display	ns9xxx_disp		15,18-41			

10.11.2. Include video support in the kernel

When using the ConnectCore 9C/Wi-9C templates for creating an OS design, the NetSilicon VGA display driver is included by default into the kernel.

To include it by hand or substitute it with a TFT display go to the Catalog view of the OS design and select **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I > Device Drivers > Display**



The catalog allows the choice of a VGA or TFT driver.

Depending on the display selection, remember to set micro switch SW7.1 in the development board accordingly.

A wrong combination of the selected driver and SW7.1 could lead to a kernel crash while booting.

10.11.3. Manage the display from the user space

The display drivers work with the Windows Embedded CE display API. For more information about the structures, methods, and functions, see these Windows Embedded CE 6.0 online help topics:

- Display Driver Functions
- Display Driver IOCTLs
- Display Driver Methods
- Display Driver Structures

10.12. Watchdog

The NS9XX0 processor contains a watchdog timer that can produce a reset signal if not refreshed on time. This mechanism is useful for preventing an uncontrolled kernel from blocking the system forever.

10.12.1. Enable/Disable the watchdog in the kernel

The kernel scheduler is the responsible for refreshing the watchdog. In Debug versions, where the debug information slows down the system, the watchdog is disabled to avoid system resets. This is done by setting or unsetting the variable **ENABLE_WATCH_DOG** (see topic 3.3.1) in the batch file of the platform, at **%_WINCEROOT%\PLATFORM\CCX9C\ccx9c.bat**, as seen in this excerpt:

```
REM
REM If you enable the Watchdog it will be enabled directly at
REM the beginning when the kernel starts. When enabling the watchdog you
need to verify
REM watchdog refresh time to adjust it to your platform needs.
REM
IF /I "%WINCEDEBUG%"=="retail" set ENABLE_WATCH_DOG=1
IF /I "%WINCEDEBUG%"=="debug" set ENABLE_WATCH_DOG=
```

This code enables the watchdog in **Release** versions and disables it in **Debug** versions, but its use can be changed as needed.

10.12.2. Manage the watchdog from user space

The processor watchdog is entirely managed by the kernel. Applications can use the Win32 watchdog API for creating their own software watchdog timers.

To use native functions from a C# application, the functions of some DLLs must be imported (see topic 14.3).

To use pointers and addresses in a C# application, the project must be configured for unsafe code (see topic 14.4).

10.12.2.1. Create the handle

To use a software watchdog timer, a device handle is needed. This handle is created by calling the **CreateWatchDogTimer** function, which has six arguments:

Type	Argument	Description
string	lpFileName	Name of the watchdog timer to be created.
uint	dwPeriod	Watchdog period (in milliseconds).
uint	dwWait	Time to wait (in milliseconds) after the watchdog timer has expired, before the default action is executed.
uint	dwDfltAction	Default action to execute when the watchdog timer expires. Possible values are: WDG_NO_DFLT_ACTION: do nothing WDG_KILL_PROCESS: terminate the process WDG_RESET_DEVICE: reset the device by calling IOCTL_HAL_REBOOT
uint	dwPara	Parameter to pass to IOCTL_HAL_REBOOT
uint	dwFlags	Reserved, must be set to 0

If the default action is WDG_RESET_DEVICE, the standard IO Control IOCTL_HAL_REBOOT is called. This IO Control uses the hardware watchdog to produce a software reset.

10.12.2.2. Start the watchdog timer

After creating the handle, the watchdog timer is started by calling the **StartWatchDogTimer** function:

```
if (!StartWatchDogTimer(hWatchDog, 0))
{
    /* ERROR */
}
```

10.12.2.3. Refresh the watchdog timer

The watchdog timer needs to be refreshed before the watchdog period expires; otherwise, the default action is triggered. The watchdog is refreshed by calling the **RefreshWatchDogTimer** function:

```
if (!RefreshWatchDogTimer(hWatchDog, 0))
{
    /* ERROR */
}
```

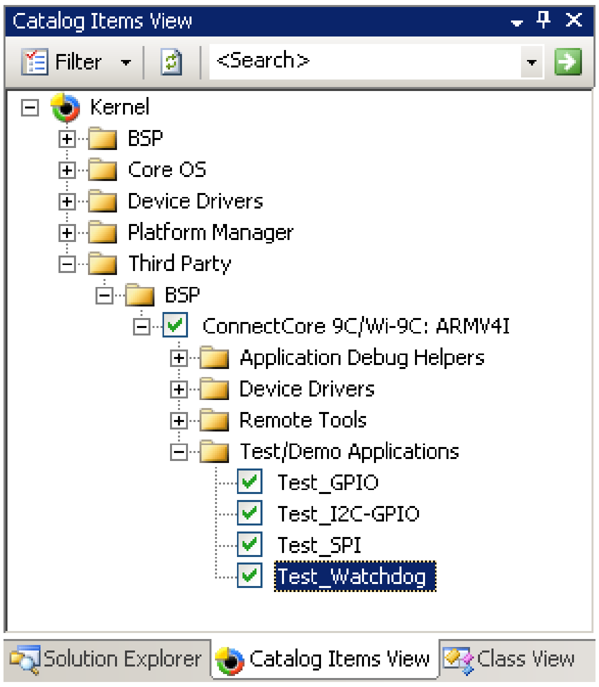
10.12.2.4. Close the handle

When done working with the watchdog, close its handle to free the resources using the **CloseHandle** function:

```
CloseHandle( hWatchDog );
```

10.12.2.5. Watchdog test application

The ConnectCore 9C/Wi-9C BSP includes a test application that uses a software watchdog. This application is included in the ConnectCore 9C and Wi-9C templates by default. The source code is in **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\Test_WatchDog**.



Kernel
- BSP
- Core OS
- Device Drivers
- Platform Manager
- Third Party
  - BSP
    - ConnectCore 9C/Wi-9C: ARMV4I
      - Application Debug Helpers
      - Device Drivers
      - Remote Tools
      - Test/Demo Applications
        - ✓ Test_GPIO
        - ✓ Test_I2C-GPIO
        - ✓ Test_SPI
        - ✓ Test_Watchdog

Test_WatchDog is a C# multi-thread application that uses some functions to handle and refresh a watchdog timer.

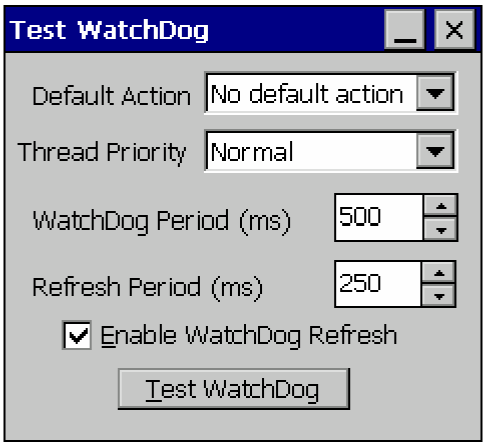
The application contains a combo box for selecting the default action to execute when the software watchdog expires. The possible values are:

- **No default action**: take no default action.
- **Kill process**: terminate the process.
- **Device reset**: reset the device by calling IOCTL_HAL_REBOOT.

The **thread priority** combo box is used to assign the thread one out of these priorities:

- **Highest**
- **Above normal**
- **Normal**
- **Below normal**
- **Lowest**

Two additional controls are used for assigning the watchdog period and the refresh period (in milliseconds).



Test WatchDog

Default Action: No default action

Thread Priority: Normal

WatchDog Period (ms): 500

Refresh Period (ms): 250

[✓] Enable WatchDog Refresh

Test WatchDog

When **Test_WatchDog** is selected, a Watchdog timer is created and started. If **Enable WatchDog refresh** is checked and the refresh period is smaller than the watchdog period, the software watchdog timer is refreshed in time, and nothing happens.

On the other hand, if **Enable WatchDog refresh** is not checked or the refresh period is bigger than the watchdog period, as soon as the watchdog timer expires without the proper refreshment, the action selected is executed.

11. Using the Wireless LAN adapter

If your module does not have a WLAN adapter, disregard this topic.

The ConnectCore Wi-9C embedded module includes a Wireless LAN adapter integrated in the module. This adapter complies with IEEE 802.11b/g Wireless LAN standard. This topic explains the configuration and security details of the WLAN adapter.

For your network to work and communicate with the ConnectCore Wi-9C using the wireless interface, a wireless Access Point (AP) must be installed and configured.

11.1. Concepts

One of the most important concerns in wireless communications is the security and integrity of the data. For this reason, it is necessary to introduce two concepts: encryption and authentication.

Encryption makes data unreadable without a certain deciphering key.

Authentication confirms the identity or origin of something or someone.

11.2. Features of the WLAN adapter

- Complies with the IEEE 802.11b and IEEE 802.11g 2.4Ghz (DSSS) standards
- High data transfer rate – up to 54Mbps.
- Supports 64/128-bit WEP, TKIP and AES encryption.
- Supports open, shared, WPA, WPA-PSK, WPA2 and WPA2-PSK authentication.
- Driver complies with the NDIS 5.0 standard

The card supports 64/128-bit WEP data encryption, which protects a wireless network from eavesdropping. It also supports the WPA (Wi-Fi Protected Access) feature, which combines IEEE 802.1x, PSK (Pre-Shared Key), and TKIP (Temporal Key Integrity Protocol) technologies. Client users are required to authorize before accessing to APs or AP Routers, and the data transmitted is encrypted/decrypted by a dynamically changed secret key. Furthermore, this adaptor supports WPA2 function, which provides a stronger encryption mechanism through AES (Advanced Encryption Standard), which is a requirement for some corporate and government users.

11.3. Include the wireless interface in the Windows CE kernel

If an OS design was created using the ConnectCore Wi-9C template (as seen in topic 3.1), all the necessary components will be included to support the wireless interface in the kernel.

On the other hand, if the OS design was not based in ConnectCore Wi-9C template and the components were selected by hand, the required components shown in the next topic must be included from the catalog:

11.3.1. Required components

Go to the Catalog and expand **Core OS > CEBASE**. Then include these elements:

- Communication Services and Networking
  - Networking - General
    - Extensible Authentication Protocol
  - Networking – Local Area Network (LAN)
    - Wireless LAN (802.11) STA - Automatic Configuration 802.1x
- Security
  - Authentication Services
    - Schannel (SSL/TLS)

Now expand **Third Party > BSP > ConnectCore 9C/Wi-9C: ARMV4I** and select this element:

- Device Drivers
  - Networking
    - Local Area Networking (LAN) devices
      - ConnectCore Wi-9C Wireless

11.3.2. Recommended catalog components

Other recommended networking utilities and services are:

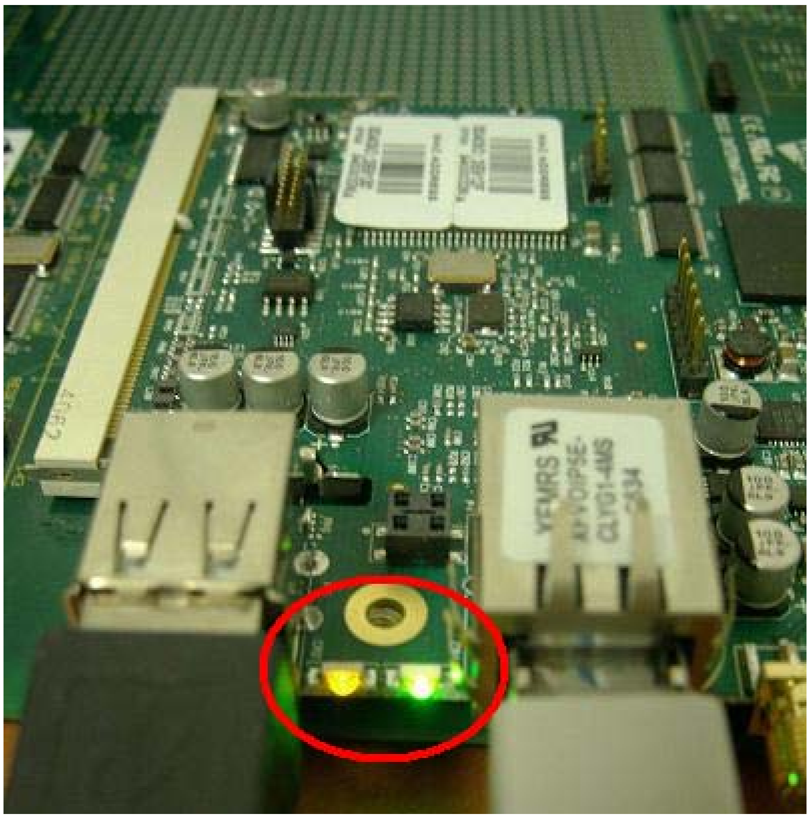
- Communication Services and Networking
  - Networking - General
    - Network utilities (ipconfig, ping, route)
  - Servers
    - FTP Server
    - Telnet Server
- Shell and User Interface
  - User Interface
    - Network User Interface

If using a WPA Enterprise configuration, the following component may also be needed for certificate management:

- Security
  - Microsoft Certificate Enrollment Tool Sample

11.4. Wireless interface LEDs

The ConnectCore Wi-9C module has two LEDs (one green, one yellow) between the Ethernet and USB host connectors, related with the wireless interface.



The green LED, which implements the Wireless Status, can represent three states:

- Solid ON: Connected to an access point
- Slow Blinking: Connected to an ad-hoc computer
- Fast Blinking: Scanning
- Solid OFF: Not connected

The yellow LED implements the network activity and blinks when packets are being received or transmitted.

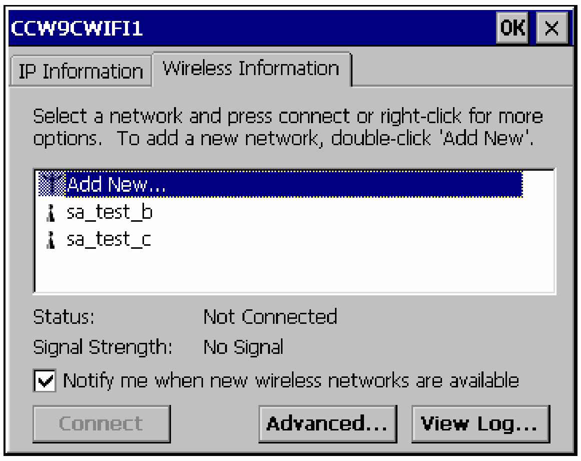
11.5. Driver start

If the wireless interface is included in the kernel, the driver starts automatically when the Windows Embedded CE system starts.

If running a debug version, the **Output** view displays this output:

```
[ccw9cWifi]: Loading Wireless Driver Version 1.1 ... OK
```

If a display is available, the first time, a window opens showing the Access Points in range, from which an access point to connect to can be selected. Also, a small connection icon with a red cross appears in the taskbar. The red cross means that the ConnectCore Wi-9C is not yet connected to any access point.



Select a network and press connect or right-click for more options. To add a new network, double-click 'Add New'.

Add New...
sa_test_b
sa_test_c

Status: Not Connected
Signal Strength: No Signal
☑ Notify me when new wireless networks are available

If a preferred network is saved in the Registry, the target automatically connects to it. The preferred network is managed with the **WifiConf** tool, covered in topic 11.11.

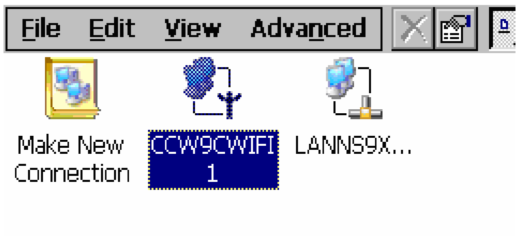
*To prevent this dialog from showing, uncheck **Notify me when new wireless networks are available**. This setting is stored in the Registry in RAM. To save the Registry permanently into NVRAM, you need to execute the **regtool** utility, covered in topic 12.*

11.6. WLAN network settings

The network settings for the WLAN adapter are taken from these U-Boot variables:

- **ipaddr_wlan**: IP address of the WLAN interface
- **netmask_wlan**: Network mask for WLAN interface

To modify the WLAN network settings in Windows Embedded CE, go to the Control Panel in the target device: **Start > Settings > Control Panel > Network > Dial-up Connections**.



Network settings modified in Windows Embedded CE are stored in RAM memory and remain valid until the target is reset. To save these settings permanently in NVRAM, the **regtool** utility must be used. See topic 12 for more information.

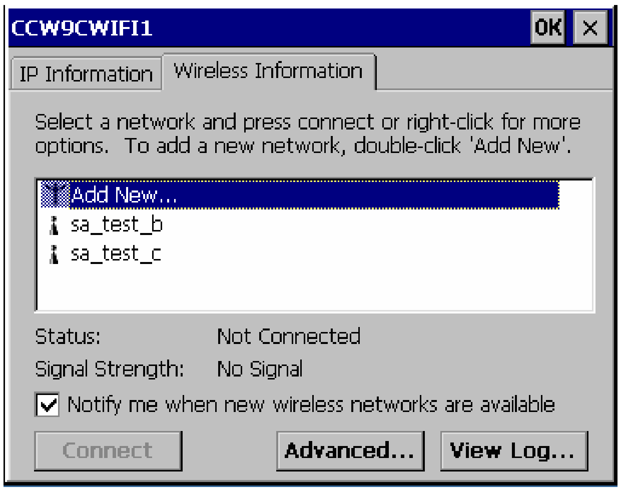
11.7. Connect to an access point (infrastructure mode)

The ConnectCore Wi-9C wireless interface can be connected to an Access Point (AP) in several ways. All of them go through the WZCSAPI (see official Microsoft Windows CE online help documentation for more information about the API).

11.7.1. Graphic mode

Double-clicking the wireless icon on the taskbar shows the window with a list of the wireless APs in range.

If the wireless icon is selected shortly after the driver was loaded, the AP list might be empty because the ConnectCore Wi-9C wireless interface is still scanning for APs.


CCW9CWIFI1

IP Information | Wireless Information

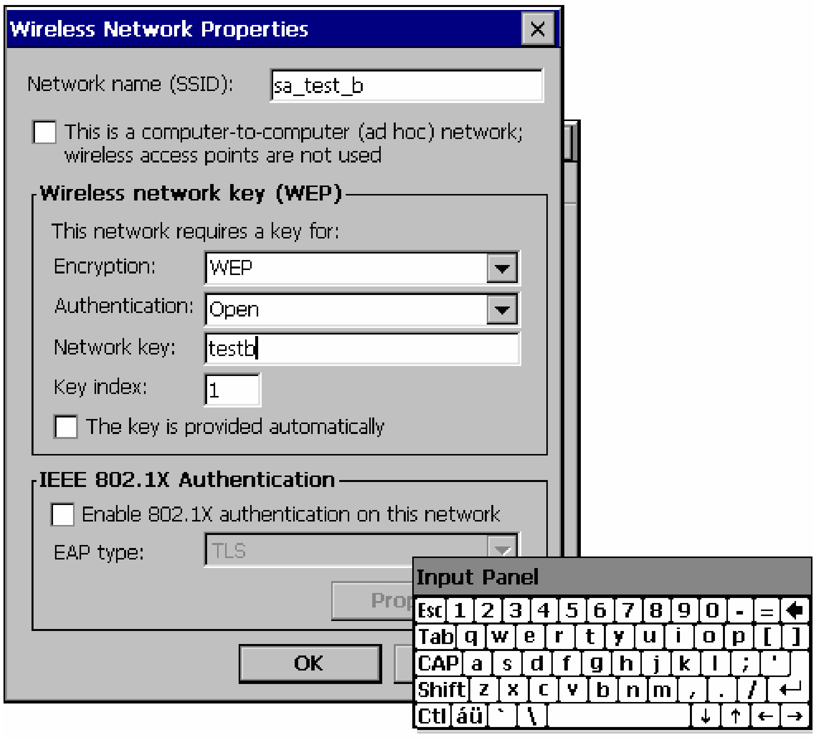
Select a network and press connect or right-click for more options. To add a new network, double-click 'Add New'.

Add New...
sa_test_b
sa_test_c

Status: Not Connected
Signal Strength: No Signal

☑ Notify me when new wireless networks are available

Click the desired AP, and click **Connect**. The **Wireless Network Properties** window opens. Depending on the AP authentication and encryption configuration, different information may need to be entered.


Wireless Network Properties

Network name (SSID): sa_test_b

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: WEP

Authentication: Open

Network key: testb

Key index: 1

☐ The key is provided automatically

IEEE 802.1X Authentication

☐ Enable 802.1X authentication on this network

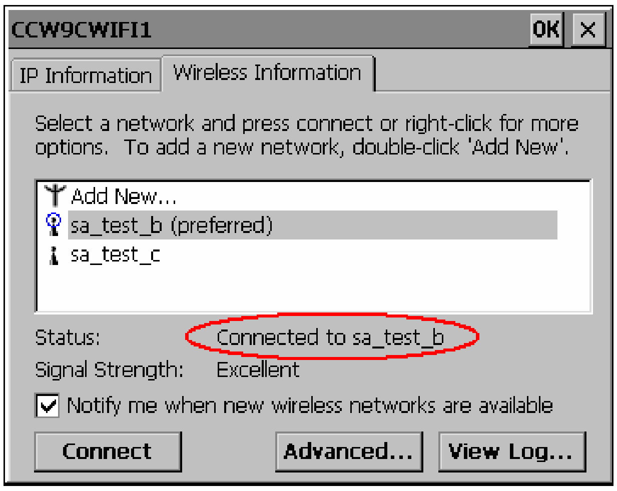
EAP type: TLS

Prop
OK
Input Panel
Esc 1 2 3 4 5 6 7 8 9 0 - =
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . /
Ctl áü ` \

The virtual keyboard can be used for introducing characters.

After clicking **OK**, the Properties window closes and previous window is displayed again.

The status text beside the selected AP passes through several states (depending on the authentication): **Scanning**, **Associating**, **Associated**, **Authenticating**, **Authenticated…**, and finishes with **Connected**.


CCW9CWIFI1

Select a network and press connect or right-click for more options. To add a new network, double-click 'Add New'.

Add New...
sa_test_b (preferred)
sa_test_c

Status: Connected to sa_test_b

Signal Strength: Excellent

☑ Notify me when new wireless networks are available

Then, the WiFi Taskbar icon changes to blue and the red cross disappears.

Now the target can be accessed from any device on the network segment. For example, attempt a ping from a wireless PC to the ConnectCore Wi-9C.

If a simple ping does not work, it is probable that the AP and the IP of the wireless interface are not within the same network segment. Check the network settings of WLAN, as seen in topic 11.6.

11.7.2. Command line mode

If a graphic display is not available, there is a command line application named **wzctool** for connecting to an AP. Execute the **wzctool** application with **/help** option to learn its syntax:

```
\> wzctool /help
wzctool usage:
options:
 -e               Enumerate wireless cards.
 -q <Card Name>  Query wireless card.
 -c <Card Name> -ssid AP-SSID -auth open -encr wep -key 1/0x1234567890
     connect to AP-SSID with given parameters.     Use -c -? for detail.
 -reset           Reset WZC configuration data. Wireless card will disconnect
    if it was connected.
 -set <Card Name> <parameter> Set WZC variables.
     Use -set -? for detail.
 -refresh         Refresh entries.
 -registry        configure as registry.
     Use -registry -? for detail.
 -enablewzcsvc    enable WZC service.
 -disablewzcsvc   disable WZC service.
 -?  shows help message
if no arg is given, wzctool will reads and set as settings in the registry.
Use '-registry -?' for detail
if no <Card Name> is given, wzctool will find the first WiFi card and use
this card.
\>
```

11.7.2.1. wzctool syntax

To get information about the available wireless interfaces, execute **wzctool –q**. This example shows that two networks are available and one of them is a preferred network.

```
\> wzctool -q
wireless card found: CCW9CWIFI1
WZCQueryInterfaceEx() for CCW9CWIFI1
In flags used       = [0x7FFFFFFF]
Returned out flags  = [0x07EFFFFF]
wzcGuid             = [CCW9CWIFI1]
wzcDescr            = [ccw9cWifi1]
BSSID = 00:17:94:FD:99:C0 (this wifi card is associated state)
Media Type          = [0]
Configuration Mode  = [0000A002]
   zero conf enabled for this interface
   802.11 OIDs are supported by the driver/firmware
Infrastructure Mode = [1]  Infrastructure net (connected to an Access Point)
Authentication Mode = [4]  Ndis802_11AuthModeWPAPSK
rdNicCapabilities   = 96 bytes
   dwNumOfPMKIDs           : [3]
   dwNumOfAuthEncryptPairs  : [11]
   Pair[1]
      AuthmodeSupported          [Ndis802_11AuthModeOpen]
      EncryptStatusSupported     [Ndis802_11WEPDisabled]
   Pair[2]
      AuthmodeSupported          [Ndis802_11AuthModeOpen]
      EncryptStatusSupported     [Ndis802_11WEPEnabled]
   Pair[3]
      AuthmodeSupported          [Ndis802_11AuthModeShared]
      EncryptStatusSupported     [Ndis802_11WEPEnabled]
   Pair[4]
      AuthmodeSupported          [Ndis802_11AuthModeWPA]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[5]
      AuthmodeSupported          [Ndis802_11AuthModeWPA]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
   Pair[6]
      AuthmodeSupported          [Ndis802_11AuthModeWPAPSK]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[7]
      AuthmodeSupported          [Ndis802_11AuthModeWPAPSK]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
   Pair[8]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[9]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
   Pair[10]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2PSK]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[11]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2PSK]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
rdPMKCache    = 0 bytes
WEP Status          = [4]  <unknown value>
SSID = sa_test_c
Capabilities =
     WPA/TKIP capable
     WPA2/AES capable

[Available Networks] SSID List [2] entries.

******** List Entry Number [0] ********
   Length                    = 196 bytes.
   dwCtlFlags                = 0x00000010
   MacAddress                = 00:17:94:FD:99:C0
   SSID                      = sa_test_c
   Privacy                   = 4  Privacy enabled (encrypted with
                               [Ndis802_11Encyption2Enabled])
   RSSI                      = -37 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse          = NDIS802_11FH
   Configuration:
      Struct Length    = 32
      BeaconPeriod     = 90 kusec
      ATIMWindow       = 0 kusec
      DSConfig         = 2437000 kHz (ch-6)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure            = Ndis802_11Infrastructure
   SupportedRates            = 1.0,2.0,5.5,11.0,6.0,12.0,18.0,24.0, (Mbit/s)
   KeyIndex                  = <not available> (beaconing packets don't have
                               this info)
   KeyLength                 = <not available> (beaconing packets don't have
                               this info)
   KeyMaterial               = <not available> (beaconing packets don't have
                               this info)
   Authentication            = 4  Ndis802_11AuthModeWPAPSK
   rdUserData length         = 0 bytes.
******** List Entry Number [1] ********
   Length                    = 196 bytes.
   dwCtlFlags                = 0x00000010
   MacAddress                = 00:13:46:9B:A8:55
   SSID                      = sa_test_d
   Privacy                   = 6  Privacy enabled (encrypted with
                               [Ndis802_11Encyption3Enabled])
   RSSI                      = -46 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse          = NDIS802_11FH
   Configuration:
      Struct Length    = 32
      BeaconPeriod     = 100 kusec
      ATIMWindow       = 0 kusec
      DSConfig         = 2472000 kHz (ch-13)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure          = Ndis802_11Infrastructure
   SupportedRates          = 1.0,2.0,5.5,11.0,6.0,9.0,12.0,18.0, (Mbit/s)
   KeyIndex                = <not available> (beaconing packets don't have
                              this info)
   KeyLength               = <not available> (beaconing packets don't have
                              this info)
   KeyMaterial             = <not available> (beaconing packets don't have
                              this info)
   Authentication          = 7  Ndis802_11AuthModeWPA2PSK
   rdUserData length       = 0 bytes.

[Preferred Networks] SSID List [1] entries.

******** List Entry Number [0] ********
   Length                  = 196 bytes.
   dwCtlFlags              = 0x00000013
   MacAddress              = 00:17:94:FD:99:C0
   SSID                    = sa_test_c
   Privacy                 = 4  Privacy enabled (encrypted with
                              [Ndis802_11Encyption2Enabled])
   RSSI                    = -37 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse        = NDIS802_11FH
   Configuration:
      Struct Length    = 32
      BeaconPeriod     = 90 kusec
      ATIMWindow       = 0 kusec
      DSConfig         = 2437000 kHz (ch-6)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure          = Ndis802_11Infrastructure
   SupportedRates          = 1.0,2.0,5.5,11.0,6.0,12.0,18.0,24.0, (Mbit/s)
   KeyIndex                = <not available> (beaconing packets don't have
                              this info)
   KeyLength               = <not available> (beaconing packets don't have
                              this info)
   KeyMaterial             = <not available> (beaconing packets don't have
                              this info)
   Authentication          = 4  Ndis802_11AuthModeWPAPSK
   rdUserData length       = 0 bytes.

rdCtrlData length   = 0 bytes

parameter setting in Zero Config
tmTr = 3000 mili-seconds (Scan time out)
tmTp = 2000 mili-seconds (Association time out)
tmTc = 60000 mili-seconds (Periodic scan when connected)
tmTf = 60000 mili-seconds (Periodic scan when disconnected)
\>
```

11.7.2.2. Connect to an AP

To connect to a specific AP, use the **–c** option. For example, to connect to an AP with SSID **myAPname** with WPA-PSK authentication with TKIP encryption and password **fY5jHot6**, execute:

```
\> wzctool -c ccw9cwifi1 -ssid myAPname -auth wpa-psk -encr tkip -key
fY5jHot6
```

If connecting to an AP with SSID **myAPname** open authentication and no encryption, the command is:

```
\> wzctool -c ccw9cwifi1 -ssid myAPname -auth open -encr disabled
```

11.7.2.3. Registry information

Executing **wzctool** with the **–registry** option, or with no option, causes the tool to connect with the default Registry values established in this Registry key:

```
[HKEY_CURRENT_USER\Comm\WZCTOOL]
    "SSID"            = "myAPname"
    "authentication" = dword:4  ;WPA-PSK (Ndis802_11AuthModeWPAPSK)
    "encryption"      = dword:4  ;TKIP (Ndis802_11Encryption2Enabled)
    "key"             = "fY5jHot6"
    "adhoc"           = dword:0  ;CE8021X is an infrastructure network
```

This Registry key can be configured to quickly create a connection to a default AP .The **encryption** and **authentication** fields are written with numbers, according to these tables:

Authentication	Value
Ndis802_11AuthModeOpen	0
Ndis802_11AuthModeShared	1
Ndis802_11AuthModeAutoSwitch	2
Ndis802_11AuthModeWPA	3
Ndis802_11AuthModeWPAPSK	4
Ndis802_11AuthModeWPANone	5
Ndis802_11AuthModeWPA2	6
Ndis802_11AuthModeWPA2PSK	7

Encryption	Value
Ndis802_11WEPEnabled	0
Ndis802_11EncryptionDisabled	1
Ndis802_11Encryption2Enabled (TKIP)	4
Ndis802_11Encryption3Enabled (AES-CCMP)	6

Also, the **adhoc** field is a numeric entry that accepts two values:

adhoc	Value
Infrastructure (AP-connection)	0
Ad hoc (computer-to-computer)	1

The WZCTOOL Registry entry exposes the encryption key. If access to the target's Registry is not secure, this would be a security hole.

11.7.2.4. Source code

The source code of the **wzctool** utility is available in the directory **%_WINCEROOT%\PUBLIC\COMMON\OAK\DRIVERS\NETSAMP\WZCTOOL\**. It can be used as an example for controlling and configuring the ConnectCore Wi-9C wireless interface.

11.8. Connect to a computer (ad hoc mode)

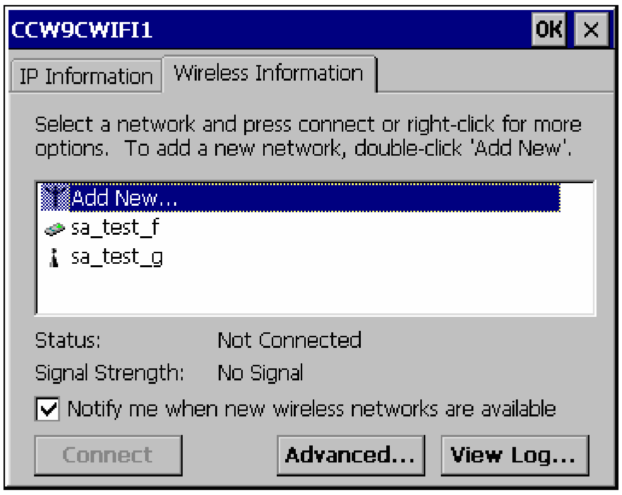
Connecting to a peer computer in ad hoc mode requires the same steps shown in previous topic, connecting to an AP in infrastructure mode.

When queried with **wzctool –q**, a device configured as ad hoc reports something like this:

```
Length                   = 196 bytes.
   dwCtlFlags               = 0x00000000
   MacAddress               = 66:3E:C2:49:AF:60
   SSID                     = sa_test_f
   Privacy                  = 0  Privacy enabled (encrypted with
                              [Ndis802_11WEPEnabled])
   RSSI                     = -55 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse         = NDIS802_11FH
   Configuration:
      Struct Length    = 32
      BeaconPeriod     = 90 kusec
      ATIMWindow       = 0 kusec
      DSConfig         = 2442000 kHz (ch-7)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure           = NDIS802_11IBSS
   SupportedRates           = 1.0,2.0,5.5,11.0,6.0,9.0,12.0,18.0, (Mbit/s)
   KeyIndex                 = <not available> (beaconing packets don't have
                              this info)
   KeyLength                = <not available> (beaconing packets don't have
                              this info)
   KeyMaterial              = <not available> (beaconing packets don't have
                              this info)
   Authentication           = 0  Ndis802_11AuthModeOpen
   rdUserData length        = 0 bytes.
```

11.8.1. Graphic mode

The only difference from connecting in infrastructure mode is that ad hoc devices have a different icon than APs. In the **Properties** window, the check box **This is a computer-to-computer (ad hoc) network** is automatically selected, and the **Encryption** and **Authentication** combo boxes are preconfigured, depending on the ad hoc device configuration.



CCW9CWIFI1

IP Information | Wireless Information

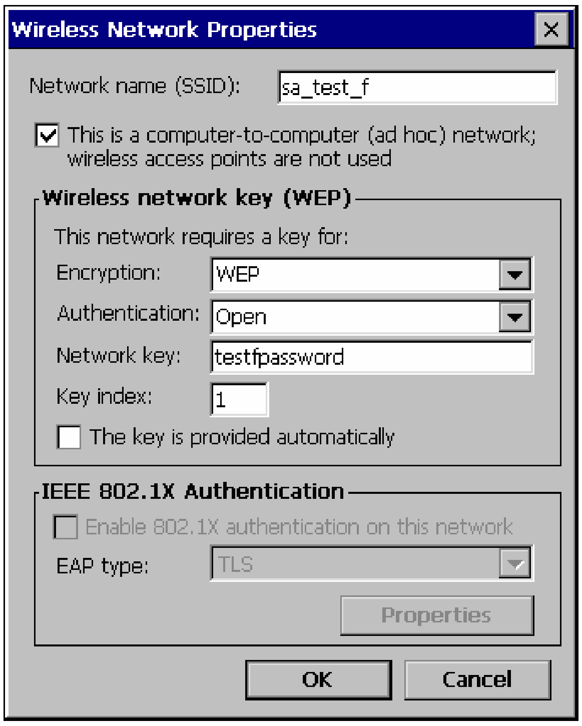
Select a network and press connect or right-click for more options. To add a new network, double-click 'Add New'.

- Add New...
- sa_test_f
- sa_test_g

Status: Not Connected

Signal Strength: No Signal

☑ Notify me when new wireless networks are available



Wireless Network Properties

Network name (SSID): sa_test_f

☑ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: WEP

Authentication: Open

Network key: testfpassword

Key index: 1

☐ The key is provided automatically

IEEE 802.1X Authentication

☐ Enable 802.1X authentication on this network

EAP type: TLS

11.8.2. Command line mode

If a graphic display is not available, use the **wzctool** application to connect to an ad hoc device For example, to connect to an ad hoc device configured with **SSID=sa_test_f** with open authentication and wep128 encryption with index 1 and password **pass7ujH,** the command is:

```
\> wzctool -c ccw9cwifi1 -adhoc -ssid sa_test_f -auth open -encr wep -key
1/pass7ujH
```

11.9. Authentication and encryption

As mentioned previously, authentication is the process of confirming the identity, and encryption is the process that makes information unreadable for unauthorized users. Here are the methods supported by the ConnectCore Wi-9C WLAN interface Windows CE 6.0 driver:

11.9.1. Supported methods

Authentication	Infrastructure mode	Ad hoc mode
open	X	X
shared	X	X
WPA-PSK (WPA Personal)	X	
WPA2-PSK (WPA2 Personal)	X	
WPA (WPA Enterprise)	X	
WPA2 (WPA2 Enterprise)	X	

Encryption	Infrastructure mode	Ad hoc mode
no encryption	X	X
WEP 64/128 bits	X	X
TKIP	X	
AES-CCMP	X	

11.9.2. Authentication and encryption combinations

There are several combinations of authentication and encryption methods. When queried with the **wzctool** the driver reports them as follows:

```
\> wzctool -q
wireless card found: CCW9CWIFI1
   dwNumOfAuthEncryptPairs  : [11]
   Pair[1]
      AuthmodeSupported          [Ndis802_11AuthModeOpen]
      EncryptStatusSupported     [Ndis802_11WEPDisabled]
   Pair[2]
      AuthmodeSupported          [Ndis802_11AuthModeOpen]
      EncryptStatusSupported     [Ndis802_11WEPEnabled]
   Pair[3]
      AuthmodeSupported          [Ndis802_11AuthModeShared]
      EncryptStatusSupported     [Ndis802_11WEPEnabled]
   Pair[4]
      AuthmodeSupported          [Ndis802_11AuthModeWPA]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[5]
      AuthmodeSupported          [Ndis802_11AuthModeWPA]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
   Pair[6]
      AuthmodeSupported          [Ndis802_11AuthModeWPAPSK]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[7]
      AuthmodeSupported          [Ndis802_11AuthModeWPAPSK]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
   Pair[8]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[9]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
   Pair[10]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2PSK]
      EncryptStatusSupported     [Ndis802_11Encryption2Enabled]
   Pair[11]
      AuthmodeSupported          [Ndis802_11AuthModeWPA2PSK]
      EncryptStatusSupported     [Ndis802_11Encryption3Enabled]
```

Encryption2 is TKIP and Encryption3 is AES-CCMP.

For explaining the connection process, these combinations can be grouped as follows:

- Open authentication and encryption.
- Open authentication with WEP encryption.
- WPA-PSK or WPA2-PSK authentication with TKIP or AES-CCMP encryption.
- WPA and WPA2 Enterprise authentication.

11.9.3. Open authentication without encryption

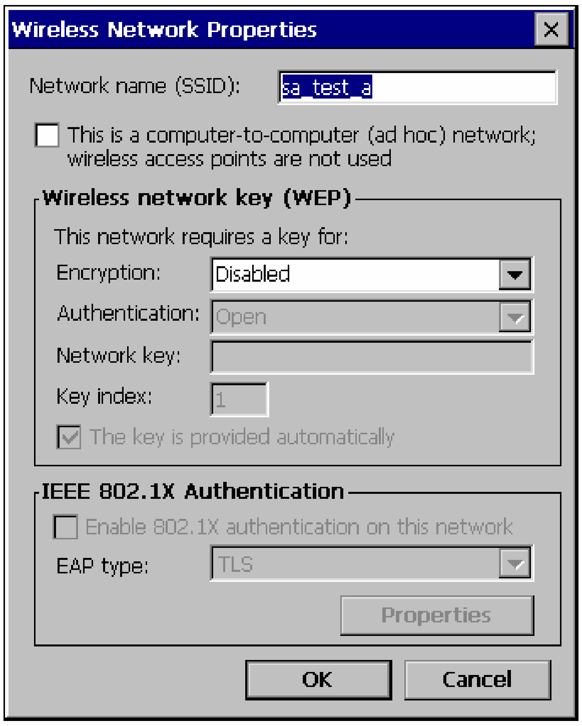
When queried with **wzctool –q**, an AP configured as previously described displays this information:

```
Length                = 196 bytes.
   dwCtlFlags         = 0x00000000
   MacAddress         = 00:13:46:9B:A8:53
   SSID               = sa_test_a
   Privacy            = 1  Privacy disabled (wireless data is not encrypted)
   RSSI               = -54 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse   = NDIS802_11FH
   Configuration:
      Struct Length    = 32
      BeaconPeriod     = 100 kusec
      ATIMWindow       = 0 kusec
      DSConfig         = 2472000 kHz (ch-13)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure      = Ndis802_11Infrastructure
   SupportedRates      = 1.0,2.0,5.5,11.0,6.0,9.0,12.0,18.0, (Mbit/s)
   KeyIndex            = <not available> (beaconing packets don't have this
                          info)
   KeyLength           = <not available> (beaconing packets don't have this
                          info)
   KeyMaterial         = <not available> (beaconing packets don't have this
                          info)
   Authentication      = 0  Ndis802_11AuthModeOpen
   rdUserData length   = 0 bytes.
```

11.9.3.1. Connect in graphic mode

To connect to an AP in graphic mode, select the AP and click **Connect**. Because the AP configuration is automatically recognized, these fields are already filled in:

- **Encryption: disabled**
- **Authentication: Open**



Wireless Network Properties

Network name (SSID): sa_test_a

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: Disabled

Authentication: Open

Network key:

Key index: 1

☑ The key is provided automatically

IEEE 802.1X Authentication

☐ Enable 802.1X authentication on this network

EAP type: TLS

11.9.3.2. Connect in command line mode

To connect in command line mode, use the wzctool command, specifying only the SSID of the AP, which in the example, is **sa_test_a**:

```
\> wzctool –c CCW9CWIFI1 -ssid sa_test_a -auth open -encr disabled
```

11.9.4. Open authentication with WEP encryption

When queried with **wzctool –q**, an AP configured with open authentication with WEP encryption with this configuration displays this information:

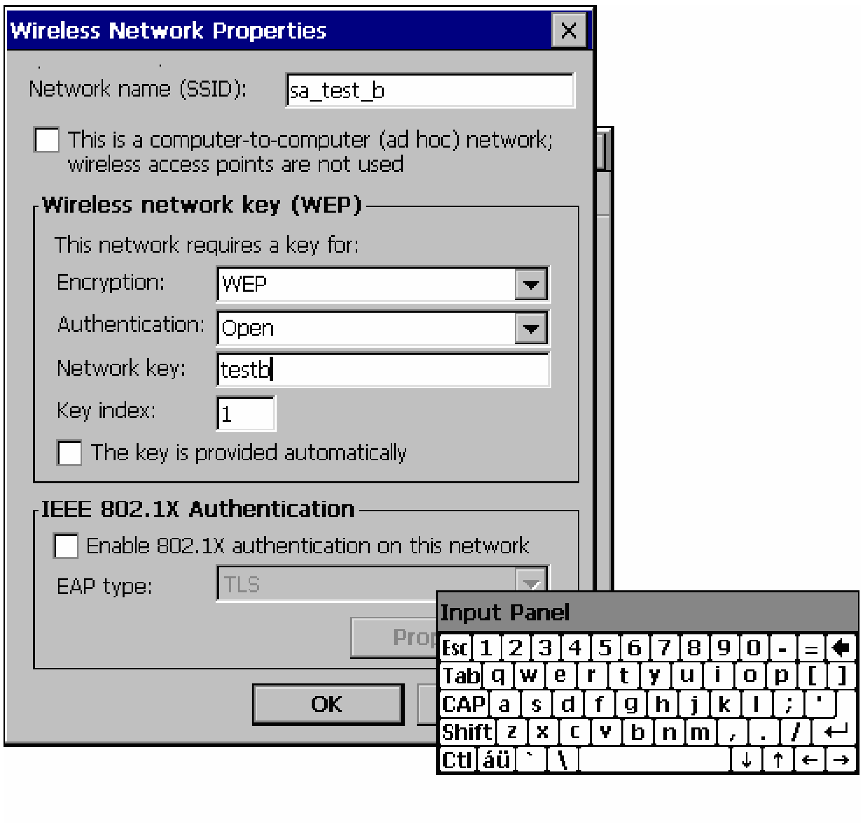
```
Length                  = 196 bytes.
   dwCtlFlags           = 0x00000000
   MacAddress           = 00:13:46:9B:A8:54
   SSID                 = sa_test_b
   Privacy                          =  0    Privacy  enabled  (encrypted  with
[Ndis802_11WEPEnabled])
   RSSI                 = -59 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse     = NDIS802_11FH
   Configuration:
      Struct Length     = 32
      BeaconPeriod      = 100 kusec
      ATIMWindow        = 0 kusec
      DSConfig          = 2472000 kHz (ch-13)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure       = Ndis802_11Infrastructure
   SupportedRates       = 1.0,2.0,5.5,11.0,6.0,9.0,12.0,18.0, (Mbit/s)
   KeyIndex             = <not available> (beaconing packets don't have this
                           info)
   KeyLength            = <not available> (beaconing packets don't have this
                           info)
   KeyMaterial          = <not available> (beaconing packets don't have this
                           info)
   Authentication       = 0  Ndis802_11AuthModeOpen
   rdUserData length    = 0 bytes.
```

11.9.4.1. Connect in graphic mode

To connect in graphic mode, select the AP and click **Connect**. Because the AP configuration is automatically recognized, these fields are already filled:

- **Encryption: WEP**
- **Authentication: Open**

The check box **The key is provided automatically** is selected by default. Deselect it and enter **Network Key** and the **Key index** values as needed for the AP. Use the virtual keyboard to introduce the password.


Wireless Network Properties

Network name (SSID): sa_test_b

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: WEP

Authentication: Open

Network key: testb

Key index: 1

☐ The key is provided automatically

IEEE 802.1X Authentication

☐ Enable 802.1X authentication on this network

EAP type: TLS

11.9.4.2. Connect in command line mode

To connect in command line mode, provide the SSID of the AP (in the example, **sa_test_a**) and the WEP password and password index (in the example, **index=1** and **password=testb**:

```
\> wzctool -c CCW9CWIFI1 -ssid sa_test_b -auth open -encr wep -key 1/testb
```

The password can also be specified in hexadecimal format (**testb = 0x7465737462**):

```
\> wzctool -c ccw9cwifi1 -ssid sa_test_b -auth open -encr wep -key
1/0x7465737462
```

11.9.5. WPA-PSK authentication with TKIP encryption

This topic shows connecting to an AP configured with WPA-PSK authentication with TKIP encryption. The connection method for other combinations of authentication (WPA-PSK or WPA2-PSK)and encryption (TKIP or AES-CCMP) vary only in the parameters specified.

When queried with **wzctool –q,** an AP configured with WPA-PSK + TKIP displays this information:

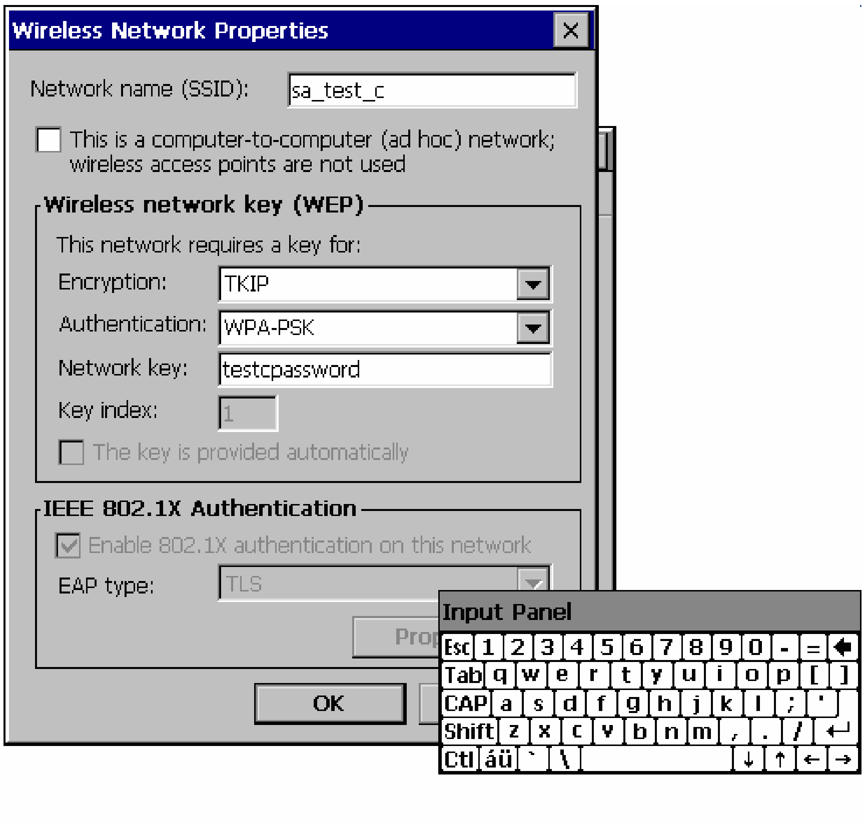
```
Length                 = 196 bytes.
   dwCtlFlags          = 0x00000010
   MacAddress          = 00:17:94:FD:99:C0
   SSID                = sa_test_c
   Privacy                         =  4    Privacy  enabled  (encrypted  with
[Ndis802_11Encryption2Enabled])
   RSSI                = -44 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse    = NDIS802_11FH
   Configuration:
      Struct Length    = 32
      BeaconPeriod     = 90 kusec
      ATIMWindow       = 0 kusec
      DSConfig         = 2437000 kHz (ch-6)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure      = Ndis802_11Infrastructure
   SupportedRates      = 1.0,2.0,5.5,11.0,6.0,12.0,18.0,24.0, (Mbit/s)
   KeyIndex            = <not available> (beaconing packets don't have this
                          info)
   KeyLength           = <not available> (beaconing packets don't have this
                          info)
   KeyMaterial         = <not available> (beaconing packets don't have this
                          info)
   Authentication      = 4  Ndis802_11AuthModeWPAPSK
   rdUserData length   = 0 bytes.
```

11.9.5.1. Connect in graphic mode

To connect to the AP in graphic mode, select the AP and click **Connect**. Because the AP configuration is automatically recognized, these fields are already filled:

- **Encryption: TKIP**
- **Authentication: WPA-PSK**

Only the **Network key** field must be entered. Enter the pre-shared key that is configured in the AP. Use the virtual keyboard to introduce the password.


Wireless Network Properties

Network name (SSID): sa_test_c

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: TKIP

Authentication: WPA-PSK

Network key: testcpassword

Key index: 1

☐ The key is provided automatically

IEEE 802.1X Authentication

☑ Enable 802.1X authentication on this network

EAP type: TLS

Pro
OK
Input Panel
Esc 1 2 3 4 5 6 7 8 9 0 - =
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . /
Ctl áü ` \ ↓ ↑ ← →

11.9.5.2. Connect in command line mode

To connect to the AP in command line mode, enter the **wzctool** command. Specify the SSID of the AP (in the example, **sa_test_c**) and the WPA-PSK password (in the example, **testcpassword**):

```
\> wzctool -c ccw9cwifi1 -ssid sa_test_c -auth wpa-psk -encr tkip -key
testcpassword
```

11.9.6. WPA2-PSK authentication with AES-CCMP encryption

WPA2-PSK authentication with AES-CCMP encryption is a different combination of the same method seen in topic 11.9.5.

When queried with **wzctool –q,** an AP configured with WPA2-PSK + AES-CCMP displays this information:

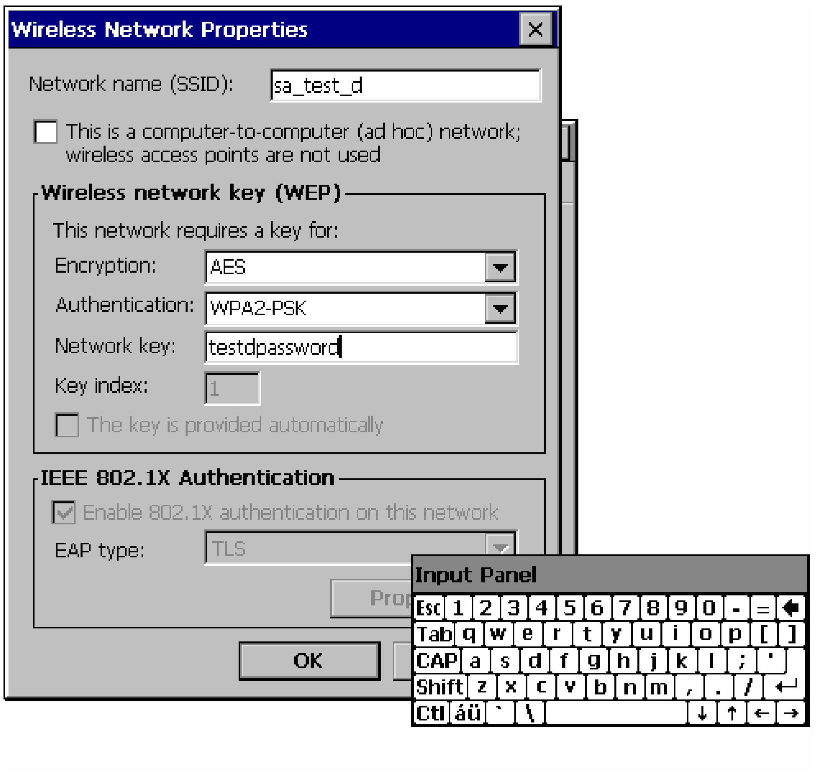
```
Length                  = 196 bytes.
   dwCtlFlags           = 0x00000010
   MacAddress           = 00:13:46:9B:A8:55
   SSID                 = sa_test_d
   Privacy              = 6  Privacy enabled (encrypted with [Ndis802_11Encr
yption3Enabled])
   RSSI                 = -37 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse     = NDIS802_11FH
   Configuration:
      Struct Length     = 32
      BeaconPeriod      = 100 kusec
      ATIMWindow        = 0 kusec
      DSConfig          = 2472000 kHz (ch-13)
      FHConfig:
         Struct Length = 0
         HopPattern     = 0
         HopSet         = 0
         DwellTime      = 0
   Infrastructure       = Ndis802_11Infrastructure
   SupportedRates       = 1.0,2.0,5.5,11.0,6.0,9.0,12.0,18.0, (Mbit/s)
   KeyIndex             = <not available> (beaconing packets don't have this
                           info)
   KeyLength            = <not available> (beaconing packets don't have this
                           info)
   KeyMaterial          = <not available> (beaconing packets don't have this
                           info)
   Authentication       = 7  Ndis802_11AuthModeWPA2PSK
   rdUserData length    = 0 bytes.
```

11.9.6.1. Connect in graphic mode

To connect to an AP in graphic mode, select the AP and click **Connect**. The AP configuration is automatically recognized, so these fields are already filled:

- **Encryption: AES**
- **Authentication: WPA2-PSK**

Only the **Network key** field must entered. Enter the pre-shared key configured in the AP. Use the virtual keyboard to introduce the password.



Wireless Network Properties

Network name (SSID): sa_test_d

This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: AES

Authentication: WPA2-PSK

Network key: testdpassword

Key index: 1

The key is provided automatically

IEEE 802.1X Authentication

Enable 802.1X authentication on this network

EAP type: TLS

11.9.6.2. Connect in command line mode

To connect to the AP in command line mode, enter the **wzctool** command. Specify the SSID of the AP (in the example, **sa_test_d**) and the WPA2-PSK password (in the example, **testdpassword**):

```
\> wzctool -c ccw9cwifi1 -ssid sa_test_d -auth wpa2-psk -encr aes -key
testdpassword
```

11.9.7. WPA Enterprise authentication

When queried with **wzctool –q,** an AP configured with WPA or WPA2 Enterprise displays this information:

```
Length                  = 196 bytes.
   dwCtlFlags           = 0x00000010
   MacAddress           = 00:13:46:9B:A8:55
   SSID                 = sa_test_d
   Privacy              = 6  Privacy enabled (encrypted with [Ndis802_11Encr
yption3Enabled])
   RSSI                 = -37 dBm (0=excellent, -100=weak signal)
   NetworkTypeInUse     = NDIS802_11FH
   Configuration:
      Struct Length     = 32
      BeaconPeriod      = 100 kusec
      ATIMWindow        = 0 kusec
      DSConfig          = 2472000 kHz (ch-13)
      FHConfig:
         Struct Length = 0
         HopPattern    = 0
         HopSet        = 0
         DwellTime     = 0
   Infrastructure       = Ndis802_11Infrastructure
   SupportedRates       = 1.0,2.0,5.5,11.0,6.0,9.0,12.0,18.0, (Mbit/s)
   KeyIndex             = <not available> (beaconing packets don't have this
                           info)
   KeyLength            = <not available> (beaconing packets don't have this
                           info)
   KeyMaterial          = <not available> (beaconing packets don't have this
                           info)
   Authentication       = 6  Ndis802_11AuthModeWPA
   rdUserData length    = 0 bytes.
```

11.9.7.1. Connect in graphic mode

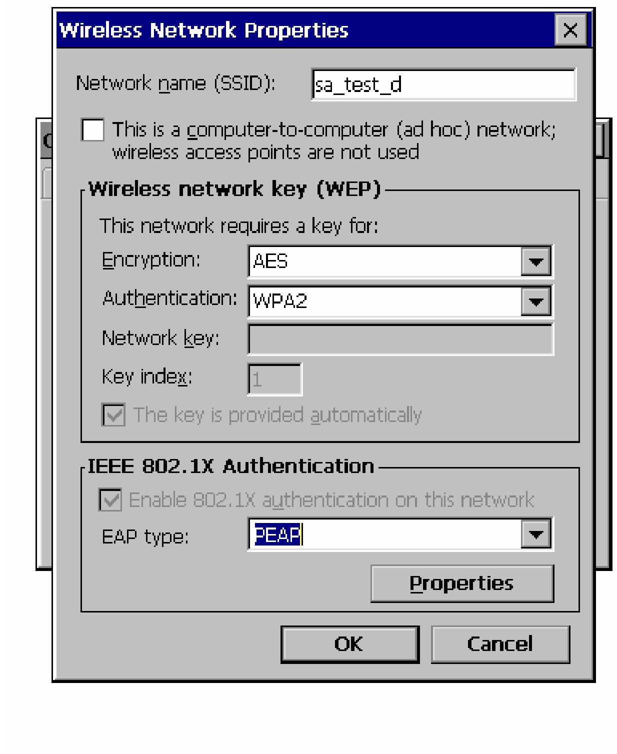
To connect to the AP in graphic mode, select the AP and click **Connect**. The AP configuration is automatically recognized, so these fields are already filled in:

- **Encryption: AES**
- **Authentication: WPA2**

Then select the desired EAP type to use. The settings and requested information displayed depend on the EAP type.

For PEAP:

From the **EAP type** combo box, select **PEAP**.


Wireless Network Properties

Network name (SSID): sa_test_d

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: AES

Authentication: WPA2

Network key:

Key index: 1

☑ The key is provided automatically

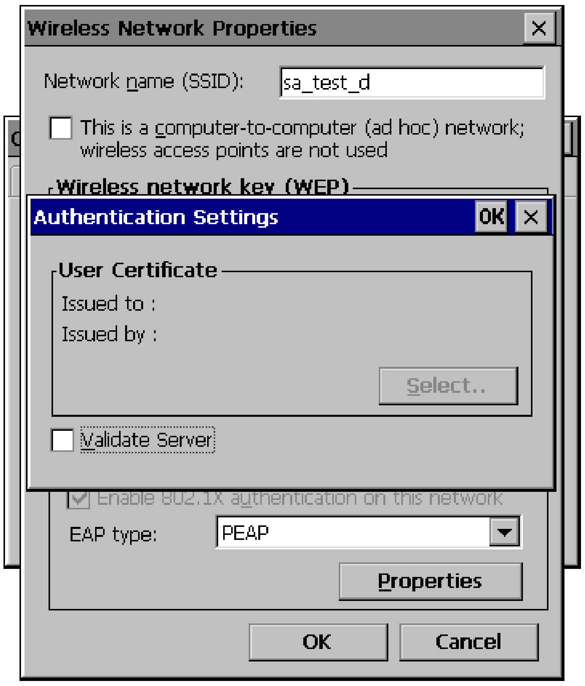
IEEE 802.1X Authentication

☑ Enable 802.1X authentication on this network

EAP type: PEAP

Click the **Properties** button. The **Authentication Settings** dialog is displayed. For the **Validate Server** checkbox, if the correct server certificates are installed, leave it selected. If Server Certificates are not installed or the server does not support this feature, deselect it.

The date of the system must be correctly set and match the server certificate valid period


Wireless Network Properties

Network name (SSID): sa_test_d

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

Authentication Settings

User Certificate

Issued to :

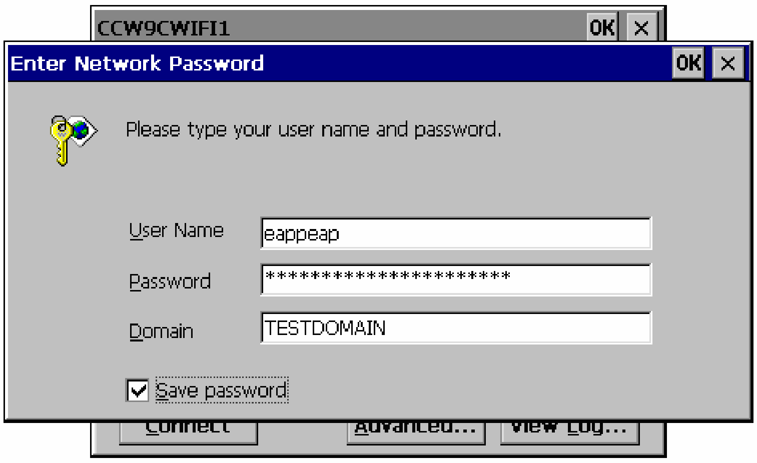
Issued by :

☐ Validate Server

☑ Enable 802.1X authentication on this network

EAP type: PEAP

Click **OK** buttons twice. If the network has never been reached using this method, the system will prompt for **User Name**, **Password**, and **Domain**:


CCW9CWIFI1

Enter Network Password

Please type your user name and password.

User Name: eappeap

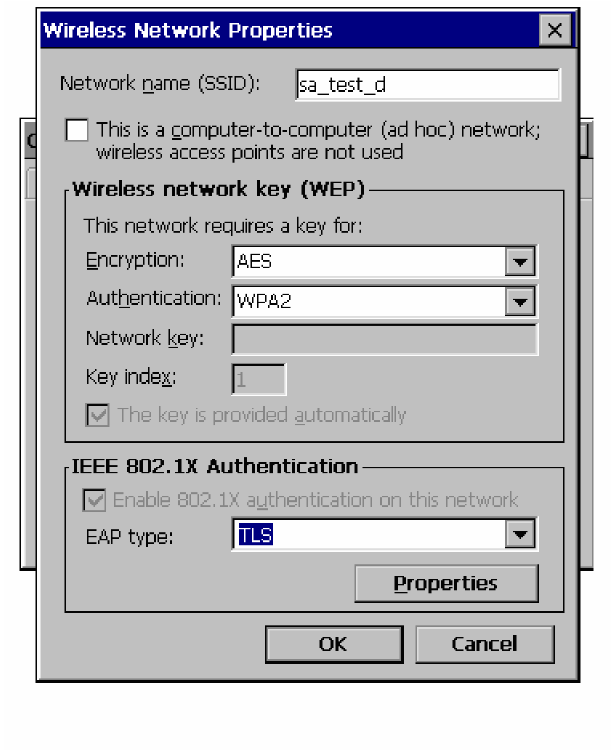
Password: **********************

Domain: TESTDOMAIN

☑ Save password

For TLS:

From the **EAP type** combo box, select **TLS**.



Wireless Network Properties

Network name (SSID): sa_test_d

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

This network requires a key for:

Encryption: AES

Authentication: WPA2

Network key:

Key index: 1

☑ The key is provided automatically

IEEE 802.1X Authentication

☑ Enable 802.1X authentication on this network

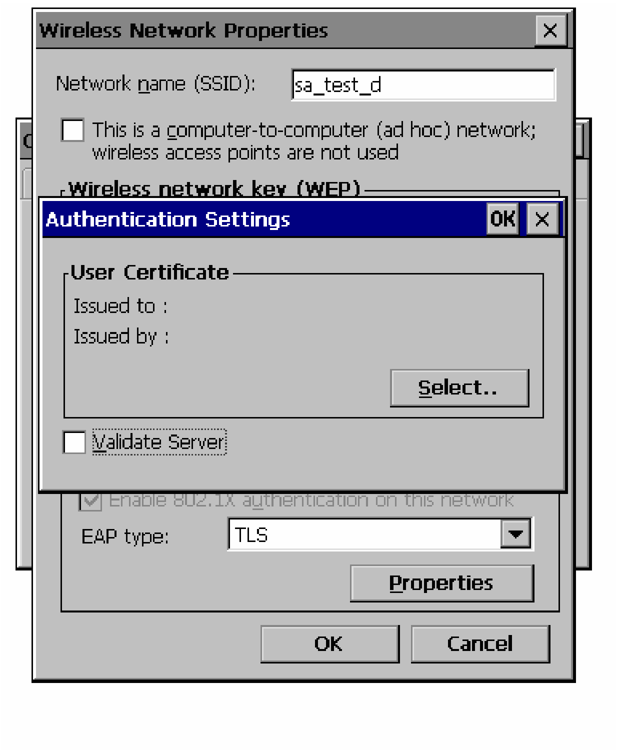
EAP type: TLS

Click the **Properties** button.

The **Authentication Settings** dialog is displayed. For the **Validate Server** checkbox, if the correct server certificates are installed, leave this setting selected. If Server Certificates are not installed or the server doesn't support this feature, deselect it.

The date of the system must be correctly set and match the server certificate valid period.

Click the **Select** button.


Wireless Network Properties

Network name (SSID): sa_test_d

☐ This is a computer-to-computer (ad hoc) network; wireless access points are not used

Wireless network key (WEP)

Authentication Settings

User Certificate

Issued to :

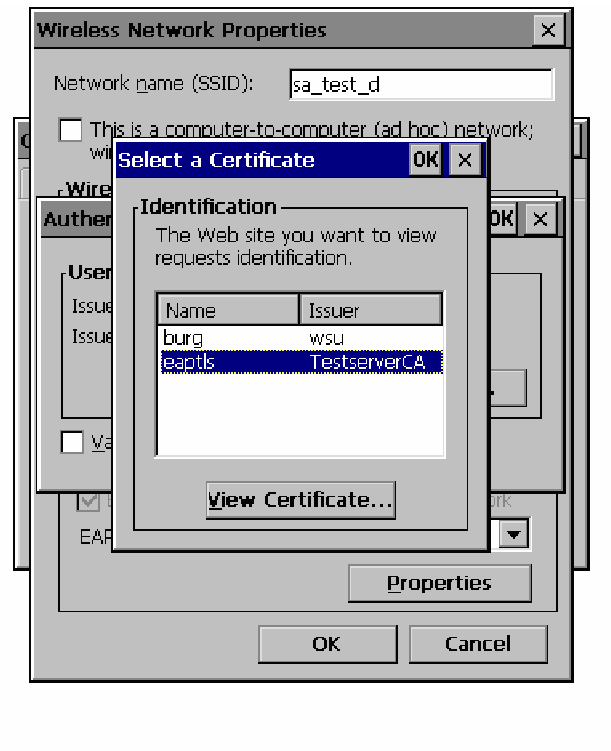
Issued by :

☐ Validate Server

☑ Enable 802.1X authentication on this network

EAP type: TLS

Select the Client Certificate to use.


Wireless Network Properties

Network name (SSID): sa_test_d

☐ This is a computer-to-computer (ad hoc) network;

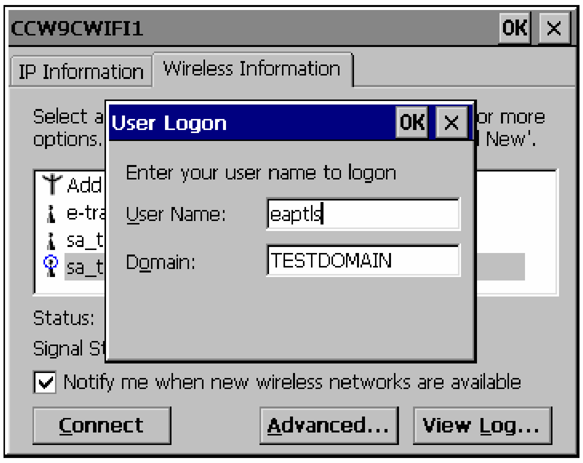
Select a Certificate

Identification

The Web site you want to view requests identification.

Name	Issuer
burg	wsu
eaptls	TestserverCA

Click **OK** buttons three times. If the network has never been reached using this method, the system will prompt for **User Name** and **Domain**:


CCW9CWIFI1

IP Information | Wireless Information

Select a ... or more options. ... New'.

User Logon

Enter your user name to logon

User Name: eaptls

Domain: TESTDOMAIN

Status:

Signal St

☑ Notify me when new wireless networks are available

Generating and installing certificates is done using a standard WinCE process. Description of the process is out of the scope of this document.

11.10. APs supporting several authentication and encryption methods

Modern Access Points can be configured to support several authentication and encryption methods at the same time. For example, an AP can be configured to support, at the same time, any combination of WPA-PSK or WPA2-PSK authentication and TKIP or AES-CCMP encryption.

Such configuration offers the possibility to connect to the AP with any of the four resultant combinations.

11.11. Wireless configuration tool

WifiConf.exe is a custom application provided to:

- Give access to non-standard functionality/features of the driver that cannot be accessed through standard Windows CE tools like graphic **NetUI** or command-line **wzctool**.
- Save the settings of the preferred wireless network in the Registry, allowing the automatic connection to the wireless network after start-up.

This application is in the Windows folder of the target

To execute the application and see its syntax, enter **wificonf** in a console or telnet session:

```
\> wificonf
Application to configure and save the wireless settings. Revision 1.2
Copyright(c) 2007 by Digi International Inc.

Usage: WifiConf <options>

Where options are:
  -status
        Displays driver internal current status.

  -stats clear|read
        Clears or reads statistics.

  -tx_power [HexValue]
        Reads/Sets tx_power.
  -chan_mask [HexValue]
        Reads/Sets chan_mask.
  -tx_rate [DecValue]
        Reads/Sets tx_rate.
  -rts_thresh [DecValue]
        Reads/Sets rts_thresh.
  -frag_thresh [DecValue]
        Reads/Sets frag_thresh.
  -options [HexValue]
        Reads/Sets options.

  -save_params
        Save wireless settings in system registry.

  -wzctool <wzctool-like command_line>
        Reads/Sets/Clears the Preferred Network in Registry.
        type 'wzctool -help' to learn about the <wzctool-like command_line>.
        See User's Guide documentation for more info.

Examples:
WifiConf -status
        Reads and display driver status
WifiConf -stats read
        Reads and display driver statistics
WifiConf -chan_mask
        Reads current value of chan_mask. Ex: 0x1002=Only ch13 & ch2 enabled
WifiConf -chan_mask 0x1003
        Sets value of chan_mask to 0x1003: ch13, ch2 and ch1 enabled
wificonf -wzctool –ssid MyAP_-auth wpa-psk -encr tkip -key MyPassword
        Stores MyAP as the Preferred Network in the registry
wificonf -wzctool -reset
        Delete the Preferred Network in the registry
```

11.11.1. Display wireless status information

To display status information, enter this command:

```
\> wificonf -status
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
OID_WIFIMAC_STATION_STATE=2 (Associated with ESS)
OID_WIFIMAC_GET_CURRENT_TX_RATE=180
OID_802_11_RSSI=-41 (Fair)
\>
```

11.11.2. Display transmission driver statistics

This command reads, or clears, reception and transmission driver statistics, which are more detailed than standard NDIS statistics:

```
\> wificonf -stats clear
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
OID_WIFIMAC_RESET_STATS Statistics Reseted
\> wificonf -stats read
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
OID_WIFIMAC_GET_STATS:
txBytes       = 105872866
txFrames      = 80694
txBCFrames    = 802
rxBytes       = 7253725
rxFrames      = 46558
rxBCFrames    = 46236
txRTS         = 0
txRetries     = 8375
txDropRetry   = 0
txDropBC      = 0
txDropAssoc   = 18
rxRTS         = 0
rxRetries     = 448
rxDropSize    = 0
rxDropBuffer  = 0
rxDropInvalid = 86
rxDropDup     = 319
rxDropAge     = 0
rxDropDecrypt = 53
rxDropOverrun = 0
rxDropReplay  = 0
rxDropMICFail = 53
\>
```

11.11.3. Commands for configuring driver parameters

To display or modify driver parameters, use these commands:

- WifiConf tx_power [HexValue]
- WifiConf chan_mask [HexValue]
- WifiConf tx_rate [DecValue]
- WifiConf rts_thresh [DecValue]
- WifiConf frag_thresh [DecValue]
- WifiConf options [HexValue]

Some commands take decimal values, while others take hexadecimal values.

To read the current parameter value, type this command without any argument:

```
\> WifiConf -chan_mask
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
OID_WIFIMAC_GET_CHAN_MASK=0x1001
\>
```

To modify the current parameter, add a value to the command line:

```
\> WifiConf -chan_mask 1003
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
OID_WIFIMAC_GET_CHAN_MASK=0x1001
OID_WIFIMAC_SET_CHAN_MASK set chan_mask=0x1003
\>
```

11.11.4. Store parameters to Registry

If any parameter of the WLAN interface is modified with the WifiConf tool, the new configuration can be stored into the Registry key [**HKEY_LOCAL_MACHINE\Comm\ccw9cWifi1\Parms**] with this command:

```
\> WifiConf -save_params
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
Configuration saved in registry. Remember to save registry in persistent
storage (type: regtool -h)!!!
\>
```

With this command, the new settings are stored in the Registry in RAM. To save this Registry permanently into NVRAM, you need to execute the regtool utility, covered in topic 12.

11.11.5. Preferred Network Configuration

As explained in topic 11.7.2.3, **wzctool.exe** retrieves the preferred network configuration from the registry, but it doesn't have any mechanism to delete it or to store a modified one.

This functionality is supplied by the **WifiConf.exe** tool, which has a wzctool-like submenu with options for:

- Storing the preferred network in the Registry
- Deleting all networks from the Registry
- Retrieving the preferred network from the Registry
- Automatically configuring the preferred network after start-up

11.11.5.1. Store the preferred network in the Registry

The command works by appending to **wificonf** application the **wzctool** command line (excluding the interface).

```
\>  wificonf  -wzctool  -ssid  sa_test_d  -auth  wpa2-psk  -encr  aes  -key
testdpassword
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
Deleting previous Preferred Network from registry
Storing Preferred Network in registry
Done Successfully.
Configuration  saved  in  registry.  Remember  to  save  registry  in  persistent
storage (type: regtool -h)!!!
\>
```

This command creates a connection like the one seen in topic 11.9.6.2 but, additionally, this entry is added to the Registry:

```
[HKEY_CURRENT_USER\Comm\WZCTOOL]
    "SSID"            = "sa_test_d"
    "authentication" = dword:7  ;WPA2-PSK (Ndis802_11AuthModeWPA2PSK)
    "encryption"     = dword:6  ;AES (Ndis802_11Encryption3Enabled)
    "key"            = "testdpassword"
    "adhoc"          = dword:0  ;CE8021X is an infrastructure network
```

With this command, the new settings are stored in the Registry in RAM. To save the Registry permanently into NVRAM, execute the regtool utility, covered in topic 12.

The WZCTOOL Registry entry exposes the encryption key. If access to the target's Registry is not secure, this would be a security hole.

11.11.5.2. Delete all networks from the Registry

To delete all networks from the Registry, enter this command:

```
\> wificonf -wzctool -reset
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
Deleting Preferred network from registry
Done Successfully
Configuration saved in registry. Remember to save registry in persistent
storage (type: regtool -h)!!!
\>
```

This command deletes all networks of **wzcsvc** as if **wzctool -reset** command was used; in addition, the **[HKEY_CURRENT_USER\Comm\WZCTOOL]** Key entry, with the preferred network seen before, is deleted from the Registry.

With this command, the new settings are stored in the Registry in RAM. To save the Registry permanently into NVRAM, you need to execute the regtool utility, covered in topic 12

11.11.5.3. Retrieve the preferred network from the Registry

To retrieve the preferred network from the Registry, enter this command:

```
\> wificonf -wzctool -registry
CCW9CWIFI1 Adapter Detected: Digi ccw9cWifi Wireless LAN Adapter.
Recovering Preferred network from Registry
Done Successfully
\>
```

This command retrieves the preferred network from the registry for **wzcsvc just** as if the **wzctool -registry** was used. This command is a wrapper function to bring the whole functionality to **WifiConf** application.

11.11.5.4. Automatic configuration of the preferred network after start-up.

WifiConf.exe tool is automatically executed at start-up (after device.dll has been loaded) because of this entry in **platform.reg** file:

```
[HKEY_LOCAL_MACHINE\Init]
        "Launch85"="WifiConf.exe"
        "Depend85"=hex:14,00
```

This startup entry executes **WifiConf** tool in a special mode to retrieve the preferred network from the Registry but with these considerations:

- Silent mode: No messages are displayed to avoid opening command windows on the screen.
- The attempt to add the preferred network is not done until CCW9CWIFI1 reports that an AP is available.
- The automatic configuration expires after a timeout of 30 seconds.

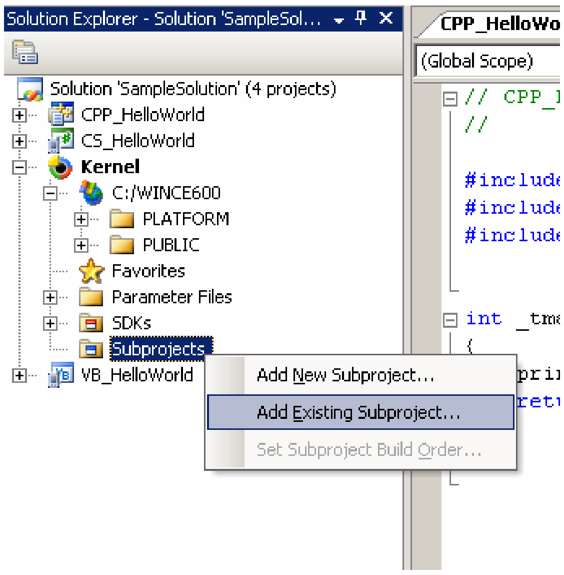
11.11.6. Source code for WifiConf

The **WifiConf** source code is in the folder **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Apps\Source Code\WifiConf\**. Either modify it or use it as example to create other applications that access the WLAN driver.

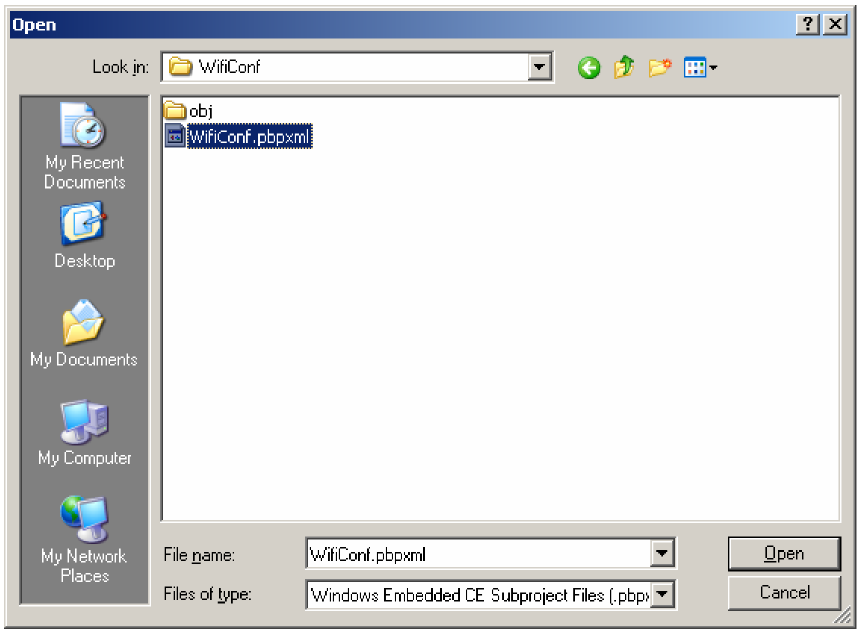
11.11.6.1. Build the WifiConf tool

WiFiConf can be built as a Windows CE native code C++ application. To add the **WifiConf** application to an OS design:

1. Select the OS design in the Solution explorer (in this case, **Kernel**) and select the **Subprojects** element in it. Right-click and select **Add Existing Subproject.**



2. Open the folder containing the WifiConf source code, select file **WifiConf.pbpxml** and click **Open**.



The **WifiConf** tool is added to the OS design.

11.11.6.2. About the code

The application gets access to the standard NDIS ccw9cwifi driver through the NDISUIO interface. After the interface is opened, both standard and not standard OIDs can be executed. The standard OIDs are those described in the standard NDIS specification, such as OID_802_11_SSID, OID_802_11_INFRASTRUCTURE_MODE, etc.

The driver's non standard OIDs are described in the file **%_WINCEROOT%\PLATFORM\CCX9C\SRC\DRIVERS\wifimac\oidwifimac.h**. Description of the OIDs follow.

Name	Description	Related Registry entry	Range
OID_WIFIMAC_GET_TX_POWER OID_WIFIMAC_SET_TX_POWER	Transmit Power	"tx_power"	0 to 15
OID_WIFIMAC_SET_CHAN_MASK OID_WIFIMAC_GET_CHAN_MASK	Bitmap of allowed channels.	"chan_mask"	Bit 0 is channel 1, and so on.
OID_WIFIMAC_GET_TX_RATE OID_WIFIMAC_SET_TX_RATE	Maximum transmit rate (in units of 100 kbps), so 540 == 54 mbps	"tx_rate"	Max 540
OID_WIFIMAC_GET_RTS_THRESH OID_WIFIMAC_SET_RTS_THRESH	RTS threshold, 0 to use default.	"rts_thresh"	0 to 2347
OID_WIFIMAC_GET_FRAG_THRESH OID_WIFIMAC_SET_FRAG_THRESH	Fragmentation threshold, 0 to use default.	"frag_thresh"	0 to 2346
OID_WIFIMAC_GET_OPTIONS OID_WIFIMAC_SET_OPTIONS	Bitmap of options: 0x0001 Enable antenna diversity 0x0002 Enable short preamble 0x0004 Enable server certificate verification 0x0008 Use only 802.11b rates in 2.4 GHz band 0x0010 Use RTS/CTS protection frames for 802.11g 0x0020 Use fixed transmit rate 0x0040 Enable 802.11 Multi domain capability(802.11d)	"options"	N/A
OID_WIFIMAC_GET_STATS OID_WIFIMAC_RESET_STATS	Get/Reset ccw9cWifi internal statistics	N/A	N/A
OID_WIFIMAC_STATION_STATE	Gets internal driver state: WLN_ST_STOPPED, WLN_ST_SCANNING, WLN_ST_ASSOC_ESS, WLN_ST_AUTH_ESS, WLN_ST_JOIN_IBSS, WLN_ST_START_IBSS	N/A	0 to 5
OID_WIFIMAC_GET_CURRENT_TX_RATE	Get current TX rate	N/A	Max 540

12. Persistent Registry

This topic describes the **regtool** application, used to save Registry settings to NVRAM.

12.1. Regtool application

While working with the target, interface settings and OS parameters can be modified, for example, network settings and memory configuration. These changes are saved to the target's Windows Registry, a database that stores settings and options for your target's operating system. Because the target's Registry resides in RAM memory, these settings are lost when the target is powered off.

The **regtool** application stores the Registry into NVRAM so the settings are not lost.

To see the application’s syntax, execute it with the **–h** option:

```
\> regtool -h
Application to save or erase the Registry. Revision 1.1
Copyright(c) 2007 by Digi International Inc.

  Usage: regtool <options>

  Where options are:
    -s    Save registry to flash
    -e    Erase registry in flash
    -h    Show this helps

\>
```

Options include:

- **-s**: Saves the Registry to the first flash partition of type **WinCE-Registry**.
- **-e**: Erases the first flash partition of type **WinCE-Registry**.

When the system boots, if it finds a correct Registry in the first flash partition of type **WinCE-Registry**, it loads these settings for the running system. Otherwise, the system generates a Registry in RAM with default values.

13. Boot loader development

This topic describes how to work with the boot loader. It covers how to customize, build, and install a boot loader image, then update the flash memory with the newly generated boot loader image.

13.1. Development environment

U-Boot sources are prepared to be built in a Linux environment. To develop with U-Boot under a Windows based host PC, you must install **Digi U-Boot SDK** from the ConnectCore BSP for Windows Embedded CE 6.0 CD-ROM, which installs the Cygwin^{TM} environment, Microcross GNU X-Tools and the U-Boot boot loader source files.

For the U-Boot boot loader to build correctly, its source code must be installed in a path without blank spaces.

13.2. Platform specific source code

Several files are used to customize U-Boot:

File	Description
include/configs/*yourplatform*.h	Default configuration for the modules
include/configs/digi_common*.h	Configuration for all Digi modules
board/*yourplatform*/*yourplatform*.c	Platform initialization

13.3. Customize U-Boot

13.3.1. Default environment variables

U-Boot has a set of default environment variables that are defined in the environment variable **CONFIG_EXTRA_ENV_SETTINGS** in **include/configs/*yourplatform*.h**.

Digi has extended U-Boot by **dynamic** environment variables. These variables are auto-generated, depending on the platform and the module U-Boot runs, and are used by the **dboot** and **update** commands.

The list of dynamic variables and their current values can be retrieved with U-Boot command **printenv_dynamic.** The standard command **printenv** does not list the variables unless they are overwritten by user action, as seen in this example:

```
#   printenv_dynamic
eimg=eboot-CCX9C
#   printenv eimg
## Error: "eimg" not defined
#   setenv eimg myeimage
#   printenv_dynamic
eimg=myeimage
```

To return dynamic variables to their default values, set them to a value of nothing with **setenv**:

```
#   setenv eimg
#   printenv_dynamic
eimg=eboot-CCX9C
```

13.3.1.1. Windows Embedded CE-related environment variables

Digi has implemented specific variables for different OS implementations (Linux, Windows Embedded CE, NET+OS). These are the specific Windows Embedded CE related variables:

Variable	Description
ebootaddr	The RAM address in which to place the EBOOT image.
wceloadaddr	The RAM address in which to place the Windows Embedded CE kernel image.
wimg	Windows Embedded CE kernel image filename; used for kernel updates.
eimg	EBOOT image filename; used for kernel updates.

13.4. Build U-Boot

A script is provided to help compile the U-Boot boot loader. To display the script's syntax, open **Cygwin** in the U-Boot installation folder and execute:

```
$   ./userbuild_cyg.sh –h

Usage: userbuild_cyg.sh [options] <platform>

        -l           List available platforms
        -c           Configure Project
        -b           Build project
        -i <path>    Install path for the U-Boot images

Available platforms:
       ccw9cjsnand
       ccw9cjsnand_dbg
       cc9cjsnand
       cc9cjsnand_dbg
```

*Platforms with the **_dbg** suffix are special versions for running with a hardware debugger.*

13.4.1. Configure U-Boot for the target platform

Before U-Boot is compiled, it needs to be configured for one of the available platforms. This is done only one time, with the **–c** option. For example, for the ConnectCore Wi-9C, execute:

```
$   ./userbuild_cyg.sh –c ccw9cjsnand
Configuring ConnectCore Wi-9C with NAND Flash on Development Board
Configuring for ccw9c board...
```

13.4.2. Compile U-Boot

After configuring the target platform, to build U-Boot, execute:

```
$   ./userbuild_cyg.sh –b ccw9cjsnand
```

The build command compiles the boot loader sources and generates the U-Boot image. The image is stored with the name **u-boot-*platform*.bin** (where ***platform*** is substituted with the platform name) in the folder in which U-Boot sources are installed (the same folder from which the command was launched).

13.4.3. Install U-Boot image

The installation folder for the image can be specified with the **–i** option. This folder can be a TFTP exposed folder or a USB disk that will be used to make the image accessible for the target.

```
$   ./userbuild_cyg.sh –i /cygdrive/c/tftproot ccw9cjsnand
```

13.5. Updating U-Boot

13.5.1. Update from a running Windows Embedded CE system

If a Windows Embedded CE system is running on the target, the U-Boot partition can be updated with a new U-Boot image by using the **update_flash** tool, as seen in topic 8.2.

13.5.2. Update from U-Boot

A useful feature of U-Boot is its ability to update itself. As demonstrated in topic 8.3, the U-Boot **update** command, can directly write to the flash memory.

Depending on the setting for the **type** parameter, the **update** command gets the image file from either from a USB flash disk or a TFTP exposed folder in the host. The command acquires the file name from the value stored in this U-Boot environment variable:

U-Boot image filename: **uimg**

The default value for this variable corresponds to the default image filename generated during compilation of the boot loader. If the image filename was changed, this U-Boot variable must be set accordingly.

The U-Boot update command transfers the image file to RAM, erasing flash sectors and writing the new image. For example, if the boot loader is in the TFTP exposed folder on the development computer, the update command is:

```
#   update uboot tftp
```

At the reboot, the new U-Boot start message is displayed. To check it, look at the compilation date.

14. Troubleshooting

Here are common issues and solutions when using Windows Embedded CE.

14.1. Language settings

The default locale for the OS designs is **English (United States)**. If a different locale is selected for the OS design, it is important to pay attention to the paths when copying files to the target. Some folders have different names depending on the selected locale; for example **My Documents** changes to **Mis Documentos** in Spanish language.

Windows Embedded CE supports these locales:

- Arabic
- English (U.S.)
- English (Worldwide)
- French
- German
- Hebrew
- Indic
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese
- Thai

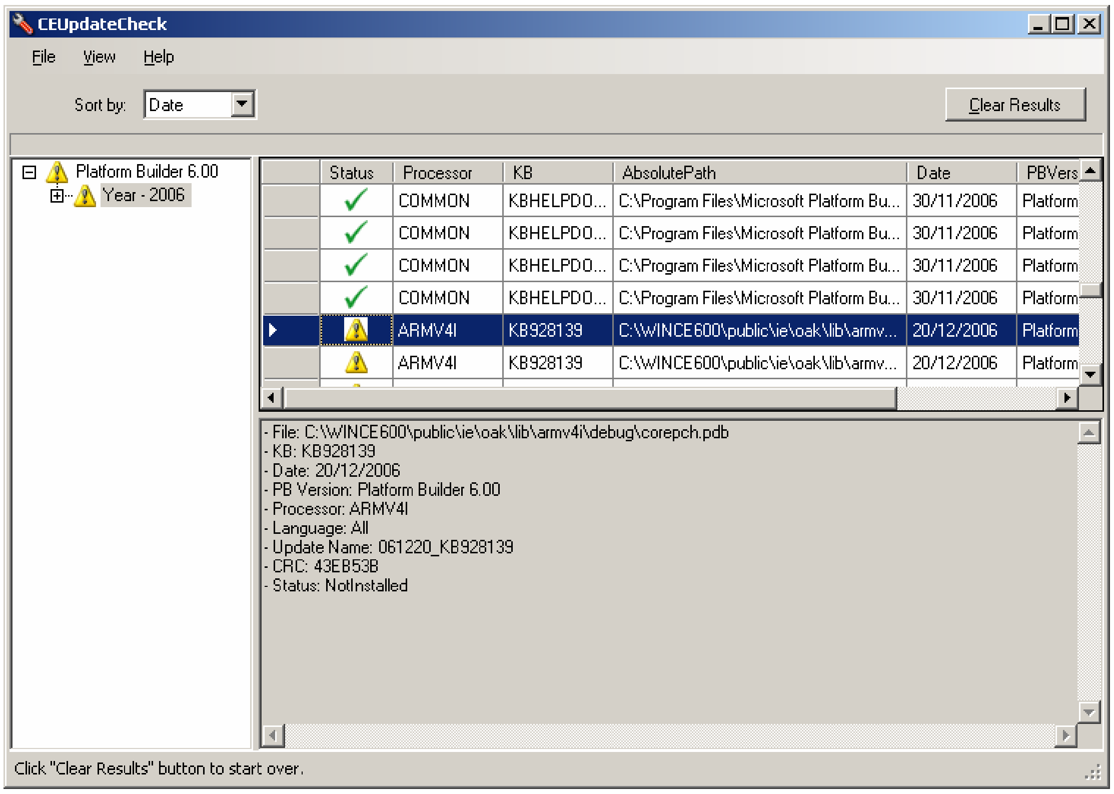
14.2. Monthly updates

Microsoft releases periodic updates of the Platform Builder plug-in and the Windows Embedded CE source code. Periodically, check the Microsoft Web site for new monthly updates:

http://msdn.microsoft.com/embedded/downloads/ce/wince/default.aspx

To check which updates are currently installed and which ones are available:

1. In Visual Studio, select **Tools > Platform Builder for CE 6.0 > CE Update Check**.
2. In the window that opens, click **Verify Updates**. A list of installed updates is displayed, indicated by a green check symbol, as well as available uninstalled items, indicated by a warning symbol. If new items are available, visit the URL above, download the items, and install them to make sure the latest release of Windows Embedded CE 6.0 is installed.



	Status	Processor	KB	AbsolutePath	Date	PBVers
	✓	COMMON	KBHELPDO...	C:\Program Files\Microsoft Platform Bu...	30/11/2006	Platform
	✓	COMMON	KBHELPDO...	C:\Program Files\Microsoft Platform Bu...	30/11/2006	Platform
	✓	COMMON	KBHELPDO...	C:\Program Files\Microsoft Platform Bu...	30/11/2006	Platform
	✓	COMMON	KBHELPDO...	C:\Program Files\Microsoft Platform Bu...	30/11/2006	Platform
▶	⚠	ARMV4I	KB928139	C:\WINCE600\public\ie\oak\lib\armv...	20/12/2006	Platform
	⚠	ARMV4I	KB928139	C:\WINCE600\public\ie\oak\lib\armv...	20/12/2006	Platform

```
- File: C:\WINCE600\public\ie\oak\lib\armv4i\debug\corepch.pdb
- KB: KB928139
- Date: 20/12/2006
- PB Version: Platform Builder 6.00
- Processor: ARMV4I
- Language: All
- Update Name: 061220_KB928139
- CRC: 43EB53B
- Status: NotInstalled
```

Click "Clear Results" button to start over.

14.3. Using native functions in managed applications

Managed code applications, like those developed in C#, do not have easy access to drivers functions, which are developed in native C code and packed in DLL libraries.

To make calls to native code from a managed application, use the **DllImport** attribute.

For more information, visit *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/vcwlkSysimportAttributeTutorial.asp* and examine the driver's sample applications.

14.4. Using pointers and addresses in managed applications

Using pointers and addresses in managed C# applications running on the .NET Framework is forbidden because it is considered unsafe code.

To use pointers and addresses, the piece of code must be enclosed within **unsafe** braces { }, as the example shows:

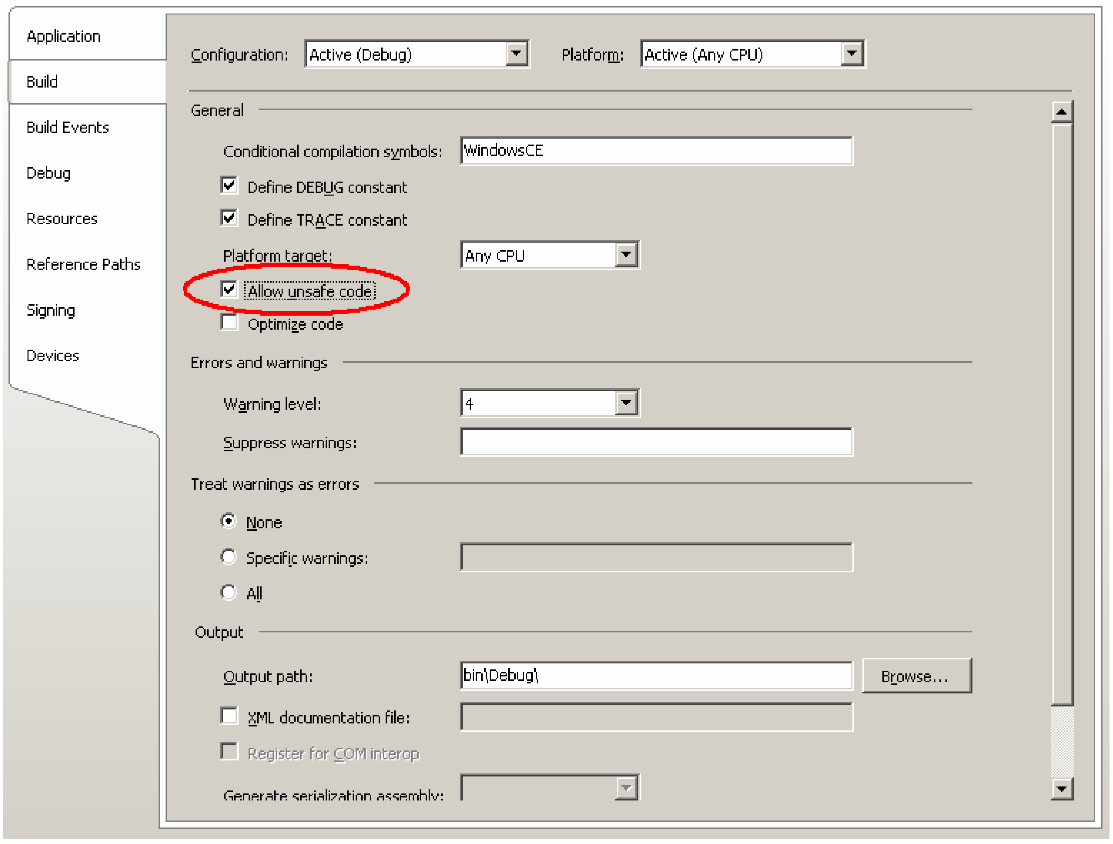
```
    int     variable;
unsafe{
    int     *pointer;
}

    variable = 3;
unsafe{
    pointer = &variable;
}
```

In addition, the project must be configured to allow compilation of unsafe code:

1 In the **Solution Explorer**, select the C# project.

2. Right-click it and select **Properties**.

3. Select the **Build** tab. In the **General** section, check **Allow unsafe code**.



Configuration: Active (Debug)
Platform: Active (Any CPU)

General

Conditional compilation symbols: WindowsCE
[✓] Define DEBUG constant
[✓] Define TRACE constant
Platform target: Any CPU
[✓] Allow unsafe code
[ ] Optimize code

Errors and warnings

Warning level: 4
Suppress warnings:

Treat warnings as errors

[● None]
○ Specific warnings:
○ All

Output

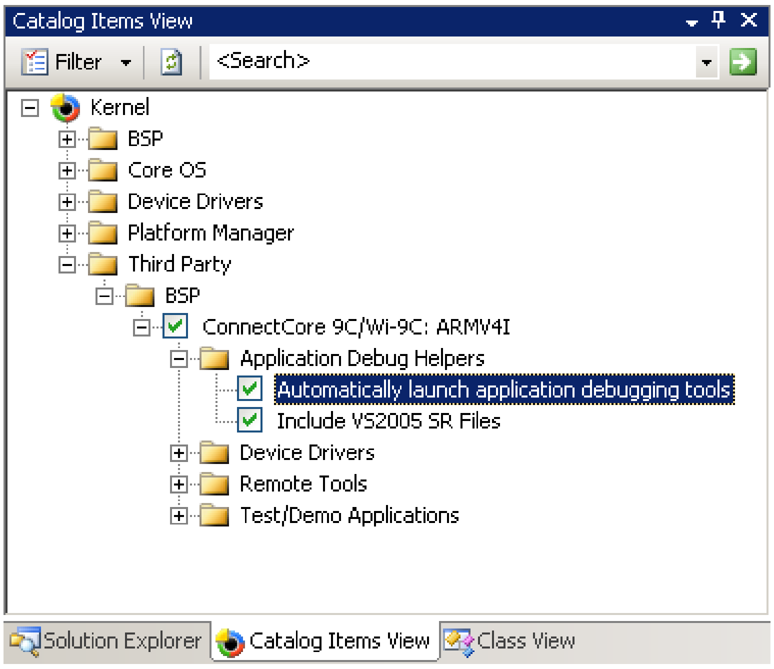
Output path: bin\Debug\
[ ] XML documentation file:
[ ] Register for COM interop
Generate serialization assembly:

14.5. Including and launching debugging services

As explained in topic 2.3.3, two services must be running in the target platform to make it listen to debug connections: **conmanclient2.exe** and **cmaccept.exe**. If the OS design was based in the ConnectCore 9C or Wi-9C template, these services are included and launched by default.

To include and launch these services in custom OS designs, go to the Catalog view and expand **Third Party > BSP > ConnectCore 9C/Wi-9C:ARMV4I > Application Debug Helpers**. Check these two items:

- **Automatically launch application debugging tools**: Includes application tools to automatically launch the **conmanclient2.exe** and **cmaccept.exe** services, to listen for debug connections.
- **Include VS2005 SR files**: Includes Visual Studio 2005 SR files into the OS design for application debugging.



Then recompile and download the new OS design. The new kernel launches the applications automatically to listen for debug connections.

14.6. Writing large files to flash from U-Boot

The **update** command in U-Boot transfers files to RAM, erases the flash partition, and writes the files from RAM into flash memory.

The transferred file is copied to a certain physical address in RAM, therefore the maximum length of the file to update is:

update file size limit = total RAM memory – RAM offset where the file was loaded

As a general rule, U-Boot does not allow updating a flash partition with a file whose size exceeds the available RAM memory. For example, for a module with 32MB RAM and 64MB flash, U-Boot will not update a partition with a file that is 35MB.

Note that this limitation is due to the RAM memory size, as U-Boot first needs to transfer the file to RAM before copying it to flash.

To update files bigger than the available RAM, use the **update_flash** tool (explained in topic 8.2).

14.7. Run-time licenses

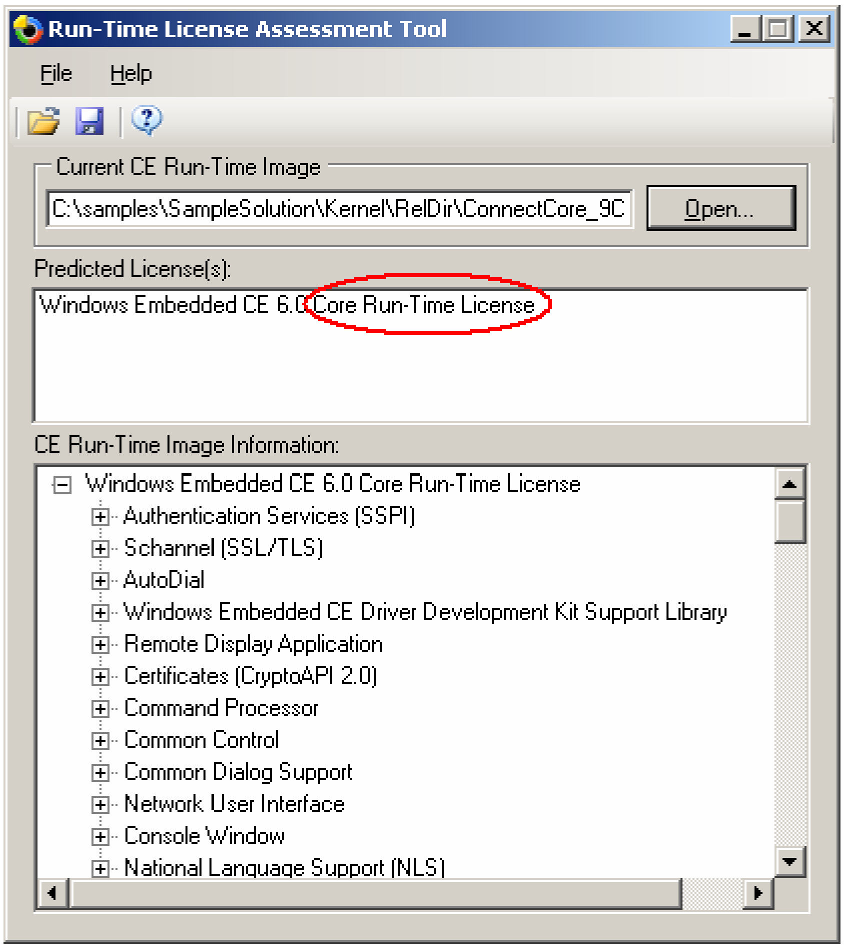
The licensing model for Microsoft Windows Embedded CE offers two run-time license options:

- **Core**: targeted to the low-end device market
- **Professional**: targeted to devices that need the full set of Windows Embedded CE features

Depending on the components included in the OS designed, one license or the other is required.

To determine which runtime license a custom OS design needs:

1. Select **Tools > Platform Builder for CE 6.0 > Run-Time License Assessment Tool**. A dialog opens.
2. Read the Terms of Use and click **Accept**.
3. In the window that appears, click **Open** and select file **ceconfig.h** or **nk.bin** of the OS design. These files are generated during compilation of the OS design and can be located at the "Flat Release Directory." The tool predicts the required run-time license.



Current CE Run-Time Image
C:\samples\SampleSolution\Kernel\RelDir\ConnectCore_9C

Predicted License(s):
Windows Embedded CE 6.0 Core Run-Time License

CE Run-Time Image Information:
- Windows Embedded CE 6.0 Core Run-Time License
  - Authentication Services (SSPI)
  - Schannel (SSL/TLS)
  - AutoDial
  - Windows Embedded CE Driver Development Kit Support Library
  - Remote Display Application
  - Certificates (CryptoAPI 2.0)
  - Command Processor
  - Common Control
  - Common Dialog Support
  - Network User Interface
  - Console Window
  - National Language Support (NLS)

14.8. CE 5.0 application compatibility on CE 6.0

Managed applications are abstracted from the underlying operating system by the .NET framework and run in both CE 5.0 and in CE 6.0 kernels. Native applications developed for CE 5.0, on the other hand, might have compatibility problems when being executed in a CE 6.0 kernel.

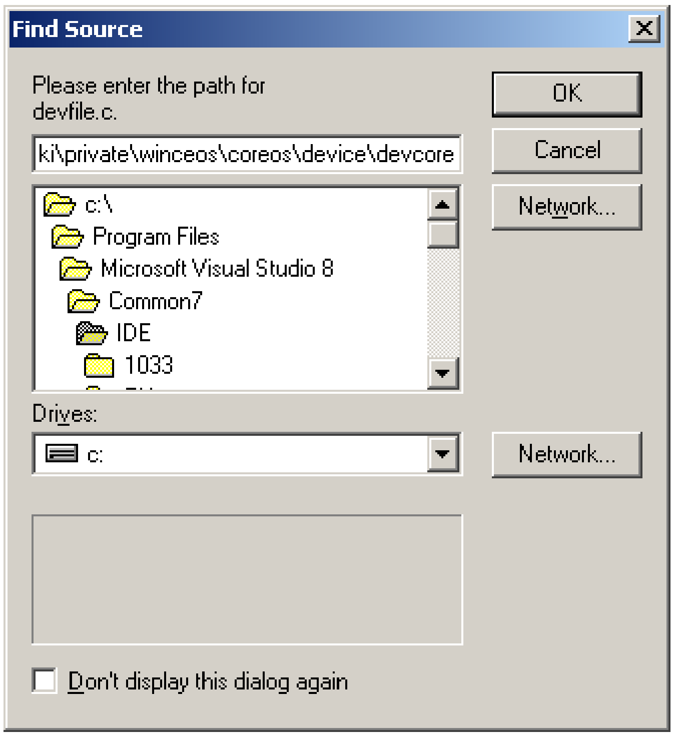
A command line tool, **CeAppCompat**, determines whether the binary applications developed for Windows CE 5.0 have compatibility problems on Windows Embedded CE 6.0. This tool can be used for applications and DLLs and folders full of applications and DLLs. The syntax of the command looks like this:

```
#   CeAppCompat -i Input -o Foo
```

where **Input** is the executable, DLL, or folder that must be checked and **Foo** is the name of the HTML report that will be generated with the compatibility information.

14.9. Finding source code for debugging

When debugging a Windows Embedded CE kernel, Visual Studio 2005 sometimes cannot find the path to the source file of the process being executed. In those cases, the **Find Executable** dialog prompts for the right path:


Find Source
Please enter the path for devfile.c.
ki\private\winceos\coreos\device\devcore
c:\
Program Files
Microsoft Visual Studio 8
Common7
IDE
1033
Drives:
c:
OK
Cancel
Network...
Network...
Don't display this dialog again

If the source file is available but in a different path, select the proper path and click **OK** to open the source code for debugging.

If the source file is not available, or debug the source file is not desired, click **Cancel** for the debug process to continue.

14.10. Application and Device Debugging

Deploying a Windows CE solution and debugging a custom application cannot be done from the same instance of Visual Studio 2005. To do both, different instances of Visual Studio 2005 must be opened.

14.11. Shorten build process

To shorten the build process during development phase, when adding new drivers or application to the Windows CE kernel in the **platform.bib/project.bib** or changing the registry **platform.reg/project.reg** for testing, it may be easier to make the changes in the corresponding files in the **%FLATRELEASEDIR%**.

Once the changes are made, in the Visual Studio 2005 Build menu, select the menu option **Make Run-Time Image** to build the new Windows CE kernel with added files or registry modifications.

*Changes made in the **%FLATRELEASEDIR%** files are overwritten the next time a SYSGEN is executed for the project if the changes are not moved to files in Visual Studio 2005.*

14.12. Windows CE Image Size

Because of the image size of the delivered Windows CE kernel, the system may run out of virtual memory on a module with 32MB of RAM. On the target, the memory configuration between application memory and file system memory can be configured. Go to **Control Panel** > **System**. Change the slider position to increment the amount of memory available for the application memory and confirm it by clicking on the **OK** button.

15. Recovering a device

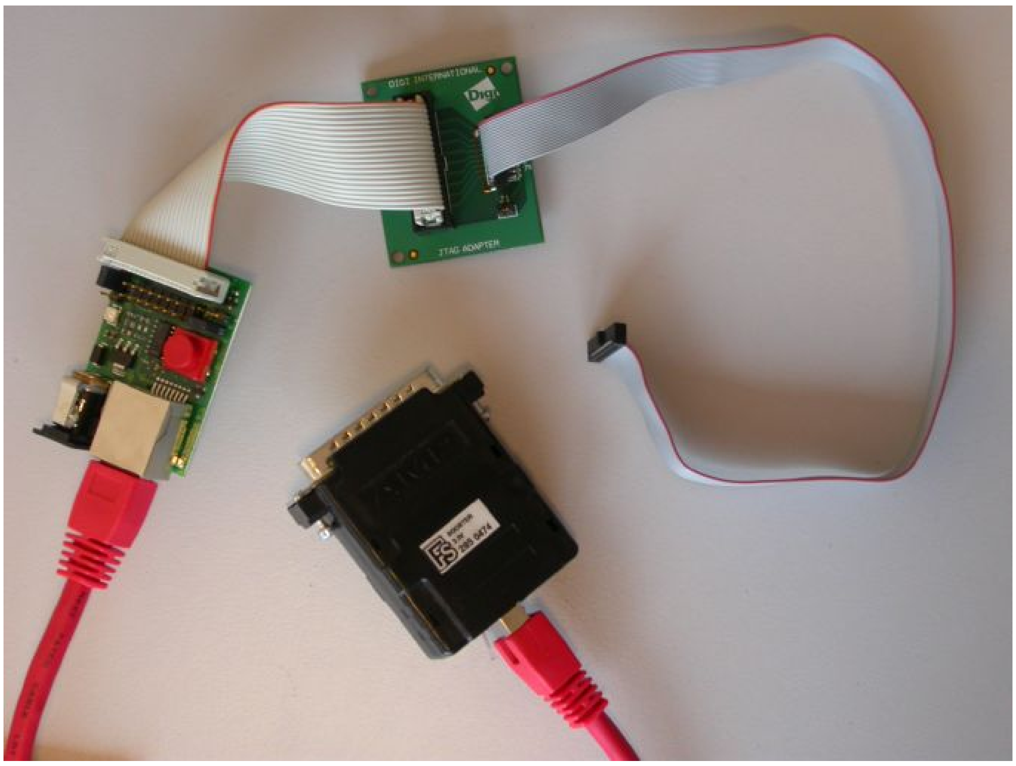
Normally, Windows CE application development involves creating an OS design and developing applications. Even bad a kernel images are written that cannot boot, new images can be rewritten e from the U-Boot monitor shell. Nonetheless, it may be best to create your custom boot loader and update it, as seen in topic 12. If a custom boot loader is not able to boot, or if the boot loader is erased from flash, a special hardware tool, called the JTAG booster is needed to recover this fundamental part of software.

This topic describes how to use the JTAG booster.

15.1. JTAG tool and software

The JTAG-Booster is a hardware tool with a DB25 connector in one end (to be attached to a PC) and 8 lines on the other (JTAG lines). The tool ships with cables and adaptors for connecting it to the development board. The logic in the JTAG tool allows for control of all the lines of the microprocessor over the JTAG bus.

Together with its DOS software, the JTAG-Booster is used to program flash memory. This is useful to recover a module that cannot even boot the U-Boot boot loader.



The JTAG-Booster tool and its software are sold as a separate product. Contact your Digi distributor for purchasing information.

The JTAG tool software runs from both the DOS operating system and from a standard Microsoft Windows installation (98, NT, 2000 or XP). To run the tool from Windows, a special driver, the **Kithara DOS enabler**, is required to access the parallel port, which has restricted access in NT platforms.

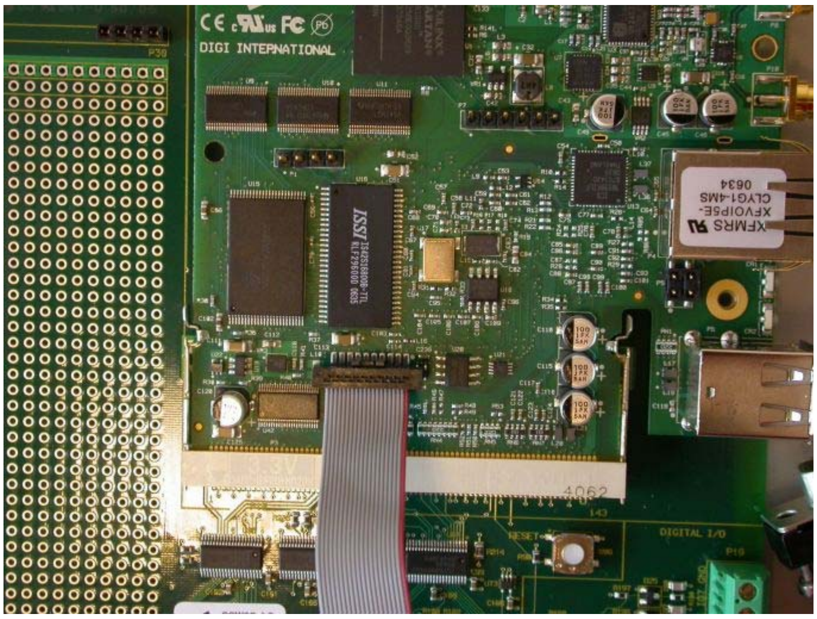
To install the Kithara DOS enabler:

1. Insert the JTAG tool CD-ROM in the Windows host machine. Double-click the file **JTAG\DOSenabler\ksetup.exe**. A reboot of the computer is required before continuing.
2. Reboot the computer and enter in the BIOS configuration screen. Verify that the BIOS parallel port settings are set to **Standard Parallel Port (SPP)** and not **EPP**.
3. Copy the JTAG directory from CD-ROM to the Windows host hard drive; for example, into **C:\JTAG**.

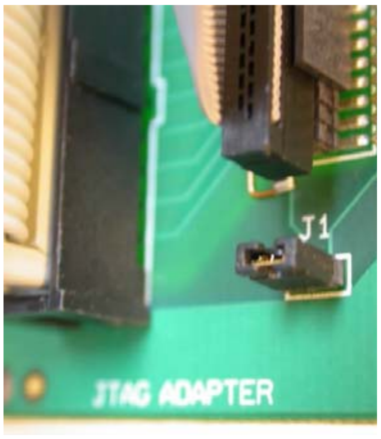
Not installing the Kithara DOS Enabler software or setting the BIOS parallel port to other than SPP can result in the error "cable not connected or target power fail" when running the JTAG tool from Windows.

15.2. Program a U-Boot into flash with the JTAG tool

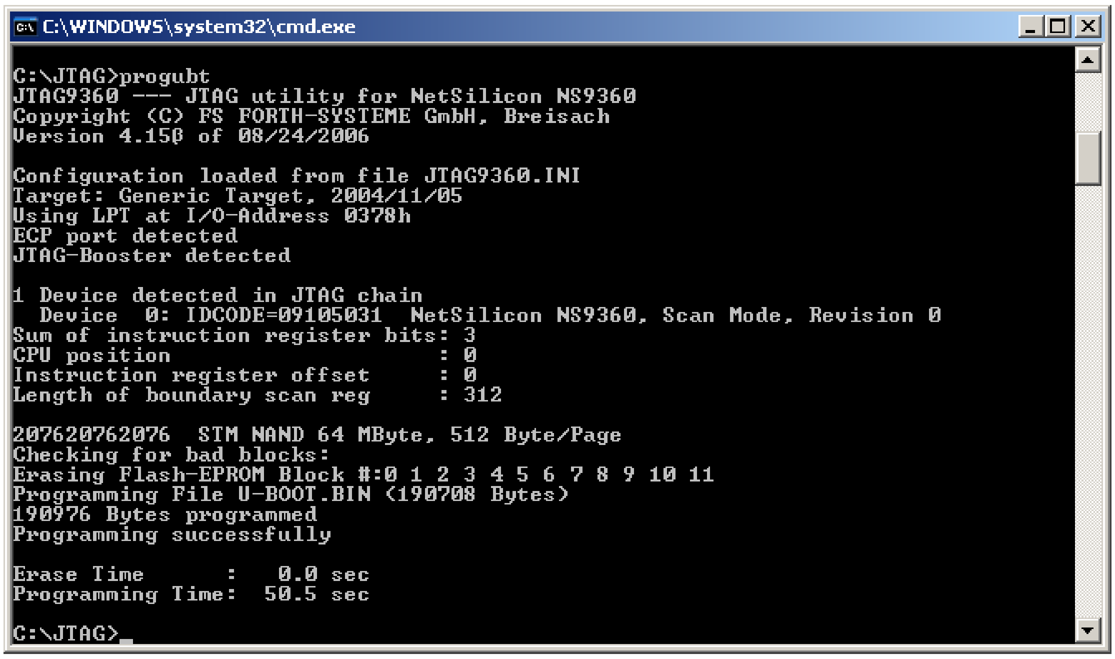
1. Connect the JTAG Booster to the parallel port of the Windows/DOS PC.
2. Connect the adapter cable to the hardware module plugged into the development board.



3. Plug jumper J1 on the JTAG adapter to switch to JTAG "boundary scan" mode. This is needed for the JTAG tool software (if unplugged, the adapter is using JTAG debug mode).



4. Copy the boot loader image file that is desired to be stored in flash memory (for example, **u-boot-ccw9cjsnand.bin**) to the installed JTAG directory on the Windows/DOS PC.
5. Rename the file **u-boot.bin**. (Because the JTAG tool is DOS software, it manages only 8 + 3 filenames.)
6. Power on the module on the development board.
7. Launch a command prompt or DOS shell on the Windows/DOS PC and change to the **JTAG** directory.
8. Start the batch file **progubt.bat** from the DOS command prompt. Output like this should be displayed:



```
C:\JTAG>progubt
JTAG9360 --- JTAG utility for NetSilicon NS9360
Copyright (C) FS FORTH-SYSTEME GmbH, Breisach
Version 4.15β of 08/24/2006

Configuration loaded from file JTAG9360.INI
Target: Generic Target, 2004/11/05
Using LPT at I/O-Address 0378h
ECP port detected
JTAG-Booster detected

1 Device detected in JTAG chain
  Device  0: IDCODE=09105031  NetSilicon NS9360, Scan Mode, Revision 0
Sum of instruction register bits: 3
CPU position                    : 0
Instruction register offset     : 0
Length of boundary scan reg     : 312

207620762076  STM NAND 64 MByte, 512 Byte/Page
Checking for bad blocks:
Erasing Flash-EPROM Block #:0 1 2 3 4 5 6 7 8 9 10 11
Programming File U-BOOT.BIN (190708 Bytes)
190976 Bytes programmed
Programming successfully

Erase Time       :   0.0 sec
Programming Time:  50.5 sec

C:\JTAG>_
```

The programming takes approximately 60 seconds, after which the message **Programming successfully** is displayed.

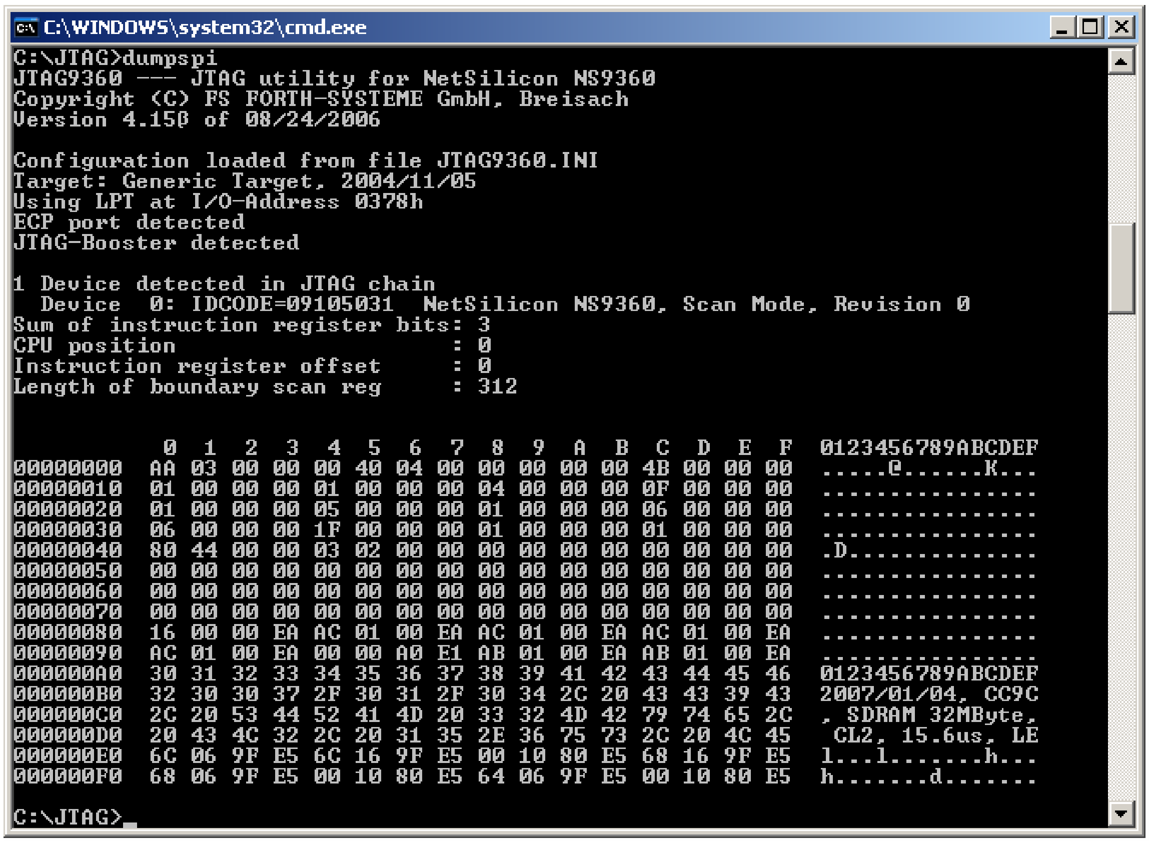
9. Reset or power off/on the module. The command prompt of U-Boot on the serial port should be displayed. If the prompt is not displayed, check the tool's output for error messages, recheck cable connections, power, and jumper settings, and retry flash programming. If the procedure still fails, try using the factory default boot loader image **u-boot.bin** located at **%PROGRAM_FILES%\Digi\ConnectCore\ConnectCore 9C and Wi-9C\Images\bootloader\*platform*,** where ***platform*** is substituted with the platform name, or at the Digi support web site: *http://www.digi.com/support/*.

15.3. Update the SPI loader

The boot loader itself cannot be executed directly from NAND flash. Instead, a small part of software, the SPI loader, is stored in an SPI EEPROM and executed at power-on. The SPI loader configures the SDRAM and the NAND flash, switches to little endian mode, and copies the first KBytes from NAND (U-Boot partition) to SDRAM. Then the U-Boot in SDRAM is executed.

Very rarely, the SPI loader needs to be updated, or the EEPROM contents are destroyed by accident. In these cases, the SPI loader must be reprogrammed using the JTAG tool.

1. Connect the JTAG-Booster as seen in topic 15.2.
2. On the Windows/DOS PC, launch a command prompt or DOS shell.
3. Change to the **JTAG** directory.
4. Start the batch file **dumpspi.bat** to check the first bytes of the SPI loader. Data like this is displayed:



```
C:\JTAG>dumpspi
JTAG9360 --- JTAG utility for NetSilicon NS9360
Copyright (C) FS FORTH-SYSTEME GmbH, Breisach
Version 4.15β of 08/24/2006

Configuration loaded from file JTAG9360.INI
Target: Generic Target, 2004/11/05
Using LPT at I/O-Address 0378h
ECP port detected
JTAG-Booster detected

1 Device detected in JTAG chain
  Device  0: IDCODE=09105031  NetSilicon NS9360, Scan Mode, Revision 0
Sum of instruction register bits: 3
CPU position                    : 0
Instruction register offset     : 0
Length of boundary scan reg     : 312


           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00000000  AA 03 00 00 00 40 04 00 00 00 00 00 4B 00 00 00   .....@......K...
00000010  01 00 00 00 01 00 00 00 04 00 00 00 0F 00 00 00   ................
00000020  01 00 00 00 05 00 00 00 01 00 00 00 06 00 00 00   ................
00000030  06 00 00 00 1F 00 00 00 01 00 00 00 01 00 00 00   ................
00000040  80 44 00 00 03 02 00 00 00 00 00 00 00 00 00 00   .D..............
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000080  16 00 00 EA AC 01 00 EA AC 01 00 EA AC 01 00 EA   ................
00000090  AC 01 00 EA 00 00 A0 E1 AB 01 00 EA AB 01 00 EA   ................
000000A0  30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46   0123456789ABCDEF
000000B0  32 30 30 37 2F 30 31 2F 30 34 2C 20 43 43 39 43   2007/01/04, CC9C
000000C0  2C 20 53 44 52 41 4D 20 33 32 4D 42 79 74 65 2C   , SDRAM 32MByte,
000000D0  20 43 4C 32 2C 20 31 35 2E 36 75 73 2C 20 4C 45    CL2, 15.6us, LE
000000E0  6C 06 9F E5 6C 16 9F E5 00 10 80 E5 68 16 9F E5   l...l.......h...
000000F0  68 06 9F E5 00 10 80 E5 64 06 9F E5 00 10 80 E5   h.......d.......

C:\JTAG>_
```

Watch the output to see which version of the SPI loader is on the module. Check that the correct date, module, and SDRAM size for the module are displayed.

If the SPI loader version information is correct, do not update the SPI loader.

To update the SPI loader:

1. Copy the new SPI loader image to the **JTAG** folder of the Windows host:
   - For modules with 64MB SDRAM: **spi64.bin**
   - For modules with 32MB SDRAM: **spi32.bin**.
2. If the image has a different name, rename it either **spi64.bin** or **spi32.bin**, according to its memory size.
3. Run either of these:
   - For modules with 64 MB SDRAM:

```
$   progspi.bat spi64.bin
```

   - For modules with 32 MB SDRAM

```
$   progspi.bat spi32.bin
```

16. Uninstalling

The different components of the BSP can be uninstalled separately using Windows Control Panel **Add or Remove Programs**:

- **Digi JumpStart BSP for Windows Embedded CE 6.0**: Uninstalls the ConnectCore 9C/Wi-9C BSP (Board Support Package), documentation and Digi TFTP server.
- **Digi U-Boot SDK**: Uninstalls the U-Boot source code.
- **Digi JumpStart SDK for Windows Embedded CE 6.0**: Uninstalls the SDK.
- **Microcross Cygwin & X-Tools 3.40 GNU Tools**
- **Microsoft Visual Studio 2005**
- **Windows Embedded CE 6.0**

Uninstalling these components does not remove the OS designs or custom applications created during development. These must be removed by hand.

17. References

U-Boot Reference Manual

Manual of Digi's implementation of the U-Boot boot loader, with description of built-in commands and environment variables.

Index