Working with the Porting Kit for the .NET Micro Framework

A developers guide

Abstract

This document will show you how to work with the Porting Kit for the .NET Micro Framework. This document is a supplement to the PK documentation and assumes that you read the PK documentation already.

1 Abstract

The .NET Micro Framework Porting Kit (PK from here on) is a command line based development environment based on msbuild. The PK is designed to work with both the C++ and C# Visual Studio compilers and a set of native embedded compilers for ARM, SH and Blackfin processors.

The PK is a snapshot of the .NET MF internal build environment and can be used to do almost anything that is possible with the .NET MF. Here are the subjects we will talk about:

- Recompile the .NET MF assemblies
- Add a library for a driver
- Add a new processor for a supported instruction set
- Add a new tool chain

Combining the four tasks above allows creating a complete MF port for any processor.

In analyzing those tasks we will dive deeply into the PK structure and build system. All future references to path in the PK assume that a path is located under the %SPOCLIENT% directory of the PK unless otherwise specified.

2 Recompile the .NET MF Assemblies

The .NET MF assemblies are written in C#. VB.NET is not yet supported officially, although you can make it work rather easily (!).

All system assemblies (e.g. mscorlib or Microsoft.SPOT.Hardware) live under the directory Framework\Subset_of_Corlib or Framework\Core. The first contains the mscorlib support and you will rarely change it, as it provides basic types that are deeply rooted in the runtime architecture. The second contains all the rest. Every assembly compiles from C# project file with extension .csproj. In order to compile any assembly you will need to build the PK distribution first, and only once, by issuing the command 'msbuild build.dirproj' from %SPOCLIENT%. This command will generate among other things Metadataprocessor.exe, which is the tool the PK uses to transform the DLL format into the .NET MF proprietary PE format. After building the PK distribution, you will be able to re-build any assembly by targeting directly the assembly project with the command 'msbuild csproj /t:build'.

2.1 Version numbers

Sometimes you may want to be able to impose your build version number, for example to be compatible with an assembly you already distributed by increasing or maintaining the build number. We do require that you do not change the major and minor version numbers if you are participating in a community development project. You can use the build number and revision number to track your private distribution. The runtime will act strictly on versioning and require that all numbers do match, so your assembly is guaranteed to be used at all times. This is a

different behavior than the .NET Framework for the desktop, where build number and revision are considered more loosely.

To impose you version numbers A.B.C.D you can use the following command:

msbuild csproj /p:VersionMajor=A /p:VersionMinor=B /p:BuildNumber=C /p:RevisionNumber=D

Whatever property you omit it will be filled in with the defaults in the file %SPOCLIENT%\ReleaseInfo.settings, which we craft based on the latest official distribution. An indication of the version also is in each project file but it is overridden by the previous file.

2.2 Project file

Recompiling assemblies produces a number of files under the directory BuildOutput\public \debug\client.

The more interesting are the DLL, loadable on the desktop framework, the stub files collection for internal calls, the proprietary PE files that are loaded by the .NET MF, their symbols (PDBX files) for the Visual Studio debugger, and the annotated assembly under the txt directory. The PE files comes in little endian and big endian format, and we will see later how the system chooses the appropriate ones.

Let's analyze the csproj file for the Microsoft.SPOT.Native assembly.

```
E:\src\client_v4_3_xbox\Framework\Core\Native>more Core_Native.csproj
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="TinyCLR_Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
ToolsVersion="4.0">
  <PropertyGroup>
    <AssemblyName>Microsoft.SPOT.Native</AssemblyName>
    <OutputType>Library</OutputType>
    <RootNamespace>Microsoft.SPOT</RootNamespace>
    <ProjectTypeGuids>{b69e3092-b931-443c-abe7-7e7b65f2a37f};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}<
/ProjectTypeGuids>
    <ProductVersion>9.0.21022</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{1436D112-41D9-4BD4-9FA9-E38709CFF861}</ProjectGuid>
    <ComponentGuid>{c1031b9e-f16a-4862-a1f9-7cf25941f831}</ComponentGuid>
    <TinyCLR_CSharp_Documentation>true</TinyCLR_CSharp_Documentation>
    <AllowUnsafeBlocks>true</AllowUnsafeBlocks>
    <NoWarn>$(NoWarn),0169,0649,1591</NoWarn>
    <AssemblyBothEndian>true</AssemblyBothEndian>
  </PropertyGroup>
  <Import Project="$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CSharp.Targets" />
  <PropertyGroup>
    <!-- MMP_PE overridden options -->
    <MMP_PE_NoBitmapCompression>true</MMP_PE_NoBitmapCompression>
    <!-- MMP_STUB options -->
    <MMP_STUB_SKIP>false</MMP_STUB_SKIP>
    <MMP_STUB_GenerateSkeletonFile>$(BUILD_TREE_STUBS)\spot_native</MMP_STUB_GenerateSkeletonFile>
    <MMP_STUB_GenerateSkeletonProject>SPOT</MMP_STUB_GenerateSkeletonProject>
    <MMP_STUB_LegacySkeletonInterop>true</MMP_STUB_LegacySkeletonInterop>
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="Debug.cs" />
    <Compile Include="Delegates.cs" />
    <Compile Include="EventArgs.cs" />
    <Compile Include="ExecutionConstraint.cs" />
    <Compile Include="HW_SystemInfo.cs" />
    <Compile Include="HW_Utility.cs" />
```

```
    <Compile Include="Interfaces.cs" />
    <Compile Include="Math.cs" />
    <Compile Include="Messaging.cs" />
    <Compile Include="Reflection.cs" />
    <Compile Include="Resources.cs" />
    <Compile Include="Timers.cs" />
    <Compile Include="WeakReference.cs" />
    <Compile Include="Native_Resources.Designer.cs">
      <AutoGen>True</AutoGen>
      <DesignTime>True</DesignTime>
      <DependentUpon>Native_Resources.resx</DependentUpon>
    </Compile>
    <Compile Include="X509Certificate.cs" />
    <EmbeddedResource Include="Native_Resources.resx">
      <Generator>ResXFileCodeGenerator</Generator>
      <LastGenOutput>Native_Resources.Designer.cs</LastGenOutput>
      <SubType>Designer</SubType>
    </EmbeddedResource>
  </ItemGroup>
  <ItemGroup>
    <AppDesigner Include="Properties\" />
  </ItemGroup>
  <ItemGroup>
    <MMP_PE_ExcludeClassByName Include="Microsoft.SPOT.PublishInApplicationDirectoryAttribute">
      <InProject>false</InProject>
    </MMP_PE_ExcludeClassByName>
    <MMP_PE_ExcludeClassByName Include="Microsoft.SPOT.FieldNoReflectionAttribute">
      <InProject>false</InProject>
    </MMP_PE_ExcludeClassByName>
  </ItemGroup>
  <ItemGroup>
    <Folder Include="Properties\" />
  </ItemGroup>
</Project>
```

This tells the build system where to find the targets for the SPOT C# build system.

In that file we will import Microsoft.SPOT.Build.Targets (renamed Device.Targets in the SDK installation), which defines all the commands for Metadataprocessor.exe. Let's see some other interesting tags:

AssemblyBothEndian: tells the build system to generate both Big- and Little-endian PE files.

MMP_STUB_GenerateSkeletonFile and **MMP_STUB_GenerateSkeletonProject**: tells the build system to generate the project and file for the interop project in native code.

MMP_PE_ExcludeClassByName: tells metadata processor to prune a class from the DLL. This is only a taste of what you can do with Metadataprocessor.exe, see the target file above for more and the Metadataprocessor.exe help for the corresponding command line equivalents.

2.3 Deploying assemblies bundles with database files (DAT files)

One other interesting file is the following for a PK sample, which describes a managed code project executable:

```
%SPOCLIENT%\tools\scripts\port\Sample\LCD>more lcd.csproj
<Project DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="4.0">
  <PropertyGroup>
    <!--
```

```
    Project description for an executible with well-known (non-unique) project GUID

    Microsoft.SPOT.Sample namespace defined in source file

-->
    <AssemblyName>Microsoft.SPOT.LCD</AssemblyName>
    <OutputType>Exe</OutputType>
    <RootNamespace>Microsoft.SPOT.Sample</RootNamespace>
    <ProjectTypeGuids>{b69e3092-b931-443c-abe7-7e7b65f2a37f};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids>
    <ProductVersion>9.0.21022</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{332F8643-496C-44C1-9C19-3A75119DB3FC}</ProjectGuid>
    <NetMfTargetsBaseDir Condition="'$(NetMfTargetsBaseDir)'==''">$(MSBuildExtensionsPath32)\Microsoft\.NET Micro
Framework\</NetMfTargetsBa
seDir>

  </PropertyGroup>
  <PropertyGroup Condition="!Exists('$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CSharp.Targets') And '$(Configuration)|$(Platform)' ==
'Debug
|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition="!Exists('$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CSharp.Targets') And '$(Configuration)|$(Platform)' ==
'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <Import Project="$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CSharp.Targets"
Condition="Exists('$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CS
harp.Targets')" />
  <Import Project="$(NetMfTargetsBaseDir)$(TargetFrameworkVersion)\CSharp.Targets"
Condition="!Exists('$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CSharp.Targets')" />
  <PropertyGroup>
    <MMP_DAT_SKIP>false</MMP_DAT_SKIP>
    <MMP_DAT_CreateDatabaseFile>$(BUILD_TREE_DAT)\tinyclr_LCD.dat</MMP_DAT_CreateDatabaseFile>
    <MMP_DAT_CreateDatabaseFileOtherEnd>$(BUILD_TREE_DAT_OTHEREND)\tinyclr_LCD.dat</MMP_DAT_CreateDatabaseFileOtherEnd>
    <MMP_XML_SKIP>false</MMP_XML_SKIP>
    <MMP_XML_GenerateDependency>$(BUILD_TREE_XML)\DependencyGraph_Microsoft.xml</MMP_XML_GenerateDependency>
    <!--

  NOT needed for Porting Kit Builds

    <MMP_PE_NoBitmapCompression>true</MMP_PE_NoBitmapCompression>
    <MMP_DAT_SKIP>false</MMP_DAT_SKIP>
    <MMP_DAT_SKIP>false</MMP_DAT_SKIP>
    <MMP_DAT_CreateDatabaseFile>$(BUILD_TREE_DAT)\tinyclr_LCD.dat</MMP_DAT_CreateDatabaseFile>
    <MMP_DAT_CreateDatabaseFileOtherEnd>$(BUILD_TREE_DAT_OTHEREND)\tinyclr_LCD.dat</MMP_DAT_CreateDatabaseFileOtherEnd>
    <MMP_XML_SKIP>false</MMP_XML_SKIP>
    <MMP_XML_GenerateDependency>$(BUILD_TREE_XML)\DependencyGraph_Microsoft.xml</MMP_XML_GenerateDependency>
-->
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="lcd.cs" />
  </ItemGroup>
  <ItemGroup>
    <!--

    Specify the library references during compile and link (to PE) phases

-->
    <Reference Include="Microsoft.SPOT.Native">
      <HintPath>$(BUILD_TREE_DLL)\Microsoft.SPOT.Native.dll</HintPath>
    </Reference>
    <Reference Include="System">
      <HintPath>$(BUILD_TREE_DLL)\System.Dll</HintPath>
    </Reference>
    <Reference Include="Microsoft.SPOT.TinyCore">
      <HintPath>$(BUILD_TREE_DLL)\Microsoft.SPOT.TinyCore.Dll</HintPath>
```

```
    </Reference>
    <Reference Include="Microsoft.SPOT.Graphics">
      <HintPath>$(BUILD_TREE_DLL)\Microsoft.SPOT.Graphics.Dll</HintPath>
    </Reference>
  </ItemGroup>
  <ItemGroup>
    <!--

    Specify the managed code assemblies to be added to DAT database

-->
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\mscorlib.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\Microsoft.SPOT.Native.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\Microsoft.SPOT.Net.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\Microsoft.SPOT.LCD.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\System.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\Microsoft.SPOT.Graphics.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\Microsoft.SPOT.TinyCore.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>
    <MMP_DAT_CreateDatabase Include="$(BUILD_TREE_PE)\Microsoft.SPOT.Hardware.pe">
      <InProject>false</InProject>
    </MMP_DAT_CreateDatabase>

    <MMP_XML_Load Include="@(MMP_DAT_CreateDatabase)">
      <InProject>false</InProject>
    </MMP_XML_Load>
  </ItemGroup>
  <!--
  <ItemGroup>
    <AppDesigner Include="Properties\" />
  </ItemGroup>

  <ItemGroup>
    <Folder Include="Properties\" />
  </ItemGroup>
-->
</Project>
```

This project is meant to build in the PK and the SDK as well. We check where the project is loaded with the simple condition

```
PropertyGroup
Condition="!Exists('$(SPOCLIENT)\tools\Targets\Microsoft.SPOT.CS
harp.Targets'…
```

Then we list the references and we issue a sequence of MMP_DAT_CreateDatabaseFile and MMP_DAT_CreateDatabase commands. As much as a PE file is one assembly, a database file (DAT file) is a collection of assemblies concatenated and aligned at the 4-byte boundary. You can generate one as in the project above or by creating a text files that list the PE files you want to include in a DAT file and passing it to Metadataprocessor .exe with the command ‘

create_database'. You can also inspect the3 contents of a DAT file with the command '-dump_dat'.

A DAT file is a self contained bundle of assemblies that is very useful to debug RAM builds with a native debugger. In fact you can create a DAT file with a name following the naming conversion "tinyclr_ my name .dat" and then create the environment variable 'set FORCEDAT= my name ' in the PK environment before building your solution. The .NET MF build system will look for FORCEDAT and embed the corresponding DAT file in the runtime execution region so that you will be able to deploy it to the device and debug it natively. This is especially handy when using RAM builds and trouble-shooting drivers and although does not allow setting breakpoints directly in managed code it is still a precious help when you want to trouble shoot drivers instead.

3 Add a library for a driver

Adding a driver library to the .NET MF collection is rather easy. Each driver has its own dotnetmf.proj file and one can just copy one of the existing ones. Sometimes a driver is actually a collection of libraries and one might need to build an entire sub-tree. There are two way of accomplishing building a sub-tree. One could simply insert the following snippet in any dotnetmf.proj file that is it by the dependency driven build system:

ItemGroup

SubDirectories Include="my_sub_dir"/

SubDirectories Include="my_other_sub_dir "/

…

/ItemGroup

This snippet will cause the build system to trickle down the directory tree to look for another dotnetmf.proj file in the directories listed. Another way of accomplishing the same task would be to mention a project as a dependency of another with

RequiredProjects Include="$(SPOCLIENT)\my_project_path\dotnetmf.proj" /

You can add as many dependencies as you need. Let's analyze a driver's project file and see how we can modify it or create a new one.

```
%SPOCLIENT%\DeviceCode\Drivers\Ethernet\enc28j60>more dotNetMF.proj
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <AssemblyName>Ethernet_enc28j60</AssemblyName>
    <Size>
    </Size>
    <ProjectGuid>{30c302e4-9feb-43d5-8868-44cbba27d983}</ProjectGuid>
```

```
    <Description>ENC28J60 Ethernet Driver</Description>
    <Level>HAL</Level>
    <LibraryFile>ETHERNET_enc28j60.$(LIB_EXT)</LibraryFile>
    <ProjectPath>$(SPOCLIENT)\DeviceCode\Drivers\Ethernet\enc28J60\dotNetMF.proj</ProjectPath>
    <ManifestFile>ETHERNET_enc28j60.$(LIB_EXT).manifest</ManifestFile>
    <Groups>Network</Groups>
    <LibraryCategory>
      <MFComponent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
Name="EthernetDriver_HAL" Guid="{7FD79853-56A2-4CB8-843B-57939B7468F4}" ProjectPath="" Conditional="" xmlns="">
        <VersionDependency xmlns="http://schemas.microsoft.com/netmf/InventoryFormat.xsd">
          <Major>4</Major>
          <Minor>0</Minor>
          <Revision>0</Revision>
          <Build>0</Build>
          <Extra />
          <Date>2009-04-30</Date>
        </VersionDependency>
        <ComponentType xmlns="http://schemas.microsoft.com/netmf/InventoryFormat.xsd">LibraryCategory</ComponentType>
      </MFComponent>
    </LibraryCategory>
    <Documentation>
    </Documentation>
    <PlatformIndependent>False</PlatformIndependent>
    <CustomFilter>
    </CustomFilter>
    <Required>False</Required>
    <IgnoreDefaultLibPath>False</IgnoreDefaultLibPath>
    <IsStub>False</IsStub>
    <Directory>DeviceCode\Drivers\Ethernet\enc28j60</Directory>
    <OutputType>Library</OutputType>
    <PlatformIndependentBuild>false</PlatformIndependentBuild>
    <Version>4.0.0.0</Version>
  </PropertyGroup>
  <Import Project="$(SPOCLIENT)\tools\targets\Microsoft.SPOT.System.Settings" />
  <Import Project="$(SPOCLIENT)\Framework\Features\ENC28J60_Config_HAL.libcatproj" />
  <PropertyGroup />
  <ItemGroup>
    <Compile Include="enc28j60.cpp" />
    <Compile Include="enc28j60_driver.cpp" />
    <IncludePaths Include="DeviceCode\arm\Drivers\Ethernet\ENC28J60" />
    <IncludePaths Include="DeviceCode\pal\net" />
    <IncludePaths Include="DeviceCode\pal\rtip" />
    <IncludePaths Include="DeviceCode\pal\rtip\drivers" />
    <IncludePaths Include="DeviceCode\pal\rtip\protocol" />
    <IncludePaths Include="DeviceCode\pal\rtip\rtpcore" />
    <IncludePaths Include="DeviceCode\pal\rtip\tinyclr" />
    <IncludePaths Include="DeviceCode\pal\rtip\vfile" />
  </ItemGroup>
  <ItemGroup>
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\net\network_defines.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\net\net_decl.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\bget.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\debugapi.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\dhcp.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\dhcpcapi.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\dhcpconf.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\dhcpext.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\ethconf.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\igmp.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\os.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\pollos.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtip.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtipapi.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtipconf.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtp.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpchar.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpdate.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpdb.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpdbapi.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpenv.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpirq.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpkern.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpmem.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpmemdb.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpmtxdb.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpnet.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpprint.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpscnv.h" />
```

```
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpsemdb.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpsignl.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpstr.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpterm.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtpthrd.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtptime.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtptypes.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\rtpcore\rtputil.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\servconf.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\sock.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\socket.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\tinyclr\rtpport.h" />
    <HFiles Include="$(SPOCLIENT)\devicecode\pal\rtip\xnconf.h" />
    <HFiles Include="enc28j60.h" />
    <HFiles Include="enc28j60_driver.h" />
  </ItemGroup>
  <Import Project="$(SPOCLIENT)\tools\targets\Microsoft.SPOT.System.Targets" />
</Project>
```

When you duplicate a project file you always have to remember to change the GUID, so that SolutionWizard can pick up your project. Also for SolutionWizard you need to identify the project level (HAL or PAL) and let the build system know whether you want to compile the project as Platform Independent or not, using the PlatformIndependent tag. The platform independent project end up in the ANY_MEDIA branch under the BuildOutput tree, as opposed to the FLASH or RAM branch. There is no particular advantage in doing one or the other thing if you have no peculiar memory requirements, but that you do not need to recompile the library for all memory configurations. Generally it is safer not to declare a component platform independent if your system needs special coding attention depending on the location of the driver code in memory. This is for example the case of calibration sensitive code such as a delay loop.

The version tag is a default version for the component, but it is overridden by the two mechanism previously described. It is interesting to note that you can make a driver depend from another driver by adding an import for a library category (see in framework\features for SolutionWizard) or a RequiredProject tag. Finally the IncludePaths tag adds to the include directory the compiler will look into.

3.1 Where to add a library for a driver

A driver code base can live under four different places: DeviceCode\Drivers, DeviceCode\Targets, DeviceCode\PAL and Solutions\ my solution \DeviceCode.

The last one is mainly for configuration libraries, generally implemented as a global variable or implementation of customization functions (such as adding lock storage drivers to the list of available storages for the File System).

DeviceCode\Targets instead is for those drivers that belong to a SoC. DeviceCode\PAL is for those middleware drivers and have not tight dependencies on any specific controller but that rather depend on a specific HAL functionality. All the other drivers go under DeviceCode\Drivers. It is highly recommended that you place the code in the right directories to maximize you chance of reusing it effectively.

4 Add a new processor for a supported instruction set or RTOS

Adding support for new processors requires adding at least the following:

1) One directory under DeviceCode\Targets for the RTOS code or the MCU/Soc code
2) One new Solution

Optionally, bug arguably so, you will need also

a) A DeviceCode directory and libraries under your solution directory for the configuration libraries and
b) Some drivers libraries under DeviceCode\Drivers, e.g. for a flash chip or an LCD

The latter are not required, but add to the flexibility and maintainability of the overall code base. In facts is way easier to reuse drivers across solutions if you provide the right configuration points under your solution directory and if you move the drivers for those controllers that are not part of your SoC under the DeviceCode\Drivers directory.

For every new driver you write you always need a stub. Most stub for existing components already exist and allow you to build an image even if you do not include your driver code in it. SolutionWizard can help you choosing stubs or actual drivers using the IsStub tag in the project above.

After adding the code base for your SoC you will need to configure it for the compiler at hand. You will notice that there is a setting file in each SoC directory under DeviceCode\Targets. We can check the AT91_SAM7X.Settings file:

```
%SPOCLIENT%\DeviceCode\Targets\Native\AT91>more AT91SAM7X.settings
∩╗┐<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Name>AT91SAM7X</Name>
    <CpuName>ARM7TDMI</CpuName>
    <DefaultISA>THUMB</DefaultISA>
    <Guid>{5a9f48c9-f9f3-4473-8b76-b4fca51eb0e8}</Guid>
    <Description>
    </Description>
    <Documentation>
    </Documentation>
    <ProjectPath>$(SPOCLIENT)\devicecode\Targets\Native\AT91\AT91SAM7X.settings</ProjectPath>
    <PLATFORM_FAMILY>ARM</PLATFORM_FAMILY>
    <CustomFilter>ARM7;AT91SAM7X;AT91</CustomFilter>
    <INSTRUCTION_SET Condition="'$(INSTRUCTION_SET)'==''">THUMB</INSTRUCTION_SET>
    <TARGETPROCESSOR>AT91SAM7X</TARGETPROCESSOR>
    <TARGETCODEBASE>AT91</TARGETCODEBASE>
    <TARGETCODEBASETYPE>Native</TARGETCODEBASETYPE>
  </PropertyGroup>
  <PropertyGroup Condition="'$(COMPILER_TOOL)'=='RVDS'">
    <DEVICE_TYPE Condition="'$(DEVICE_TYPE)' == ''">ARM7TDMI</DEVICE_TYPE>
    <BUILD_TOOL_GUID>{1942C531-3AAC-4abb-8B4F-C3111012F9D9}</BUILD_TOOL_GUID>
  </PropertyGroup>
  <PropertyGroup Condition="'$(COMPILER_TOOL)'=='MDK'">
    <DEVICE_TYPE Condition="'$(DEVICE_TYPE)' == ''">DARMAT</DEVICE_TYPE>
    <BUILD_TOOL_GUID>{CD24C1A5-2641-4460-AC5A-093B1C6D3D8B}</BUILD_TOOL_GUID>
  </PropertyGroup>
  <PropertyGroup Condition="'$(COMPILER_TOOL)'=='GCC'">
    <DEVICE_TYPE Condition="'$(DEVICE_TYPE)' == ''">ARM7TDMI</DEVICE_TYPE>
    <BUILD_TOOL_GUID>{722B0D5D-1243-4557-913F-61FAB04E9209}</BUILD_TOOL_GUID>
  </PropertyGroup>
```

```
  <ItemGroup>
    <IncludePaths Include="devicecode\Targets\Native\AT91" />
  </ItemGroup>
</Project>
```

Most properties are meant for the compiler/linker tool chain following the rules of the targets and settings files under the tools\targets directory.

CpuName: this property is mostly used by SolutionWizard and can be used as a filter in project files to select driver specific to a processor core.

DEVICE_TYPE: this property will be passed to the compiler as an optimization switch or to generate a specific instruction set.

INSTRUCTION_SET: the instruction set to generate code for.

TARGETPROCESSOR: the specific processor type. Used by the target files to select optimizations or patches.

TARGETCODEBASE: the code base for the source tree of the specific MCU under the Codebase type. This is where the build system will be looking for the settings file to identify the device type and other parameters.

TARGETCODEBASETYPE: this property can be set to *Native* or *OS* and it identifies the code base for the MCU source tree under DeviceCode\Targets.

Under the solution directory we find another two important files that complete our MCU definition. As a solution is a container of libraries and defines a device image, all the files under solutions mostly deal with configuration properties. The first item is the platform_selector.h file that defines items such as the high-level hardware and memory configuration for the runtime.

For example the selector file defines the debugging transport for the CLR (DEBUGGER_PORT and MESSANGING _PORT) and the transport for the debug prints (STDIO for hal_printf and DEBUG_TEXT_PORT for debug_printf and Debug.Print). It also defines how many USB endpoints, COM ports, where the FLASH memory resides and how big it is, the GP I/O configuration and the clocks. The last interesting entry is the runtime memory profile, which can be selected as medium, small or large using the macros NETWORK_MEMORY_PROFILE__medium, NETWORK_MEMORY_PROFILE__small, NETWORK_MEMORY_PROFILE__extrasmall, and NETWORK_MEMORY_PROFILE__large. The runtime memory profiles impacts the CLR internal caches size and the TCP/IP stack. In the smallest configuration the runtime will only have a 9KB overhead in runtime memory.

The last file to examine one solution settigns file.

```
E:\src\client_v4_3_xbox\Solutions\iMXS>more iMXS.settings
∩╗┐<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Author>
    </Author>
```

```
    <Description>Freescale iMXS full functionality (except networking)</Description>
    <Documentation>
    </Documentation>
    <PlatformGuid>{7895a3c9-aafb-47b0-9b66-da3ea37527c2}</PlatformGuid>
    <TARGETPLATFORM>iMXS</TARGETPLATFORM>
    <PLATFORM>iMXS</PLATFORM>
    <IsSolutionWizardVisible>True</IsSolutionWizardVisible>
    <ENDIANNESS>le</ENDIANNESS>
  </PropertyGroup>
  <ItemGroup>
    <IncludePaths Include="Solutions\iMXS" />
  </ItemGroup>
  <Import Project="$(SPOCLIENT)\devicecode\Targets\Native\MC9328\MC9328.settings" />
</Project>
```

The most important tags are:

TARGETPLATFORM: this is in facts the solution

PLATFORM: this is the compilation switch to pass to msbuild for the /p:platform switch. It should match TARGETPLATFORM and we use it mostly for legacy reasons

ENDIANNESS: this is the endianness of the device. The system will automatically choose the right assemblies based on this switch.

Also note that the solution settings directly import the MCU settings.

5 Add a new tool chain

After knowing all of the above, adding a new tool chain is a less challenging than one might think. When adding a new tool chain, one does not necessarily need to add a new platform or MCU code base, but you need to make your solution work with the new tool chain.

A user initializes the PK build environment by calling a setenv_ tool chain .cmd script at the very beginning. You will need to craft an equivalent initialization script that works with your tool chain installation. Most tool chains create environment variables you can use to set the tools in your path.

After the environment is initialized, you only will need to act in the tools\targets directory. You will notice that each tool chain we support (RVDS, MDK, HEW for SH, Visual DSP for Blackfin, GCC, etc.) comes with a pair of settings/targets file with the name Microsoft.Spot.system. tool chain .[targets | settings].

Your initialization script will have to set correctly two environment variables called COMPILER_TOOL and COMPILER_TOOL_VERSION so that the infrastructure in Microsoft.SPOT.System.Targets will pick up the right pair of settings/targets for you tool chain.

The settings file defines the extensions of the output files and the build output paths under the BuildOutput directory. The target file hosts all compiler and linker switches for the build flavor (debug,release,rtm) and is no different than a make equivalent.

NOTE: When working with the CodePlex distribution one can also add toll chains under the %SPOCLIENT%\Tools directory, provided that a specific layout of the executable and library files is implemented. Please see the information at this regard on the Documentation page of the .NET MF CodePlex distribution at http://netmf.codeplex.com, under the general information about the distribution.