APPLICATION NOTE

Atmel AVR10006: XDK – User Guide

Atmel Microcontrollers

Features

- Extension Developer's Kit user guide
- Generate and publish an Atmel® Studio extension
- 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 and how to become an Atmel Gallery developer can be found at:

http://gallery.atmel.com

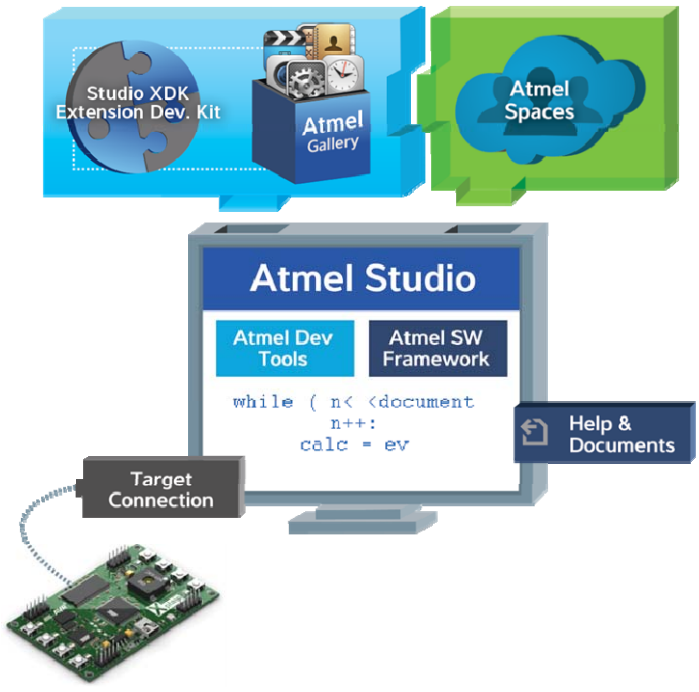


Table of Contents

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 below) which is targeted to provide embedded developers with all the development tools and embedded software needed 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.

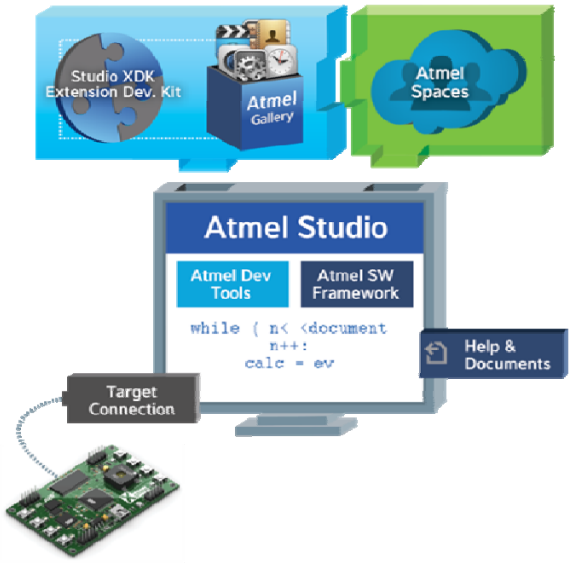
The Atmel Software Framework (ASF) is closely integrated with Atmel Studio 6 and delivers a collection of production-ready source code such as drivers, communication stacks, graphic services and touch functionality. Over 1,600 project examples, built on ASF can be selected and configured in Atmel Studio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

The Atmel Gallery apps store provides development tools and embedded software for MCU-based application design that extend 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 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.



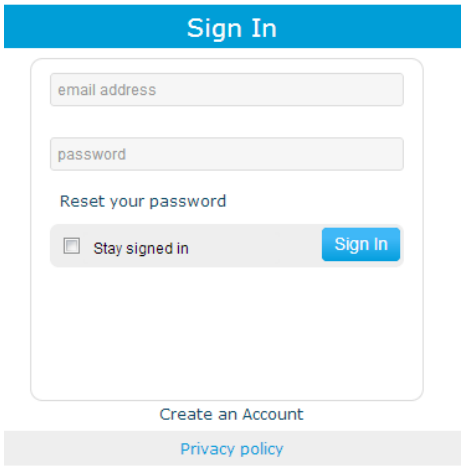
1.2 Installation

1.2.1 Developer registration

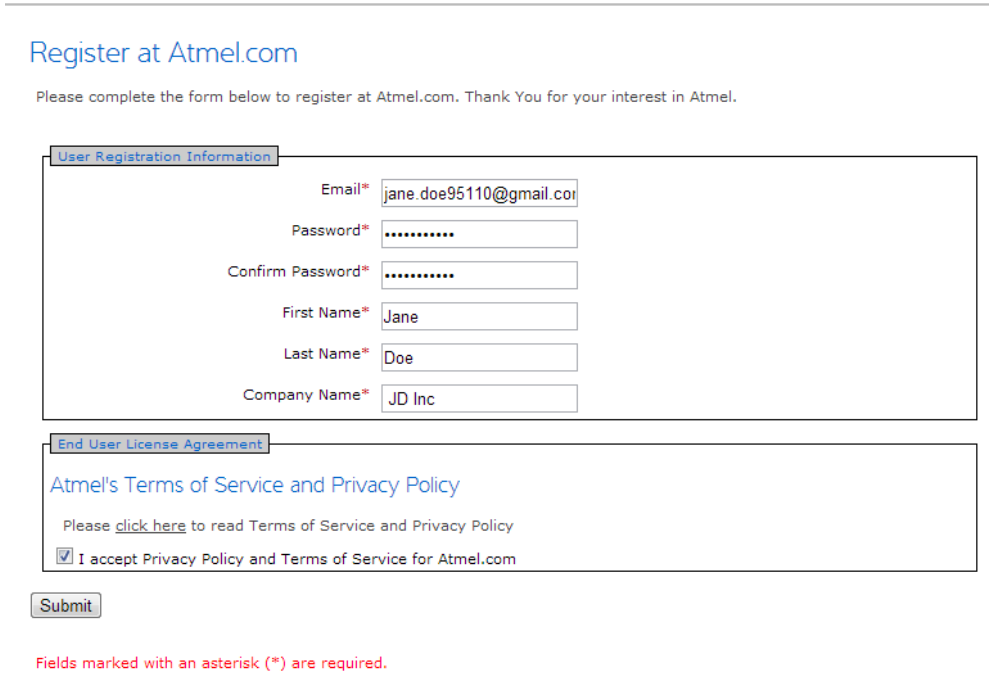
This section describes how you can get started with distributing your Atmel Studio-related extensions via the Atmel Gallery. The process is the same whether you are offering free, trial or commercial extensions.

Before you can start uploading your extensions to Atmel Gallery, you must have a user account:

- Sign in at http://gallery.atmel.com
- Click on “Create an Account”



This takes you to the registration form on atmel.com. Complete the required information and accept the Atmel End User License Agreement (EULA).

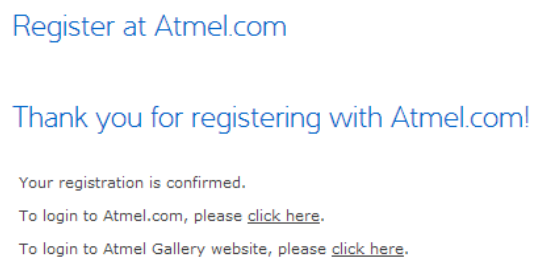


Within a few minutes, you should receive an email asking you to confirm your subscription. If you do not receive this email, please check your junk filter. If you cannot find it there either, go to http://www.atmel.com -> Buy -> Atmel Gallery, enter your email and password and click sign in. This will resend the confirmation email.

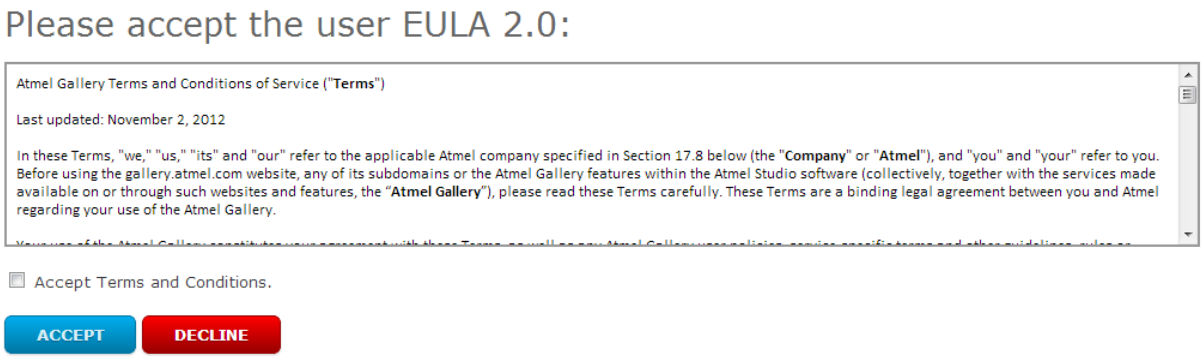
If you still do not receive a confirmation email, please contact us at gallery@atmel.com.

If you have already signed up for the Extension Manager in Atmel Studio, you can use the same email address and password to sign in for Atmel Gallery. The same goes for users who have an account for the Atmel Samples Center. These credentials are also valid for Atmel Gallery.

Click the link in the confirmation email. This directs you back to atmel.com where you need to click the link for login to Atmel Gallery website. The link leads you back to http://gallery.atmel.com where you enter your email and password to sign in.

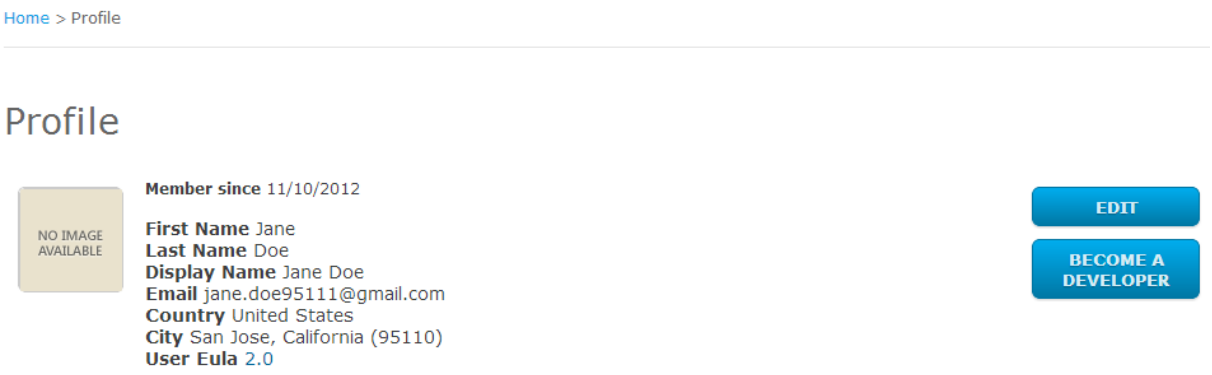


Accept the Atmel Gallery EULA and click Submit. Depending on your country of residence, you might be required to complete your address details before you are able to submit.

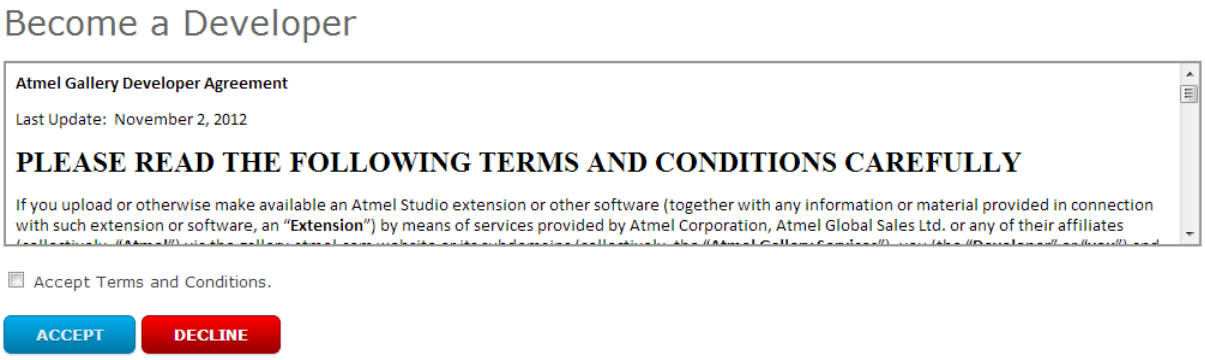


Once you have completed your login, go to Profile.

Click the Become a developer button.



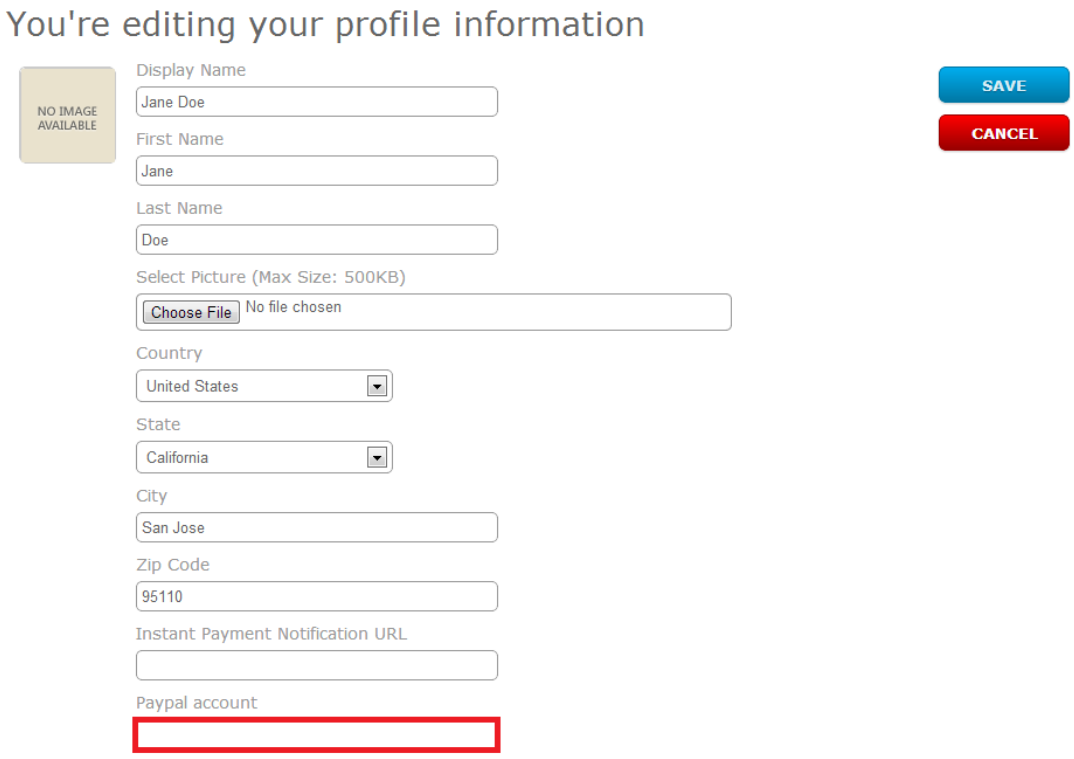
Accept the Atmel Gallery Developer Agreement.



Your developer account will be pending for Atmel approval. Once the moderator has reviewed and approved the account, we will send you a notification email. You can expect to receive the approval within five business days.

When you return to your account profile in Atmel Gallery, http://gallery.atmel.com/Profile, you will notice that more buttons have been added to the far right. From here you can edit your profile information, view payment notifications and upload your extensions.

If you are offering a commercial extension, you will need to have a PayPal™ account in order for us to make payments to you on a quarterly basis. This information must be added by editing your profile. Please ensure that the information you submit in the PayPal account field is correct before saving.



1.2.2 Software tools

1.2.2.1 Atmel Studio IDE SDK

- Atmel Studio 6 SP2: http://www.atmel.com/tools/atmelstudio.aspx
- Atmel Studio IDE SDK: 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 6.0 IDE SDK
- Visual Studio® 2010: http://msdn.microsoft.com/en-us/library/dd831853(v=vs.100).aspx

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

- Visual Studio 2010 SDK (Use Visual Studio SDK SP1 for Visual Studio 2010 SP1): http://www.microsoft.com/en-us/download/details.aspx?id=2680

1.2.2.2 Embedded SDK

Will come soon.

1.2.3 Hardware tools

1.2.3.1 Embedded SDK – supported devices

Will come soon.

1.2.3.2 Embedded SDK – supported boards

Will come soon.

2. The Atmel Studio IDE SDK

2.1 Introduction

This section is targeting those who want to extend Atmel Studio 6.0. It describes how to write a simple extension, and contains some example code illustrating the possibilities offered by the Atmel Studio 6.0 SDK.

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

- Write your own Atmel Studio 6.0 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 6.0 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 6.0 use the same tools as for extending 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\6.0\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 6.0

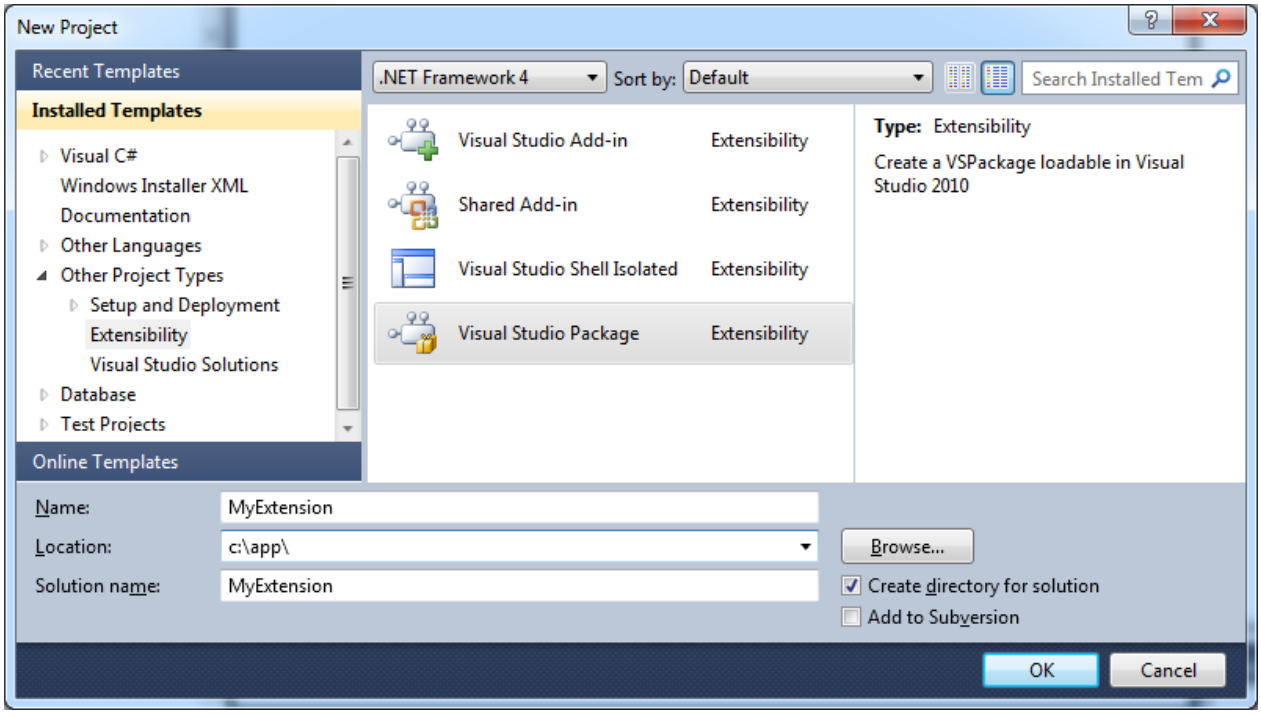
Extensions for Visual Studio are called packages (or VSPackages) and add-ins. In this document we are focusing on creating a VSPackage. The Visual Studio Package project template creates a basic VSPackage (see creating a VSPackage). The template can add code to create a menu command or a tool window – 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 6.0, do the following steps:

1. Start Visual Studio 2010 and select **New Project**.
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.



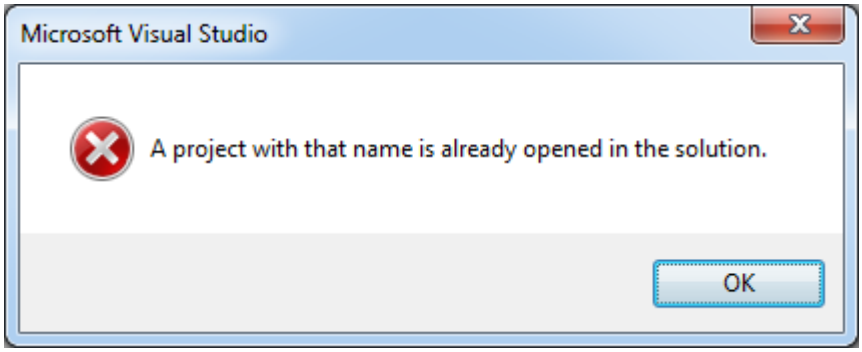
3. On the **Select a Programming Language** page, select either Visual C#®, Visual C++® or Visual Basic®. (In our examples, C# is used) Have the template generate a key.snk file to sign the assembly. Alternatively, click **Browse** to select your own key file. The template makes a copy of your key file 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 dock able 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

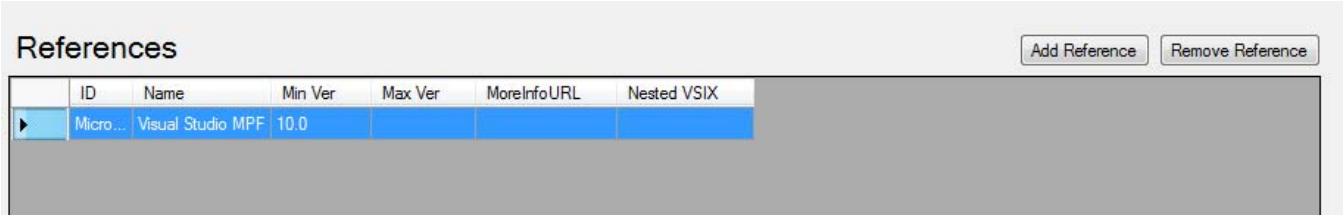
2.2.1.2 Remove reference to Visual Studio MPF

This reference is not needed for most Atmel Studio 6.0 extensions and can cause issues later.

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

This is a test.

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

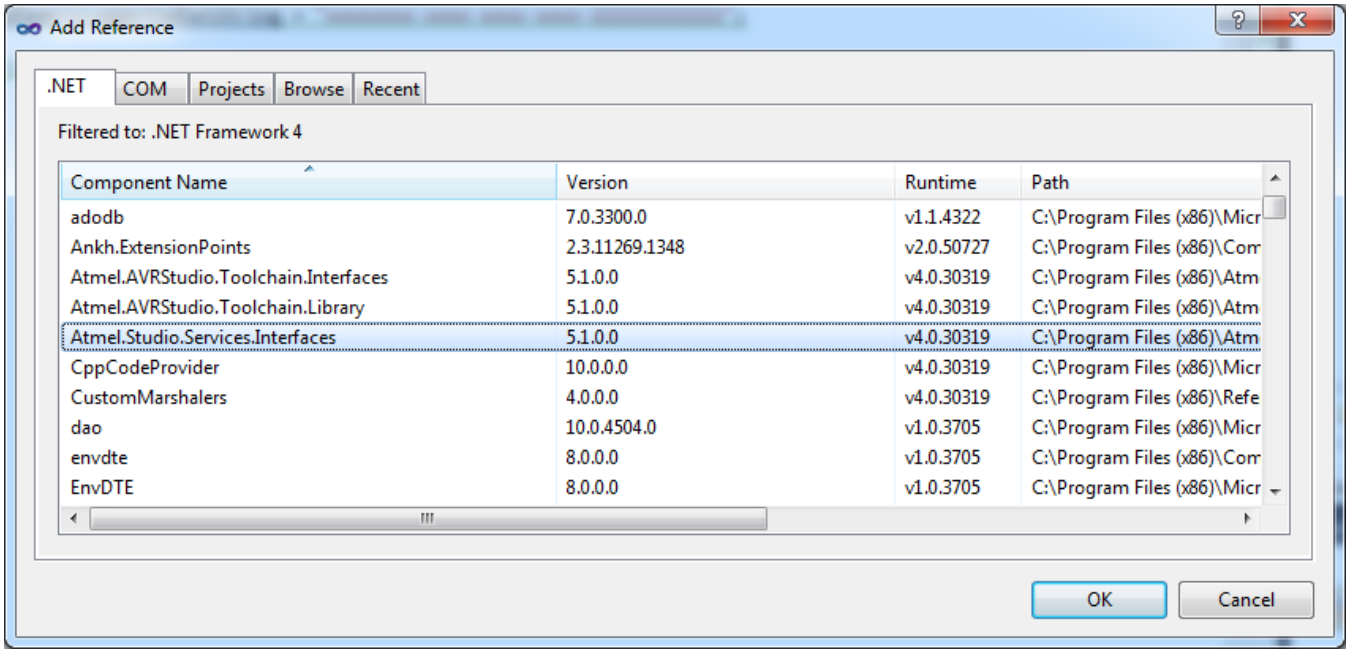


2.2.2 Add Atmel Studio IDE SDK assembly reference

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**.

Figure 2-4. Select assembly.



Select Atmel.Studio.Services.Interfaces in the .NET tab.

If you do not see this assembly, please make sure that you have installed Atmel Studio IDE SDK.

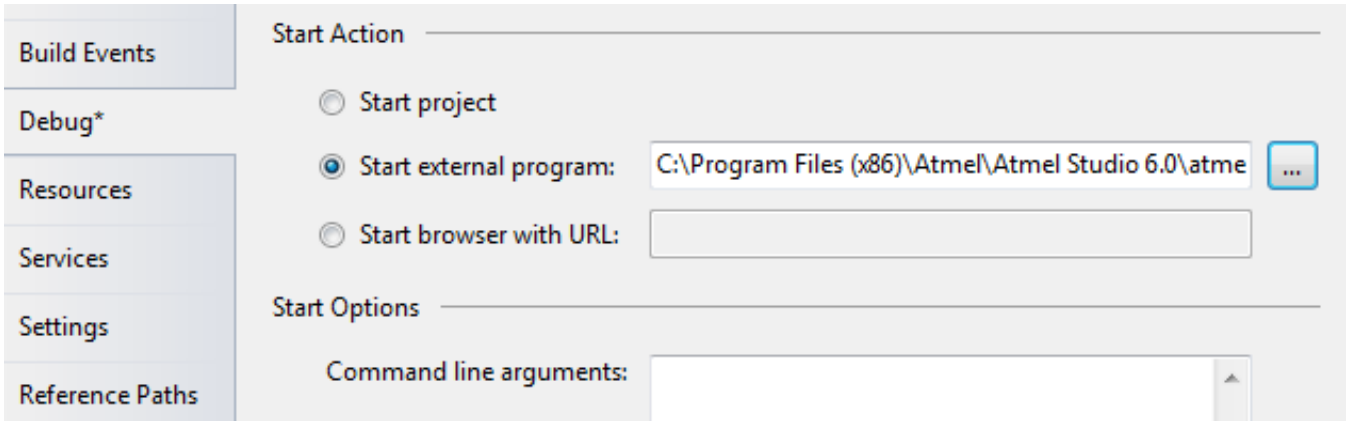
2.2.3 Set up debugging

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 6.0)

Remove any command line arguments (if present).

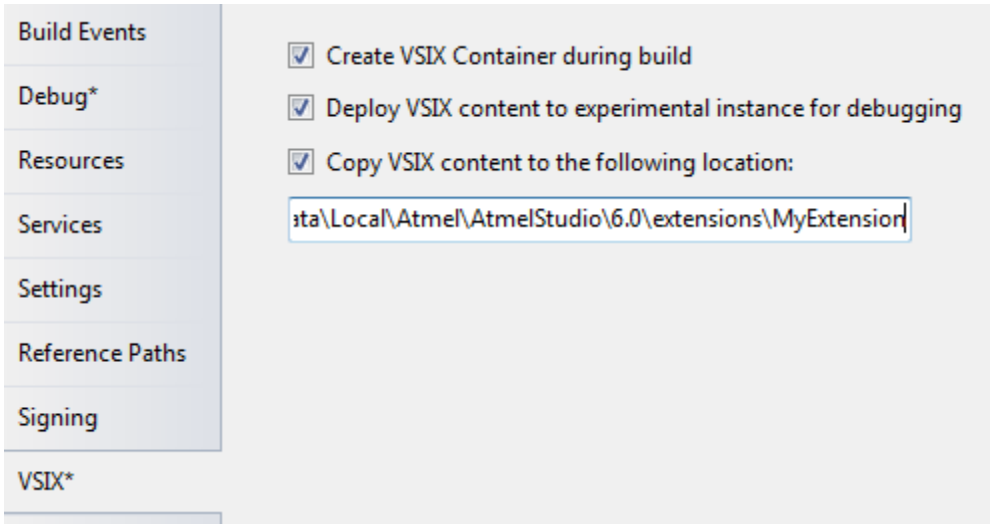
Figure 2-5. Project properties – debug tab.



- VSIX – Check Copy VSIX content to the following location:
  - For Windows Vista® / Windows 7 / Windows 8:
    C:\Users\user\AppData\Local\Atmel\AtmelStudio\6.0\extensions*myextension*
  - For Windows XP:
    C:\Documents and settings\user\Application Data\Atmel\AtmelStudio\6.0\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.

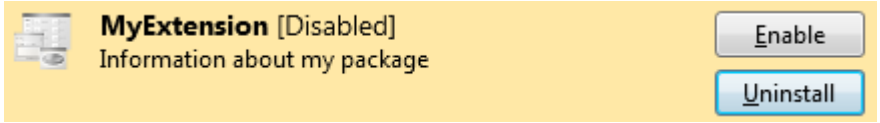


Initially the extension is disabled from execution in Atmel Studio 6.0. Extensions are normally installed and the user must accept the product. If the files are copied directly (as they are when debugging), the extension must be manually enabled to be accepted.

To enable the extension:

- Start debugging in Visual Studio. Atmel Studio 6.0 will start up with the extension files copied
- Open the **Extension Gallery** from **Tools** menu in Atmel Studio 6.0. Select **Installed Extensions** and the extension:

Figure 2-7. Enable extension.



- Click **Enable**. Atmel Studio 6.0 will now suggest restarting itself. **Do not select that option**. Restart will just start Atmel Studio 6.0 outside of the debugging environment
- Close Atmel Studio 6.0 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

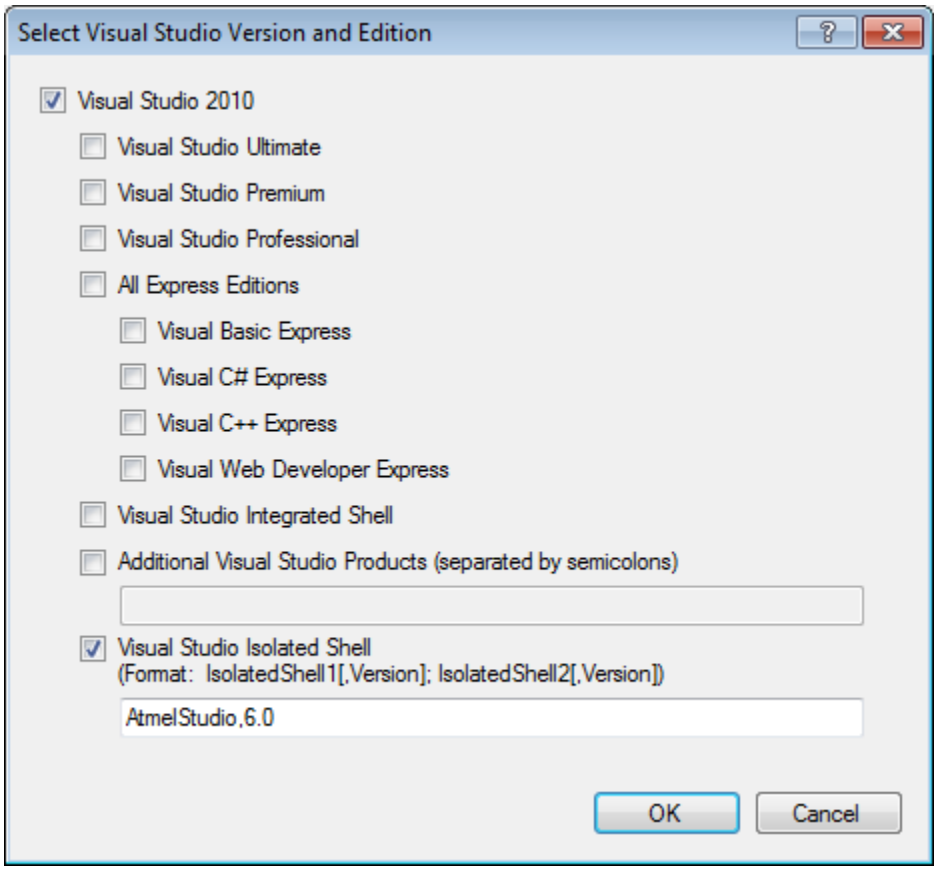
Extensions are distributed as a file of type ".vsix". The VSIX file is the unit of deployment for a Visual Studio 2010 Extension. Visual Studio will recognize the VSIX extension and install the contents of the file to the right location.

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.
8. Check **Visual Studio Isolated Shell**.
9. Add "AtmelStudio,6.0" and press **OK**.

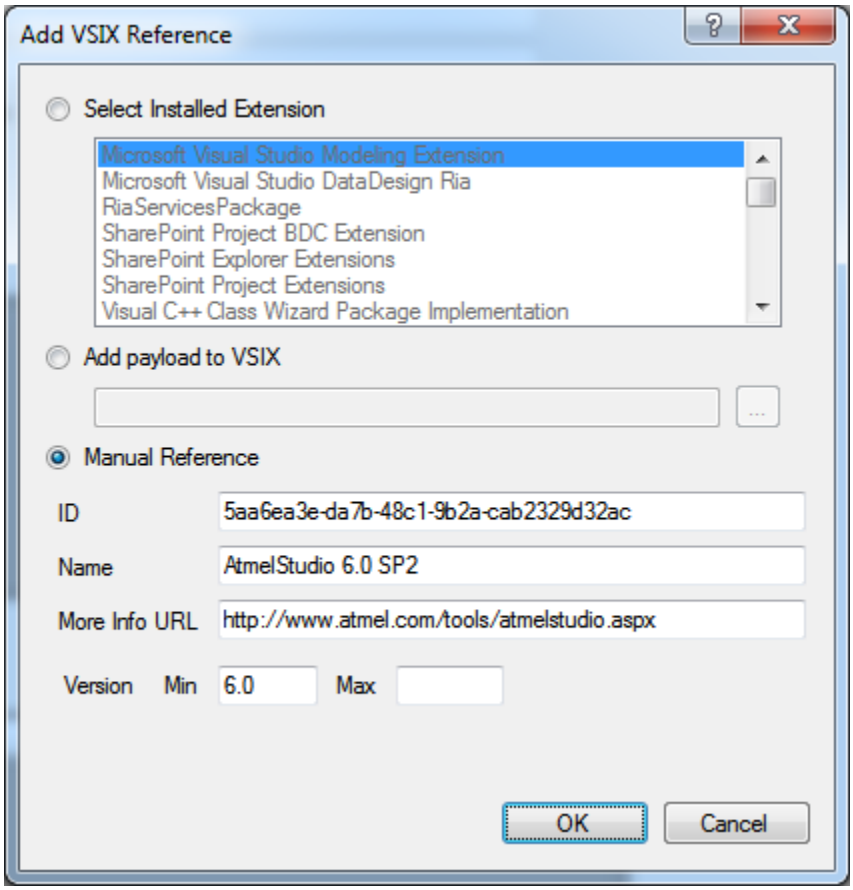
Figure 2-8. Select editions: Add support for Atmel Studio 6.0.



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. Press **Add Reference**.
14. Select **Manual Reference**.
15. Add the following to add a reference to Atmel Studio 6.0 Service Pack 2:
    ID: 5aa6ea3e-da7b-48c1-9b2a-cab2329d32ac
    Name: Atmel Studio 6.0 SP2
    More Info URL: http://www.atmel.com/tools/atmelstudio.aspx
    Version min.: 6.0

Figure 2-9. Add reference to Atmel Studio 6.0 SP2.



16. Build the solution.

The manual reference to Atmel Studio 6.0 SP2 is to ensure that that the end user have SP2 installed (which is required for Atmel Studio 6.0 IDE SDK extensions).

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 6.0. 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 for how to add menus, toolbars etc.

To enable the extension for Atmel Studio 6.0, 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.

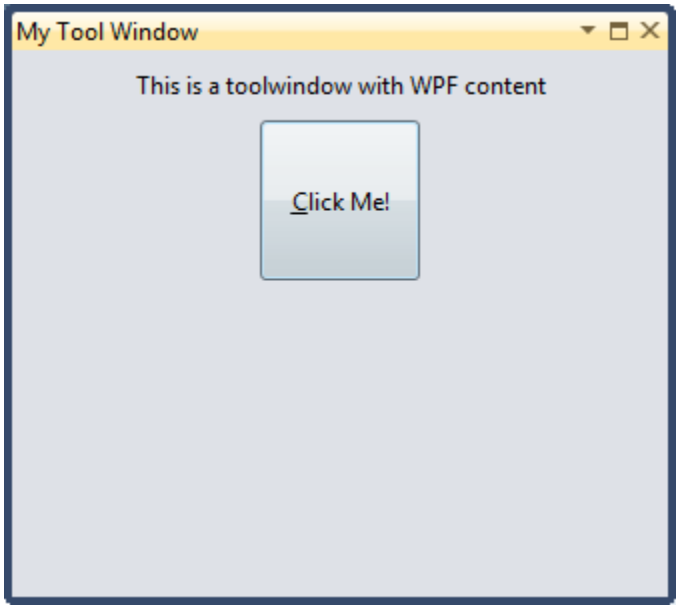
At this point we have a default extension with a menu and a tool window. The file MyControl.xaml.cs looks like this:

```
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.Media.Imaging;
using System.Windows.Navigation;
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 6.0 will start and load the extension for debugging. In Atmel Studio 6.0, select **View -> Other Windows -> My Tool Window**.

Figure 2-10. 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:

```
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 6.0.

1. Set a breakpoint on the function **DebuggerEventsOnEnterBreakMode()**.
2. Start debugging of *MyExtension*.
3. Create an Atmel Studio 6.0 project (if you don't know how to do this – see Getting Started creating a project).
4. Select **Debug -> Start Debugging and Break** in Atmel Studio 6.0.
5. Select **View -> Other Windows -> My Tool Window** in Atmel Studio 6.0 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:

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

}
```

With:

```
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 6.0 from Visual Studio project.
2. Create an Atmel Studio 6.0 project.
3. Enter debug mode in Atmel Studio 6.0 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 6.0 performs a single step.

Note: If no stepping occurs, try to turn Optimalization Off in the Atmel Studio 6.0 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 Atmel.Studio.Services.Device;

  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 exist, read eeprom values
      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;
      }

      // 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:

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

byte value = 22;
var data = new byte[1];
data[0] = (byte)value;
ulong address = 100;

int count = 1;
int size = 1;
int mode = 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,
                                    address,
                                    size,
                                    count,
                                    mode,
                                    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:

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

Get memory segments for an address space:

```
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:

```
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 Section Submit Extensions to Atmel Gallery.

2.6 Atmel Studio IDE SDK references

- Visual Studio 2010 SDK: http://www.microsoft.com/en-us/download/details.aspx?id=2680

Note: If SP1 of Visual Studio is installed use the SDK for SP1

- Extending Visual Studio http://msdn.microsoft.com/en-US/vstudio/ff718165.aspx
- Get Started with Extending Visual Studio: http://msdn.microsoft.com/en-us/vstudio/ff677564
- Creating a VSPackage: http://msdn.microsoft.com/en-us/library/cc138589(v=vs.100).aspx
- Videos: http://msdn.microsoft.com/en-us/vstudio/gg132841
- Forum: http://social.msdn.microsoft.com/forums/en-US/vsx/threads/
- Creating a package: http://msdn.microsoft.com/en-us/library/bb164725(v=vs.100).aspx
- Getting Started creating a project in Atmel Studio: http://www.atmel.no/webdoc/atmelstudio/atmelstudio.AVRStudio.GettingStarted.Newbie.CreateAndRun.html

3. The Embedded SDK

This section will be updated by a new XDK release in January 2013. 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.

Web site: http://gallery.atmel.com/Products/Partners.

4.1 Uploading the extension

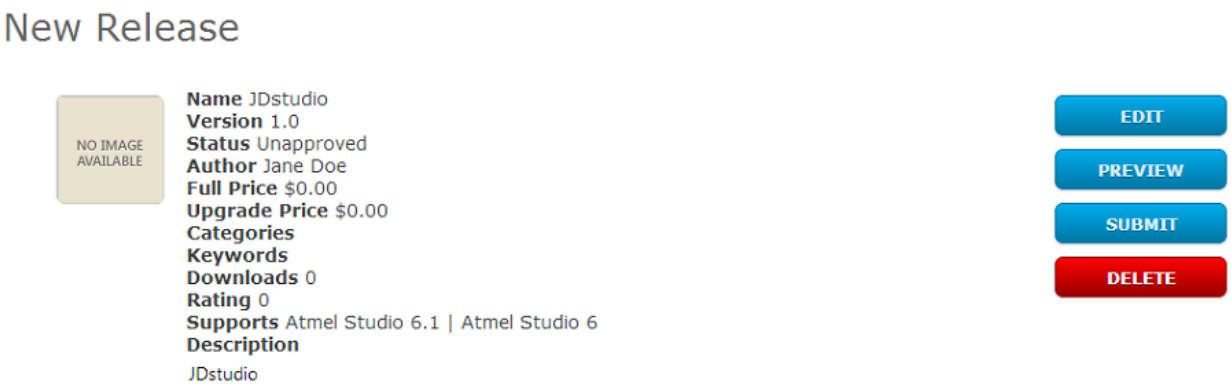
To upload a new extension, go to Profile in the Atmel Gallery banner, and click on MY PRODUCTS.

From here you can choose whether you want to upload as .vsix, .msi, or some other file type. We recommend that you use a .vsix file because Extension Manager in Atmel Studio can recognize it, download it, and install it correctly. For large files however, we recommend to package your extension as a .msi install file. MSI files will download from the Gallery into Studio using the user's default browser download capability and will not block Studio 6 during the download. The "Other" category should be used for any other content you would like to add, e.g. documentation, and these files could be .exe, .pdf, .zip, etc.

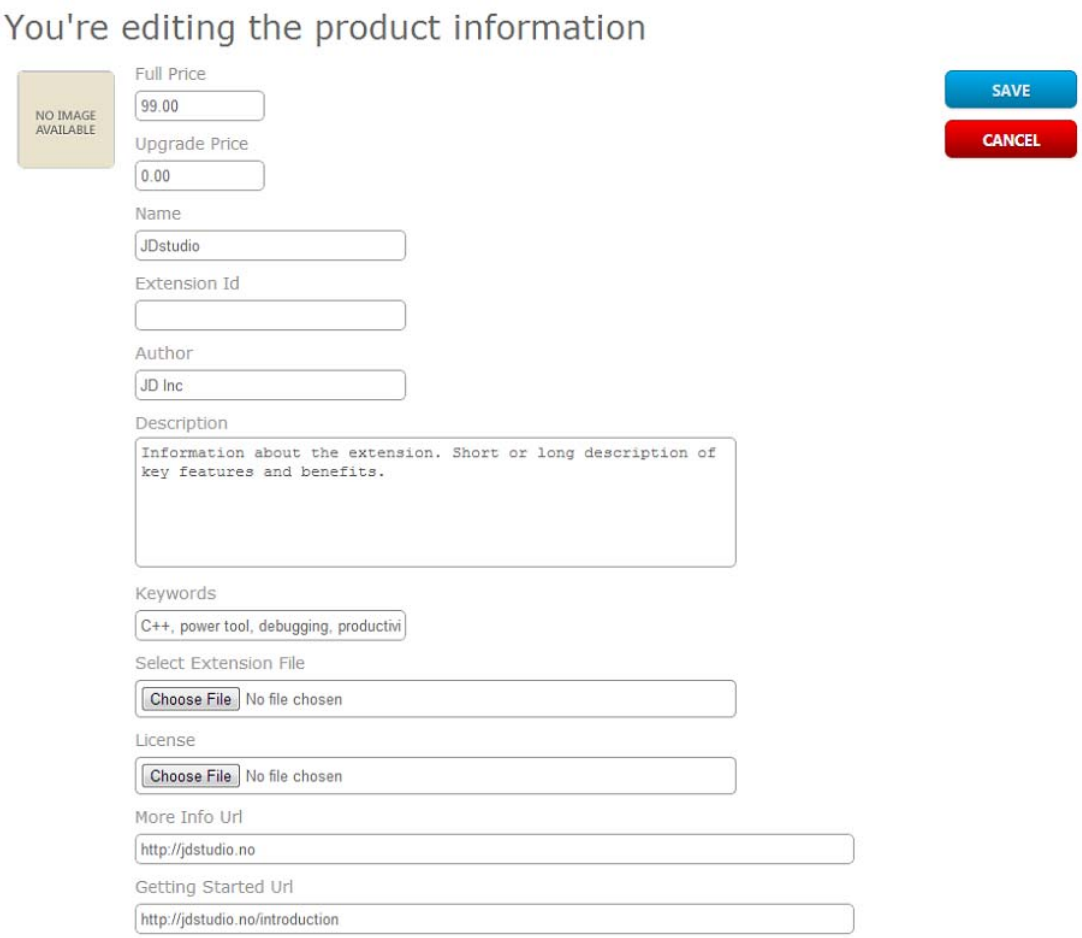
Browse to find the file you want to upload, click Submit.

When submitting .msi and other file types, it is mandatory to fill out the Version field.

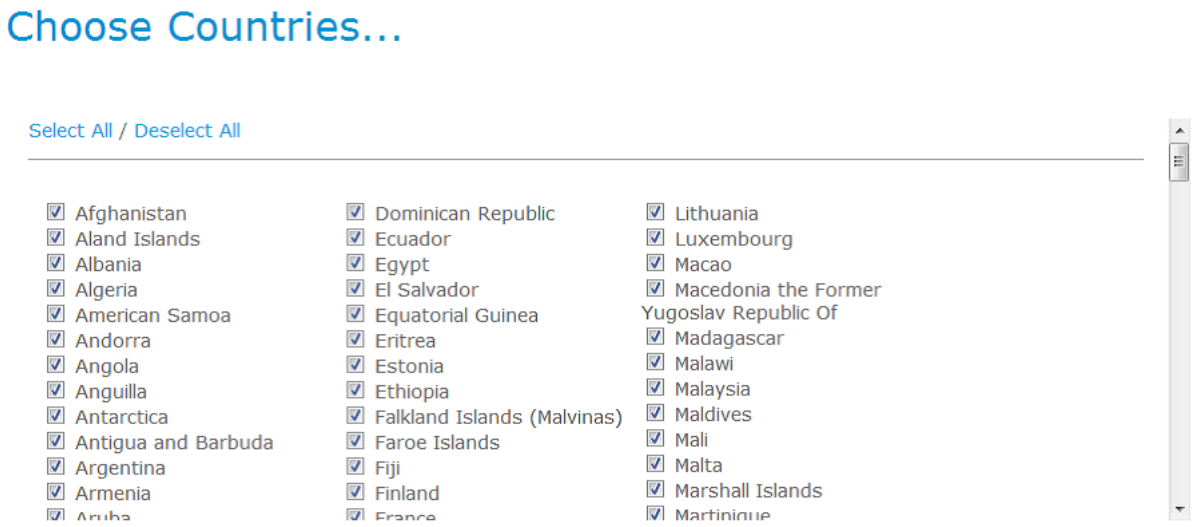
Clicking on the “Submit” button will take you to the New Release screen. In the new release screen, please click the Edit button to provide more information, to be displayed for your extension in the Gallery.



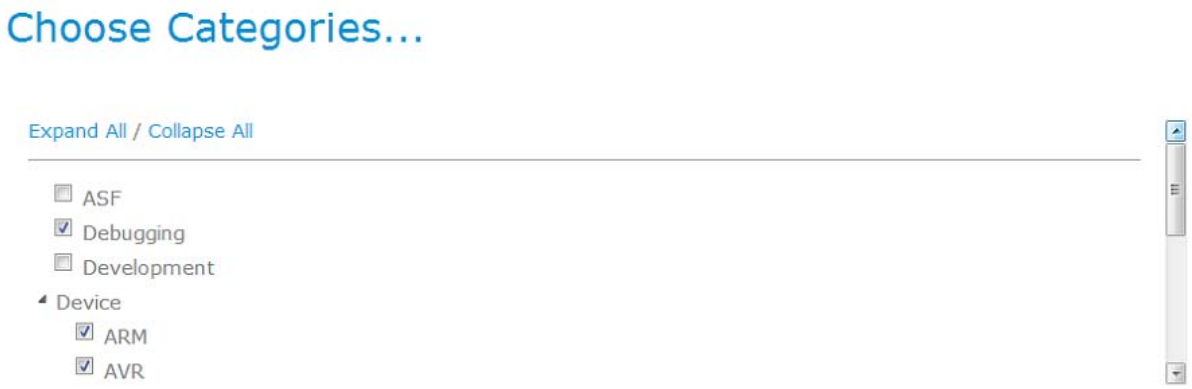
In the below screen you should enter as much information about your extension as possible. This is your opportunity to promote your extension, and tell potential users about its features and benefits.



- Full price: Extension price. Leave as 0.00 if you are offering a free or trial version extension
- Upgrade price: Cost for upgrading to a newer version of your extension, if applicable
- Name: Extension name
- Extension ID: Leave blank
- Author: This can be edited, for extensions submitted by a company, please use the company name
- Description: Information about the key features and benefits of the extension. Will be displayed next to extension in the Extension Manager and in the Gallery. Specification: Max. 4000 characters
- Keywords: Words you enter here are searchable in the Browse section of the Gallery
- Select Extension File: This field should only be used if you have submitted your extension and we find something during our review that needs to be modified. We will then ask you to go back to this field to make another upload, and all the information and images you have already added will still be there
- License: Upload your End User License Agreement. This is the license between you as a developer and the end user. End users must accept the license in order to be able to download the extension
- Specification: .rtf or .txt
- More Info URL: Link to your product page. This allows you to provide more detailed information about your extension. The link can be accessed both from the Extension Manager and the Atmel Gallery, and will be the natural starting point for a potential user for finding more information about your product
- Getting Started URL: Link to online documentation. Can be accessed both from Extension Manager and the Atmel Gallery



In the Choose Countries you can deselect countries in which your extension cannot be downloaded, if applicable. Certain countries with restricted download capabilities have already been removed.

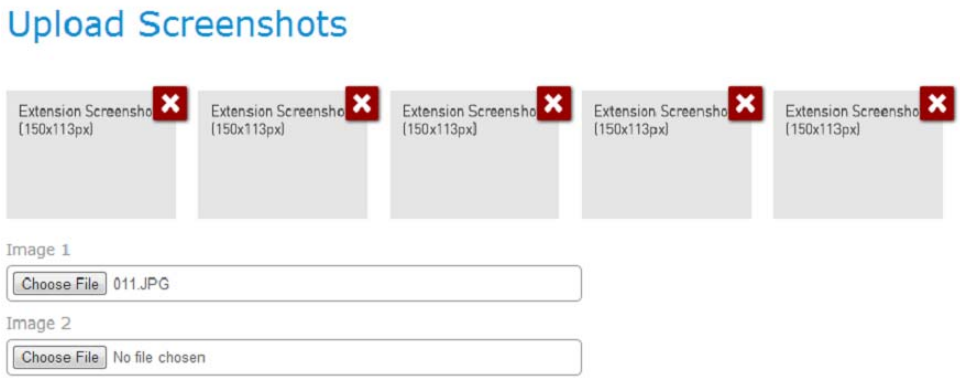


Categories are browsable both in the Extension Manager and in the Atmel Gallery; hence it is important that you tie your extension to the appropriate categories to enable users to find it when searching. If you cannot find a suitable category in the current selection, please email your suggestion us at gallery@atmel.com, and we will review if the suggested category can added.

Currently there is only one available Platform, Atmel Studio 6.0. Eventually there will be more options, and you should check the ones compatible with your extension.

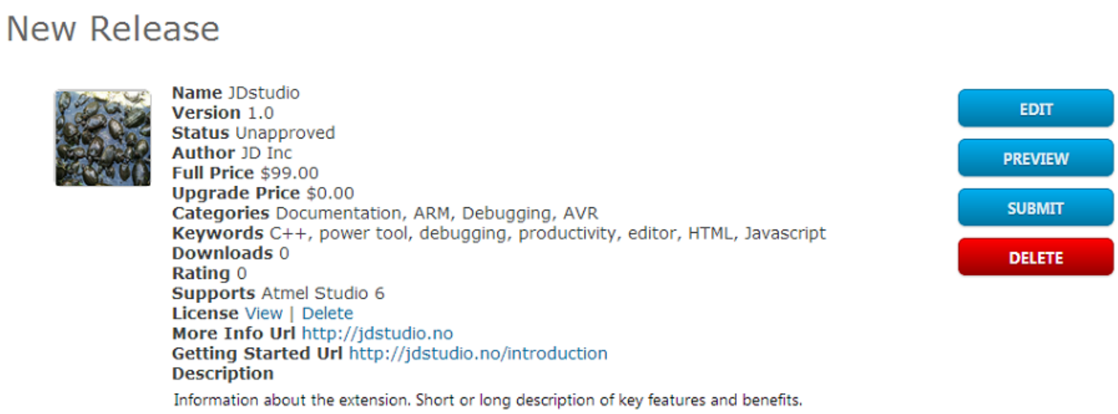
Upload the image you want to be displayed as the icon for your extension. This will be displayed both in the Atmel Studio Extension Manager and in the Gallery. Specification: 80x80 pixels.

Here you can add your company logo, which will be displayed in the Atmel Studio Extension Manager as part of the product information.



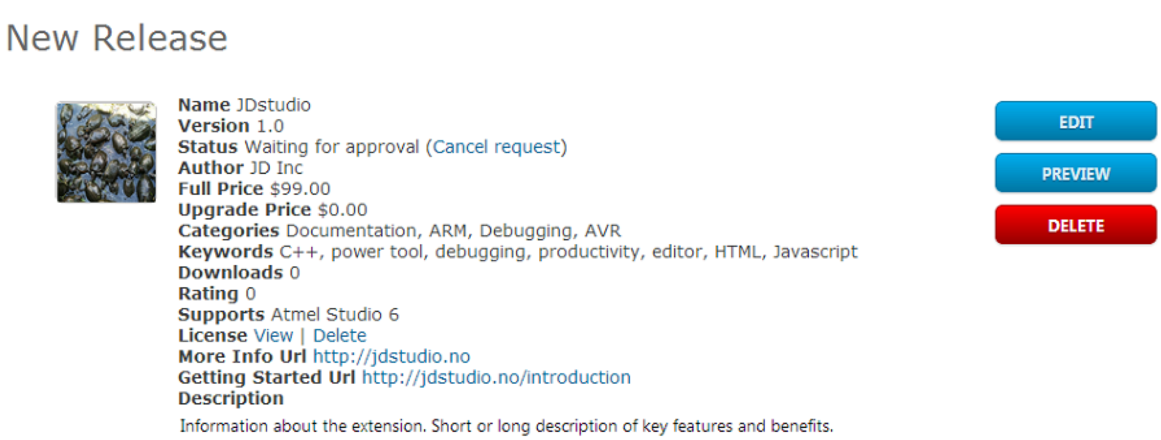
You can upload up to five screenshots which will be visible to users under the product information in the Gallery. Users will be able to view full size versions by right-clicking each screenshot. Specification: 200x200 pixels.

When you have added all information and files, make sure you save everything before your session times out. You can also save without submitting, and return to complete the rest and then submit later on. Your extension will be displayed as a new release, with status Unapproved.



Preview will allow you to see how the extension will appear in the gallery. If you are satisfied with the preview you press Submit. This will notify the Atmel Gallery moderator that your extension is pending for release.

Once you submit an extension, the status will change to Waiting for approval.

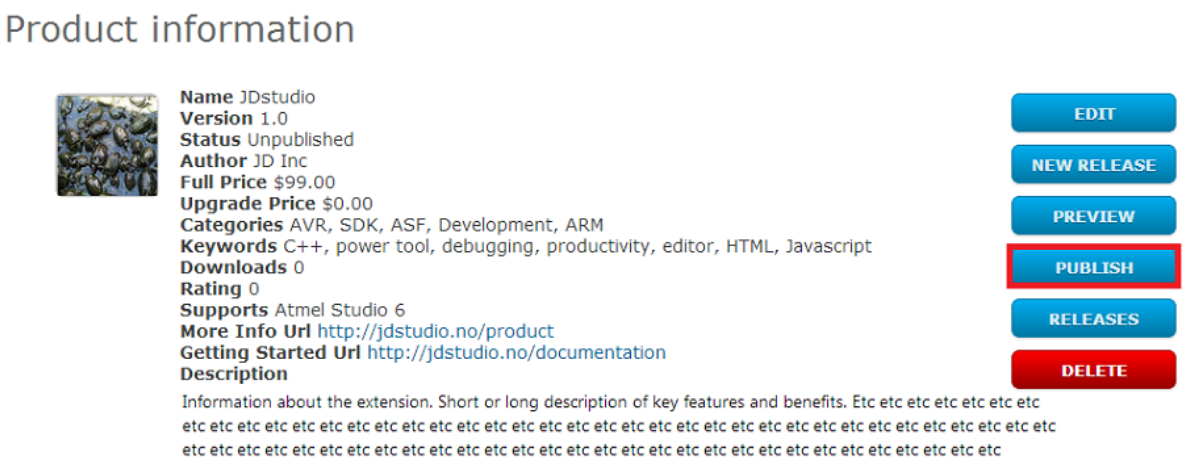


4.2 Publishing the extension

Before the extension can be released, it will be tested by our technical support team. This test will involve:

- Install/uninstall
- General product functionality
- Product information (checking for compliance with Terms and Conditions of Service and Developer Agreement)

If we discover bugs or irregularities during the testing, we will notify you and may request you to resubmit the extension with the identified issues corrected.If the testing is successful, you will receive a notification email informing you that your extension has been approved. When you get this email you will need to access the extension via your Gallery profile to publish it.



When you publish, the extension will immediately be visible to Extension Manager and Atmel Gallery users.

You will be able to unpublish the extensions to make changes, or delete it entirely, at your convenience.

4.3 Download notification

Every time someone downloads or purchases your extension, you will receive a notification email from us. The email will contain user name and email, in case your extension requires providing the user with a license key. It also presents a breakdown of the financial details.

4.4 Payments

If you are offering a commercial extension, you will need to provide your PayPal account by editing the information in your account profile in order to receive your payments from Atmel on a quarterly basis.

4.5 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.

5. References and further information

5.1 Atmel Gallery

- http://gallery.atmel.com/

5.2 Atmel Studio

- http://www.atmel.com/tools/atmelstudio.aspx

5.3 ASF

- http://www.atmel.com/tools/AVRSOFTWAREFRAMEWORK.aspx

6. Revision History

Doc. Rev.	Date	Comments
42050B	11/2012	Added how to upload an extension information
42050A	11/2012	Initial document release

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki
Shinagawa-ku, Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370