Features

- Extension Developer’s Kit user guide
- Atmel® Studio IDE SDK
- Embedded SDK

Description

The Atmel Studio Extension Developer’s Kit (XDK) supports 3rd parties to independently extend the Atmel Studio 6 platform with both development tools and embedded software and prepares them for submission to the Atmel Gallery.

By using the XDK an integrated user experience of 3rd party extensions is delivered to users of Studio 6. For partners, extensions developed with the XDK provide the opportunity to offer their development tools or embedded software directly to the Atmel Studio 6 user base through the Atmel Gallery.

This application note together with additional information about the Atmel Gallery, how to become an Atmel Gallery developer and how to submit an extension can be found at: http://gallery.atmel.com and http://gallery.atmel.com/Partner.
Table of Contents

1. Getting Started with the XDK ............................................................... 3
   1.1 Atmel Studio Integrated Development Platform Overview .................. 3
   1.2 Registration ..................................................................................... 4
   1.3 Software tools .................................................................................. 4
      1.3.1 Atmel Studio IDE SDK ............................................................... 4
      1.3.2 Embedded SDK ....................................................................... 4

2. The Atmel Studio IDE SDK .................................................................. 5
   2.1 Introduction ...................................................................................... 5
      2.1.1 Extending Atmel Studio ............................................................ 5
      2.1.2 Example projects ..................................................................... 5
   2.2 How to create an extension for Atmel Studio 6.1 ............................... 6
      2.2.1 Create a Visual Studio package ............................................... 6
      2.2.1.2 Remove reference to Visual Studio MPF ............................. 7
      2.2.2 Add Atmel Studio IDE SDK assembly reference ..................... 7
      2.2.3 Set up debugging .................................................................... 8
      2.2.4 Package the extension for distribution .................................... 10
   2.3 Using the Visual Studio SDK ........................................................... 11
      2.3.1 How to make GUI extensions .................................................... 11
      2.3.2 How to use automation for basic run control ......................... 11
      2.3.3 How to use expression evaluation .......................................... 14
   2.4 Using the Atmel Studio IDE SDK ..................................................... 14
      2.4.1 How to read/write memory ....................................................... 14
      2.4.1.1 Naming of memories on a device ...................................... 16
   2.5 Distributing extensions on Atmel Gallery ......................................... 17
   2.6 Atmel Studio IDE SDK references .................................................. 17

3. The Embedded SDK .......................................................................... 18
   3.1 Introduction ..................................................................................... 18
      3.1.1 Extending the Embedded Software Realm ................................ 18
   3.2 Embedded Application extensions ................................................... 18
      3.2.1 How to Create an Embedded Application Extension ................ 18
      3.2.2 How to use an Embedded Application Extension .................... 23
      3.2.3 How to publish an Embedded Application Extension ............... 24
      3.2.4 Embedded SDK configuration ............................................... 24
   3.3 Embedded Module/Library Extensions ............................................. 25

4. Submit extensions to Atmel Gallery ................................................... 26

5. References and further information .................................................... 27
   5.1 Atmel Gallery .................................................................................. 27
   5.2 Atmel Studio ................................................................................... 27
   5.3 ASF ................................................................................................. 27
   5.4 Support ............................................................................................ 27

6. Revision History ................................................................................ 28
1. Getting Started with the XDK

1.1 Atmel Studio Integrated Development Platform overview

The XDK is an important component of the Atmel Studio Integrated Development Platform (see Figure 1-1), which is targeted to provide embedded developers with all the development tools and embedded software needed to address the challenges of ever increasing complexity in embedded designs: Atmel Studio 6 is the development environment for developing and debugging Atmel ARM® Cortex™-M and Atmel AVR® microcontroller (MCU) based applications. The Atmel Studio 6 IDE provides developers with a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. Atmel Studio 6 is available free of charge and can be downloaded at http://www.atmel.com/Microsite/atmel_studio6/software.aspx.

The Atmel Software Framework (ASF) is closely integrated with Atmel Studio 6 and delivers a collection of production-ready source code such as drivers, communication stacks, graphic services and touch functionality. Over 1,600 project examples, built on ASF can be selected and configured in Atmel Studio 6. Detailed information and documentation for ASF can be found at http://asf.atmel.com.

The Atmel Gallery provides development tools and embedded software for MCU-based application design that extends Atmel Studio 6. Users can download and securely purchase both Atmel and 3rd party extensions such as compilers, advanced debugging tools, real-time operating systems, communication stacks that integrated directly into Atmel Studio 6. The Atmel Gallery can be directly accessed from within Atmel Studio 6 or also at http://gallery.atmel.com. Atmel Spaces is a cloud-based collaboration workspace for hosting software and hardware projects targeting Atmel MCUs. Atmel Spaces helps the Atmel design community to collaborate by providing all the tools that enable collaborative development easy. An extension for Studio 6 to access Atmel Spaces projects is available in the Atmel Gallery; Atmel Spaces is also available at http://spaces.atmel.com.

The XDK enables 3rd parties to independently extend the Atmel Studio platform with both development tools and embedded software and to prepare them for submission to the Atmel Gallery. Collaboration projects developed on Atmel Spaces can equally benefit from the XDK for integrations with Studio.

The XDK enables 3rd party developers to integrate their tools and embedded software with Atmel Studio 6, and to package and submit their integrations into the Atmel Gallery, for both commercial and non-commercial integrations.

Figure 1-1. Atmel Studio Integrated Development Platform.
1.2 **Registration**
Refer to XDK - How to become a developer on [http://gallery.atmel.com/Partner](http://gallery.atmel.com/Partner).

1.3 **Software tools**

1.3.1 **Atmel Studio IDE SDK**
- Atmel Studio IDE SDK: [http://www.atmel.com/tools/atmelstudio.aspx](http://www.atmel.com/tools/atmelstudio.aspx) or through Atmel Studio Extension Manager. Please note that local help should be used when installing Atmel Studio 6 IDE SDK

Note: Visual Studio 2010 Express does not support creation of VSPackages.

1.3.2 **Embedded SDK**
2. The Atmel Studio IDE SDK

2.1 Introduction

This section is targeting those who want to extend Atmel Studio 6.1. It describes how to write a simple extension, and contains some example code illustrating the possibilities offered by the Atmel Studio 6.1 SDK.

After reading this document you should be able to do the following:

- Write your own Atmel Studio 6.1 extension and prepare it for upload to Atmel Gallery
- Use Visual Studio SDK for:
  - Extending the GUI
  - Basic run control
  - Expression evaluation
- Use Atmel Studio IDE SDK for:
  - Read and write memory in an extension

The reader is expected to be familiar with how to write a C# program and how to use the Visual Studio programming environment.

2.1.1 Extending Atmel Studio

Atmel Studio 6.1 is a Visual Studio Isolated Shell application. The engine behind is Visual Studio, taking care of the Windows® system, project system, menus, toolbars and the ecosystem in general. To make extensions to Atmel Studio 6.1 use the same tools as for Extend Visual Studio. Extensions written for Visual Studio will work in Atmel Studio as long as they do not use features not available in the Isolated Shell; mainly the compilers.

Visual Studio SDK provides interfaces for extending Visual Studio, which also can be used to extend Atmel Studio. Some of the functionalities offered are:

- Visual Studio automation
- Access to debugger run control (step, run, break…)
- Debugger events (enter runmode, breakmode, edit mode…)
- Adding GUI elements (windows, menu items, toolbars, buttons…)

Atmel Studio IDE SDK provides additional interfaces:

- Read/write different memories available on a device
- Get information about the device and tool used
- Add a toolchain (not covered in this document)

2.1.2 Example projects

After installing the Atmel Studio IDE SDK, example projects for Visual Studio 2010 are located under the example folder in the SDK, default to:

C:\Program Files (x86)\Atmel\Atmel Studio SDK\6.1\examples

The following examples are available:

- MemoryList Shows how to step, select memory type and read/write values
- MemoryLogger Select memory type, read memory and write to hex-file

Note: Extensions are uniquely identified by a guid number. To avoid mix up of using the same guid for several extensions, the guids in the examples have been invalidated. This shows as compilation error. Please use menu Tools->Create Guid to generate unique identifiers to use.
2.2 How to create an extension for Atmel Studio 6.1

Extensions for Visual Studio are called packages (or VSPackages) and add-ins. In this document we are focusing on creating a VSPackage. The Visual Studio Package project template creates a basic VSPackage (see creating a VSPackage - http://msdn.microsoft.com/en-us/library/cc138589(v=vs.100).aspx). The template can add code to create a menu command or a tool window – editor automation is not covered in this document.

2.2.1 Create a Visual Studio package

In order to create a VSPackage that can be installed as an extension to Atmel Studio 6.1, do the following steps:

2. Select the Visual Studio Package project template, found in "Installed Templates->Other Project Types->Extensibility". In the Name box, type a name for the solution (e.g. MyExtension) and then click OK.

Figure 2-1. New Project dialogue.

3. On the Select a Programming Language page, select either Visual C#®, Visual C++® or Visual Basic®. (In our examples, C# is used) Have the template generate a key.snk file to sign the assembly. Alternatively, click Browse to select your own key file. The template makes a copy of your key file and names it key.snk.
4. On the Basic VSPackage Information page, specify details about your VSPackage.
5. Click Next to specify package options for your VSPackage.
6. Select the Menu Command option to create a command (will appear in the Tools menu) for your VSPackage, and Tool Window option to create a dockable window (accessible from View Other Windows) for your VSPackage, click Next.
7. The Command Options page is displayed.
   a. In the Command Name box, type a name for the new command. If you later want to host the command as a button on the toolbar, this name is also used as the tooltip for the button.
   b. In the Command ID box, type the command ID for your command. The command ID is the name of a constant that represents this command in the generated code.
   c. Click Next.
8. The **Tool Window Options** page is displayed.
   a. In the **Window Name** box, type a name for the new window.
   b. In the **Command ID** box, type the command ID for your command.
      The command ID is the name of a constant that represents this command in the generated code.
   c. Click **Next**.
9. Optionally, select **Integration Test Project** and **Unit Test Project** to create test projects for your solution.
10. Click **Finish** to create your VSPackage.

**Note:** If this message appears:

**Figure 2-2. Error message.**

- Restart Visual Studio
- Open last Solution
- In Solution Explorer: Add Existing Project – add newly created .csproj

### 2.2.1.2 Remove reference to Visual Studio MPF

This reference is not needed for most Atmel Studio 6.1 extensions and can cause issues later.

- Select source.extension.vsixmanifest In the Solution Explorer
- Select Visual Studio MPF in References section
- Press **Remove Reference**

**Figure 2-3. Remove reference to Visual Studio MPF.**

### 2.2.2 Add Atmel Studio IDE SDK assembly reference

In order to be able to use the interfaces defined in the Atmel Studio SDK, you need to add a reference to the assembly.

In the solution explorer right click **References**.
2.2.3 Set up debugging

To be able to debug the newly created project, do the following.

- In **Project Properties** set the following option:
  - Debug – Start external program
    - Select atmelstudio.exe (default in C:\Program Files (x86)\Atmel\Atmel Studio 6.1)

Remove any command line arguments (if present).
Figure 2-5. Project properties – debug tab.

- VSIX – Check Copy VSIX content to the following location:
  - For Windows Vista® / Windows 7 / Windows 8:
    C:\Users\user\AppData\Local\Atmel\Atmel Studio\6.1\extensions\myextension
  - For Windows XP:
    C:\Documents and settings\user\Application Data\Atmel\Atmel Studio\6.1\extensions\myextension

Note: The Application Data folder is initially hidden (turn on View hidden folders in Windows). The folder “extensions\MyExtension” must be created before compiling.

Figure 2-6. Project properties – VSIX tab.

Initially the extension is disabled from execution in Atmel Studio 6.1. Extensions are normally installed and the user must accept the product. If the files are copied directly (as they are when debugging), the extension must be manually enabled to be accepted.
To enable the extension:

- Start debugging in Visual Studio. Atmel Studio 6.1 will start up with the extension files copied
- Open the Extension Gallery from Tools menu in Atmel Studio 6.1. Select Installed Extensions and the extension.

Figure 2-7. Enable extension.

- Click Enable. Atmel Studio 6.1 will now suggest restarting itself. Do not select that option. Restart will just start Atmel Studio 6.1 outside of the debugging environment
- Close Atmel Studio 6.1 and start the debugging session in Visual Studio again – you will now be able to debug your extension

Note: If you get a warning on MPF, answer Enable anyway. See Section 2.2.1.2.

2.2.4 Package the extension for distribution

Extensions are distributed as a file of type ".vsix". The VSIX file is the unit of deployment for a Visual Studio 2010 Extension. Visual Studio will recognize the VSIX extension and install the contents of the file to the right location.

This is a zip-file with a defined content. Double-clicking it starts the Visual Studio Package installer. It determines what version of Visual Studio and Atmel Studio the package supports and copies the files to the correct location.

Although a VSPackage project creates a ".vsix" file as output, this might not always work properly. It is recommended to make a separate VSIX project. Do the following:

1. Add New Project to the solution.
2. Select Visual C# -> Extensibility -> VSIX Project.
3. Write the name of the project, e.g. MyExtensionInstall, click OK.
4. The project is created and opens the source.extension.vsixmanifest page.
5. Add Author (required) and set information like License Terms and Icon for the product.
6. Press Select Editions to add support for AtmelStudio.
7. Remove the checkmarks under Visual Studio 2010 except the top node.
9. Add "AtmelStudio,6.1" and press OK.
10. Press Add Content.
11. Select Content Type **VS Package**.
12. Select **Project** and the project(s) to add. Click **OK**. Do this once for each project that you want to add. **Do not add the VSIX project.**
13. Build the solution.

In the output folder (defined in the project options) for the VSIX project a ".vsix" file is created. Double-clicking this will install the extension to Atmel Studio 6.1. This file can be uploaded to Atmel Gallery for distribution.

### 2.3 Using the Visual Studio SDK

#### 2.3.1 How to make GUI extensions

See the Microsoft® Walkthroughs for Commands, Menus, and Toolbars (http://msdn.microsoft.com/en-us/library/bb165440(v=vs.100).aspx) for how to add menus, toolbars etc.

To enable the extension for Atmel Studio 6.1, see Section 2.2.4.

#### 2.3.2 How to use automation for basic run control

This section builds on the previous created extension project, see Section 2.2. It is shown here how to add listener for debugger events, perform basic run control and read/write device memory.

At this point we have a default extension with a menu and a tool window. The file MyControl.xaml.cs looks like this:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
```
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace MyCompany.MyExtension
{
    /// <summary>
    /// Interaction logic for MyControl.xaml
    /// </summary>
    public partial class MyControl : UserControl
    {
        public MyControl()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show(string.Format(System.Globalization.CultureInfo.CurrentUICulture, "We are inside {0}.button1_Click()", this.ToString()), "My Tool Window");
        }
    }
}

To display the default window included so far in the project, do this:

1. In your Visual Studio project (e.g. MyExtension) choose **Start debugging**.
2. Atmel Studio 6.1 will start and load the extension for debugging. In Atmel Studio 6.1, select **View -> Other Windows -> My Tool Window**.

Figure 2-9. **My Tool Window.**
We now want to add some functionality. Use automation to add a listener to when the debugger breaks. The following listing shows the new MyControl.xaml.cs after doing changes:

```csharp
using System.Windows;
using System.Windows.Controls;

namespace MyCompany.MyExtension
{
    using EnvDTE;
    using Microsoft.VisualStudio.Shell;

    /// <summary>
    /// Interaction logic for MyControl.xaml
    /// </summary>
    public partial class MyControl : UserControl
    {
        private readonly DebuggerEvents debuggerEvents;

        public MyControl()
        {
            InitializeComponent();
            var myDte = (DTE)Package.GetGlobalService(typeof(DTE));
            this.debuggerEvents = myDte.Events.DebuggerEvents;
            this.debuggerEvents.OnEnterBreakMode += this.DebuggerEventsOnEnterBreakMode;
        }

        private void DebuggerEventsOnEnterBreakMode(dbgEventReason Reason, ref dbgExecutionAction ExecutionAction)
        {
            //
        }

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

We now want to see our new code in action, i.e. trigger our listener when a break event occurs in Atmel Studio 6.1.

1. Set a breakpoint on the function `DebuggerEventsOnEnterBreakMode()`.
2. Start debugging of `MyExtension`.
3. Create an Atmel Studio 6.1 project (if you don’t know how to do this – see Getting Started creating a project - http://www.atmel.no/webdoc/atmelstudio/atmelstudio.AVRStudio.GettingStarted.Newbie.CreateAndRun.html).
5. Select `View -> Other Windows -> My Tool Window` in Atmel Studio 6.1 to activate your extension window.
6. Select `Debug -> Continue` and then `Debug -> Break All`.
7. Breakpoint in Visual Studio is hit.

By default packages are not loaded until they are used. This can be overridden in the package main file:

In the above example add this to MyExtensionPackage.cs.

Add: `[ProvideAutoLoad(UIContextGuids.NoSolution)]`

Before: `public sealed class MyExtensionPackage : Package`

Now we want to use automation for run control, e.g. step into when clicking on the button:

Replace the code:

```csharp
private void button1_Click(object sender, RoutedEventArgs e)
{
}
```
private void button1_Click(object sender, RoutedEventArgs e)
{
    // Get Visual Studio automation object model
    var myDte = (DTE)Package.GetGlobalService(typeof(DTE));

    // Only StepInto if debugger is in breakmode
    if (myDte.Debugger.CurrentMode == dbgDebugMode.dbgBreakMode)
    {
        myDte.Debugger.StepInto();
    }
}

To see the effect of the change:
1. Start Atmel Studio 6.1 from Visual Studio project.
2. Create an Atmel Studio 6.1 project.
3. Enter debug mode in Atmel Studio 6.1 normally.
4. If not visible, make the extension window visible select View -> Other Windows -> My Tool Window.
5. Click on the Click me button.
6. Atmel Studio 6.1 performs a single step.

Note: If no stepping occurs, try to turn Optimization Off in the Atmel Studio 6.1 project settings.

2.3.3 How to use expression evaluation

Expression evaluation gives you the possibility to e.g. look at variable myvar:

Expression expression = myDte.Debugger.GetExpression("myvar");

Refer to Visual Studio help (F1) for more information about Expression.

2.4 Using the Atmel Studio IDE SDK

2.4.1 How to read/write memory

This section builds on the previously created extension project. See also Section 2.1.2 for examples using the Atmel Studio IDE SDK.

Use this code in MyControl.xaml.cs to read the EEPROM memory from the device:

using System.Windows;
using System.Windows.Controls;
namespace MyCompany.MyExtension
{
    using System.Collections.Generic;
    using System.Linq;
    using Atmel.Studio.Services;
    using EnvDTE;
    using Microsoft.VisualStudio.Shell;
    /// <summary>
    /// Interaction logic for MyControl.xaml
    /// </summary>
    public partial class MyControl : UserControl
    {
        // Debugger events from Visual Studio SDK
        private readonly DebuggerEvents debuggerEvents;

        // Target service from Atmel Studio SDK
        private readonly ITargetService2 targetService;
public MyControl()
{
    InitializeComponent();

    // Handle to The top-level object in the Visual Studio automation object model.
    var myDte = (DTE)Package.GetGlobalService(typeof(DTE));

    if (myDte == null)
    {
        return;
    }

    // Get handle to debugger events
    this.debuggerEvents = myDte.Events.DebuggerEvents;
    if (this.debuggerEvents == null)
    {
        return;
    }

    // Add delegate for enter break mode event in debugger
    this.debuggerEvents.OnEnterBreakMode += this.DebuggerEventsOnEnterBreakMode;

    // Get handle to target service
    this.targetService = ATServiceProvider.TargetService2;
}

private void DebuggerEventsOnEnterBreakMode(dbgEventReason reason, ref dbgExecutionAction executionAction)
{
    if (this.targetService == null) return;

    // Get reference to launched target
    ITarget2 target = this.targetService.GetLaunchedTarget();

    // Get reference to address space eeprom
    IAddressSpace addressSpace = target.Device.GetAddressSpace(MemoryTypes.Eeprom);

    if (addressSpace != null)
    {
        // Get correct name of eeprom memory for current device to use in call to read memory
        string addressSpaceName = target.GetAddressSpaceName(MemoryTypes.Eeprom);

        // Array for storing errors when reading memory.
        // Each read byte with error gets a new entry.
        MemoryErrorRange[] errorRange;

        // Read memory bytes; type, start address, wordsize, number of bytes to read, mode, errordata
        byte[] result = target.GetMemory(
            addressSpaceName,
            addressSpace.Start,
            1,
            (int)addressSpace.Size,
            0,
            out errorRange);
    }
}

private void button1_Click(object sender, RoutedEventArgs e)
{
    // Handle to The top-level object in the Visual Studio automation object model.
    var myDte = (DTE)Package.GetGlobalService(typeof(DTE));

    if (myDte == null)
    {
        return;
    }
```csharp
// Only StepInto if debugger is in breakmode
if (myDte.Debugger.CurrentMode == dbgDebugMode.dbgBreakMode)
{
    myDte.Debugger.StepInto();
}
}
}

The following describe how to write the value 22 to byte number 100 in EEPROM:

```csharp
string addressSpaceName = target.GetAddressSpaceName(MemoryTypes.Eeprom);
var data = new byte[22];
ulong startAddress = 100;

int byteCount = 1;
int wordSize = 1;
int writeMode = 0;
IStatus status;

// Write a byte to memory, type, address, size, number of bytes, mode, bytearray, status)
// Each byte write that result in error will be listed in errorBytes.
List<string> errorBytes = target.SetMemory(addressSpaceName, startAddress, wordSize, byteCount, writeMode, data, out status);
```

The following modes are supported by GetMemory() and SetMemory():

- normal
- continue on error
- verify after reading

For a complete example using memory read and write, see the MemoryList example (see Section 2.1.2).

### 2.4.1.1 Naming of memories on a device

To read or write to memory, the memory type must be given. Memory types differ between the devices. Device memory is defined as a collection of address spaces. Each address space can be divided into memory segments.

- **Address space**
  - Memory segment
  - Memory segment

ATmega88 looks like this:

- **Prog**
  - Flash
  - BOOT_SECTION_1
  - BOOT_SECTION_2
  - BOOT_SECTION_3
  - BOOT_SECTION_4

- **Signatures**
  - Signatures

- **Fuses**
  - Fuses

- **Lockbits**
• Lockbits
• Data
  • Registers
  • IO
  • RAM
• EEPROM
  • EEPROM

Get all address spaces:

```csharp
IList<IAddressSpace> addressSpaces = target.Device.AddressSpaces;
```

Get memory segments for an address space:

```csharp
IList<IMemorySegment> memorySegments = target.Device.MemorySegments;
```

Address space names used to call memory functions differs across devices. To get the name call GetAddressSpaceName() on the active target:

```csharp
string addressSpaceName = target.GetAddressSpaceName(MemoryTypes.Eeprom);
```

Only address space names are used as memory type, memory segments are defined by a start address and a size inside the memory space.

### 2.5 Distributing extensions on Atmel Gallery

Refer to the XDK – How to submit an extension to Atmel Gallery on http://gallery.atmel.com/Partner.

### 2.6 Atmel Studio IDE SDK references

- Visual Studio 2010 SDK
- Note: If SP1 of Visual Studio is installed use the SDK for SP1
- Extend Visual Studio
- Get Started with Extending Visual Studio
- Creating a VSPackage
- Videos
- Forum
- Creating a package
- Getting Started creating a project in Atmel Studio
3. **The Embedded SDK**

3.1 **Introduction**

This section is targeting those who want to extend the Atmel Studio Integrated Development Platform with embedded software and prepare them for submission to the Atmel Gallery. It describes how to create an embedded application extension or an embedded module extension illustrating the possibilities offered by the Atmel Studio Embedded SDK.

After reading this section the user should be able to do the following:

- Write an embedded application extension and prepare it for upload to the Atmel Gallery
- Write an embedded module/library extension and prepare it for upload to the Atmel Gallery

The reader is expected to be familiar with embedded systems and the C language.

3.1.1 **Extending the Embedded Software Realm**

The Atmel Software Framework (ASF) is closely integrated with Atmel Studio 6 and delivers a collection of production-ready source code such as drivers, communication stacks, graphic services and touch functionality. Over 1,600 project examples, built on ASF can be selected and configured in Atmel Studio 6.

The Embedded SDK is a mean to extend the default embedded software realm represented by the ASF by offering the possibility to create embedded applications and embedded modules/libraries in an extension format ready for submission to the Atmel Gallery. These extensions may be based on the ASF or not and may have dependencies to other extensions or not.

The Embedded SDK provides interfaces and tools:

- To package an embedded application project into an Atmel Gallery extension format
- To package an embedded module/library project into an Atmel Gallery extension format
- To seamlessly browse and install extensions from the Atmel Gallery
- To seamlessly use locally installed extensions into an embedded project

3.2 **Embedded Application extensions**

Embedded Application extensions are packaged in a vsix format. In this section we are focusing on creating, using and submitting an embedded application extension to the Atmel Gallery.

3.2.1 **How to Create an Embedded Application Extension**

The starting point is to create an Atmel Studio project. There are several ways to do that whether the project should use the ASF or not: in all cases the New Project Wizard should be used.

1. Start Atmel Studio 6 and select **New Project** or **New Example Project from ASF or Extensions**.
   a) In the case of **New Project**, select an Atmel board or a user board or an Arduino board the empty project will be based on or select a plain empty new project (no ASF support in the latter).
Figure 3-1. ASF-based Empty New Project Dialogue (choose one of {Atmel board, User board (empty customizable board), Arduino board}).

Figure 3-2. Empty New Project Dialogue (no ASF support).

b) In the case of **New Example Project from ASF or Extensions**, select an example.

Figure 3-3. New Example Project from ASF Dialogue.
2. Develop the application from scratch or by customizing the Example project. In this phase, existing extensions may be added to the project (more on this later) or ASF modules may be imported into the project.

3. Once the application is ready for release, create a VSIX extension package.
   
a) It is important to mention that one extension = one version = one example. Exporting a solution with several projects is currently not supported.

b) Select the project and choose Export as an example extension (.vsix). Atmel Studio will automatically launch a build of the associated project: if the build fails, the export will be canceled.

Figure 3-4. Export Action.

   
c) Fill-in the extension manifest. The data entered here will be displayed in the Atmel Gallery as first hand information on the extension so they should be carefully specified. Note that these can be edited later if required. The fields of the manifest are:

   i. **Product Name**: this will be the name of the extension as displayed in the Atmel Gallery.

   ii. **Short Name**: a short form of the Product Name.

   iii. **Author**: a company or individual name.

   iv. **Org**: a short form of Author, used for representation in the New Example Project dialog.

   v. **Version**: the version of the extension.

   vi. **Description**: short description of the main features of this extension. Mention also the supported part number(s), the board it is running on, the compiler it supports.

   vii. **License Term**: by default, the vsix will contain an empty License.txt file which can be edited (more on this later). Another file in text format can be specified here. The content of this field will be displayed upon extension installation.

   viii. **Icon**: icon that will be associated with this extension in the Atmel Gallery. If not specified, a default extension icon will be used.

   ix. **Preview Image**: image representing this extension.

   x. **More Info Url**: a URL where more info on the extension may be found.

   xi. **Getting Started Guide**: a URL to the Getting Started guide for this extension.

   xii. **All Users**: controls whether this extension will be displayed for all users or only for the current user. Default is set to public (all users). This control applies only for Atmel Studio and is valid only locally (i.e. on the computer where the extension is installed).
xiii. **Id**: this is pre-filled by Atmel Studio: it is not supposed to be changed and is designed to be unique. This field may be edited an if ID conflict happen. The unique ID is used by Atmel Gallery to identify the extension. Whenever an extension update is published, the ID must never change.

xiv. Further information can be added by clicking on the “More…” button. The following three fields are available for edit:

1. **Kit**: kits associated with this example, used in the “Kits” filter view of the new Example Project dialog of Atmel Studio. Multiple values can be separated with commas.

2. **Category**: categories associated with this example, used in the “Category” filter view of the New Example Project dialog of Atmel Studio. Multiple values can be separated with commas.

3. **Technology**: Technologies associated with this example, used in the “Technology” filter view of the New Example Project dialog of Atmel Studio. Multiple values can be separated with commas.

**Figure 3-5. The Manifest Form and its “More…” pop-up.**

d) When done editing the manifest, click **Ok**. Atmel Studio will build the vsix extension based on the project and the Manifest. Check the result of that build in the Output window (choose Output from “Extension Packaging”); this output also provides the exact location where the vsix extension has been saved. This operation will create in all cases a vsix folder that contains a license text file and the extension.fdkmanifest file. If the vsix packaging succeeded this will also contain an xml file called asf.xml.
i. If the build failed or if further editing is required, the manifest may be edited again by double-clicking on extension.fdkmanifest: this will re-open the Manifest form. Clicking on Ok will not trigger a vsix extension build: the action Export as an example extension (.vsix) shall be launched again to effectively build the extension after the editing has been done. The same applies for the license text: double-click on the license.txt file to edit it with the Atmel Studio editor, save it then execute the action Export as an example extension (.vsix) again to rebuild the extension.

Once the extension packaging is successful, note the vsix extension package location: this file will need to be directly accessed to install or publish that extension.
3.2.2 How to use an Embedded Application Extension

The vsix format used to package an embedded application extension is a Microsoft Visual Studio 2010 Extensions file format. Visual Studio will recognize the vsix extension and install the contents of the file to the right location. Once an Embedded Application Extension has been installed, it is possible to create a new project from that extension.

1. Install the vsix extension. Locate the extension file and double-click on the vsix file. The Visual Studio Extension Installer does the installation for Atmel Studio and displays the content of the license.txt file provided when the manifest was filled in during the extension creation phase.

2. Click on **Install** to install the extension in Atmel Studio. Note that Atmel Studio must be closed then restarted for the extension installation to be effective.
3. Launch Atmel Studio. Select **New Example Project from ASF or Extensions**. The just installed extension appears in the list of example projects next to the ASF or other extensions. Since the projects are organized by Company name at the top, the newly added extension will appear under the name chosen for the field Author in the extension Manifest form. Select the application provided by the extension and click **OK**. If the license file is not empty, Atmel Studio will display a license agreement window that the user must accept before going any further.

4. This newly created project is now usable as with any other Atmel Studio project.

### 3.2.3 How to publish an Embedded Application Extension

Each Embedded Application Extension can apply for publication on the Atmel Gallery. See the application note XDK – How to submit an extension to Atmel Gallery on [http://gallery.atmel.com/Partner](http://gallery.atmel.com/Partner).

### 3.2.4 Embedded SDK configuration

The Embedded SDK engine configuration is available under **Tools → Options → Atmel Software Framework → XDK Settings**.

The available configuration is:

- **AutoGenerate asf.xml**: enabled by default. When set to False, the extension packaging will exclusively rely on asf.xml files added by the developer of the extension.
3.3 Embedded Module/Library Extensions

An Embedded Module Extension (aka Embedded Library Extension) is a set of files or pre-compiled libraries that may be used by Executable projects. Embedded Module/Library extensions are packaged in a vsix format. In this section we are focusing on creating, using and submitting an embedded module/library extension to the Atmel Gallery.

*This section will be updated by a new XDK release. For short term needs, contact us at gallery@atmel.com.*
4. **Submit extensions to Atmel Gallery**

The Atmel Gallery is targeted at developers who want to provide integrations to the Atmel Studio platform in the form of development tools or embedded software. With Atmel Gallery, engineers who develop with Atmel MCUs can easily extend the Atmel Studio development platform with just the right tool or software library for their projects.

As an Atmel Gallery developer, you can integrate your tool or embedded software with Studio and make it available, for free download or commercial purchase, to the over 100,000 users of Atmel Studio.

This section describes how to upload an extension via the Atmel Gallery, once you have signed up as a developer.

The Atmel Gallery is subject to Atmel moderation. Submitted extensions will be verified for install, uninstall, and program execution without error messages, program aborts, or other unwanted behavior.

Check the “How to submit extensions to Atmel Gallery” document on [http://gallery.atmel.com/Partner](http://gallery.atmel.com/Partner).
5. References and further information

5.1 Atmel Gallery

- http://gallery.atmel.com/

5.2 Atmel Studio


5.3 ASF


5.4 Support

If you have any questions you cannot locate the answer to in the above description, or if you require technical assistance, please contact us through gallery@atmel.com.
## Revision History

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>42050C</td>
<td>02/2013</td>
<td>Added embedded XDK section. Moved how to upload an extension and how to become a XDK developer to separate documents.</td>
</tr>
<tr>
<td>42050B</td>
<td>11/2012</td>
<td>Added how to upload an extension information</td>
</tr>
<tr>
<td>42050A</td>
<td>11/2012</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>