eBox-3350MX
Windows Embedded Compact 7 Jump Start

Getting Started Guide
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>6</td>
</tr>
<tr>
<td>eBox-3350MX Compact 7 Jump Start Kit</td>
<td>7</td>
</tr>
<tr>
<td>Preconfigured Software</td>
<td>8</td>
</tr>
<tr>
<td>BIOSLoader</td>
<td>8</td>
</tr>
<tr>
<td>EBOOT.BIN</td>
<td>8</td>
</tr>
<tr>
<td>NK.BIN and NK2.BIN</td>
<td>8</td>
</tr>
<tr>
<td>Part 1 – Development Environment Overview</td>
<td>9</td>
</tr>
<tr>
<td>Windows Embedded Compact 7</td>
<td>9</td>
</tr>
<tr>
<td>Platform Builder</td>
<td>9</td>
</tr>
<tr>
<td>Windows Embedded Compact 7 Remote Tools</td>
<td>9</td>
</tr>
<tr>
<td>Visual Studio 2008 Integrated Development Environment</td>
<td>10</td>
</tr>
<tr>
<td>Develop Compact 7 Devices: Typical Development Steps</td>
<td>10</td>
</tr>
<tr>
<td>Develop Compact 7 Applications with VS2008</td>
<td>11</td>
</tr>
<tr>
<td>Silverlight for Windows Embedded</td>
<td>11</td>
</tr>
<tr>
<td>The Target Device</td>
<td>11</td>
</tr>
<tr>
<td>Common Terminology</td>
<td>13</td>
</tr>
<tr>
<td>Compact 7 Environment and Directory Variables</td>
<td>14</td>
</tr>
<tr>
<td>OEM Environment Variables</td>
<td>14</td>
</tr>
<tr>
<td>Part 2 – Development Environment Connectivity</td>
<td>15</td>
</tr>
<tr>
<td>Ethernet Connectivity</td>
<td>15</td>
</tr>
<tr>
<td>Develop Compact 7 OS Run-time Image – OS Design</td>
<td>15</td>
</tr>
<tr>
<td>Develop Compact 7 Applications with Visual Studio 2008</td>
<td>15</td>
</tr>
<tr>
<td>Serial Debug Connectivity</td>
<td>15</td>
</tr>
<tr>
<td>Development Environment Setup with DHCP</td>
<td>16</td>
</tr>
<tr>
<td>Using Wireless Access Point Router</td>
<td>16</td>
</tr>
<tr>
<td>Development Environment Setup with Static IP</td>
<td>17</td>
</tr>
<tr>
<td>Local Area Network without DHCP Service</td>
<td>17</td>
</tr>
<tr>
<td>Direct Connection using Cross-Over RJ-45 Ethernet Cable</td>
<td>17</td>
</tr>
<tr>
<td>Static IP Addresses</td>
<td>17</td>
</tr>
<tr>
<td>Part 3 – Required Software &amp; Installation</td>
<td>18</td>
</tr>
<tr>
<td>Recommended Software Installation Sequence</td>
<td>18</td>
</tr>
<tr>
<td>Windows Embedded Compact 7 Installation</td>
<td>18</td>
</tr>
<tr>
<td>Board-Support-Package Installation</td>
<td>19</td>
</tr>
<tr>
<td>SDK Installation</td>
<td>20</td>
</tr>
</tbody>
</table>
AutoLaunch Component Installation.................................................................20
CoreCon Connectivity Component ........................................................................20
Part 4 – OS Design (Platform Builder Project) ..........................................................21
Visual Studio 2008 .................................................................................................21
Windows Embedded Compact 7 OS Design Wizard..................................................21
OS Design Wizard – Board Support Package (BSP) ...............................................22
OS Design Wizard – Design Templates ...................................................................23
OS Design Wizard – Applications & Media ............................................................23
OS Design Wizard – Networking & Communications ............................................24
Security Warning - Catalog Item Notification .......................................................25
Part 5 – Configure and Customize Compact 7 OS Design ........................................26
Customize the OS Design – Add Additional Components ......................................27
Customize the OS Design – Locate Component by Search......................................31
Customize the OS Design – Configuration Manager ..............................................32
Customize the OS Design – Build Options ............................................................32
Customize the OS Design – The Registry ...............................................................33
  Registry Entries for the FTP Server Component ................................................33
  Registry Entries for the Telnet Server Component .............................................34
  Registry Entries for the AutoLaunch Component ...............................................34
  Adding Registry Entries to the OS Design .........................................................34
Static IP Address (Optional) ..................................................................................35
Other Compact 7 Components .................................................................................35
Part 6 – Build & Generate OS Run-time Image .......................................................37
  The Build Process – Starting ............................................................................37
  The Build Process – Completed .........................................................................37
Part 7 – Download Compact 7 OS Run-time to Target Device .................................39
  Target Device Preparation ..................................................................................39
  Create Connectivity Profile for a Target Device .................................................40
  Associate eBox to MyTargetDevice Connectivity Profile ..................................41
  Download Compact 7 OS Run-time Image to eBox ..........................................44
  Deploy Compact 7 OS Run-time Image to Target Device ..................................45
Part 8 – Software Development Kit (SDK) ...............................................................47
  Create and Configure Compact 7 SDK ..............................................................47
  Build and Generate Compact 7 SDK ..................................................................49
Part 9 – Develop Native Code App with VS2008 ....................................................50
Create a New Native Code Application Project ................................................................. 50
Add Codes and Build the Application Project ................................................................. 53
Preparing Target Device to Connect to VS2008 IDE ......................................................... 54
Configure VS2008 Device Connectivity Settings ............................................................ 56
Establish Connectivity to the Target Device ................................................................. 58
Download Application to Target Device ........................................................................ 59
Part 10 – Develop Managed Code App with VS2008 ......................................................... 60
Create a New Managed Code Application Project ......................................................... 60
Add Codes and Build the Application Project ................................................................. 62
Preparing Target Device to Connect to VS2008 IDE ......................................................... 63
Establish Connectivity to Compact 7 Target Device ..................................................... 63
Download Application to Target Device ........................................................................ 64
Debug Application Running on Target Device ............................................................ 67
Part 11 – Debug and Remote Tools .................................................................................. 70
Download KITL Enabled Compact 7 OS Image to Target Device ..................................... 71
Remote Tool: Process Viewer ......................................................................................... 71
Remote Tool: Registry Editor ......................................................................................... 73
Remote Tool: System Information .................................................................................. 74
Part 12 – Silverlight for Windows Embedded ................................................................ 76
Congratulations! – You’ve completed all the steps ......................................................... 77
Appendix A – Development station Setup with DHCP .................................................... 78
Connecting to Local Area Network with DHCP .............................................................. 78
Using Wireless Access Point Router .............................................................................. 78
Appendix B – Development station Setup with Static IP .................................................. 79
Connecting to Ethernet Hub or Switch without DHCP .................................................. 79
Direct Connection with Cross-Over RJ-45 Ethernet Cable ............................................. 79
Static IP Address ............................................................................................................... 79
Appendix C – Windows Embedded Compact Resources .................................................. 81
Appendix D – eBox-3350MX Technical Information ........................................................ 84
  FRONT CONNECTORS OUTLINE FOR eBOX-3350MX ................................................ 86
  REAR CONNECTORS OUTLINE FOR eBOX-3350MX .................................................. 86
Appendix E – BIOSLoader ............................................................................................... 88
Primary BIN File ............................................................................................................. 88
Alternative BIN File ......................................................................................................... 88
BIOSLoader Configuration File for eBox-3350MX .......................................................... 89
Appendix F – Using Static IP Address ................................................................. 91
Configure Static IP for Prebuilt Compact 7 OS Run-time ........................................... 91
Configure OS Design with Static IP Address .............................................................. 92
Appendix G – Deploy OS Run-time with DiskPrep ...................................................... 93
Appendix H – CoreCon Connectivity ........................................................................... 96
Introduction

This Windows Embedded Compact 7 (Compact 7) getting started guide, with hands-on exercises, is written in step-by-step format to show how to create a Compact 7 OS design, build customized OS run-time image from the OS design, configure the development environment to download the OS run-time image to a target device, develop Compact 7 applications using the Visual Studio 2008 (VS2008) Integrated Development Environment (IDE).

The following subjects are covered in this guide:

- What’s included in the jump start kit
- Development environment overview
- Required software and recommended installation sequences
- Develop and configure Compact 7 OS design project
- Build customized Compact 7 OS run-time image
- Establish connectivity and download OS run-time image to a target device
- Develop manage and native code applications for Compact 7 using Visual Studio 2008, establish connectivity and download the application to target device for testing and debugging
- Using Kernel Independent Transport Layer (KITL) and Remote Tools to debug Compact 7 OS run-time on a target device

While it’s possible to use a Virtual PC as the target device and work through similar exercises, using a real hardware provides a practical environment, and helps improve the learning process. The eBox-3350MX, configured with the necessary software components, is used as the target device for the exercises in this guide.

To minimize unnecessary problem caused by missing software components and improper development environment setup, follow the recommended software installation sequences and setup provided as part of this getting started guide to install the necessary software.

The primary objective for this guide is to show how to use Compact 7 development tools. It’s not within this guide’s objective to teach programming languages and concepts. Short and simple exercises are used in this guide to keep the subject matter simple and easy to understand.

Update to this getting started guide and other Compact 7 resources are available from the following URLs:

http://www.embeddedpc.net/eBox3350MX/
http://www.embeddedpc.net/download/eBox3350MX.htm
eBox-3350MX Compact 7 Jump Start Kit

The eBox-3350MX Compact 7 jump start kit includes the following:

- eBox-3350MX, a 1.0 GHz compact computing device built with the following features:
  - 1.0 GHz Vortex86MX System-On-Chip
  - 512MB soldered on DDR2 system memory
  - VGA output support broad range of display resolution
  - HD Audio
  - SD flash storage interface
  - 10/100Mbps Ethernet
  - Three USB 2.0 Host interfaces

- SD flash storage
  The SD flash storage, includes with this kit, is preconfigured with bootloader to launch a prebuilt Compact 7 OS run-time image. In addition, flash storage also includes an Ethernet bootloader, needed to establish connectivity to the Platform Builder development station to download OS run-time image.

- One RJ-45 Ethernet Crossover cable
  With a crossover Ethernet cable and proper static IP address settings, the target device can be connected directly to the development station and create a standalone development environment with support to download Compact 7 OS run-time image and deploy application to the target device.

- eBox-3350MX-MSJK Windows Embedded Compact 7 jump start CD – This jump start CD includes the following:
  - eBox-3350MX Board-Support-Package for Compact 7
  - eBox-3350MX SDK for Compact 7
  - AutoLaunch component for Compact 7
  - CoreCon component for Compact 7
  - Sample project and codes for the exercises in this guide
Preconfigured Software

The eBox-3350MX (eBox) in this jump start kit includes a SD flash storage preconfigured with BIOSLoader (bootloader) to launch the evaluation Compact 7 OS image. When power on, the BIOSLoader display a Compact 7 splash screen, delay 10 seconds and wait for keyboard input. When input from the keyboard is not detected after the 10 seconds delay, the BIOSLoader launches the default Compact 7 OS run-time image, NK.BIN.

While the splash screen is showing, before the 10 seconds delay expires, you can perform the following:

- Press “1” to terminate the time delay and launch the Ethernet bootloader, EBOOT.BIN.
- Press “2” to terminate the time delay and launch an alternative evaluation OS run-time image, NK2.BIN.
- Do nothing, after the 10 seconds delay expires, the BIOSLoader launches the default evaluation OS run-time image, NK.BIN.

BIOSLoader

BIOSLoader is a Windows Embedded Compact bootloader for x86 device and is dependent on the x86 system BIOS to function. During boot up, the BIOSLoader can be configured to launch a Windows Embedded Compact OS run-time image or an Ethernet bootloader via the BOOT.INI configuration file.

EBOOT.BIN

EBOOT.BIN is an Ethernet bootloader. EBOOT.BIN can be launched by BIOSLoader or Loadcepc (DOS bootloader). When launched on a Windows Embedded Compact target device, this Ethernet bootloader acquires an IP address from an available DHCP server, broadcast a series of BOOTME messages to establish connectivity with a Platform Builder development station and download OS run-time image from the development station.

NK.BIN and NK2.BIN

These are Windows Embedded Compact evaluation OS run-time images. By default, a Windows Embedded Compact OS run-time image is generated with the NK.BIN file name. The NK2.BIN and NK3.BIN file names are modified from the default NK.BIN when it was generated.

- The NK.BIN OS image is generated from an OS design based on the Industrial Controller template with Hive-based registry enabled. The AutoLaunch, CoreCon and Remote display application components are included and configured to launch during startup.
- The NK2.BIN OS image is generated from the same OS design that generated the NK.BIN image above, without the AutoLaunch, CoreCon, Remote display application and Hive-based registry components.
Part 1 – Development Environment Overview

The Windows Embedded Compact 7 Platform Builder (Platform Builder) development tool is a plug-in to the Visual Studio 2008 integrated development environment (VS2008 IDE). All Compact 7 development tasks are supported within the VS2008 IDE, a developer friendly, efficient and intuitive development environment with templates and wizard to help simplify complicate development tasks.

Windows Embedded Compact 7

Windows Embedded Compact 7 is a hard Real-time operating system with ability to handle 32,000 concurrent processes and 2GB memory footprint for each process. Compact 7 delivers reliable, secure performance in a small footprint package along with the latest networking, multimedia and communications technologies. Compact 7 provides broad range of device support with enhanced features, including robust file system, Web services for device, Silverlight for Windows Embedded, voice over IP, network gateway configurations, platform development tool enhancements, greater application compatibility with other Windows Embedded Compact based devices, Internet Explorer, Windows Media CODECs, Microsoft .NET Compact Framework, and a number of other newly supported protocols and services.

Combining large pool of production quality BSPs, device drivers, programming libraries and effective development tools, Compact 7 provides an ideal rapid application development environment to help create the next generation of smart, media rich, connected and service oriented devices. For more information about Compact 7, visit:


Platform Builder

Platform Builder is the development tool used to create Compact 7 OS design projects, build custom Compact 7 OS run-time images and develop Compact 7 device drivers. It also provides the remote tools to debug Compact 7 OS run-time image, device drivers and software components on a target device.

Platform Builder for Compact 7 is a plug-in to the VS2008 IDE, and requires the VS2008 IDE to function.

Note:

The previous version, Platform Builder for Windows Embedded CE 6.0, is a plug-in to the Visual Studio 2005 IDE.
For Windows CE 5.0 and earlier versions, Platform Builder is a standalone development tool.

Windows Embedded Compact 7 Remote Tools

Using remote tools provided as part of the Compact 7 development environment, developer is able to remotely debug Compact 7 OS run-time image on a target device.

The following remote tools are provided as part of the Compact 7 development environment:

- File Viewer
- Kernel Tracker
- Performance Monitor
- Power Monitor
• Process Viewer
• Profiler
• Registry Editor
• Resource Consumer
• Resource Leak Detector
• System Information
• Timeline Viewer
• Zoom

Visual Studio 2008 Integrated Development Environment
Visual Studio 2008 is a popular developer friendly development tool to develop broad range of applications for different version of Microsoft Windows operating system. From the VS2008 IDE, developer is able to create applications for the following Windows operating system:

• Windows 7, Vista and XP
• Windows Embedded Standard 7, Windows Embedded Standard 2009 & XP Embedded
• Windows 2003 and 2008 Server
• Windows Phone 7, Windows Mobile Smartphones and Windows Embedded Handheld devices
• Windows Embedded CE 6.0 and Windows Embedded Compact 7 devices
• More...

The VS2008 IDE provides a centralized development environment to develop broad range of applications, such as:

• Windows Form application
• Win32, ATL and MFC project
• DLL, ActiveX control, ASP.NET Web service, Windows service
• SQL Server project, Console application, Class libraries
• Windows Embedded Compact and Mobile Smart Device application
• Windows Embedded Compact OS design
• More...

Develop Compact 7 Devices: Typical Development Steps
After the hardware platform is selected, a typical Compact 7 device development project goes through the following development phases:

• Develop Compact 7 bootloader, device drivers and BSP for the selected hardware (some hardware vendors provide bootloader, device driver and BSP to support their hardware)
• Create and customize an OS design project for the hardware
• Build customized Compact 7 OS run-time image for the hardware
• Deploy Compact 7 OS run-time image to the hardware for testing and debugging
• Generate Compact 7 SDK from the OS design to support application development
• Develop application for the Compact 7 device
• Application development can take place concurrently, as the hardware and OS design being fine-tuned
• After the satisfied Compact 7 OS run-time image and application are created, deploy the solution to the final hardware for distribution

**Develop Compact 7 Applications with VS2008**
Visual Studio 2008 supports application development for Compact 7 devices. There are multiple options to develop Compact 7 applications from the VS2008 IDE:

• Develop Native code application with Visual C++
• Develop Managed code application with Visual C#
• Develop Managed code application with Visual Basic

A Software Development Kit (SDK), generate from the OS design for the target device, is needed to support application development for the target device.

The Visual Studio IDE provides an efficient development environment, making it possible to download the OS run-time image and application from the development station to the target device for testing and debugging.

From the Visual Studio 2008 IDE with KITL and CoreCon connectivity, developer is able to download OS run-time image and application to a Compact 7 target device, launches the application, set breakpoint and step through the codes one-line-at-a-time as the application runs on the target device.

**Silverlight for Windows Embedded**
Silverlight is a user interface (UI) development framework designed to separate the graphical presentation from the programming logic. Silverlight enables graphic designer to design the graphical presentation for the application without the need to understanding the programming logic and provides a structured system for the application developer and graphic designer to work together, doing what they do best.

While the desktop and Windows Phone 7 Silverlight application’s code behind is based on managed code, Silverlight for Windows Embedded application’s code behind is based on native code, which provides an environment to develop small and efficient code to support embedded devices typically built with limited hardware resources.

Silverlight for Windows Embedded enables UI designers to develop stunning UI for the device without the need to understand complicated application code. At the same time, application developer can focus on their core expertise, develop the application logic and leverage UI graphic designer’s expertise to design user interface for the project.

Silverlight for Windows Embedded application for Compact 7 is developed using the combination of C++ and XAML code, using Visual Studio 2008 and Expression Blend 3.

**The Target Device**
Within the Compact 7 development environment, the term target device is referring to the hardware platform. For the exercises in this getting started guide, the eBox-3350MX (eBox) is used as the target device.
The ICOP_eBox3350MX_70C BSP which includes the necessary hardware adaptation codes, device drivers and configuration files to support Compact 7 OS development for the eBox, is used for the exercises in this guide to develop the OS design project, as shown in Figure 1.

Fig. 1 – ICOP_eBox3350MX_70C BSP for Compact 7
Common Terminology

To minimize the need to write long description and the need to use lengthy term repeatedly, it’s a common practice for developer in different industries to use abbreviated key words. To the new comer, without knowing the terminologies used and what the abbreviated key words represent can cause confusion.

The following table contains abbreviation for some of the common key words used in the Compact 7 development environment and this getting started guide:

<table>
<thead>
<tr>
<th>Abbreviation/ key word</th>
<th>Description and Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compact 7</td>
<td>Windows Embedded Compact 7</td>
</tr>
<tr>
<td>Platform Builder</td>
<td>Windows Embedded Compact 7 Platform Builder</td>
</tr>
<tr>
<td>VS2008</td>
<td>Visual Studio 2008</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package: A group of Windows Embedded Compact software components that includes the OEM adaptation layer code, device drivers and configuration files, needed to create the OS design project to generate OS run-time image for the targeted hardware.</td>
</tr>
<tr>
<td>OAL</td>
<td>OEM Adaptation Layer</td>
</tr>
<tr>
<td>Component</td>
<td>Component can be a device driver, BSP, programming library, application, utility, configuration settings, fonts, etc... The OS design is made up by a group of components.</td>
</tr>
<tr>
<td>Component Catalog</td>
<td>The component catalog lists all of the components provided by Platform Builder and installed 3rd party components. The component catalog provides the interface to add components to the OS design project and remove components from the project. This is also the interface to view which components are included to the OS design project.</td>
</tr>
<tr>
<td>OS design</td>
<td>OS design is a Platform Builder project, containing components that make up the OS run-time image. Windows Embedded Compact OS run-time image is generated from an OS design project.</td>
</tr>
<tr>
<td>Target Device</td>
<td>Refer to the device used in the Compact 7 development environment. When working on exercise using a VirtualPC, the VirtualPC is the target device. For the exercise in this guide, the eBox-3350MX is the target device.</td>
</tr>
<tr>
<td>OS Run-time Image</td>
<td>The binary file generated from the OS design project to deploy to the target device.</td>
</tr>
<tr>
<td>Release Directory or Build Release Directory</td>
<td>Referring to the directory where all files related to the OS design projects are placed by the build tools prior to compiling the OS run-time image.</td>
</tr>
</tbody>
</table>
Compact 7 Environment and Directory Variables

Within the Compact 7 development environment, environment variables are used to configure the OS design to include or exclude associated components. Environment variable is also used to control certain system behaviors. Directory variables representing the associated directory are used to help make the codes and script easier to understand and read. Here are some of these variables:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description - Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP_DISPLAY_NOP</td>
<td>Configure the OS design to include the stub display driver (DDI_NOP.dll) to support headless device.</td>
</tr>
<tr>
<td>BSP_NOAUDIO</td>
<td>Configure the OS design to exclude all audio components from the build.</td>
</tr>
<tr>
<td>IMGRAM256</td>
<td>Configure the OS design to generate run-time image supporting 256MB of RAM</td>
</tr>
<tr>
<td>PRJ_ENABLE_FSREGHIVE</td>
<td>Configure the OS design to enable Hive-based registry support</td>
</tr>
<tr>
<td>_WINCEROOT</td>
<td>Defines the root directory for the Compact 7 install, default is: C:\WINCE700</td>
</tr>
<tr>
<td>_PLATFORMROOT</td>
<td>Represent the following directory: $(_WINCEROOT)\PLATFORM Or C:\WINCE700\PLATFORM</td>
</tr>
</tbody>
</table>

The above table lists a small sampling of variables used in the Compact 7 development environment. For more information about Compact 7 environment variables, visit the following URLs:

About Environment Variables:

BSP Environment Variables:

IMG Environment Variables:

PRJ Environment Variables:

Miscellaneous Environment Variables:

OEM Environment Variables

In addition to the environment variables established by Microsoft, hardware manufacture can establish environment variables unique to their BSP to support the intended target device. The following environment variables are unique to the ICOP_eBox3350MX_70C BSP:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description - Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP_VORTEX86DX_IDE</td>
<td>Enable support for IDE storage device.</td>
</tr>
<tr>
<td>BSP_VORTEX86DX_9120A</td>
<td>Enable support for IDE device built with 9120A chip.</td>
</tr>
<tr>
<td>BSP_NIC_R6040</td>
<td>Enable support for R6040 Ethernet controller.</td>
</tr>
<tr>
<td>BSP_VDX_DisableAggressiveFlush</td>
<td>When enabled, it configures the OS design to disable aggressive registry flushing.</td>
</tr>
</tbody>
</table>
Part 2 – Development Environment Connectivity

To develop Compact 7 OS design and application, it’s important to establish a proper development environment and understand the different options available to establish connectivity between the development station and the target device. Connectivity needed to download OS run-time image from the development station to the eBox is different from the connectivity needed to download Visual Studio application to the eBox for testing and debugging.

**Note:**

Within the Platform Builder development environment, the term “target device” is used to identify the hardware platform in use. In the case for this getting started guide, the eBox is the target device.

Throughout this guide, the term “target device” is used to identify the eBox.

**Ethernet Connectivity**

For the exercises in this guide, Ethernet is the primary connectivity used in the development environment with both the development station and target device connected to the same Local Area Network (LAN).

Ethernet connectivity is used to download the OS run-time image from the development station to the target device and deploy application to the target device for testing and debugging.

**Develop Compact 7 OS Run-time Image – OS Design**

Compact 7 OS design is one of the project type supported by the VS2008 IDE, to generate custom OS run-time Image. As part of the OS design development process, connectivity between the development station and target device is needed to download OS run-time image to the target device. While it’s possible to use the serial port and other connectivity, Ethernet connection is efficient and provides the bandwidth to download Compact 7 OS run-time image quickly. Comparing to a serial connection, where the transfer rate is in the hundred kilo-byte (KB) range, Ethernet connection’s transfer rate is in the mega-byte (MB) range.

**Develop Compact 7 Applications with Visual Studio 2008**

The VS2008 IDE supports Compact 7 managed and native code applications development and provides the environment to deploy application from the development station to the target device. As part of the application development exercises in this guide, with CoreCon connectivity established over an Ethernet connection, the VS2008 IDE provides the facility to deploy application to the target device, set breakpoint and enables the developer to step through the code, line by line, as the application executes on the target device.

**Serial Debug Connectivity**

Serial Debug connectivity is established by connecting a null RS-232 serial modem cable between a serial port on the target device and an available serial port on the development station. Serial connection is useful for debugging headless device and the device’s startup process during the bootloader phase where the Compact 7 OS run-time has not been fully launched.

A terminal emulation program, such as Hyper-Terminal or similar application, running on the development station, is used to capture serial debug messages from the target device.
Development Environment Setup with DHCP

One of the common development environments to support Compact 7 development is to attach both the development station and target device to the same LAN with DHCP service to provide IP addresses dynamically with a null RS-232 serial modem cable connected between one of the serial port on the target device and an available serial port on the development station.

Here is a typical setup to connect both the development station and target device to a LAN with DHCP service, as shown in Figure 2.

![Diagram of development environment with DHCP service provider](image)

Fig. 2 - Development environment with DHCP service provider

If the target device fails to establish connectivity with the development station as expected with this configuration, it may be caused by one of the following:

- The development station’s firewall software may be blocking the connection
- DHCP service for the target device needs to be enabled on your network. Some secured network may require the target device’s MAC address to be added to the authorized device list to be serviced by the DHCP server

Using Wireless Access Point Router

When using a wireless-access-point-router with multiple Ethernet ports, connecting both the development station and target device directly to the Ethernet port on the wireless-access-point-router may be problematic with certain model of access point, and prevent the development environment from functioning as expected.

**Note:**

| The wireless-access-point-router device’s routing function filter and route network packets based on the packet’s associated origin, destination and other information. In some router, the routing algorithm may prevent some of the packets between the development station and target device from reaching its destination and cause problem. |

To minimize connectivity problem, instead of connecting directly to the wireless-access-point-router device’s Ethernet ports, attach an Ethernet network hub to the wireless-access-point-router device, to access the DHCP service provided by the wireless-access-point-router, connect both the development station and target device to the Ethernet network hub. With both the development station and target device connecting to the same Ethernet hub, the network traffics between them are not filtered.
Development Environment Setup with Static IP

It’s possible to establish a Compact 7 development environment with Static IP addresses. Here are two scenarios for setting up the development environment using static IP addresses:

**Local Area Network without DHCP Service**

The development station and target device are connected to the same LAN or Ethernet Hub, without DHCP service, as shown in Figure 3.

![Fig. 3 - LAN without DHCP](image)

**Direct Connection using Cross-Over RJ-45 Ethernet Cable**

Connectivity can be established by attaching the target device to the development station directly, using a cross-over RJ-45 Ethernet cable, as shown in Figure 4.

![Fig. 4 - Direct connection with cross-over RJ-45 Ethernet cable](image)

**Static IP Addresses**

Without DHCP service to assign IP addresses dynamically, the target device and development station must be configured with appropriate static IP addresses in order to establish connectivity. Both the development station and target device must be configured with static IP addresses within the same subnet.

Refer to appendix F for more information about using static IP addresses.
Part 3 – Required Software & Installation

The following software components are needed to work through the exercises in this getting started guide:

- Visual Studio 2008
- Visual Studio 2008 service pack 1
- Windows Embedded Compact 7
- Visual Studio 2008 update for Windows Embedded Compact 7
- ICOP_eBox3350MX_70C BSP
- eBox3350MX_SDK_Compact7.msi
- AutoLaunch_v300_Compact7.msi

Recommended Software Installation Sequence

It’s important to install the software in their proper sequences. Here is the recommended software installation sequence:

- Visual Studio 2008
- Visual Studio 2008 service pack 1
  Visual Studio 2008 service pack 1 is available from the following URL:
- Windows Embedded Compact 7
  A 180 days evaluation version of Compact 7 software is available for download from Microsoft, at the following URL:
- Visual Studio 2008 update for Windows Embedded Compact 7
  VS2008 update for Compact 7 is available for download from Microsoft, at the following URL:
- ICOP_eBox3350MX_70C_BSP.msi
  This BSP is provided on the jump start kit CD, in the \WinCE700 folder.
- eBox3350MX_SDK_Compact7.msi
  This SDK is provided on the jump start kit CD, in the \WinCE700 folder.
- AutoLaunch_v300_Compact7.msi
  This AutoLaunch component is provided on the jump start kit CD, in the \WinCE700 folder.

Windows Embedded Compact 7 Installation

Since the Compact 7 development tool, Platform Builder, is a plug-in to the VS2008 IDE, VS2008 must be installed to the develop workstation prior to installing the Compact 7 software.

When installing the Compact 7 software, during the installation options selection, select the *Custom install* option, as shown in figure 5.
During the supported processor selection step, include support for the x86 Architecture, as shown in Figure 6.

**Note:**
It’s strongly recommended to install all software to the default installation directory. Some of the 3rd party components used for the exercises in this guide assume all software components are installed to the default directory. When the software components are installed to a different directory, these 3rd party components may not function as expected.

**Board-Support-Package Installation**
VS2008 and Compact 7 must be installed prior to installing the Board-Support-Package (BSP). The **ICOP_eBox3350MX_70C BSP** is provided on the jump start CD, in the `\WinCE700` folder.

`WinCE700\ICOP_eBox3350MX_70C_BSP.msi`
After installation, this BSP shows up on the Platform Builder 2008 component catalog as “ICOP_eBox3350MX_70C : x86” under the “\Third Party\BSP” folder.

**SDK Installation**

VS2008 and Compact 7 must be installed prior to installing the SDK. A Compact 7 SDK is needed to support application development exercises in this guide. The `eBox3350MX_SDK_Compact7.msi` SDK is provided on the jump start CD, in the `\WinCE700` folder.

```
\WinCE700\eBox3350MX_SDK_Compact7.msi
```

**AutoLaunch Component Installation**

The AutoLaunch component for Compact 7 in self-installable file format, `AutoLaunch_v300_Compact7.msi`, is provided with the jump start kit CD. To install, locate and launch this component on the jump start CD, in the `\WinCE700` directory.

```
\WinCE700\AutoLaunch_v300_Compact7.msi
```

After installation, this component shows up on the Compact 7 Platform Builder component catalog as “AutoLaunch_v300_Compact7” under the “\Third Party\Embedded101” folder.

**CoreCon Connectivity Component**

In the previous version of the jump start kit, the CoreCon component was provided as a separate component. For the exercise in this getting started guide, the CoreCon component is a sub-component to the AutoLaunch component:

- AutoLaunch_v300_Compact7/AutoLaunch CoreCon

**Note:**

Files for the CoreCon component are installed to the development station as part of the Visual Studio software installation.

For 32-bit Windows 7, XP and Vista development station, the CoreCon component files are installed to the following directory:

```
\Program Files\Common Files\Microsoft Shared\CoreCon
```

For 64-bit Windows 7 development station, the CoreCon component files are installed to the following directory:

```
\Program Files (x86)\Common Files\Microsoft Shared\CoreCon
```

When the `AutoLaunch CoreCon` sub-component is included in the OS design, it contains the necessary command script and associated configuration parameters to include CoreCon files to the final OS run-time image.
Part 4 – OS Design (Platform Builder Project)

In this section, with help from the OS Design wizard, you will work through the steps to create a new OS design project.

Visual Studio 2008

The VS2008 IDE provides support to create different types of projects, such as Windows Application, Console Application, Class Library, smart device application, Windows Services, Web Control, etc... When starting a new project with VS2008, the VS2008 IDE provides different wizards and templates to help create the initial workspace for the project. Platform Builder is one of the available project types, to develop Compact 7 OS design project.

To begin, launch VS2008. From the VS2008 menu, select File → New → Project to bring up the new project screen, as shown in Figure 7.

Fig. 7 - VS2008 New Project – Platform Builder 2008 (OS design)

- From the New Project screen’s left pane, click to highlight the Platform Builder option.
- From the right pane, click to highlight the OS Design option.
- Enter eBox3350MX as the name for the project.
- Make sure the Create directory for solution check box is checked.
- Click OK to continue.

Windows Embedded Compact 7 OS Design Wizard

When a new Platform Builder OS design project is selected, the OS Design Wizard launches, as shown in Figure 8, and guides you through the process to configure the initial OS design project using templates available as part of the VS2008 IDE.
Fig. 8  -  OS Design Wizard

- Click **Next** to continue and bring up the *Create an OS Design - Board Support Packages* selection screen.

**OS Design Wizard – Board Support Package (BSP)**

In the BSP selection step, the *OS Design Wizard* provides the options to select one or more BSP for the new project from the list of available BSPs. All of the installed BSPs, including BSPs from Microsoft and third party companies are listed on the *Create an OS Design - Board Support Packages* screen, as shown in Figure 9.

Fig. 9  -  OS design Wizard – Select BSP

- From the *Select one or more BSPs* pane, select the **ICOP_eBox3350MX_70C: x86** BSP.
- Click **Next** to continue and bring up the *Create an OS Design - Design Templates* screen.
OS Design Wizard – Design Templates
In the design template selection step, the *OS Design Wizard* provides multiple design templates to choose from, as shown in Figure 10.

![Design Templates](image)

Fig. 10 - OS Design Wizard – Design Templates
- Expand the *Enterprise Device* node and select the **Industrial Controller** template.
- Click **Next** to continue and bring up the *Create an OS Design - Application & Media* screen.

OS Design Wizard – Applications & Media
In the applications & media selection step, the *OS Design Wizard* provides the options to select and include the .NET Compact Framework, Internet Explorer, Network User Interface and other components to the OS design, as shown in Figure 11.

![Applications and Media](image)

Fig. 11 - OS Design Wizard – Applications & Media
In addition to the .NET Compact Framework 3.5 component selected by default, include the following components to the OS design:

- Console Window
- Internet Explorer 7.0
- Network User Interface
- Waveform Audio
- Windows Internet Services

Click **Next** to continue and bring up the *Create an OS Design - Networking & Communication* screen.

**OS Design Wizard – Networking & Communications**

In the networking & communication selection step, the *OS Design Wizard* provides the options to select communication, networking and security components, as shown in Figure 12.

![OS Design Wizard – Networking & Communications](image)

Fig. 12 - OS Design Wizard – Networking & Communications

- For the networking & communication selection step, keep the default selection.
- Click **Next** to continue and bring up the *Create an OS Design - OS Design Project Wizard Complete* screen, as shown in Figure 13.

At this point, the *OS Design Wizard* included the necessary OS components needed to configure the initial OS design project.
Fig. 13 - OS Design Wizard – Completed

- Click Finish to complete the OS Design Wizard step.

**Security Warning - Catalog Item Notification**

At the completion of OS Design Wizard step, the Catalog Item Notification screen is raised, with security warning, to warn one or more of the components included in the OS design may pose security risk, as shown in Figure 14.

Fig. 14 - Catalog Item Notification – Security Warning

- Click Acknowledge to acknowledge the warning and close the warning screen.

At the completion of the OS Design Wizard step, the initial project workspace for the eBox3350MX OS design is created in the following directory:

- C:\WINCE700\OSDesigns\eBox3350MX\
Part 5 – Configure and Customize Compact 7 OS Design

At this point, the initial workspace for the eBox3350MX OS design project is created using the Industrial Controller design template along with the ICOP_eBox3350MX_70C BSP and components selected during the OS Design Wizard steps.

The following project folder and sub-folders are created for the OS design, under the main Compact 7 OS design project directory:

- C:\WINCE700\OSDesigns\eBox3350MX\n  This is the folder for the eBox3350MX solution. VS2008 supports different project types. A solution provides a centralized work space to keep different project types supporting the same solution in one location.
  For example, the eBox3350MX solution may include the “eBox3350MX OS design”, “Visual Basic managed code application”, “Visual C# managed code application” and “Visual C++ native code application”.

- C:\WINCE700\OSDesigns\eBox3350MX\eBox3350MX\n  This is the folder for the eBox3350MX OS design project.

The VS2008 IDE should look similar to the screen as shown in Figure 15.

![VS2008 IDE after OS Design Wizard](image)
Customize the OS Design – Add Additional Components

The OS design can be further customized with the following:

- Add component(s) to the OS design.
- Remove component(s) from the OS design.
- Add application and library as subproject to the OS design.
- Modify system configuration and registry files to customize system behavior.

With the eBox3350MX OS design project active, from the VS2008 menu, select View → Other Windows → Catalog Items View to bring up the Catalog Item View window, as shown in Figure 16.

![Catalog Items View](image)

Fig. 16 - Catalog Items View (Component Catalog)

The Catalog Item View window on the VS2008 IDE contains all of the Compact 7 components, including application, library, driver, utility and 3rd party components installed to the development station which can be added to the OS design project.

From the Catalog Items View window, expand the component nodes and review the components selected during OS Design Wizard steps:

- Component with a green check mark to the left indicates it was selected during the OS Design Wizard steps, or manually added after the OS Design Wizard steps.
- Component with a solid green square to the left indicates the component is included to the OS design as the result of being a dependency to another selected component.
- Component with a red cross to the left indicates the component is selected and is dependent on one or more other components currently not included to the OS design. (Component with a red cross to the left will not be included to the final image.)

Work through the following steps to customize the eBox3350MX OS design project:

- From the Catalog Items View window, expand the \Third Party\BSP node.
- Expand all nodes under the ICOP_eBox3350MX_70C_X86 node, as shown in Figure 17.
Verify the following BSP components are selected and included to the OS design:

- **ATAPI (IDE) Storage driver**
  
  **Note:**
  The BSP’s ATAPI (IDE) Storage component set the SYSGEN_ATAPI variable to include the ATAPI storage driver, and the SYSGEN_FATFS variable to include FAT file system support.

- **Hive-based registry support**
  
  **Note:**
  The BSP’s Hive-based registry component is needed to save registry settings to non-volatile flash storage between power reset. The Hive-based registry component is a sub-component under the ATAPI (IDE) Storage driver component.

- **HD Audio driver**
  
  **Note:**
  This component set the BSP_VORTEX86MX_HD_AUDIO and SYSGEN_AUDIO environment variables to include the audio driver.

- **Vortex86MX display driver**
  
  **Note:**
  Device driver to support the Vortex86MX integrated video controller.

- **VGA-01 640x480x16 @ 60Hz**
  
  **Note:**
  This is a sub-component to the Vortex86MX display driver. The display setting is selected arbitrary for this HOL guide. You can select a different display setting supported by the display monitor you are working with.
- **R6040 Ethernet driver**
  
  **Note:**
  
  Device driver to support the R6040 Ethernet controller, built-in to the Vortex86MX System-On-Chip.

- **PS/2 Keyboard & Mouse driver**
  
  **Note:**
  
  This BSP component set the BSP_KEYBD_8042 environment variable to include the PS/2 Keyboard and Mouse driver to the OS run-time image.

- **512MB RAM**
  
  **Note:**
  
  There are different models of eBox, built with 256MB, 512MB or 1024MB of system memory. This component set the IMGRAM512 environment variable and configure the OS run-time image to use the 512MB of available system memory. Improper system memory configuration can cause the Compact 7 OS run-time image not able to complete the boot up process and reset the device.

- **USB Keyboard & Mouse (HID) class driver**
  
  **Note:**
  
  This component set the SYSGEN_USB_HID_CLIENTS environment variable to include the USB Keyboard and Mouse HID class driver to the OS run-time image.

- **USB Mass Storage class driver**
  
  **Note:**
  
  This component set the SYSGEN_USB_STORAGE environment variable to include the USB storage class driver to the OS run-time image to support external USB storage.

Expand the `\Core OS\Windows Embedded Compact\Applications – End User` node, locate and include the following components to the OS design to support CAB file installation and application deployment from the Visual Studio 2008 IDE:

- **CAB File Installer/Uninstaller**
  
  **Note:**
  
  This component provides application installation & removal support. It’s needed to support VS2008 application debug and deployment.

Expand the `\Core OS\Windows Embedded Compact\Applications and Services Development` node, locate and include the following components to the OS design to support managed code application:

- **.NET Compact Framework 3.5**
- **OS Dependencies for .NET Compact Framework 3.5**
  
  **Note:**
  
  .NET Compact Framework components are needed to support managed code application. During the OS Design wizard steps, .NET Compact Framework 3.5 components are included to the OS design project. Expand the `\Core OS\Windows Embedded Compact\Applications and Services Development` node to validate these two .NET Compact Framework components are selected.
Expand the \Core OS\ Windows Embedded Compact\Communication Services and Networking\Servers node, select and include the FTP server component to the OS design:

- **FTP Server**

  **Note:**
  
  The FTP server provides a convenience mechanism to upload file(s) to the target device, and download file(s) from the target device, using an FTP client.

  In the later section, the following registry entries will be added to the project to enable the FTP server and enable support for anonymous login to the FTP server:

  ```
  [HKEY_LOCAL_MACHINE\COMM\FTPD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0
  "UserList"="@.*;"
  "AllowAnonymous"=dword:1
  "AllowAnonymousUpload"=dword:1
  "AllowAnonymousVroots"=dword:1
  "DefaultDir"="\\"
  ```

Expand the \Third Party\ Embedded101 node, select and include the following AutoLaunch components to the OS design, a utility to launch application during startup:

- **AutoLaunch_v300_Compact 7**

  **Note:**
  
  With the appropriate registry entries, the AutoLaunch utility can be configured to launch one or more application automatically when the Compact 7 OS starts.

  The following sample registry entries configure the AutoLaunch utility to launch App1.exe and App2.exe with 5 and 10 seconds delay respectively.

  ```
  [HKEY_LOCAL_MACHINE\Startup]
  "Process2"="App1.exe"
  "Process2Delay"=dword:00001388 ; delay 5 seconds before launching App1.exe
  "Process3"="App2.exe -c"
  "Process3Delay"=dword:00002710 ; delay 10 seconds before launching App2.exe
  ```

- **AutoLaunch_v300_Compact 7\Autolaunch CoreCon**

  **Note:**
  
  When this sub-component is added, it includes the command script to copy the necessary CoreCon connectivity files to the OS design’s build release directory, and include the following registries to launch CoreCon during startup:

  ```
  [HKEY_LOCAL_MACHINE\Startup]
  "Process0"="ConmanClient2.exe"
  "Process0Delay"=dword:00001388
  [HKEY_LOCAL_MACHINE\System]
  "CoreConOverrideSecurity"=dword:1
  ```

- **AutoLaunch_v300_Compact 7\Autolaunch Remote Display application**

  **Note:**
  
  When this sub-component is added to the OS design, it sets the SYSGEN_CERDISP variable to include the Remote Display application to the OS design and include the following registry entries to launch the Remote Display application during startup:

  ```
  [HKEY_LOCAL_MACHINE\Startup]
  "Process1"="cerdisp -c"
  "Process1Delay"=dword:00002710
  ```
Customize the OS Design – Locate Component by Search

The VS2008 IDE provides the search function to locate Compact 7 component from the catalog by searching the catalog using key-word or partial key-word associated with the component. In this section, you will use the search function to locate a component from the catalog and add the component to the OS design.

From the Catalog Item View tab, enter Telnet in the search text box on the top right, and click the green arrow to the right of the search text box to locate the Telnet Server component, as shown in Figure 18.

![Figure 18 - Locate component using the search function](image)

The search engine locates and highlights the Telnet Server component, as shown in Figure 19.

![Figure 19 - Locate the Telnet Server component using the search function](image)

- Select and include the Telnet Server component in the OS design.

**Note:**
The Telnet Server component provides a convenience mechanism to access and the compact 7 device remotely. When this component is added to the OS design, it does not include the necessary registry entries to launch the Telnet server.

In the later section, the following registry entries will be added to the project to enable the Telnet server and enable support for anonymous login:

```
[HKEY_LOCAL_MACHINE\COMM\TELNETD]
"IsEnabled"=dword:1
"UseAuthentication"=dword:0
```
Customize the OS Design – Configuration Manager

Using the configuration manager, the OS design can be configured to generate an OS run-time image in checked, debug or release mode. An OS image built in release mode is optimized for distribution, does not contain ASSERT and not able to output DEBUGMSG. An image built in debug mode is not optimized and include additional resource to support debugging. However, a debug mode image, without optimization, can be difficult to debug certain type of timing and performance related problems. An image built in Checked mode (Checked mode is new to Compact 7), a hybrid between the debug and release modes, is built with compiler optimization, contains ASSERT, DEBUGMSG and RETAILMSG to provide the necessary resources needed for debugging.

A debug mode image will provide more detailed debug messages when the OS image loads and executes applications and modules. The size of an OS image built in debug mode is generally about 50% larger than an OS image built in release mode, from the same OS design project. An OS image built in release mode, with KITL enabled, provides sufficient debug information to support most of the general application development environment. For the exercise in this guide, you will configure the OS design to generate an OS run-time image in release mode.

- From the VS2008 menu, select Build → Configuration Manager to bring up the Configuration Manager screen, as shown in Figure 20.

![Configuration Manager](image)

By default, the eBox3350MX OS design is configured to generate run-time image in release mode. The exercise in this section demonstrates the available options to configure the OS design to generate run-time image in checked, debug and release modes.

Customize the OS Design – Build Options

The OS design can be further customized by editing the project’s build options.

From the VS2008 menu, select Project → EBox3350MX Properties to bring up the eBox3350MX Property Pages screen, as shown in Figure 21.
Fig. 21 - OS design Property – Build Options

From the left pane, expand the **Configuration Properties** node and select **Build Options**.

From the right pane, change the following build option:

- **Set the Enable KITL build option to No**

  **Note:**
  When deploying a Compact 7 OS run-time image, with KITL enabled, to the target device’s local flash storage, the system will search for an unavailable KITL connection during start up and will not be able to complete the boot process.

Click **Apply** and **OK** to close the *eBox3350MX Property Pages* screen.

**Customize the OS Design – The Registry**

The registry plays a key role in controlling how the Compact 7 OS run-time behave, loads driver, application and more. Improper registry entries can cause serious problem and can cause the system failing to complete the boot process.

In the earlier steps, the FTP server and Telnet server components were included to the OS design. In order for these components to function as intended, appropriate registry entries need to be added to the OS design project.

**Registry Entries for the FTP Server Component**

By default, the FTP server is not configured to launch during startup. For the exercise in this guide, the following registry entries need to be added to the OS design and configure the FTP server to launch during startup, with user authentication disabled and allow anonymous login:

```
[HKEY_LOCAL_MACHINE\COMM\FTPD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0
  "UserList"="@;"
  "AllowAnonymous"=dword:1
  "AllowAnonymousUpload"=dword:1
  "AllowAnonymousVroots"=dword:1
  "DefaultDir"="\"
```
Registry Entries for the Telnet Server Component
By default, the Telnet server is not configured to launch during startup. For the exercise in this guide, the following registry entries are added to configure the Telnet server to launch during startup and allow anonymous login:

```
[HKEY_LOCAL_MACHINE\COMM\TELNETD]
"IsEnabled"=dword:1
"UseAuthentication"=dword:0
```

Registry Entries for the AutoLaunch Component
When the AutoLaunch component is added to the OS design, the following registry entries are added to the project as part of the AutoLaunch component, to launch the AutoLaunch.exe executable during startup:

```
[HKEY_LOCAL_MACHINE\Init]
"Launch99"="AutoLaunch.exe"
"Depend99"=hex:0a,00,14,00
```

When the AutoLaunch utility launches, it searches the following registry key and launch application with proper entries under this key:

```
[HKEY_LOCAL_MACHINE\Startup]
```

As part of the AutoLaunch_v300_Compact7 release, two sub-components are included which you can select and include CoreCon connectivity and Remote Display application components to the OS design. “Process0” and “Process1” under the HKLM\Startup key are used by these two components.

To configure the AutoLaunch utility to launch other application component, start with “Process2” as follow:

```
[HKEY_LOCAL_MACHINE\Startup]
"Process2"="App_01.exe"
"Process2Delay"=dword:3A98
"Process3"="App_02.exe"
"Process3Delay"=dword:4E20
```

Note:
The process delay entries above are randomly selected. You can configure the delay based on your need.

3A98 in HEX equal to 15000 in decimal, which represent 15000 ms or 15 seconds
You can configure the delay to control startup sequence between multiple applications.

Adding Registry Entries to the OS Design
Work through the following steps to add the necessary registry entries for the FTP Server and Telnet Server to the OS design:

- From the VS2008 menu, select View → Solution Explorer to bring up the Solution Explorer window.
- From the Solution Explorer window, expand the \Parameter Files node, and double click on OSDesign.reg to open this file in the code editor window.
- On the code editor window’s lower left, click on Source to view the OSDesign.reg registry file in source code format.
• Scroll to the end of the file and add the following entries to OSDesign.reg:

```plaintext
; Registry entries for the FTP server component
[HKEY_LOCAL_MACHINE\COMM\FTPD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0
  "UserList"="*;"
  "AllowAnonymous"=dword:1
  "AllowAnonymousUpload"=dword:1
  "AllowAnonymousVroots"=dword:1
  "DefaultDir"="\"

; Registry entries for the Telnet server component
[HKEY_LOCAL_MACHINE\COMM\TELNETD]
  "IsEnabled"=dword:1
  "UseAuthentication"=dword:0
```

**Static IP Address (Optional)**

By default, the OS design generates a Compact 7 OS run-time image with DHCP enable to request IP address dynamically. During the development process, it may be convenient to configure a Compact 7 OS run-time image preconfigured with static IP address, and provide a known IP to work with.

The OS design can be configured to generate an OS run-time image with preconfigured static IP address by adding the following registry entries to OSDesign.reg:

```plaintext
[HKEY_LOCAL_MACHINE\Comm\PCI\R60401\Parms\TcpIp]
  "EnableDHCP"=dword:0
  "DefaultGateway"=multi_sz:"192.168.2.1"
  "UseZeroBroadcast"=dword:0
  "IpAddress"=multi_sz:"192.168.2.232"
  "Subnetmask"=multi_sz:"255.255.255.0"

[HKEY_LOCAL_MACHINE\Comm\VMINI1\Parms\TcpIp]
  "EnableDHCP"=dword:0
  "IpAddress"=multi_sz:"192.168.2.233"
  "Subnetmask"=multi_sz:"255.255.255.0"
```

**Note:**

When the KITL build option is enabled, the Compact 7 OS run-time image is generated with the VMINI driver to support debugging, which is different from the driver for the R6040 Ethernet controller.

The registry entries under the HKEY_LOCAL_MACHINE\Comm\VMINI1\Parms\TcpIp key are needed to support the KITL connectivity.

Static IP is not required for the exercises in this guide.

***Important:*** Improper IP addresses assignment and firewall can prevent the development workstation to establish connectivity to the target device.

**Other Compact 7 Components**

In addition to the components selected during the *OS design Wizard* step and the components automatically included in the OS design by the template, additional components from the catalog can be added to the OS design to provide additional function and features.

For example, the File Server component can be added to provide files and folders sharing over a network connection. The FTP Server component can be added to provide remote file upload and download services. The RAS Server/PPTP Server (Incoming) component can be added to provide inbound dialup network connection via the serial port.
The following table lists the eBox’s I/O peripherals and their associated Compact 7 device drivers and support components:

<table>
<thead>
<tr>
<th>eBox Peripherals</th>
<th>Compact 7 Drivers &amp; Support Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vortex86MX display</td>
<td>eBox display driver (Driver included with BSP)</td>
</tr>
<tr>
<td>HD Audio</td>
<td>eBox audio driver (Driver included with BSP)</td>
</tr>
<tr>
<td>R6040 Ethernet Driver</td>
<td>eBox Ethernet driver (Driver included with BSP)</td>
</tr>
<tr>
<td>USB Ports (USB 2.0 Host)</td>
<td>USB host drivers (Driver available from Platform Builder’s component catalog)</td>
</tr>
<tr>
<td>SD*¹</td>
<td>ATAPI (Driver available from Platform Builder’s component catalog)</td>
</tr>
<tr>
<td>IDE</td>
<td>ATAPI (Driver available from Platform Builder’s component catalog)</td>
</tr>
</tbody>
</table>

*¹ The Micro-SD slot on eBox is link to the IDE interface, and does NOT support hot-swap. When a Micro-SD card is plugged into the slot prior to power on, the system will detect and recognize the card as an IDE storage device. When booting to Compact 7, the Micro-SD card will shows up as “Hard Disk”

The eBox used to develop the exercise for this guide is configured to boot from an SD storage card. The SD flash slot on the eBox is link to the IDE interface. The eBox’s SD slot does not support I/O module. SD storage card inserted to the SD slot on the eBox behaves just like a typical IDE hard disk and requires the ATAPI driver and FAT file system components to function.

To learn more about other Compact 7 components on the Platform Builder component catalog, refer to the Compact 7 help document on the development station, or the following MSDN URL:

Windows Embedded Compact Documentation
Part 6 – Build & Generate OS Run-time Image

In the previous sections, you created an OS design project, configured and customized the OS design. In this section, you will work through the steps to build and generate a custom Compact 7 OS run-time image from the OS design.

The Build Process – Starting

From the VS2008 menu, with the eBox3350MX OS design project active, select Build → Advanced Build Commands → Sysgen (blddemo -q) to build and generate an OS run-time image from the OS design.

Depending on the development station’s processor speed and available system resources, the build process may take anywhere from 10 to well over 30 minutes.

During the build process, the output tab on the VS2008 IDE displays compilation activities, as shown in Figure 22.

The Build Process – Completed

When the build process is completed, the output tab on the VS2008 IDE displays build succeeded message, as shown in Figure 23.

![Fig. 22 - VS2008 IDE showing the OS design being built.]

![Fig. 23 - VS2008 IDE – Build completed]

The Output tab on the VS2008 IDE shows the result of the build process, with “0” failed task.

**Note:**

When the build process ended with 1 or more error, the build process failed and will not generate a Compact 7 OS run-time image file. When the build process ended with warnings and without any error, the build process is completed with a Compact 7 OS run-time image file generated. The number of warning may vary depending on the selected components and installed QFE. When working with a known good BSP, the warnings are generally non critical and do not impact system function. However, as part of good engineering practice, it’s good to go through the warning messages to identify potential problem.
With a successful build, a Compact 7 OS run-time image file, NK.BIN, is generated in the following build release directories:

- For OS design configured to generate an image in checked mode:
  \WINCE700\OSDesigns\EBox3350MX\EBox3350MX\RelDir\ICOP_eBox3350MX_70C_x86_Checked

- For OS design configured to generate an image in debug mode:
  \WINCE700\OSDesigns\EBox3350MX\EBox3350MX\RelDir\ICOP_eBox3350MX_70C_x86_Debug

- For OS design configured to generate an image in release mode:
  \WINCE700\OSDesigns\EBox3350MX\EBox3350MX\RelDir\ICOP_eBox3350MX_70C_x86_Release

Searching the above directories, with a successful build, there should be a NK.BIN file, which is the Compact 7 OS run-time image file. After the build process is completed, Platform Builder generates the following log files for the last build process:

- C:\WINCE700\Build.log (This file list all of the build steps in detail.)
- C:\WINCE700\Build.wrn (This file list all of the warnings during the build process.)
- C:\WINCE700\Build.err (This file list all the errors during the build process. This file is not present for a successful build.)
Part 7 – Download Compact 7 OS Run-time to Target Device

To download the run-time image to the target device, you need to establish connectivity between the VS2008 development station and target device.

In this section, you will work through the steps to establish connectivity and download the Compact 7 OS run-time image created in the earlier steps to the target device.

A target device built with different type of processor and I/O peripherals may require different transport mechanism to establish the connectivity. The eBox-3350MX is built with an x86 processor (Vortex86MX) and will use the integrated Ethernet to establish connectivity to the VS2008 development station.

To work through the exercises in this guide, both the development station and target device must be connected to the same LAN, using one of the following connectivity options:

1. Both the development station and target device are connected to the same LAN with DHCP service to provide IP addresses dynamically.

2. Both the development station and target device are connected to the same LAN without DHCP service and need to configure static IP addresses.

3. The target device is connected to the development station directly using a cross-over Ethernet cable and need to configure static IP addresses.

**Note:** Refer to appendix A and B for more information about connectivity for downloading OS run-time image from the development station to target device.

The exercises in this guide are created with both the development station and target device connected to the same LAN with DHCP service to provide IP addresses dynamically.

Target Device Preparation

The target device needs to be configured with an appropriate bootloader to establish connectivity and download Compact 7 OS run-time image from the development station.

The target device provided as part of the jump start kit is shipped with an SD flash configured with a bootloader, BIOSLoader, to launch the preconfigured software, as follow:

- **NK.BIN**
  This is a Compact 7 OS run-time image generated from the eBox3350MX OS design exercise in this guide, with AutoLaunch component configured to launch the included CoreCon connectivity component to support application development using VS2008.

- **NK2.BIN**
  This is a generic Compact 7 OS run-time image without the AutoLaunch, CoreCon and Remote display application components.

- **EBOOT.BIN**
  This is the Ethernet bootloader for the target device, needed to establish connectivity with the development station to download OS run-time image. The EBOOT.BIN Ethernet bootloader needs to be launched by another bootloader, such as Loadcepc or BIOSLoader.

After power up, the target device display the Compact 7 splash screen for about 10 seconds before launching the default OS run-time image, NK.BIN.
While the splash screen is showing, you can press “1” or “2” before the 10 seconds delay timed-out to perform the following:

- Press “1” to launch EBOOT.BIN to establish connectivity to the Platform Builder development station and download OS run-time image from an active OS design.
- Press “2” to launch the alternative OS run-time image, NK2.BIN.

Otherwise, after the 10 seconds delay timed-out, the bootloader launches the default image, NK.BIN.

Create Connectivity Profile for a Target Device

To establish connectivity and download Compact 7 OS run-time image from the development station to the target device, you need to configure a target device connectivity profile and associate the profile to the target device in use.

In this section you will work through the steps to create a target device connectivity profile and associate this profile with the eBox, as the target device for the exercise in this guide.

Note:
The exercise in this section is applicable to development environment using dynamic IP addresses with DHCP, and development environment using static IP addresses.

Continue from the previous exercise, with the eBox3350MX OS design project active, from the VS2008 menu, select Target → Connectivity Options to bring up the Target Device Connectivity Options screen, as shown in Figure 24.

![Target Device Connectivity Options](image)

Fig. 24 - Target Device Connectivity Options

Work through the following steps to create a connectivity profile for the target device (eBox):

- From the Target Device Connectivity Options screen, click Add Device to bring up the screen to add a new target device profile as shown in Figure 25.
Fig. 25 - Target Device Connectivity Options – Add new device

- Enter **MyTargetDevice** as the new target device name
- Click **Add** to continue

The **Target Device Connectivity Options** screen displays the new device profile, **MyTargetDevice**, as shown in Figure 26.

Fig. 26 - Target Device Connectivity Options – MyTargetDevice created

- Select **Ethernet** for the Download option.
- Select **Ethernet** for the Transport option.
- Select **KdStub** for the Debugger option.
- Click **Apply** to continue and save the settings.

**Associate eBox to MyTargetDevice Connectivity Profile**

In this section, you will work through the steps to associate the eBox to the **MyTargetDevice** connectivity profile, created in the previous section.
From the *Target Device Connectivity Options* screen (continuing from previous section), clicks on the top most *Settings* button, to bring up the *Ethernet Download Settings* screen, as shown in Figure 27.

![Ethernet Download Settings screen](image)

**Fig. 27 - Ethernet Download Settings**

The *Ethernet Download Settings* screen is waiting for BOOTME messages from the target device’s Ethernet bootloader (*EBOOT.BIN*).

Work through the following steps to associate the eBox to this device profile:

- Apply power to the target device.
- As the target device power up and display the Windows Embedded Compact 7 splash screen, press “1” to launch the Ethernet bootloader, *EBOOT.BIN*.
- As the *EBOOT.BIN* Ethernet bootloader is launched, it broadcasts a series of BOOTME messages to the attached LAN via UDP.
- As the *Ethernet Download Settings* screen detected the BOOTME messages from the target device, it displays the device ID from the target device, as shown in Figure 28.
Click and highlight the device ID listed in the **Active target devices** windows to associate the device ID to the profile.

**Click Apply** to continue.

**Note:**
In the environment where there are multiple target devices connected to the same network segment booting up and send bootme requests at about the same time, multiple device IDs will be listed in the Active Devices windows. To identify the target device you are working with, make sure it’s the only target device booting and sending the bootme request.

After the device ID is detected and accepted in the **Active target devices** step, the same device ID is now shown on the **Target Device Connectivity Options** screen, as shown in Figure 29.

Click **Apply** and **Close** to save the settings and close the **Target Device Connectivity Options** screen.
Download Compact 7 OS Run-time Image to eBox

With the *MyTargetDevice* connectivity profile created and the eBox target device associated with this profile, you are ready to download the Compact 7 OS run-time image from the Development station to the target device.

Work through the following steps to download the OS run-time image to the target device:

- From the VS2008 IDE, select and use *MyTargetDevice* device profile as shown in Figure 30.

  ![Fig. 30 - select MyTargetDevice from VS2008 IDE](image)

- From the VS2008 menu, select **Target → Attach Device** to initiate the download process and bring up the **Device Status** screen, as shown in Figure 31.

  ![Fig. 31 - Waiting for BOOTME request from target device](image)

- Power on (or reset power) the target device.
- As the Compact 7 splash screen is showing, press “1” to launch the Ethernet bootloader, *EBOOT.BIN*, before the 10 seconds delay is timed out.
- As the *EBOOT.BIN* launches, it sends a series of BOOTME messages to the development station to trigger the OS run-time image download process.
- As the image download takes place, the **Device Status** screen displays download activities, as shown in Figure 32.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Once launched, the Ethernet bootloader, <em>EBOOT.BIN</em>, will continue to broadcast BOOTME messages to the development station and timed out after about 3 minutes. If you failed to initiate the OS run-time image before the timed out period, you need to re-initiate the download process again.</td>
</tr>
</tbody>
</table>


After the image download process is completed, the Compact 7 OS run-time image downloaded to the target device will launch. Be patient... It takes a few moments for the image to boot up.

When the Windows Embedded Compact 7 screen is displayed on the target device, it’s an indication the boot process is completed, as shown in Figure 33.

**Deploy Compact 7 OS Run-time Image to Target Device**

In the previous step, when the Compact 7 OS image is downloaded and launched on the target device, the image is placed directly to the device’s memory (RAM) and is not saved to the device’s local storage. After going through power reset, the downloaded image will be lost. To deploy a Compact 7 OS run-time image to the target device’s local storage, the OS run-time image file, *NK.BIN*, needs to be copied to the local storage and configured with an appropriate bootloader.

The *NK.BIN* OS run-time image file, in release mode, is generated in the following directory:

```
C:\WINCE700\OSDesigns\EBox3350MX\EBox3350MX\Rel\ICOP_EBox3350MX_70C_x86_Release
```
Note:

Release mode image built with KITL enabled and debug image is not intended for deployment to the target device’s local storage.
When deploying a release mode OS run-time image, generated with KITL enabled, to the target device’s local storage, the image attempt to establish connectivity to an unavailable KITL connection and will not be able to complete the boot process.

There are multiple methods to deploy the NK.bin OS run-time image file to the target device’s local storage.

To deploy a Compact 7 OS run-time image to the target device’s local storage, the OS design must be configured with the following:

- Generate OS run-time image in release mode
- Disable the KITL build option and other build option configured to support debugging

Refer to appendix G for more information about deploying a Compact 7 OS run-time image to the target device’s local storage using DiskPrep, a Windows Embedded Compact power toy.

The DiskPrep power toy is available for download from the following URL:

http://code.msdn.microsoft.com/DiskPrep
Part 8 – Software Development Kit (SDK)

In this section, you will work through the steps to create and configure an SDK from the OS design project. An SDK is needed to support Compact 7 application development for the target device.

Create and Configure Compact 7 SDK

With the eBox3350MX OS design project active, from the VS2008 menu, select Project → Add New SDK to bring up the SDK Property Pages screen, as shown in Figure 34.

Fig. 34 - New SDK Property Page

- Enter eBox3350M_X_SDK_Compact7 as SDK name
- Enter Windows Embedded Compact 7 SDK for eBox-3350MX as product name
- Fill in the company name and company website information.
- From the left pane, click on the Install node and enter eBox3350MX_SDK_Compact7.msi as the file name on the right, as shown in Figure 35.

Fig. 35 - SDK Property Page
On the left pane, click on the Development Languages node to bring up the development languages configuration, and enable support for both the Native and Managed development, as shown in Figure 36.

![SDK Property Page](image)

**Fig. 36 - SDK Property Page**

Click **Apply** follow by **OK** to complete configuring the new SDK.

To make changes to the SDK before compiling, from the VS2008 IDE’s Solution Explorer tab, expand the \SDKs folder, right click on **eBox3350MX_SDK_Compact7** and select **Properties** to bring up the SDK Properties Pages, as shown in Figure 37.

![VS2008 IDE – Edit SDK](image)

**Fig. 37 - VS2008 IDE – Edit SDK**
Build and Generate Compact 7 SDK

With the eBox3350MX_SDK_Compact7 SDK created, you can build and generate the SDK.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The OS design project must be built prior to launching the SDK build step. The SDK build process requires library files generated during the OS design build process.</td>
</tr>
</tbody>
</table>

From VS2008 menu, select Build → Build All SDKs to build and generate the installation package for the SDK. After the build process is completed, the eBox3350MX_SDK_Compact7.msi installation package is generated in the following folder:

\WINCE700\OSDesigns\eBox3350MX\eBox3350MX\SDKs\SDK1\MSI

After the eBox3350MX_SDK_Compact7.msi installation package is generated, install this SDK to the development station to support Compact 7 application development for the target device.
Part 9 – Develop Native Code App with VS2008

In this section, you will work through the steps to develop a native code application for Compact 7 using Visual C++ 2008. The following are needed to work through the exercise:

- A target device with Compact 7 OS run-time from the earlier exercise launched.
- eBox3350MX_SDK_Compact7.msi SDK from the earlier exercise installed to the development station.

Create a New Native Code Application Project

From the VS2008 menu, select File → New → Project to bring up the New Project screen, as shown in Figure 38.

Fig. 38  -  New project wizard

- On the New Project screen’s left pane, expand the Visual C++ node and select Smart Device as the project type. On the right pane, select Win32 Smart Device Project.
- Enter VS2008_Win32_HelloWorld as the project name.
- Enter C:\Lab as the location for the project.

**Note:** You can choose a different folder for the project other than C:\Lab.

- Click OK to continue and bring up the Win32 Smart Device Project Wizard screen, as shown in Figure 39.
Fig. 39 - Win32 smart device project wizard

- Click **Next** to continue and bring up the **Platforms** selection step, as shown in Figure 40.

Fig. 40 - Win32 smart device project wizard | SDK selection

- From the **Selected SDKs** pane on the right, click to highlight the **Windows Mobile 5.0 Pocket PC SDK** entry.
- Click the **single left pointing arrow**, located to the left of the **Selected SDKs** pane to remove the **Windows Mobile 5.0 Pocket PC SDK** entry.
- From the **Installed SDKs** pane on left, click to highlight the **eBox3350MX_SDK_Compact7** entry.
• Click on the **right pointing arrow**, located to the right of the *Installed SDKs* pane, to add the **eBox3350MX_SDK_Compact7** SDK to the *selected SDKs* pane, as shown in Figure 41.

![Fig. 41 - Win32 smart device project wizard | SDK selection](image)

• After the **eBox3350MX_SDK_Compact7** SDK is added to the *Selected SDKs* pane, click **Next** to continue and bring up the *Project Settings* step, as shown in Figure 42.

![Fig. 42 - Win32 smart device project wizard | Project settings](image)

• Keep the default selection to create a **Windows application**.

• Click **Finish** to complete the wizard and generate the initial files for the project.
After the new project wizard step is completed, the initial VS2008_Win32_HelloWorld project workspace is created in the following folder:

C:\Lab\VS2008_Win32_HelloWorld

Add Codes and Build the Application Project
With the VS2008_Win32_HelloWorld project active, the VS2008 IDE should look similar to the IDE as shown in Figure 43.

Work through the following steps to add codes to the VS2008_Win32_HelloWorld project:

- From the VS2008 IDE’s Solution Explorer tab, double click on the VS2008_Win32_HelloWorld.cpp file, in the |Source Files| folder to view and edit the code in the code editor window.
- Replace the codes in the “case WM_PAINT:” statement section with the following codes, as shown in Figure 44.

```cpp
Case WM_PAINT:
    //
    RECT rect;
    GetClientRect (hWnd, &rect);
    hdc = BeginPaint(hWnd, &ps);
    DrawText(hdc, TEXT("Hello World Win32 Native Application!")), -1, &rect,
    DT_CENTER|DT_VCENTER|DT_SINGLELINE);
    EndPaint(hWnd, &ps);
    break;
```
Fig. 44 - VS2008 IDE with VS2008_Win32_HelloWorld project active

From the VS2008 menu, select **Build → Build Solution** to build the **VS2008_Win32_HelloWorld** project.

**Preparing Target Device to Connect to VS2008 IDE**

To download the **VS2008_Win32_HelloWorld** application to the Compact 7 target device, you need to establish connectivity between the target device and VS2008 IDE using CoreCon.

To establish CoreCon connectivity, CoreCon files from the following directory on the VS2008 development station must be included in the Compact 7 device’s run-time image, or made accessible from the Compact 7 device’s file system:

\Program Files\Common Files\Microsoft Shared\Corecon\1.0\Target\wce400

There are multiple sub-folders, with names corresponding to the CPU architectures, under the above directory. Each of these folders contains CoreCon component files to support the corresponding CPU family. These CoreCon files are installed to the development station during the Visual Studio 2008 installation process.

**Note:**
The eBox is built with x86 CPU architecture. The CoreCon files needed to support the eBox are in the “\x86” sub folder.

As part of the exercise in the earlier steps, the **AutoLaunch_v300_Compact7** component and **Autolaunch CoreCon** sub-component are included in the OS design and compiled as part of the OS run-time image. The AutoLaunch component is configured to launch the CoreCon executable during startup.

If the CoreCon files are not included as part of the OS run-time image, you need to manually copy the necessary CoreCon files to the Compact 7 target device’s \Windows directory and manually launch the **ConmanClient2.exe** and **CMaccept.exe** executable.
Since the Compact 7 OS run-time image launched on the target device is built with the AutoLaunch and Autolaunch CoreCon components, the target device is configured with CoreCon launched and is ready to establish connectivity to the VS2008 IDE.

In order to establish CoreCon connectivity, you need to know the Compact 7 target device’s IP address. Work through the following steps to find out the IP address:

1. With Compact 7 running on the target device, from the Compact 7 desktop, click on Start → Run and launch the CMD command to launch the command console window, as shown in Figure 45.

![Fig. 45 - Compact 7 desktop on target device](image)

2. Click OK to continue and launch the command console window.

3. From the command console window, launch the IPCConfig command to retrieve the target device’s IP address information, as shown in Figure 46.

![Fig. 46 - Compact 7 desktop on target device](image)

Alternatively, you can double click on the network icon on the Compact 7 device desktop’s taskbar to bring up the network controller information screen which provides IP address information, as shown in Figure 47.
The target device’s IP address is needed in the next step. Record the IP address.

**Configure VS2008 Device Connectivity Settings**

To establish connectivity to the Compact 7 target device, you need to configure the device connectivity settings for the VS2008 development station with the target device’s IP address.

Work through the following steps to configure the device connectivity settings:

- From the VS2008 menu, select **Tools → Options** to bring up the **Options** screen, as shown in Figure 48.

- Expand the **Device Tools** node on the left pane and click to highlight the **Devices** node.

- On the right pane, click to select **eBox3350MX_SDK_Compact7 x86 Device**, as shown in Figure 49.
Fig. 49 - VS2008 Options

- Click **Properties** to bring up the *eBox3350MX_SDK_Compact7 x86 Device Properties* screen, as shown in Figure 50.

Fig. 50 - *eBox3350MX_SDK_Compact7 x86 Device Properties*

- From the *eBox3350MX_SDK_Compact7 x86 Device Properties* screen, click **Configure** to bring up the **Configure TCP/IP Transport** screen, as shown in Figure 51.

Fig. 51 - **Configure TCP/IP Transport**
- From the Configure TCP/IP Transport screen, select the **Use specific IP address** option, and enter the target device’s **IP address** from the earlier step.
- Click **OK** to continue and close the Configure TCP/IP Transport screen.
- Click **OK** to continue and close the eBox3350MX_SDK_Compact7 x86 Device Properties screen.
- Click **OK** to continue and close the **Options** screen.

### Establish Connectivity to the Target Device

To initiate connectivity to the target device, select **Tools → Connect to Device** from the VS2008 menu to bring up the **Connect to Device** screen, as shown in Figure 52.

![Connect to Device](image)

Fig. 52 - Connect to Device

- If it’s not already selected, select **eBox3350MX_SDK_Compact7** from the Platform selection.
- Click **Connect** to establish connectivity, and bring up the **Connecting** screen.
- Upon a successful connection, the **Connecting** screen displays the **Connection succeeded** message, as shown in Figure 53.

![Connecting](image)

Fig. 53 - Connection succeeded

- Click **Close** to close the **Connecting** screen and continue.
Download Application to Target Device

With CoreCon connectivity established between the VS2008 IDE and the target device, you can download the **VS2008_Win32_HelloWorld** application to the target device.

- From the VS2008 menu, select **Debug → Start Debugging** to initiate the process to download the application to the device.

- As the **VS2008_Win32_HelloWorld** application is downloaded and launched on the target device, the application screen is shown on the target device’s desktop, as shown in Figure 54.

![Fig. 54 - VS2008_Win32_HelloWorld running on the target device](image)

This concludes the exercise to create a native code application for Compact 7, from the VS2008 IDE and download the application to the target device.
Part 10 – Develop Managed Code App with VS2008

In this section, you will work through the steps to develop a managed code application for Compact 7 using Visual C#, from the VS2008 IDE.

Create a New Managed Code Application Project

From the VS2008 menu, select File → New → Project to bring up the New Project screen, as shown in Figure 55.

![New Project Wizard](image)

Fig. 55 - New project wizard

- On the left pane, expand the Other Languages\Visual C# node and select Smart Device as the project type.
- On the right pane, select Smart Device Project.
- Enter VS2008_CSharp_HelloWorld as the project name.
- Enter C:\Lab as the location for the project.

**Note:**
You can choose a different folder for the project other than C:\Lab.

- Click OK to continue and bring up the Add New Smart Device Project screen.
- From the Add New Smart Device Project screen, select eBox3350MX_SDK_Compact7 from the Target platform selection.
- From the .NET Compact Framework version selection, select .NET Compact Framework Version 3.5.
- From the Templates selection pane, select Device Application, as shown in Figure 56.
Fig. 56 - Add New Smart Device Project

- Click OK to complete the step to create a new smart device project.

After the new project wizard step is completed, the initial VS2008_CSharp_HelloWorld project workspace is created in the following folder:

C:\Lab\VS2008_CSharp_HelloWorld

The initial application project is created with a blank form, Form1.cs, and a blank module, Program.cs, as shown in Figure 57.
Add Codes and Build the Application Project

In the following steps, you will make changes and add codes to the project:

- Resize Form1 to a smaller size or to fit the target device’s display resolution. For this exercise, resize Form1 to 320 x 240.
- Change Form1’s caption to “VS2008 Managed Code Hello World”.
- Delete the mainMenu1 component from the project.
- Add a text box to Form1, change the name to textHelloWorld, clear the default content and place textHelloWorld at the center of Form1.
- Add a button control to Form1, change the name to buttonHelloWorld, change the caption to “Hello World” and place buttonHelloWorld to the center of Form1, just below textHelloWorld.
- Double click on buttonHelloWorld (Hello World button) to bring up the source code window showing the buttonHelloWorld_Click event handler.
- Add the following codes to the buttonHelloWorld_Click event handler:

```csharp
private void buttonHelloWorld_Click(object sender, EventArgs e)
{
    textHelloWorld.Text = "Hello World!";
    textHelloWorld.Text = "2nd Hello World!";
    textHelloWorld.Text = "3rd Hello World!";
    textHelloWorld.Text = "Final Hello World!";
}
```

The VS2008 IDE should look similar to the following, as shown in Figure 58.

Fig. 57 - Visual Studio 2008 IDE with new project
Preparing Target Device to Connect to VS2008 IDE

The VS2008 Managed code development environment uses CoreCon to establish connectivity between the VS2008 IDE and the target device, similar to the CoreCon used for the Native code application exercise in Part-9 of this guide. Follow and use the same steps from the native code application section to configure the VS2008 device connectivity settings.

Note:
If you are continuing from the previous VS2008 native code exercise, using the same instance of Compact 7 device, the device connectivity settings are already configured.

Establish Connectivity to Compact 7 Target Device

To initiate connectivity to the target device, from the VS2008 menu, select Tools → Connect to Device to bring up the Connect to Device screen, as shown in Figure 59.
If it’s not already selected, select eBox3350MX_SDK_Compact7 from the Platform selection. Click Connect to establish connectivity, and bring up the Connecting screen. Upon a successful connection, the Connecting screen displays the Connection succeeded message, as shown in Figure 60.

Click Close to close the Connecting screen and continue.

Download Application to Target Device

With CoreCon connectivity established between the VS2008 IDE and the target device, work through the following steps to download the VS2008_CSharp_HelloWorld application to the device:

- From the VS2008 menu, select Debug → Start Debugging to bring up the Deploy VS2008_CSharp_HelloWorld screen, as shown in Figure 61.
Fig. 61 - Deploy the VS2008_CSharp_HelloWorld application

- From the Deploy VS2008_CSharp_HelloWorld screen, select eBox3350MX_SDK_Compact7 x86 Device and click Deploy to download the application to the target device.
- As you click Deploy to download the application, the Microsoft Visual Studio screen is raised to indicate there are deployment errors, as shown in Figure 62.

Fig. 62 - Application deployment error

- Click Yes to continue and download the application to the target device.
- As the VS2008_CSharp_HelloWorld application is downloaded and executed on the target device, the application screen is shown on the device’s desktop, as shown in Figure 63.
In the earlier step, there was an error message during the process to download the application to the target device, as shown in Figure 62. Work through the following steps to change the VS2008_CSharp_HelloWorld project’s configuration to eliminate this error message:

- From the VS2008 menu, select **Project → VS2008_CSharp_HelloWorld Properties** to bring up the **VS2008_CSharp_HelloWorld properties** screen, as shown in Figure 64.
• Select the **Devices** tab from the **VS2008_CSharp_HelloWorld properties** screen’s left pane and uncheck the **Deploy the latest version of .NET Compact Framework** option on the right.

With the above configuration, you will not encounter any more error message when downloading the **VS2008_CSharp_HelloWorld** application to the target device.

### Debug Application Running on Target Device

In this section, you will work through the steps to set break point and step through the application’s codes, one line at a time, as the application executes on the target device.

There are two options to download and launch the application to the target device.

- From the VS2008 menu, select **Debug → Start Debugging**.
- From the VS2008 menu, select **Debug → Start Without Debugging**.

To work through the exercise in this section, the application must be downloaded to the target device by selecting the **Debug → Start Debugging** option.

With the **VS2008_CSharp_HelloWorld** application downloaded to the target device, work through the following steps to set a breakpoint in the **buttonHelloWorld_Click** event handler:

- From the VS2008 IDE, switch to source code viewing mode and navigate to the **buttonHelloWorld_Click** event handler.
- Click on the following line of code and press the **F9** key to set a breakpoint, as shown in Figure 65.

```csharp
textHelloWorld.Text = "2nd Hello World!";
```

![VS2008 IDE | Application breakpoint](image)

Fig. 65 - VS2008 IDE | Application breakpoint
Note:
The above breakpoint is set while the application is running on the target device.

With the breakpoint in place, work through the following steps to execute the VS2008_CSharp_HelloWorld application, and cause the application to halt when the execution reaches the breakpoint:

- From the target device’s Compact 7 desktop, with the VS2008_CSharp_HelloWorld application running, click the Hello World button to execute the codes in the buttonHelloWorld_Click event handler.

- As the code in the event buttonHelloWorld_Click executes, it halt at the breakpoint you set in the earlier step.

- On the development station’s VS2008 IDE, the breakpoint highlight color changed from Red to yellow indicating the program is halt at this line of code, as shown in Figure 66.

![Program execution halt at the breakpoint](image)

- The application running on the target device is halt on the following line of code:
  ```csharp
  textHelloWorld.Text = "2nd Hello World!";
  ```

- As the code execution is halt on the above line of code, this line of code has not been executed. The textbox on the VS2008_CSharp_HelloWorld application screen, running on the target device, is showing the “Hello World!” message, indicates the line of code just before the current breakpoint has been executed, as shown in Figure 67.
Fig. 67 - Compact 7 application halt at the breakpoint

- From the VS2008 IDE, press the F11 key to step through one line of code.
- As the F11 key is pressed, the application executes one line of code. The next line of code becomes highlighted with yellow, indicates the program is halt on this line of code. The textbox message on the VS2008_CSharp_HelloWorld application is changed to “2nd Hello World!” as shown in Figure 68.

Fig. 68 - Compact 7 application halt at the breakpoint

- The F11 key is used to step through the code one line at a time. To continue the code execution, press the F5 key from the VS2008 IDE.

From the VS2008_CSharp_HelloWorld application screen, clicking on the Hello World button again will execute the code and cause the application to reach the breakpoint and halt the execution, again.

As you can see from this simple exercise, Compact 7 and VS2008 IDE provide an effective debug environment that can help simplify complicated debugging tasks.
Part 11 – Debug and Remote Tools

While it’s not within the scope of this guide to cover in depth development and debug issues, it’s good to know about the debug and remote tools available for the Compact 7 development environment.

Using the same OS design project, created in the earlier section, work through the following steps to enable the KITL (Kernel Independent Transport Layer) build option and generate an OS run-time image with KITL enabled for the exercises in this section.

- If the project is not already active, launch the OS design project.
- From the VS2008 menu, select Project → eBox3350MX Properties to bring up the OS design property screen.
- From the eBox3350MX Property Pages screen’s left pane, expand the Configuration Properties node and click on the Build Options node to bring up the Build options selection window on right, as shown in Figure 69.

![Figure 69 - eBox3350MX Property Pages - Build options](image)

- On the right pane, double click on the Enable KITL option to enable this build option (set to “Yes”).
- Click Apply follow by OK to save the setting and close the screen.
- In the earlier section, the AutoLaunch component, CoreCon and Remote Display Application were included in the OS design. These components can cause interference to the remote tools. Remove these components from the OS design:
  - AutoLaunch_v300_Compact7
  - AutoLaunch_v300_Compact7\Autolaunch CoreCon
  - AutoLaunch_v300_Compact7\Autolaunch Remote Display application
- After the above AutoLaunch components are removed, from the VS2008 menu, select Build → Advanced Build Commands → Sysgen (blddemo –q) to generate an OS run-time image with KITL enabled.
Download KITL Enabled Compact 7 OS Image to Target Device

After the build process in the previous step is completed and successfully generated an OS run-time image with KITL enabled. Follow the procedure in the earlier section (Part-7) to download the Image to the target device.

After the image is downloaded to the target device, the VS2008 IDE’s Output tab should look similar to the following screen, as shown in Figure 70.

![Fig. 70 - VS2008 IDE showing debug output message](image)

With KITL enabled, the Platform Builder’s debug output window displays additional information as the target device startup. The additional information is useful for debugging and can help identify potential problems. Debug output message can be copied to a text file to be analyze in detail. Refer to Compact 7 documentation for more information about various debug and troubleshooting resources available.

In the following section, you will work through a series of exercise using remote tools from the VS2008 IDE to access the Compact 7 OS run-time on the target device remotely.

Remote Tool: Process Viewer

The Remote Process Viewer makes it possible to view running processes, associated threads and modules on the Compact 7 target device remotely.

Continuing from the previous section, with the KITL enabled OS image downloaded to the target device, work through the following steps to launch the Remote Process Viewer:

- From the VS2008 menu, select **Tools → Remote Tools → Process Viewer** to bring up the **Select a Windows CE Device screen**, as shown in Figure 71.
From the Select a Windows CE Device screen, select **MyTargetDevice** under the Windows CE node and click **OK** to establish connectivity to the target device.

After the connection is established, the following **Windows CE Remote Process Viewer** screen will appear, as shown in Figure 72.

Using the Remote Process Viewer, you can view all of the running processes and each process’ associated threads and modules on the target device. You can use the Remote Process Viewer to terminate running process. Try the following exercise:

- Launch **Control Panel** from the target device’s Compact 7 desktop.
- From the Remote Process Viewer remote tools running on the development station, select **Target → Refresh** from the menu to collect updated running process from the target device.
- After refresh, the **Control.exe** process shows up in the Process section.
- Click to highlight the **Control.exe** process and select **File → Terminate Process** to terminate the **Control.exe** process.
• The Windows CE Process Viewer Warning screen is raised.
• As you click Yes to continue and terminate the process, the control panel running on the target device is terminated.

From the menu, select **File → Exit** to terminate this remote tool session.

**Remote Tool: Registry Editor**

The Remote Registry Editor makes it possible to view and edit a Compact 7 device’s registry entries remotely from the development station. This is a useful tool to view registry entries and check whether the registry entries for device drivers and applications included to the OS design are compiled as part of the OS run-time image as intended.

Continuing from the previous section, with the KITL enabled OS image downloaded to the target device, work through the following steps to launch the **Remote Registry Editor**:

• From the VS2008 menu, select **Tools → Remote Tools → Registry Editor** to launch the **Remote Tools Shell**.
• The **Connecting to device** screen is shown as it attempt to connect to the target device, as shown in Figure 73.

![Fig. 73 - Connecting to device](image)

• After connection is established, the **Remote Tools Shell** is launched with the **Remote Registry Editor**, as shown in Figure 74.

![Fig. 74 - Remote Registry Editor](image)
To view which device drivers are loaded, take a look at registry entries under the following registry key:

\[HKEY_LOCAL_MACHINE\Drivers\Active\]

To terminate the *Remote Registry Editor*, select **File → Exit** from the Remote Tools Shell menu.

**Remote Tool: System Information**

The System Information remote tool is a useful utility to view the target device’s system information, including the following:

- **System summary:**
  - OS information
  - Processor information
  - System time and time zone
  - Etc.

- Memory
- Storage
- Drivers
- Owner Info
- Etc.

Continuing from the previous section, with the KITL enabled OS image downloaded to the target device, work through the following steps to launch the *Remote System Information* tool:

- From the VS2008 menu, select **Tools → Remote Tools → System Information** to launch the *Remote Tools Shell*.

- The *Connecting to device* screen is shown as it attempt to connect to the target device, as shown in Figure 7.5.

![Fig. 7.5 - Connecting to device](image)

- After connection is established, the *Remote Tools Shell* is launch with *Remote System Information* remote tool, as shown in Figure 7.6.
Fig. 76 - Remote System Information

You can use similar process to check out the other remote tools.
Part 12 – Silverlight for Windows Embedded

Silverlight is a user interface (UI) development framework designed to separate the graphical presentation from the programming logic. Different from the Silverlight for the desktop environment, where the code behind is based on managed code, SWE code behind uses native code. Using native code, you can develop compact and efficient SWE application to support resource constrained embedded devices.

Silverlight for Windows Embedded (SWE) was initially introduced as part of the Windows Embedded CE 6.0 R3 release. The initial release for CE 6.0 was based on Silverlight 2 and uses Expression Blend 2 to develop the XAML code project for the UI. For this release, you need to manually convert the XAML code from Expression Blend 2 to a CE 6.0 application project, as a subproject within an OS design.

SWE for Compact 7 is based on Silverlight 3 and uses Expression Blend 3 to develop the XAML code project for the UI. The Compact 7 release includes addition SWE tools to help convert the Expression Blend 3 XAML code project to a Compact 7 application subproject to the OS design. Here is the list of the additional SWE software components included as part of the Compact 7 software:

- **Windows Embedded Silverlight Tools (WEST)**
  The WEST tool installed as plug-in to the VS2008 IDE, provides the following functions:
  - Create Platform Builder Subproject wizard: This wizard is provided to convert Expression Blend 3 XAML code into a Compact 7 application subproject.
  - Update Silverlight for Windows Embedded Project wizard: This wizard is provided to update the codes in a Compact 7 application subproject (initially converted from a SWE Expression Blend 3 XAML code project), when the source project, the Expression Blend XAML code project, is changed.
  - Windows Embedded Events wizard: This wizard is provided to add C++ event handlers to the user interface elements parsed from the SWE Expression Blend 3 XAML code.

While SWE for Compact 7 is based on Silverlight 3, it’s a subset of the Silverlight 3 for the desktop and does not support all of the Silverlight 3 features. SWE Expression Blend 3 project is created using the Silverlight for Windows Embedded Template, which limit the project to use features that are supported by the SWE for Compact 7 environment.

Here is the process to develop a SWE for Compact 7 application:

1. Create a SWE for Compact 7 application from the Expression Blend 3 IDE, using the Silverlight for Windows Embedded template.
2. Convert the Expression Blend 3 SWE XAML code project to a Compact 7 subproject, using WEST.
3. Use the Windows Embedded Events wizard to create the necessary event handler and add code to support the application’s function and logic.

SWE application development is a broad subject involving multiple technologies. It’s not within this getting started guide’s intention to cover SWE application development. For more information about SWE for Compact 7 and sample application, please visit the following URL:

- [http://www.silverlight.net/](http://www.silverlight.net/)
- [http://www.embeddedpc.net/SWE/](http://www.embeddedpc.net/SWE/)
Congratulations! – You’ve completed all the steps

You completed all the steps in this guide.

- Develop, configure and customize a Compact 7 OS design project.
- Generate Compact 7 OS run-time image from the OS design, establish connectivity and download the OS run-time image to a target device.
- Develop Compact 7 native and managed code application using Visual Studio 2008, establish CoreCon connectivity and deploy the application to the target device.
- Configure the OS design with KITL to generate a Compact 7 OS run-time with KITL enabled, download the KITL enabled OS run-time to the target device and use remote tools to debug the image remotely.

This is just the beginning. The primary objective for this getting started guide is to help developer new to the Compact 7 development environment to get started with Compact 7 development quickly. There are advance subjects, not covered in this guide, which you need to pursue to further your knowledge.

Every embedded development project involves different hardware and application scenario. When starting an embedded development project, it’s important to select the appropriate hardware platform to establish an effective and efficient development environment with all of the needed components. Here are some addition subjects Compact 7 developers should know:

- How to deploy application to a Compact 7 device for distribution
- How to support and provide update to a Compact 7 device in the field
- Develop Real-time application for Compact 7
- Web Services for Device for Compact 7
- Silverlight for Windows Embedded
- Etc...

For additional information resources relate to Windows Embedded, visit the following Microsoft Website:


Additional Windows Embedded Compact information resources are provided in appendix C.

To learn more about the eBox family of product and other Single-Board-Computer from ICOP, visit the following Web sites.

http://www.icoptech.com
http://www.compactpc.com.tw
http://www.embeddedpc.net/eBox3350MX/
Appendix A – Development station Setup with DHCP

One of the common setup is to attach both the development station and Target Device to the same LAN with DHCP service to provide IP addresses dynamically, as shown in Figure A1.

Fig. A1 – Development environment with DHCP service

Connecting to Local Area Network with DHCP

It’s a typical setup to connect both the development station and the target device to a LAN with DHCP service to provide IP addresses dynamically. If the target device fails to establish connectivity with the development station and download the image as expected with this configuration, review and check the following:

- You may need to enable DHCP service for the target device on your network. Some secured network may require the target device’s MAC address to be added to the authorized device list for DHCP service in the DHCP server.
- There may be security issues relate to your development station’s firewall or security policy configured for your company’s LAN.

Using Wireless Access Point Router

When using a wireless-access-point-router with multiple Ethernet ports, connecting both the development station and target device directly to the Ethernet port on the wireless-access-point-router may be problematic with certain model of the wireless access point, and prevent the development environment to function as expected.

Note:
The wireless-access-point-router device’s routing function filter and route network packets based on the packets’ associated origin and destination information. Some of the router’s routing algorithm may prevent some of the packets between the development station and target device to reach its destination and prevent the setup from functioning properly.

Instead of connecting directly to the wireless-access-point-router device’s Ethernet ports, attach an Ethernet network hub to the wireless-access-point-router device, and connect both the development station and target device to the Ethernet network hub. With both the development station and target device connecting to the same Ethernet hub, the network traffic between them is not filtered.
Appendix B – Development station Setup with Static IP

Following are two scenarios where the development environment is setup without DHCP and require proper static IP addresses configuration to function.

Connecting to Ethernet Hub or Switch without DHCP
In this scenario, both the development station and target device are connected to the same LAN without DHCP service, and require proper static IP addresses configuration to establish connectivity.

![Diagram of development station connected to Ethernet hub and target device without DHCP](image1)

**Fig. B1** - Connecting to Ethernet Hub/Switch without DHCP

Direct Connection with Cross-Over RJ-45 Ethernet Cable
In this scenario, the target device is attached directly to the development station using a cross-over RJ-45 Ethernet cable, and require proper static IP addresses configuration to establish connectivity.

![Diagram of development station connected to target device using cross-over RJ-45 Ethernet cable](image2)

**Fig. B2** - Direct connection with cross-over RJ-45 Ethernet cable

Static IP Address
Without DHCP service to assign IP addresses dynamically, the target device and development station must be configured with appropriate static IP addresses in order to establish connectivity.

To establish connectivity, both the development station and target device must be configured with static IP addresses within same subnet.

Here are the recommended static IP addresses:
For the target device:
   IP address: 192.168.2.232
   Subnet mask: 255.255.255.0

For the development station:
   IP address: 192.168.2.132
   Subnet mask: 255.255.255.0

The prebuilt Compact 7 OS run-time images provided as part of the preconfigured software on the target device’s flash storage is built with DHCP enabled, and will attempt to send request to acquire IP address from an available DHCP server during startup. Refer to appendix F for information about how to configure a static IP address for the Compact 7 OS run-time, and how to modify the OS design’s registry entries to configure and build a Compact 7 OS run-time with static IP address.
Appendix C – Windows Embedded Compact Resources

Windows CE Reference
Windows Embedded Compact product documentation
This is the URL to the Windows Embedded Compact documentation on MSDN. If you are new to Windows Embedded Compact, this site provides valuable information and links to technical information resources.

Windows Embedded Community
http://www.embedded101.com
The Embedded101 community site provides large collection of technical information resources for Windows Embedded.

Windows Embedded Compact Community projects

AutoLaunch for Windows Embedded Compact (CE)
http://autolaunch4ce.codeplex.com
This AutoLaunch utility, provided as a Platform Builder component, helps simplify the tasks to configure an OS design to launch one or more application during startup. The AutoLaunch utility can be configured to launch multiple applications, with configurable time delay for each application, to control the launch sequence. This utility supports CE 6.0 and Compact 7.

CoreCon for Windows Embedded Compact (CE)
http://corecon4ce.codeplex.com
The project provides a Platform Builder component to include the necessary CoreCon files needed to establish connectivity between the target device and Visual Studio development station to deploy application to the target device for testing and debugging. Components for CE 6.0 and Compact 7 are available for download.

Windows Embedded CE Stream Driver Wizard
http://cedriverwiz.codeplex.com/
The CEDriverWiz driver wizard helps simplify the tasks needed to create the initial project workspace to develop a stream interface device driver for Windows Embedded Compact. This wizard supports CE 6.0 and Compact 7.

Windows Embedded CE component wizard
http://cecomponentwiz.codeplex.com
The CEComponentWiz wizard helps simplify the tasks needed to create a Windows Embedded CE catalog component.

Webcam Driver for Windows CE
http://cewebcam.codeplex.com
This project provides Windows CE device driver to support USB Webcam built to meet the USB video class specification.

Windows Embedded CE 6.0 USB Camera Driver
This is the URL to the USB camera driver for Windows Embedded CE 6.0 to support USB camera built to the USB video class specification.
Phidgets USB I/O driver shared source projects
http://phidgetswincedriver.codeplex.com
Phidgets is a family of I/O, sensors and microcontroller modules which can easily be added to a Windows Embedded CE system through the USB interface. This community project provides the Windows Embedded CE device driver needed to support Phidgets devices.

Bluetooth Wrapper for Windows CE
Provides a free Win32 API Wrapper that developers can expose in Visual Studio .NET or the .NET Compact Framework. Exposing the Win32 API Wrapper reduces the amount of code needed to develop for Bluetooth Technologies and helps make it easier to create compelling Windows Mobile and Windows CE Bluetooth applications.

32feet.Net – Personal Area Networking for .NET
http://32feet.codeplex.com
32feet.NET is a shared-source project to make personal area networking technologies such as Bluetooth, Infrared (IrDA) and more, easily accessible from .NET code.

Open SSH for Windows CE
http://cessh.codeplex.com
This shared-source project provides the resources needed to remotely access a Windows CE device in a secure manner using the SSH protocol. It helps execute remote commands on the device. It also supports tunneling, forwarding arbitrary TCP ports and it can transfer files using the associated SFTP or SCP protocols.

Windows CE Wifi Driver for Atheros AR-6000
http://cewifidriverar6000.codeplex.com
This is a shared-source Windows Embedded CE device driver project for the Atheros AR-6000 Wifi chipset.

**Windows Embedded Compact Books**

Professional Windows Embedded Compact 7
http://www.amazon.com/Professional-Windows-Embedded-Compact-7/dp/1118050460/ref=sr_1_1?s=books&ie=UTF8&qid=1312412814&sr=1-1

Professional Windows Embedded CE 6.0

Windows Embedded CE 6.0 Fundamentals

Programming Microsoft Windows CE .NET
http://www.amazon.com/Programming-Microsoft%C2%AE-Windows%C2%AE-Third-Reference/dp/0735618844/ref=sr_1_4?s=books&ie=UTF8&qid=1312412961&sr=1-4
Other Useful Links

http://www.embeddedpc.net
http://blogs.msdn.com/b/mikehall/
http://blogs.msdn.com/b/obloch/
http://blogs.msdn.com/b/ce_base/

Hardware Reference Information

http://www.embeddedpc.net/eBox3350MX
The eBox-3350MX is designed with a 1.0 GHz Vortex86MX System-On-Chip, built with 512MB DDR2 system memory and other common computer interfaces.

http://www.vortex86MX.com
Vortex86MX System-On-Chip information site.

http://www.icoptech.com
Other embedded hardware from ICOP.
Appendix D – eBox-3350MX Technical Information

The eBox-3350MX is a compact computing device designed for applications where physical space is limited. The enclosure is designed to easily secure behind a LCD monitor and other surfaces, which help save precious space in the work place, point-of-sales, point-of-information and other environment. The aluminum enclosure also function as heat sink to help dissipate heat and make it possible for eBox-3350MX to be fan-less and eliminated potential system failure caused by premature fan failure.

Designed with the Ultra-Low Power 1.0 GHz Vortex86MX System-On-Chip with integrated I/O peripherals and soldered on 512MB RAM provides sufficient system memory to support Windows Embedded Compact, Windows CE, Windows XP Embedded and WEPOS. The high bandwidth USB 2.0 connections provide broad range of expansion options, and support Operating System loading from USB flash storage device. The integrated 10/100M Ethernet can be configured to use PXE and boot from the network for diskless workstation application.

The eBox-3350MX is designed to launch supported operating system from SD flash storage devices, which are readily available in the market.

Board-Support-Package is available to support Windows Embedded CE 5.0, 6.0 and Windows Embedded Compact 7. Hardware device drivers are available to support Windows XP Embedded, Windows Embedded Standard and WEPOS.
## eBox-3350MX Specification

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1.00 GHz Vortex86MX System-On-Chip</td>
</tr>
<tr>
<td>BIOS</td>
<td>AMI BIOS</td>
</tr>
<tr>
<td>System Chipset</td>
<td>Integrated in Vortex86MX</td>
</tr>
<tr>
<td>I/O Chip</td>
<td>Integrated in Vortex86MX</td>
</tr>
<tr>
<td>(RAM) Memory</td>
<td>Soldered on 512MB DDR2 RAM</td>
</tr>
<tr>
<td>Storage</td>
<td>SD flash storage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIO</td>
<td>1 x SD flash storage slot</td>
</tr>
<tr>
<td>USB</td>
<td>3 x USB 2.0 Ports</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Display</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>VGA</td>
</tr>
<tr>
<td>Chipset</td>
<td>Integrated in Vortex86MX</td>
</tr>
<tr>
<td>Display Resolution</td>
<td>Up to 1600 x 1200</td>
</tr>
<tr>
<td>Connector</td>
<td>Standard 15-Pin VGA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Audio</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>Integrated in Vortex86MX</td>
</tr>
<tr>
<td>Audio Interface</td>
<td>Mic-in, Line-out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ethernet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>10/100M R6040, integrated in Vortex86MX</td>
</tr>
<tr>
<td>Remote Boot</td>
<td>Built-in boot ROM function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mechanical &amp; Environment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Requirement</td>
<td>+5V @ 2A</td>
</tr>
<tr>
<td>Operating Temp.</td>
<td>+5 ~ 50 °C</td>
</tr>
<tr>
<td>Operating Humidity</td>
<td>0% - 90% relative humidity, non-condensing</td>
</tr>
<tr>
<td>Size (W x H x D)</td>
<td>95<em>95</em>20 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>277g</td>
</tr>
</tbody>
</table>
FRONT CONNECTORS OUTLINE FOR eBOX-3350MX

REAR CONNECTORS OUTLINE FOR eBOX-3350MX

Pin Assignments

J16: Mini USB DC Power Input

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5V Input</td>
</tr>
<tr>
<td>2</td>
<td>USB IN -</td>
</tr>
<tr>
<td>3</td>
<td>USB IN +</td>
</tr>
<tr>
<td>4</td>
<td>NONE</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
</tr>
</tbody>
</table>

J7: USB (90°)– 4-pin USB Type 1 Connector (Vertical Type)

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
</tr>
<tr>
<td>2</td>
<td>USB0-</td>
</tr>
<tr>
<td>3</td>
<td>USB0+</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
</tr>
<tr>
<td>5</td>
<td>GGND</td>
</tr>
<tr>
<td>6</td>
<td>GGND</td>
</tr>
</tbody>
</table>
J5,J6: USB:
For connection to external USB device – 4-pin USB Type 1 Connector (H)

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
</tr>
<tr>
<td>2</td>
<td>USB2-</td>
</tr>
<tr>
<td>3</td>
<td>USB2+</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
</tr>
<tr>
<td>5</td>
<td>NC</td>
</tr>
<tr>
<td>6</td>
<td>NC</td>
</tr>
</tbody>
</table>

LED: POWER ON/OFF

<table>
<thead>
<tr>
<th>LED Color</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>Power On</td>
</tr>
</tbody>
</table>

J15: VGA – 15-pin D-Sub Connector

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal Name</th>
<th>Pin #</th>
<th>Signal Name</th>
<th>Pin #</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MR</td>
<td>2</td>
<td>MG</td>
<td>3</td>
<td>MB</td>
</tr>
<tr>
<td>6</td>
<td>GND</td>
<td>7</td>
<td>ND</td>
<td>8</td>
<td>GND</td>
</tr>
<tr>
<td>11</td>
<td>NC</td>
<td>12</td>
<td>CC</td>
<td>13</td>
<td>HYSYNC</td>
</tr>
<tr>
<td>15</td>
<td>VCC</td>
<td>14</td>
<td>VSYNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>VCC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. J9: LAN RJ-45 Connector

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal Name</th>
<th>Pin #</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>FTXD+</td>
<td>3</td>
<td>FRXIN+</td>
</tr>
<tr>
<td>2</td>
<td>FTXD-</td>
<td>4</td>
<td>NC</td>
</tr>
<tr>
<td>5</td>
<td>NC</td>
<td>6</td>
<td>FRXIN-</td>
</tr>
<tr>
<td>7</td>
<td>NC</td>
<td>8</td>
<td>NC</td>
</tr>
</tbody>
</table>
Appendix E – BIOSLoader

The SD flash storage provided as part of the eBox-3350MX Compact 7 jump start kit is preconfigured with BIOSLoader.

BIOSLoader is a bootloader designed to support x86 target device and can be used on device intended for distribution to the end-user. BIOSLoader is dependent on the x86 target device’s built-in BIOS to initialize the hardware correctly.

BIOSLoader is provided as part of the CEPC Board-Support-Package, in the following folder:

C:\WINCE700\Platform\CEPC\SRC\Bootloader\BIOSLoader

The BIOSLoader setup disk is provided in the \Diskimages folder under the above directory.

Through the BOOT.INI configuration file, the BIOSLoader can be configured to launch the NK.BIN OS run-time image or EBOOT.BIN Ethernet bootloader.

Here are some of the BIOSLoader configurable parameters from the Boot.ini file:

Primary BIN File

The primary BIN file parameter configures the BIOSLoader to launch a designated Compact 7 OS run-time image or the EBOOT.BIN Ethernet bootloader.

The following primary BIN file entry configures the BIOSLoader to launch NK.bin, a Windows Embedded Compact OS run-time image file:

# Primary BIN file name
BinFile=NK.BIN

The following primary BIN file entry configures the BIOSLoader to launch Eboot.bin, an Ethernet bootloader:

# Primary BIN file name
BinFile=EBOOT.BIN

Alternative BIN File

The alternative BIN file parameter configures the BIOSLoader to launch a designated Compact 7 OS run-time image or bootloader when the associated key is pressed during boot up, before the configured boot delay time expired.

The following alternative BIN file parameters configure the BIOSLoader to launch EBOOT.BIN, an Ethernet bootloader, when the “1” key is pressed during boot up before the designated boot delay time expired:

# Primary BIN file name
BinFile=NK.BIN

# Alternative image
# While booting hit the key corresponding to the suffix char (e.g. "1" for
# BinFile1) to boot the alternative image
BinFile1=EBOOT.BIN

# Boot delay (in seconds) in which to select the alternative image
Delay=10
The following alternative BIN file parameters configure the BIOSLoader to launch *NK2.BIN*, an alternative Compact 7 OS run-time image, when the “2” key is pressed during boot up before the designated boot delay time expired:

```plaintext
# Alternative image
# While booting hit the key corresponding to the suffix char (e.g. "1" for BinFile1) to boot the alternative image
BinFile2=NK2.BIN

# Boot delay (in seconds) in which to select the alternative image
Delay=10
```

The following parameter configures the BIOSLoader to launch using a preconfigured IP address:

```plaintext
# Debug IP address
DbgIP=192.168.2.232
```

**Note:**
The static IP address configuration is for Eboot.bin Ethernet bootloader only and does not impact the Compact 7 OS run-time image. To configure the Compact 7 OS run-time image to launch with a preconfigured static IP address, add the following registry entries to the OSDesign.reg file for the OS design:

```
[HKEY_LOCAL_MACHINE\Comm\PCI\060401\Parms\TcpIp]
 "EnableDHCP"=dword:0
 "DefaultGateway"=multi_sz:"192.168.2.1"
 "UseZeroBroadcast"=dword:0
 "IpAddress"=multi_sz:"192.168.2.232"
 "Subnetmask"=multi_sz:"255.255.255.0"
```

The following parameter configures the BIOSLoader to launch with serial debug enabled to send messages to the target device’s COM1:

```plaintext
# COM Port to pass in BootArgs
COMPort=1
```

**Note:**
In order for this configuration to work, the Compact 7 OS run-time image must be generated with the following build option enabled:
- Enable eboot space in memory

**BIOSLoader Configuration File for eBox-3350MX**

As part of this jump start kit, the eBox is preconfigured to launch the preloaded Compact 7 OS run-time image (*NK.BIN*) using BIOSLoader with 10 seconds delay during boot up. Before the 10 seconds delay expire (while the startup Compact 7 splash is still showing), you can press “1” to launch the Ethernet bootloader, *EBOOT.BIN*, or press “2” to launch the secondary OS image, *NK2.bin*.

Following is the code listing from the BIOSLoader configuration file (*BOOT.INI)*:

```plaintext
# Primary File to Load on startup:
BinFile=NK.bin

# Device name root string:
DeviceNameRoot=VDX

# Boot delay (in seconds) in which to select the alternative image
Delay=10
```
# Alternative image
# While booting hit the key corresponding to the suffix char (e.g. "1" for
# BinFile1) to boot the alternative image
BinFile1=ebott.bin
BinFile2=NK2.bin
BinFile3=NK3.bin

# Video Setting:
Video=on

# Physical screen width
# Must be >= DisplayWidth
PhysicalWidth=640

# Physical screen height
# Must be >= DisplayHeight
PhysicalHeight=480

# Display Depth
# Possible values are 8, 15, 16, 24, 32
DisplayDepth=16

# Display width
DisplayWidth=640

# Display height
DisplayHeight=480

# Debug IP address
# DbgIP=192.168.2.232

# COM Port to pass in BootArgs
# (Functionality must be supported by loaded program [ebott/nk..etc] for argument to
# have an affect)
# 0 - Suppress
# 1 - COM 1
# 2 - COM 2
Appendix F – Using Static IP Address

When configuring the development environment to use static IP address, the IP addresses for both the target device and development station must be configured to be in the same subnet.

The evaluation Compact 7 OS run-time image provides as part of the jump start kit is built with DHCP enabled, and will request IP address from an available DHCP service dynamically at boot time.

Configure Static IP for Prebuilt Compact 7 OS Run-time

Work through the following steps to configure a static IP address for the preconfigured Compact 7 OS run-time image:

- From Compact 7 desktop, select Start → Settings → Network and Dial-up Connections to bring up the Network Connections screen.

- From the Network Connection screen, right click on PCI-R60401 and select Properties.

- From the PCI\R60401 Settings screen, enter the static IP address and subnet mask.
Configure OS Design with Static IP Address

The OS design project can be configured to generate Compact 7 OS run-time image with preconfigured static IP address, by adding the necessary registry entries to the OSDesign.reg file.

Work through the following steps to configure the OS design with static IP address:

- With the OS design project active, click on the Solution Explorer window from the VS2008 IDE.
- Expand the Parameter Files node and double click on OSDesign.reg to open this file in the code editor window.
- From the code editor window, add the following registry entries to the end of the OSDesign.reg file.

```
[HKEY_LOCAL_MACHINE\Comm\PCI\R60401\Parms\TcpIp]
  "EnableDHCP"=dword:0
  "DefaultGateway"=multi_sz:"192.168.2.1"
  "UseZeroBroadcast"=dword:0
  "IpAddress"=multi_sz:"192.168.2.232"
  "Subnetmask"=multi_sz:"255.255.255.0"
```

- From the VS2008 IDE, select Build → Build Solution to generate a new Compact 7 OS run-time image.

After the build process is completed, the resulting Compact 7 OS run-time image is generated with the preconfigured static IP address.
Appendix G – Deploy OS Run-time with DiskPrep

DiskPrep is a Windows Embedded Compact power toy developed from Microsoft, available for download freely from the following URL:

http://code.msdn.microsoft.com/DiskPrep

The DiskPrep utility can be used to configure different type of storage device with BIOSLoader to launch Compact 7 OS run-time image and Ethernet bootloader with customizable splash screen.

The DiskPrep utility can configure the following type of storage device to launch Compact 7 OS run-time with BIOSLoader:

- IDE hard disk and IDE flash storage (such as EmbedDisk).
- Compact Flash storage
- SD memory card
- Micro-SD memory card
- USB flash storage

**Note:**

Not all x86 target devices are designed to support launching the OS run-time image from all of the above storage devices.

An appropriate “USB to Flash Storage Reader/Writer” adapter is needed to attach the flash storage media to the DiskPrep development station.

For the exercise in this section, using DiskPrep, you will work through the steps to configure the flash storage to launch **NK.BIN**, a Compact 7 OS run-time image, after 10 seconds delay with the following options:

- Before the 10 seconds delay expired, press “1” to launch **EBOOT.BIN**, an Ethernet bootloader.
- Before the 10 seconds delay expired, press “2” to launch **NK2.BIN**, the secondary OS run-time image.

To configure the flash storage using DiskPrep, you need to have the following:

- To configure an SD flash storage card, you need a USB to SD flash storage adapter to attach the SD flash to the development station. (The built-in SD slot works just fine.)
- To configure an IDE flash storage module, you need a USB to IDE adapter to attach the IDE flash to the development station.

**Note:**

Before deploying the NK.bin Compact 7 OS run-time image to the local flash storage, you need to be sure the image is functioning as expected.

Being able to download the NK.bin image to the target device and launch the Compact 7 OS run-time on the device does not necessary mean the NK.bin file can be deploy and boot from the target device’s local storage.

NK.bin image built with KITL enabled, even in release mode, is not able to boot from the target device’s local flash storage.

Make sure to generate a Compact 7 OS run-time image with the KITL built option disabled.

Here are the steps to deploy the **NK.BIN** Compact 7 OS run-time image using the DiskPrep power toy:

- Create a directory on your development station and name the directory as follow:

  C:\MyOSImage
- Copy the NK.BIN OS run-time image file to the C:\MyOSImage directory.
- Attach the flash storage to the development station using an adapter.
  - To configure SD flash storage, use a USB to SD flash adapter.
  - To configure IDE flash storage, use a USB to IDE adapter.
  - To configure Compact Flash storage, use a USB to Compact Flash adapter.
- After the development station detected and recognized the flash storage, launch the DiskPrep.exe executable from the C:\MyOSImage directory, and bring up the DiskPrep program screen, as shown in Figure H1.

![DiskPrep utility](image)

Fig. H1 - DiskPrep utility

- The DiskPrep utility detected the flash storage’s present and listed the detected flash storage in the Disk Selection drop down list box.
- The DiskPrep utility can prepare the flash storage with FAT, FAT32 or exFAT file system.
- For this exercise, select FAT.
- It’s possible to configure the device to display a custom splash screen during startup. For this exercise, select the Use the default Windows CE splash option.
- In the BOOT.INI section, select Place a BOOT.INI file on the disk.

**Note:**
The BOOT.INI configuration file is not required for the BIOSLoader to launch the NK.BIN Compact 7 OS run-time image. With a BOOT.BIN configuration file included, BIOSLoader can be configured with...
additional options.

- Select “640 x 480” for the “Physical Video Mode” and 16bpp for “Bit Depth”.
- Use the default “CEPC” as the device prefix or select “Use custom device name prefix” and enter a customized device prefix. The device prefix along with the last portion of the MAC address is used as the device ID to identify the device to the development station. For this exercise, use the default “CEPC” selection.
- To deploy the NK.BIN OS run-time image on to the flash storage, select the “Load specific image file copied from” option, click the Browse button to bring up the file selection dialog box, navigate to the “C:\MyOSImage” directory and select the NK.BIN file. At this point, the DiskPrep utility screen should look similar to the following, as shown in Figure H2.

![DiskPrep utility](image)

- Click OK to format and configure the flash storage with BIOSLoader and NK.BIN Compact 7 OS run-time image file.
- After the DiskPrep completed the process, go through the proper step to disconnect the flash storage from the development station, to prevent file corruption.

The flash storage is now configured with BIOSLoader and a Compact 7 OS run-time image.
Appendix H – CoreCon Connectivity

CoreCon connectivity is needed to establish link between the Compact 7 target device and the Visual Studio 2008 IDE to support application development for the target device.

The required CoreCon files are installed to the development station as part of the VS2008 installation to the following folder on the development station:

32-bit development station
\Program Files\Common Files\Microsoft shared\CoreCon\1.0\Target\WCE400\<CPU>

64-bit development station
\Program Files (x86)\Common Files\Microsoft shared\CoreCon\1.0\Target\WCE400\<CPU>

There are multiple set of CoreCon files installed to sub-folders under the above directory, to support ARM, MIPS and x86 processors, with the processor family as the sub-folder name. Each of these CoreCon files supports a designated processor family.

The eBox is engineered with an x86 processor. The required CoreCon files to support eBox are in the following folder:

32-bit development station
\Program Files\Common Files\Microsoft shared\CoreCon\1.0\Target\WCE400\x86

64-bit development station
\Program Files (x86)\Common Files\Microsoft shared\CoreCon\1.0\Target\WCE400\x86

To establish connectivity between the Visual Studio IDE and Compact 7 target device, the appropriate CoreCon files need to be launched on the target device when Compact 7 starts. In order to use these CoreCon files, they must be included to the Compact 7 OS run-time image, copied to the target device’s local storage or make available to the target device via Network share or external storage attached to the target device.

To help simplify the process to include the required CoreCon files to the OS design, a CoreCon component, provided as a sub-component to the AutoLaunch_v300_Compact7 component, is included with this jump start kit.

The AutoLaunch_v300_Compact7 component is installed to the C:\WINCE700\3rdParty directory. After installation, the AutoLaunch_v300_Compact7 component is available as part of the Compact 7 catalog component, in the \Third Party\Embedded101 folder. The CoreCon component is a sub-component, included as part of the AutoLaunch_v300_Compact7 component, as follow:

- AutoLaunch_v300_Compact7\Autolaunch CoreCon

The CoreCon component does not include the actual CoreCon binary files. When the CoreCon component is included in the OS design, it references and adds the necessary build steps to include the CoreCon files already installed to the development station, to the OS run-time image.

In addition to including the necessary CoreCon files to the OS run-time image, in order for the Visual Studio IDE to establish connectivity to the target device, the following CoreCon executable must be launched from the target device, after the Compact 7 OS is started:

- Conmanclient2.exe
- CMaccept.exe
The `CMaccept.exe` executable temporarily disable Compact 7 security to enable CoreCon connection to take place. After launching the `CMaccept.exe` executable, CoreCon connection must be established within 3 minutes. Otherwise, the `CMaccept.exe` executable needs to be launched again.

The AutoLaunch_v300_Compact7 component, along with the CoreCon sub-component, is designed to simplify the tasks needed to include CoreCon to the OS run-time image, and configure the OS run-time to launch CoreCon during startup.

When the `AutoLaunch_v300_Compact7` and the `AutoLaunch_v300_Compact7\Autolaunch CoreCon` components are included in the OS design, the following registry entries are added to the OS design.

```
[HKEY_LOCAL_MACHINE\Init]
  "Depend99"=hex:14,00,1e,00,32,00
  "Launch99"="AutoLaunch.exe"

[HKEY_LOCAL_MACHINE\System]
  "CoreConOverrideSecurity"=dword:1

[HKEY_LOCAL_MACHINE\Startup]
  "Process0"="ConmanClient2.exe"
  "Process0Delay"=dword:00001388
```

With the above registry, “CoreConOverrideSecurity”, added to disable security, the CMaccept.exe executable is not needed.

The registry entries under the HKLM\Init key configure the OS run-time image to launch the `AutoLaunch.exe` utility during startup.

As the `AutoLaunch.exe` utility launches, it scans the HKLM\Startup key to launch software components with proper entries. The following HKLM\Startup entries configure the AutoLaunch.exe to launch the `ConmanClient2.exe`, executable for the CoreCon component, during startup.

```
[HKEY_LOCAL_MACHINE\Startup]
  "Process0"="ConmanClient2.exe"
  "Process0Delay"=dword:00001388
```