Software

Atmel Studio

USER GUIDE

Preface

Atmel® Studio is an Integrated Development Environment (IDE) for writing and debugging AVR®/ARM® applications in Windows® XP/Windows Vista®/ Windows 7/8 environments. Atmel Studio provides a project management tool, source file editor, simulator, assembler, and front-end for C/C++, programming, and on-chip debugging.

Atmel Studio supports the complete range of Atmel AVR tools. Each new release contains the latest updates for the tools as well as support for new AVR/ARM devices.

Atmel Studio has a modular architecture, which allows interaction with 3rd party software vendors. GUI plugins and other modules can be written and hooked to the system. Contact Atmel for more information.

Table of Contents

Preface 1

1. Introduction 8
   - 1.1. Features 8
   - 1.2. New and Noteworthy 8
     - 1.2.1. Atmel Studio 7.0 8
     - 1.2.2. Atmel Studio 6.2 Service Pack 2 11
     - 1.2.3. Atmel Studio 6.2 Service Pack 1 11
     - 1.2.4. Atmel Studio 6.2 11
     - 1.2.5. Atmel Studio 6.1 Update 2 12
     - 1.2.6. Atmel Studio 6.1 Update 1.1 12
     - 1.2.7. Atmel Studio 6.1 Update 1 12
     - 1.2.8. Atmel Studio 6.1 12
     - 1.2.9. Atmel Studio 6.0 12
     - 1.2.10. AVR Studio 5.1 13
   - 1.3. Installation 13
   - 1.4. Contact Information 14

2. Getting started 16
   - 2.1. Starting Atmel Studio 16
   - 2.2. Creating a Project 17
     - 2.2.1. Introduction 17
     - 2.2.2. Creating a new Project 17
     - 2.2.3. Choosing a Target Device 19
     - 2.2.4. Writing and Compiling Code 19

3. Project Management 22
   - 3.1. Introduction 22
     - 3.1.1. The Solution Container 22
     - 3.1.2. Save and Open Projects 22
     - 3.1.3. Project Output View 22
     - 3.1.4. Solution Explorer 22
     - 3.1.5. Toolbar Icons 23
     - 3.1.6. Hierarchical Display 23
     - 3.1.7. Item Management Commands 23
     - 3.1.8. Project Components 23
   - 3.2. GCC Projects 25
     - 3.2.1. New Project Wizard 25
     - 3.2.2. Starting a New GCC Project for AVR Device 25
     - 3.2.3. Libraries Options 29
     - 3.2.4. Starting a New GCC Project for SAM (ARM) Device 33
     - 3.2.5. Code Editing 36
     - 3.2.6. Starting a New GCC Static Library Project 37
     - 3.2.7. GCC Project Options and Configuration 40
   - 3.3. Assembler Projects 57

3.3.1. Create New Assembler Project....57
3.3.2. Assembler Options....60
3.4. Import of Projects....62
3.4.1. Introduction....62
3.4.2. Import AVR Studio 4 Project....62
3.4.3. Import AVR 32 Studio Project....65
3.4.4. Import Project Template....69
3.5. Debug Object File in Atmel Studio....71
3.5.1. Introduction....71
3.5.2. Atmel Studio Supported Debug Formats....72
3.5.3. Opening Object Debug File in Atmel Studio....72

4. Debugging....77
4.1. Introduction....77
4.1.1. Debug Platform Independent Debug Environment....77
4.1.2. Differences Between Platforms....77
4.2. Starting a Debug Session....77
4.3. Ending a Debug Session....77
4.4. Attaching to a Target....78
4.5. Start without Debugging....78
4.5.1. One Click Programming - Program and Run....78
4.5.2. Keyboard Shortcut....79
4.6. Debug Control....79
4.7. Breakpoints....81
4.7.1. General Information on Breakpoints....81
4.7.2. Operations with Breakpoints....82
4.7.3. Breakpoint Window....84
4.8. Data Breakpoints....86
4.8.1. Adding Data Breakpoint....86
4.8.2. Data Breakpoints Window....87
4.8.3. General Information on Data Breakpoint....98
4.8.4. Data Breakpoint Usage....99
4.9. QuickWatch, Watch, Locals, and Autos Windows....99
4.9.1. Watch Window....100
4.9.2. Locals Window....102
4.9.3. Autos Window....103
4.9.4. QuickWatch and Watches....104
4.9.5. Expression Formatting....105
4.10. DataTips....106
4.10.1. Expanding and Editing Information....107
4.10.2. Making a DataTip Transparent....108
4.10.3. Visualizing Complex Data Types....108
4.10.4. Adding Information to a Watch Window....108
4.10.5. Importing and Exporting DataTips....108
4.11. Disassembly View....108
4.12. I/O View....110
4.12.1. About the I/O View....110
4.12.2. Using the I/O View Tool....111
4.12.3. Editing Values and Bits in Break Mode....111

4.13. Processor View 111
4.14. Register View..... 112
4.15. Memory View..... 112
4.16. Call Stack Window..... 113
4.17. Object File Formats..... 116
4.18. Trace..... 117
4.18.1. Application Output..... 117
4.18.2. Program Counter Sampling..... 118
4.18.3. Variable Watching..... 118
4.19. Trace View..... 119
4.19.1. Trace View Options..... 119
4.19.2. Trace View Interpretation..... 122

5. Programming Dialog..... 125
5.1. Introduction..... 125
5.2. Interface Settings..... 128
5.3. Tool Information..... 131
5.4. Board Settings/Tool Settings..... 132
5.4.1. Power Debugger..... 132
5.4.2. STK600..... 132
5.4.3. QT600..... 133
5.4.4. STK500..... 133
5.5. Card Stack..... 134
5.6. Device Information..... 135
5.7. Oscillator Calibration..... 136
5.8. Memories..... 137
5.9. Fuse Programming..... 139
5.10. Lock Bits..... 140
5.11. Production Signatures..... 140
5.12. Production Files..... 141
5.13. Security..... 144
5.14. Automatic Firmware Upgrade Detection..... 145

6. Miscellaneous Windows..... 146
6.1. Device Pack Manager..... 146
6.2. User Interface Profile Selection..... 148
6.3. Available Tools View..... 149
6.3.1. Introduction..... 149
6.3.2. Tool Actions..... 150
6.3.3. Add a Non-detectable Tool..... 150
6.4. Tool Info Window..... 152
6.4.1. Xplained Pro Kits..... 154
6.4.2. Disable the Tools Info Window..... 154
6.4.3. Manually Showing the Window..... 154
6.5. Firmware Upgrade..... 154
6.5.1. Introduction..... 154
6.5.2. Automatic Upgrade..... 154
6.5.3. Manual Upgrade..... 155
6.6. Find and Replace Window..... 155

6.7. Export Template Wizard ... 159
6.7.1. Project Template ... 160
6.7.2. Item Template ... 160
6.7.3. Template Parameters ... 160
6.8. Kit Mode Setting ... 162

7. Atmel GNU Toolchains ... 163
7.1. GNU Compiler Collection (GCC) ... 163
7.2. ARM Compiler and Toolchain Options: GUI ... 163
7.3. ARM GNU Toolchain Options ... 168
7.3.1. ARM/GNU Common Options ... 168
7.3.2. Compiler Options ... 168
7.3.3. Linker Options ... 171
7.3.4. Assembler Options ... 172
7.3.5. Preprocessing Assembler Options ... 172
7.3.6. Archiver Options ... 172
7.4. Binutils ... 173
7.5. AVR Compiler and Toolchain Options: GUI ... 173
7.6. Commonly Used Options ... 178
7.6.1. Compiler Options ... 178
7.6.2. Linker Options ... 181
7.6.3. Assembler Options ... 182
7.7. 8-bit Specific AVR GCC Command Line Options ... 182
7.7.1. AVR C Compiler ... 182
7.7.2. AVR C Linker ... 183
7.8. 32-bit Specific AVR GCC Command Line Options ... 183
7.8.1. Optimization ... 183
7.8.2. Debugging ... 184
7.8.3. AVR32 C Linker ... 185
7.9. Binutils ... 186

8. Extending Atmel Studio ... 187
8.1. Extension Manager UI ... 187
8.2. Registering at Atmel Extension Gallery ... 188
8.3. Installing New Extensions in Atmel Studio ... 189
8.4. Visual Assist ... 192
8.5. Overview of QTouch Composer and Library ... 193
8.5.1. Installation ... 194
8.5.2. Overview of QTouch Project Builder ... 194
8.5.3. Overview of QTouch Analyzer ... 195
8.6. Scripting Extensions ... 196
8.6.1. Debug Scripting ... 196

9. Menus and Settings ... 199
9.1. Customizing Existing Menus and Toolbars ... 199
9.2. Reset Your Settings ... 200
9.3. Options Dialog Box ... 201
9.3.1. Environment Options ... 201
9.3.2. Project Options ... 218

9.3.3. Source Control....221
9.3.4. Text Editor Options....221
9.3.5. Debugger....237
9.3.6. Atmel Software Framework Settings....238
9.3.7. Builder....239
9.3.8. Device and Tool Libraries....239
9.3.9. Status Management....239
9.3.10. Text Templating....240
9.3.11. Toolchain....240
9.3.12. GDB Settings....241
9.4. Code Snippet Manager....242
9.4.1. Managing Code Snippets....242
9.4.2. Code Snippet Manager Layout....243
9.4.3. Modifying Existing Code Snippets....243
9.5. External Tools....244
9.5.1. Add an External Tool to the Tools Menu....244
9.5.2. Pass Variables to External Tools....245
9.5.3. Initial Directory....246
9.5.4. Run Behavior....246
9.5.5. Assign a Keyboard Shortcut....246
9.6. Predefined Keyboard Shortcuts....246

10. Command Line Utility (CLI)....262

11. Frequently Asked Questions....263
11.1. Compatibility with Legacy AVR Software and Third-party Products....265
11.1.1. How do I Import External ELF Files for Debugging?....265
11.1.2. How do I Reuse My AVR Studio 4 Projects with the New Atmel Studio?....265
11.2. Atmel Studio Interface....266
11.2.1. How can I Start Debugging My Code? What is the Keyboard Shortcut for Debugging?....266
11.2.2. What is a Solution?....266
11.2.3. What is a Project....266
11.2.4. How can I use an External Makefile for my Project?....266
11.2.5. When Watching a Variable, the Debugger says `Optimized away`....266
11.2.6. When Starting a Debug Session, I get an Error Stating that **Debug Tool is not Set**....267
11.3. Performance Issues....267
11.3.1. Atmel Studio Takes a Very Long Time to Start on My PC, but Runs Well in a VM Environment. Is there Something I Can do With This?....267
11.3.2. Verification and Programming often Fails with a Serial Port Buffer Overrun Error Message when using STK500....267
11.3.3. I've connected my Tool through a USB Hub, and now I get Error Messages and Inconsistent Results while Programming and Debugging....267
11.4. Driver and USB Issues....267
11.4.1. How do I get my Tool to be Recognized by Atmel Studio?....267
11.4.2. The Firmware upgrade Process fails or is Unstable on a Virtualized Machine....268
11.4.3. Debugging never Breaks under a Virtualized Machine....268
11.4.4. No Tool is recognized by Atmel Studio, but the Driver seems to be Working....268

11.4.5. Firmware Upgrade Fails on VirtualBox 268
11.4.6. Common Jungo USB Errors 269
11.4.7. Issues with ARM Compatible Tools 270

12. Document Revision History 272

1. Introduction

1.1. Features

Atmel Studio provides a large set of features for project development and debugging. The most notable features are listed below.

- Rich code editor for C/C++ and Assembly featuring the powerful Visual Assist extension
- Cycle correct simulator with advanced debug functionality
- Atmel Software Framework allowing creation of modular applications and providing building blocks for a prototype on any AVR platform
- Debugging on actual devices using Debugging Tools
- Rich SDK to enable tight integration of customer plugins
- Compatible with many Microsoft® Visual Studio® plugins

1.2. New and Noteworthy

New features available.

1.2.1. Atmel Studio 7.0

Atmel Studio 7.0.1006

The following changes are done in Atmel Studio 7.0.1006:

- New Atmel Start extension that allows the user to create and configure Atmel Start projects within Atmel Studio
- Ability to load multiple modules in a debug session (experimental)
- AVR 8-bit GCC Toolchain 3.5.3 with upstream versions:
  - gcc 4.9.2
  - Binutils 2.26
  - avr-libc 2.0.0
  - gdb 7.8
- ARM GCC Toolchain 5.3.1 with upstream versions:
  - gcc (ARM/embedded-5-branch revision 234589)
  - Binutils 2.26
  - gdb 7.10

Atmel Studio 7.0.1006 contains a fix for the following issues that were present in 7.0.943:

- AVRSV-6878: Atmel Studio write the write-once wdt registers on some SAM devices.
- AVRSV-7470: SAM Cortex®-M7 devices fails launch occasionally.
- AVRSV-7471: Devices with external and internal RAM lists all the RAM as available.
- AVRSV-7473: Atmel Studio hangs during startup.
- AVRSV-7474: Kits connected to Atmel Studio are not getting enumerated in the QTouch Start Page.
- AVRSV-7477: Show all files does not work from solution explorer.
- AVRSV-7482: Exception when adding breakpoint on SAM4L.
- AVRSV-7486: Debugging may fail in Cortex-M0+ SAM devices at high clock speeds.

Atmel Studio 7.0.943

Atmel Studio 7.0.943 contains a fix for the following issue:

- AVRSV-7459: Projects containing files with upper case file names can fail to build. Saving files with upper case file names converts file name to lower case.

Atmel Studio 7.0.934

The following changes are done in Atmel Studio 7.0.934:

- AVR 8-bit GCC Toolchain 3.5.2 with upstream versions:
  - gcc 4.9.2
  - Binutils 2.26
  - avr-libc 2.0.0
  - gdb 7.8
- AVR 32-bit GCC Toolchain 3.4.3 with upstream versions:
  - gcc 4.4.7
  - Binutils 2.23.1
  - Newlib 1.16.0
- ARM GCC Toolchain 4.9.3 with upstream versions:
  - gcc (ARM/embedded-4_9-branch revision 224288)
  - Binutils 2.24
  - gdb 7.8.0.20150304-cvs

Atmel Studio 7.0.934 resolves the following issues present in Atmel Studio 7.0.790:

- AVRSV-7376: Atmel-ICE slow programming.
- AVRSV-7379: Unhandled exception when writing fuses or lockbits when Auto Read is turned off.
- AVRSV-7396: Some machines shows an error regarding 'Exception in MemoryPressureReliever'.
- AVRSV-7400: When in Standard mode, **Disable debugWire and Close** are not visible in the Debug menu.
- AVRSV-7408: When using Atmel Studio in Standard mode, the **Set Startup Project** menu is missing.

Atmel Studio 7.0.790

The following features are added in Atmel Studio 7.0.790:

- Support for mass storage mode in embedded debugger (EDBG), enabling drag and drop programming
- Introduction of user interface profiles. The user can choose an interface where some of the toolbar buttons and menu items are removed.
- Support for importing libraries to previously imported sketches. Added support for Arduino Zero and Zero Pro.
- Parallel build turned on by default

Atmel Studio 7.0.790 resolves the following issues present in Atmel Studio 7.0.634:

- AVRSV-7084: Persist user settings during upgrade.
- AVRSV-7014: Some ATmega and ATtiny devices failed to start debugging with the Simulator.
- AVRSV-7230: "Show all files" in Solution Explorer not consistent.
- AVRSV-7062: Firmware upgrade of Xplained Mini kits not detected.
- AVRSV-7164: Reading flash to .bin file created incorrect .bin file.

- AVRSV-7106: Hex files with Unix® or mixed file endings fail to load.
- AVRSV-7126: Databreakpoints for ARM should not be limited to RAM.

Atmel Studio 7.0.634

This release adds device support for the SAM B11 device family.

Atmel Studio 7.0.634 resolves the following issues present in Atmel Studio 7.0.594:

- AVRSV-6873: Jungo Driver issue with Windows 10.
  Note: If you install this version of Atmel Studio in parallel with an older Studio versions or IAR Embedded Workbench® and are using AVR Dragon™, AVRISP mkII, JTAGICE mkII, AVR ONE!, STK®600, or QT600 read How to downgrade to use older Jungo drivers.
- AVRSV-6676: Launching debugging fails due to issue with Intel graphics driver.

Atmel Studio 7.0.594

Atmel Studio 7.0.594 resolves the following issues present in Atmel Studio 7.0.582:

- AVRSV-7008: Opening a 6.2 project in Atmel studio 7.0.582 persists Debug configuration settings for all the other configurations.
- AVRSV-6983: Uninstalling Studio extensions does not work in some cases.
- AVRSV-7018: Project Creation fails with some culture specific user names.
- AVRSV-7019: Help Viewer does not work on 32-bit machines.
- Issues with getting tools/debuggers recognized or visible see section 2.4 in 'Atmel Studio 7.0.594-readme.pdf' for workarounds.

Atmel Studio 7.0.582

- Updated to Visual Studio Isolated Shell 2015
- Integration with Atmel Start.
  - This tool will help you select and configure software components, drivers, middle-ware, and example projects to tailor your embedded application in a usable and optimized manner
- New device support system, CMSIS Pack compliant
- Data Visualizer, used for processing and visualizing data
- Updated help system, improved context sensitive help
- Atmel Software Framework version 3.27.3. ASF is an extensive software library of software stacks and examples.
- A major upgrade of the Visual Assist extension to Atmel Studio that assists with reading, writing, re-factoring, navigating code fast
- Import Arduino Sketch projects into Atmel Studio
- Support for Flip-compatible bootloaders in atprogram and programming dialogue. The connected device appears as a tool.
- AVR 8-bit GCC Toolchain 3.5.0 with upstream versions[1]:
  - gcc 4.9.2
  - Binutils 2.25
  - avr-libc 1.8.0svn
  - gdb 7.8
- AVR 32-bit GCC Toolchain 3.4.3 with upstream versions[1]:
  - gcc 4.4.7
  - Binutils 2.23.1

  - Newlib 1.16.0
- ARM GCC Toolchain 4.9.3 with upstream versions[1]:
  - gcc 4.9 (revision 221220)
  - Binutils 2.24
  - gdb 7.8.0.20150304-cvs

1.2.2. Atmel Studio 6.2 Service Pack 2

- Atmel Software Framework 3.21.0
- Added support for the ATSAML21 device family
- Added support for the ATSAMV7 device family, based on the ATM Cortex-M7 core

1.2.3. Atmel Studio 6.2 Service Pack 1

- Atmel Software Framework 3.19.0
- AVR 8-bit Toolchain 3.4.5 with upstream versions:
  - GCC 4.8.1
  - Binutils 2.41
  - avr-libc 1.8.0svn
  - gdb 7.8
- AVR 32-bit Toolchain 3.4.2 with upstream versions:
  - GCC 4.4.7
  - Binutils 2.23.1
- ARM GCC Toolchain 4.8.4 with upstream versions:
  - GCC 4.8.4
  - Binutils 2.23.1
  - gdb 7.8
- Support for trace buffers for ARM (MTB) and 32-bit AVR UC3 (NanoTrace)
- Support for attaching to targets

1.2.4. Atmel Studio 6.2

- Atmel Software Framework 3.17.0
- AVR 8-bit Toolchain 3.4.4 (with upstream GCC 4.8.1)
- AVR 32-bit Toolchain 3.4.2 (with upstream GCC 4.4.7)
- ARM GCC Toolchain 4.8.3
- Support for Atmel-ICE
- Support for Xplained Mini
- Support for data breakpoints
- Read OSCCAL calibration for tinyAVR® and megaAVR®
- Create ELF production files for AVR 8-bit using the programming dialogue
- Live Watch

---

1 For more information, see the readme that is installed as part of the toolchain.
2 For more information, see the readme that is installed as part of the toolchain.

- Non-intrusive trace support for SAM3 and SAM4 family of devices including
  - Interrupt trace and monitoring
  - Data trace
  - FreeRTOS™ awareness
  - Statistical code profiling
- Polled Data trace support for Cortex M0+
- Default debugger for SAM devices is now GDB. GDB does in some scenarios handle debugging of optimized code better.
- Support to create a GCC Board project (Atmel board\User board) for ALL the installed versions of ASF
- New ASF Board Wizard, to Add or Remove Board Project Template
- Improved loading time of New Example Project dialog, by loading only one ASF version by default
- IDR events now gets displayed in a separate pane in the output window
- LSS file syntax highlighting

1.2.5. Atmel Studio 6.1 Update 2

- Support for SAM D20 devices on the JTAGICE3
- Atmel Software Framework 3.11.0

1.2.6. Atmel Studio 6.1 Update 1.1

- Fix programming of boot section for XMEGA devices introduced in 6.1 update 1
- Fix SAM4LSP32 bare-bone project setup

1.2.7. Atmel Studio 6.1 Update 1

- Atmel Software Framework 3.9.1
- Extension Development Kit (XDK). Support for packaging an Embedded Application project into an Atmel Gallery Extension.
- Support for SAM D20 and SAM4N devices
- ARM GCC Toolchain 4.7.3 with experimental newlib-nano and multilibs

1.2.8. Atmel Studio 6.1

- Support for Embedded Debugger platform
- Support for Xplained Pro kits
- Atmel Software Framework 3.8.0
- AVR 8-bit Toolchain 3.4.2 (with upstream GCC 4.7.2)
- AVR 32-bit Toolchain 3.4.2 (with upstream GCC 4.4.7)
- ARM GCC Toolchain 4.7.3
- CMSIS 3.20
- Updated Visual Assist
- Command line utility for firmware upgrade
- Stimulus for simulator. Create a stimuli file to write register values while executing simulation.

1.2.9. Atmel Studio 6.0

- Support for Atmel ARM-based MCUs with Atmel SAM-ICE
- Atmel Software Framework 3.1.3
- AVR Toolchain 3.4.0

- ARM Toolchain 3.3.1
- Atmel Software Framework Explorer
- Support for QTouch Composer as extension
- Updated Visual Assist
- New extension gallery

1.2.10. AVR Studio 5.1

- New version of AVR Software Framework (ASF)
- Availability and installation of new ASF versions through extension manager, without having to upgrade Studio 5
- Support for side by side versioning of ASF, with the ability to upgrade projects
- Syntax highlighting and better debugging support for C++ projects
- Support for importing AVR 32 Studio C++ projects
- New version of AVR Toolchain
- New command line utility (atprogram) with support for all Atmel AVR tools and devices
- Enhancements to programming dialog including support for ELF programming
- New version of Visual Assist with several enhancements and bugfixes

1.3. Installation

Installation instructions.

Supported Operating Systems

- Windows 7 Service Pack 1 or higher
- Windows Server 2008 R2 Service Pack 1 or higher
- Windows 8 / 8.1
- Windows Server 2012 and Windows Server 2012 R2
- Windows 10

Supported Architectures

- 32-bit (x86)
- 64-bit (x64)

Hardware Requirements

- Computer that has a 1.6GHz or faster processor
- RAM
  - 1GB RAM for x86
  - 2GB RAM for x64
  - An additional 512MB RAM if running in a Virtual Machine
- 6GB of available hard disk space

Downloading and Installing

- Download the latest Atmel Studio installer
- Atmel Studio can be run side by side with older versions of Atmel Studio and AVR Studio®. Uninstallation of previous versions is not required.

- Verify the hardware and software requirements from the "System Requirements" section
- Make sure your user have local administrator privileges
- Save all your work before starting. The installation might prompt you to restart, if required.
- Disconnect all Atmel USB/Serial hardware devices
- Double click the installer executable file and follow the installation wizard
- Once finished, the installer displays an option to **Start Atmel Studio after completion**. If you choose to open, then note that Atmel Studio will launch with administrative privileges, since the installer was either launched as administrator or with elevated privileges.

1.4. Contact Information

Report any problems you experience with this version of Atmel Studio. We would also like to receive good ideas and requests that can help to improve further development and releases of Atmel Studio.

Check out the Atmel Knowledge Base for any issues that you might encounter. From the same page, it is possible to contact Atmel Support through the new support portal which is linked up with your myAtmel account.

For the latest updates of Atmel Studio, visit the Atmel web site: www.atmel.com.

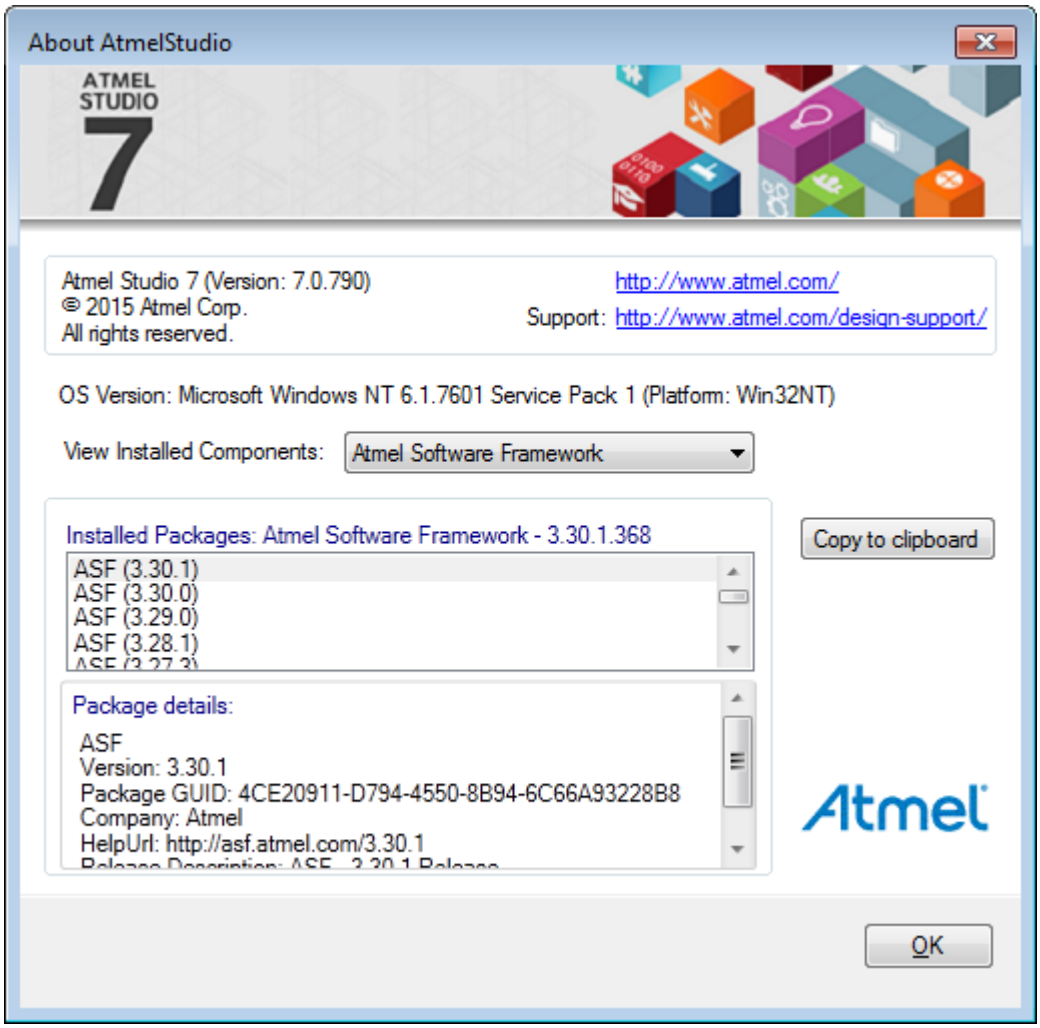
Reporting Bugs

Copy the information from the version dialog (see the figure below) and include it in the email to Atmel. Also, make sure to provide a detailed description of the problem:

1. Describe how to recreate the problem.
2. Attach any test program that causes the problem.
3. Check that the copied version information contains used debug platform and device.

The version dialog is opened by the file menu **Help → About Atmel Studio**. Debug platform and device are only displayed if you are in debug mode. Push the copy button to copy the contents to the clipboard.

Figure 1-1. Atmel Studio About Box



2. Getting started

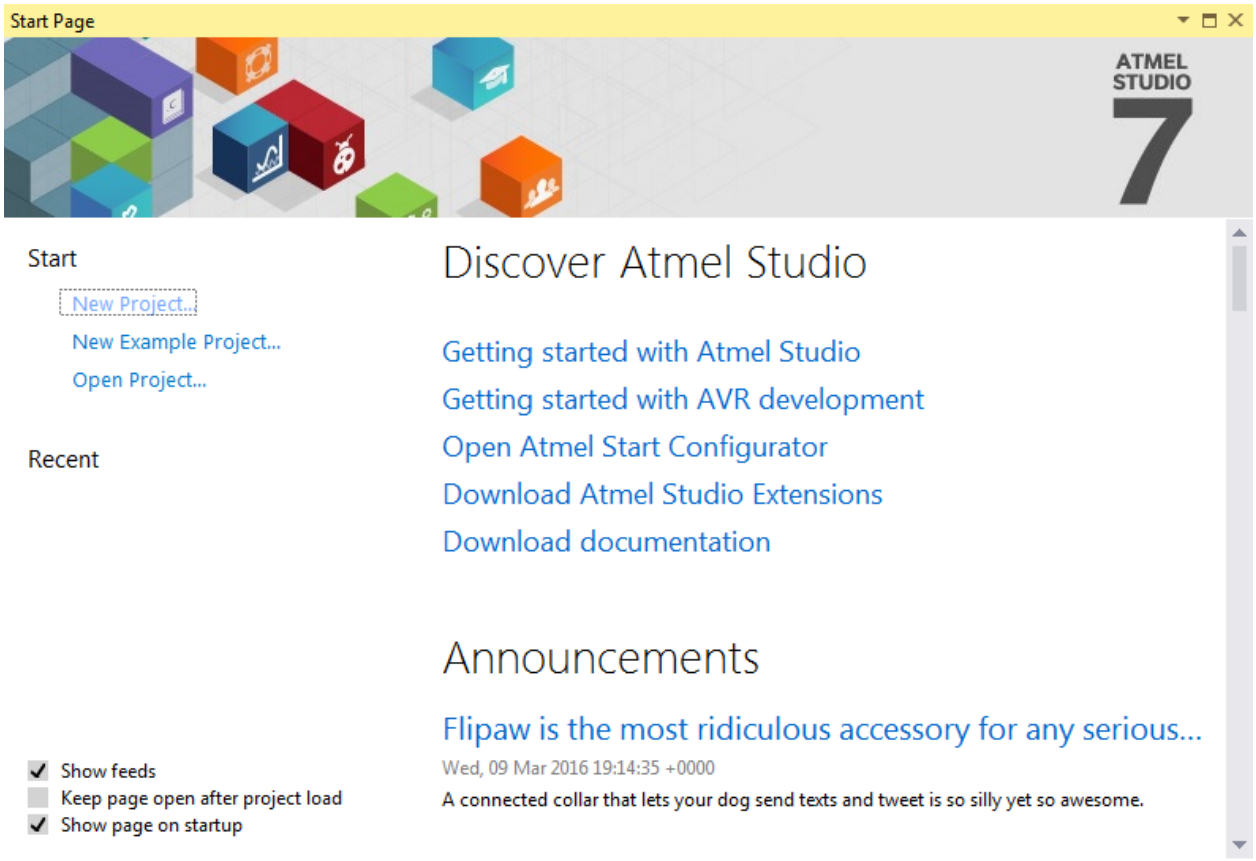
2.1. Starting Atmel Studio

Atmel Studio is started by clicking on the Atmel Studio 7.0 shortcut in the Start-up menu.

Once started, the start page is displayed. From within this page you can create new projects and reopen recently used projects, as well as browse through articles providing tutorials, help and news.

The Start page can also be accessed from **View** → **Start Page**, or Alt V G .

Figure 2-1. The Project Related Section of the Start Page



The left section of the start page contains project-related items:

- **New project** - Use this to create a new project. If you are new to the concept of software development with Atmel Studio, refer to the step-by-step guides. The project settings and available options are described in detail in Project Management.
- **New example project** - To take a step-by-step tour of the available Atmel platforms' functionalities using the Atmel Software Framework, click this button.
- **Open project** - Load an existing project, not mentioned on the **Recent projects** pane.

The **Recent projects** lists the most recently opened projects. Clicking on any of the links will open the project, restoring it and the GUI to its last saved settings. You can select the number of projects you would like to be shown in the Menus and Settings.

Discover Atmel Studio

This section contains links to helpful information about how to use Atmel Studio and related tools.

Announcements

In the **Announcements** section you can read the Atmel RSS feed or any other RSS feed. From the **Tools** > **Options...** menu, select **Start Page** > **Feeds** to configure which RSS feeds that should be seen. In order to turn ON or OFF the feeds, use the **Show feeds** check-box.

2.2. Creating a Project

2.2.1. Introduction

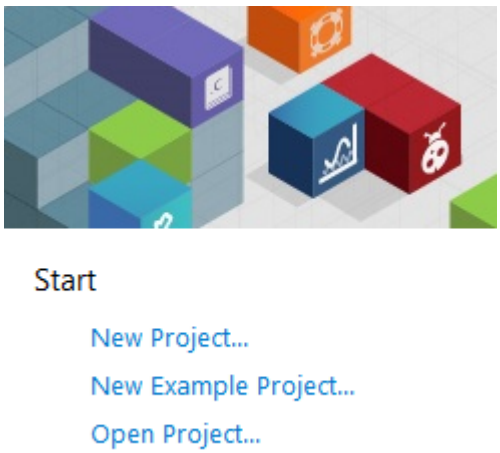
Atmel Studio is based on Visual Studio, and hence the application development process is organized into projects and The Solution Container.

The following sections demonstrates how to create a new GCC C executable project and write a simple application.

2.2.2. Creating a new Project

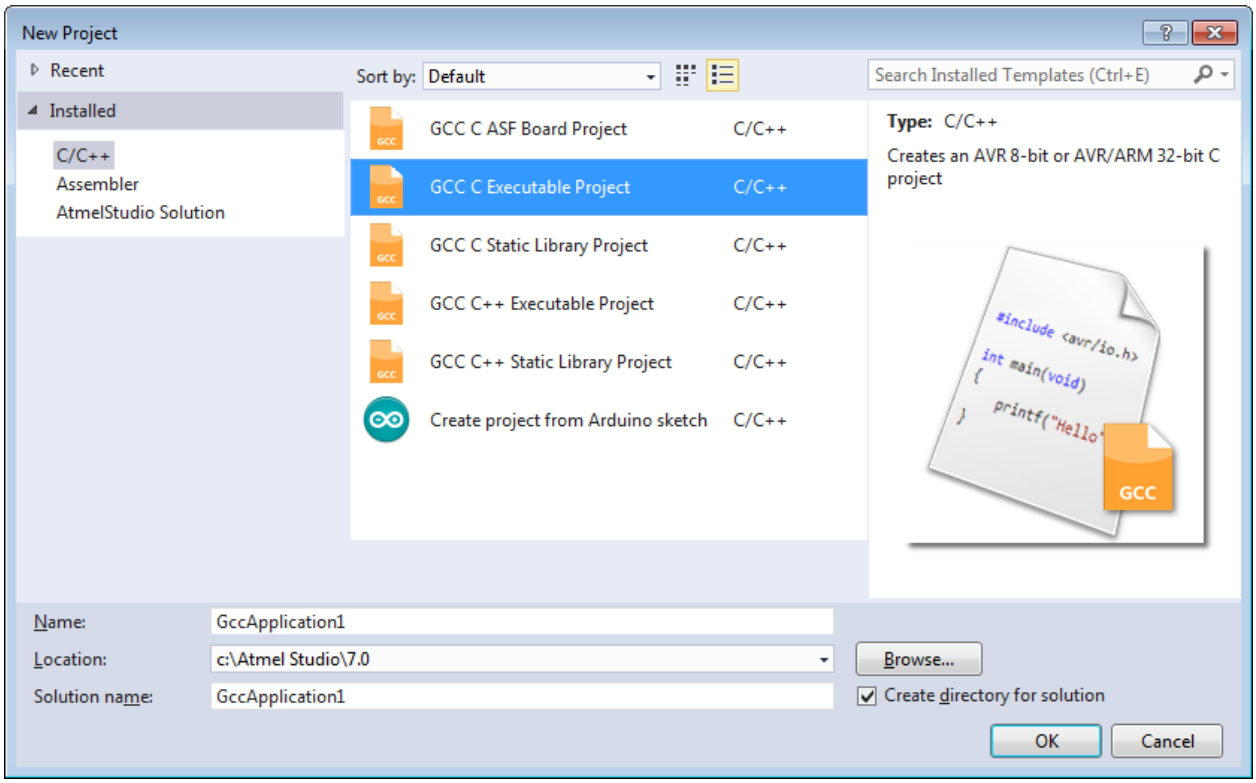
On the Start Page discussed in Getting started, click the **New Project** option.

Figure 2-2. Project Options



The project wizard appears.

Figure 2-3. Project Wizard



About project types

Table 2-1. Project Types

Category	Project templates	Description
C/C++	GCC C ASF Board Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit ASF Board project.
C/C++	GCC C Executable Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit GCC project.
C/C++	GCC C Static Library Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit GCC static library(LIB) project.
C/C++	GCC C++ Executable Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit C++ project.
C/C++	GCC C++ Static Library Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit C++ static library (LIB) project.
Assembler	Assembler Project	Select this template to create an AVR 8-bit Assembler project.
Category	Project Templates	Description

Note:
Extensions and plugins to Atmel Studio may provide new project templates.

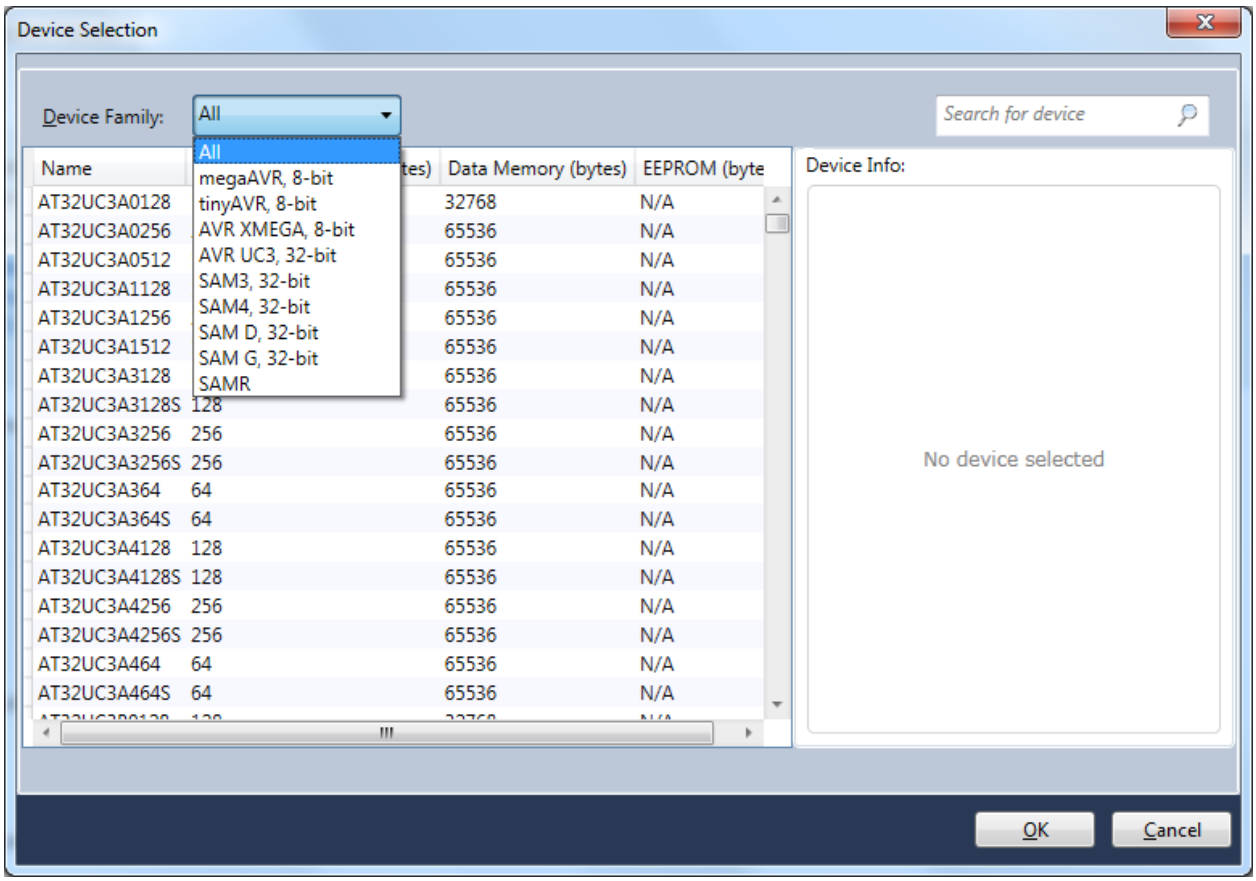
Create a project

1. In the **New Project** dialog box, select **Installed Templates**. This lists the available project types.
2. For this example, create an **GCC C Executable Project**.
3. In the **Name** box, type a name for the new project.
4. In the **Location** box, select a save location.
5. Atmel Studio will suggest a name in the **Solution name** box. You can override this name if wanted.
6. Leave the **Create directory for solution** checkbox checked.
7. Click **OK**.

2.2.3. Choosing a Target Device

When a new project is created, the **Device Selection** dialog is displayed and you will be prompted to select the project target device.

Figure 2-4. Device Selection



The device selection dialog lists all supported devices for the current project type. To narrow down the selection of devices, select the device family in the **Device Family** field, or use the **Search for Device** field to view a filtered list of devices matching your search string.

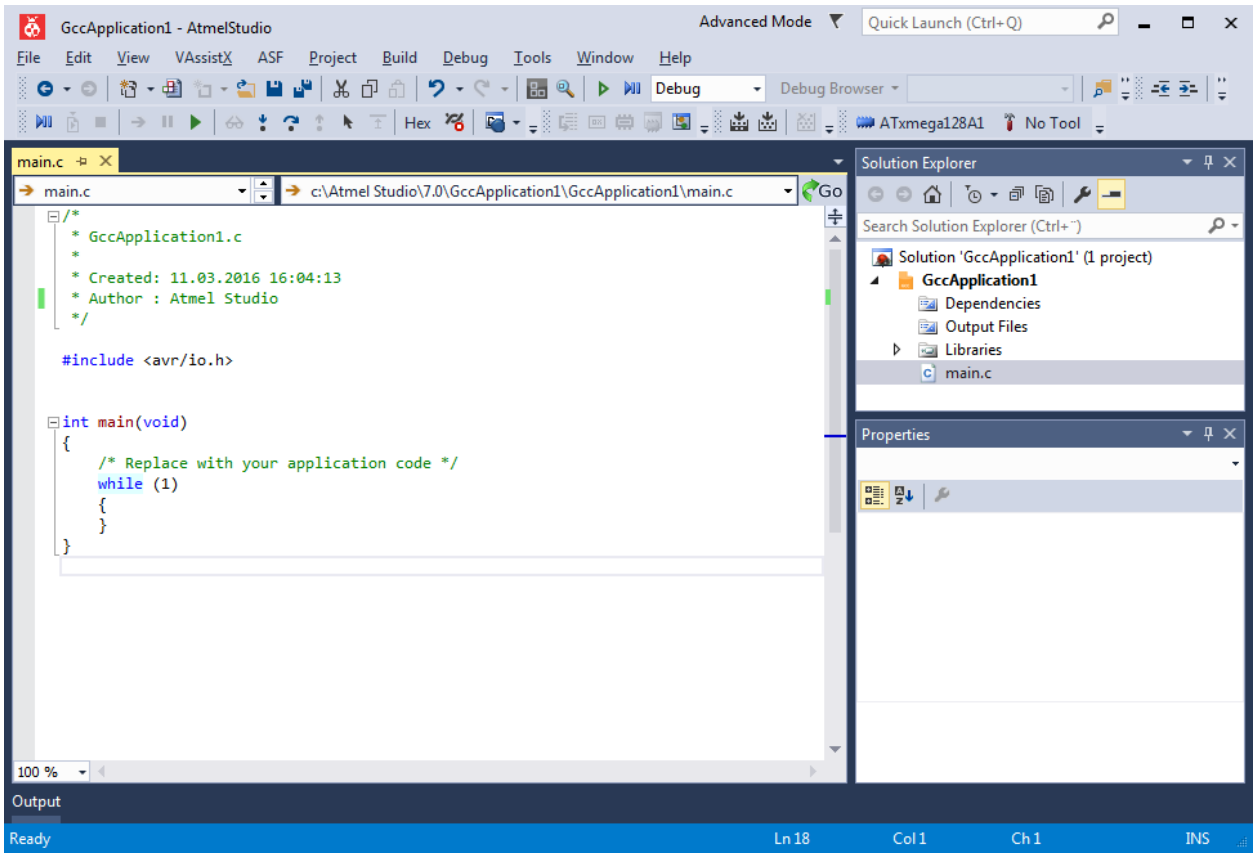
Select a device

1. In the **Device Selection** dialog, select `ATxmega128A1`.
2. Click **OK**.

2.2.4. Writing and Compiling Code

Your solution and project has been created. You can now start editing your application.

Figure 2-5. Code Editor



Atmel Studio automatically opens the newly created C file in the source editor. If the file is closed at any time, double click on `[Project_name].c` - in this case `GccApplication1.c` - to open it in the editor.

At this time the C file contains only an include statement for I/O manipulation and a simple main() function.

Create and build a simple application

1. Replace the original `main` function with the following source code:

```
#define MAXINT 200000

int main(void)
{
    unsigned int t=1000, k=0, l=5, pn=2;
    unsigned int primes[t];
    primes[0]=2;
    primes[1]=3;

    while (pn < t || primes[pn] < MAXINT)
    {
        for ( k = 0; k <= pn; k++)
        {
            if (l % primes[k] == 0)
            {
                goto otog;
            }
            else
            {
                if (k == pn)
                    primes[pn++]=l;
            }
        }
otog:
        l += 2;
    }
```

```
    return 0;
}
```

2. To compile the project, press F7 key or select **Build Solution** from the Build menu.

Atmel Studio now builds the application. All output from the compiler is listed in the output window.

This concludes the introduction to creating code projects in Atmel Studio. All aspects of projects are described in detail in Project Management.

The next section will describe how to debug this application using the built-in simulator.

3. Project Management

3.1. Introduction

Atmel Studio is an Integrated Development Environment (IDE) for writing and debugging applications for AVR/ARM platforms. Currently as a code writing environment, it supports the included AVR Assembler and any external AVRGCC/ARMGCC compiler in a complete IDE environment.

Using Atmel Studio as an IDE gives you several advantages:

1. Editing and debugging in the same application window allows for a faster error tracking.
2. Breakpoints are saved and restored between sessions, even if the code was edited in the meantime.
3. Project item management is made convenient and portable.

3.1.1. The Solution Container

With AVR Studio 5, the concept of "solution" is introduced. The solution is a container that may contain several projects. A project cannot exist outside a solution. If you try to open a project file (`.cproj` or `.asmproj` extension) a solution will be created for you. This allow you to keep for example a bootloader project, and several application projects in the same solution. In practice the Solution is stored as an `.atsln` file. In general, projects that are added to the solution are placed in a separate folder inside the folder that the `.atsln` file recides in.

3.1.2. Save and Open Projects

All projects are saved under a chosen name with the `.cproj` extension for GCC projects and `.asmproj` extension for 8-bit assembler projects. The user can reopen a project, either from the **file** menu, from the recently used projects list, or from the **Project** menu, under **Open project**.

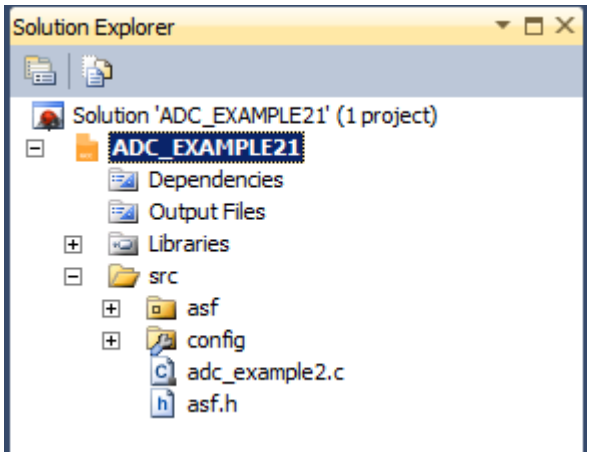
3.1.3. Project Output View

After building, assembling, or compiling the project, the operation result will be shown in the build output window. If any errors occur, the user can double-click on the message, which will position the marker over the corresponding line in the source window.

3.1.4. Solution Explorer

Solution Explorer allows you to view items and perform item management tasks in a solution or a project. It also allows you to use the Atmel Studio editors to work on files outside the context of a solution or project. By default it appears on the **right** side of the Atmel Studio GUI.

Figure 3-1. The Solution Explorer Pane



3.1.5. Toolbar Icons

Buttons specific to the item selected in the tree view appear on the Solution Explorer.

- Displays the appropriate property user interface for the selected item in the tree view.
- Shows all project items, including those that have been excluded in the project and those that are hidden.

3.1.6. Hierarchical Display

A single solution and all its projects appear in a hierarchical display. This allows you to work on several projects at the same time and at the same time keep track of all projects and items. Most source control system extensions (such as AnkhSVN) will also add icon overlays to the item icons, to signal the up-to-date status of the project items that are under revision control.

3.1.7. Item Management Commands

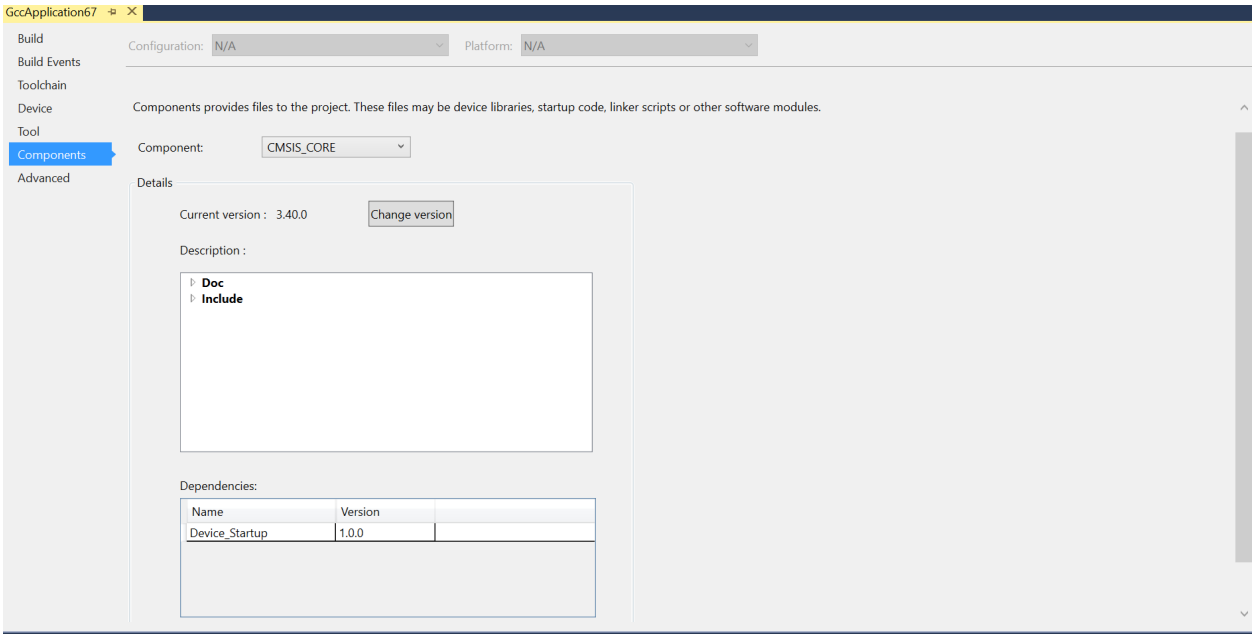
Solution Explorer supports a variety of management commands for each project or solution item. Right click on any item to get a menu with the available commands for that particular item.

3.1.8. Project Components

A project will contain a set of device specific components. This includes startup code, linker scripts, and other support libraries.

Components are small pieces of code or other supporting files that are included in any project.

Figure 3-2. Project Components



Components that are included in a project are listed in the **Component** drop-down. Selecting a component from the drop-down menu shows the component version, the files that the component is contributing with, and the dependencies that the component has.

The version of the component can be changed by clicking the **Change version** button.

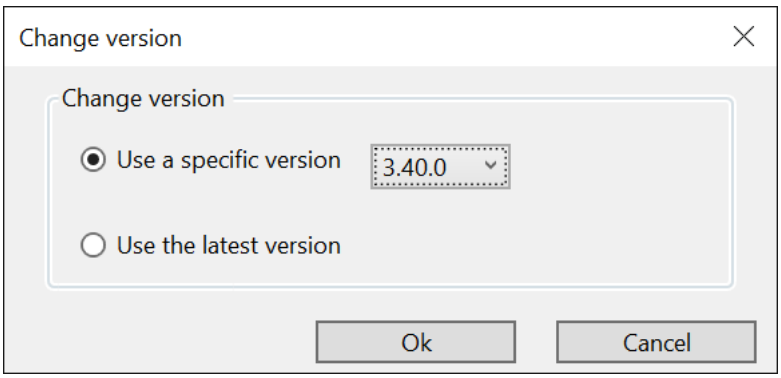
3.1.8.1. Change Version

Components are versioned when added to the project. To change the version that is used, use this dialog.

There are two options when choosing the version of a component

Use a specific version	Lock the project to a specific version of the component.
Use the latest version	Choose the most recent version of the component that is available.

Figure 3-3. Change Version



Components are part of the device packs in Atmel Studio. These device packs are managed using the **Device Pack Manager**.

Related Links

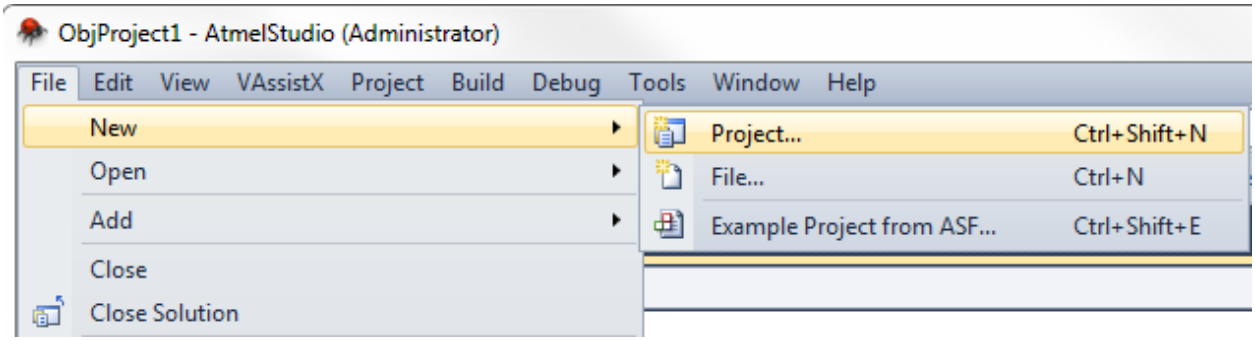
Device Pack Manager on page 146

3.2. GCC Projects

3.2.1. New Project Wizard

Select **File** → **New** from the menu, and the dialog below will appear. The startup wizard will also have an option to start a new project.

Figure 3-4. New Project



Project types

Currently several project types are available in the **Project Type** box. AVR board examples - to guide you through the usage of the AVR boards, User board project - if you have created your own product with the AVR tools, and a general AVR GCC project - a board independent project with a GNU compiler. It is also possible to create an AVR Assembler project and a general AVR Solution, which may include any supported source code type.

Tip:
Projects can also be created by loading supported object files. If you want to create such a project, you should use the **File** → **Open file** menu.

Project name and initial file

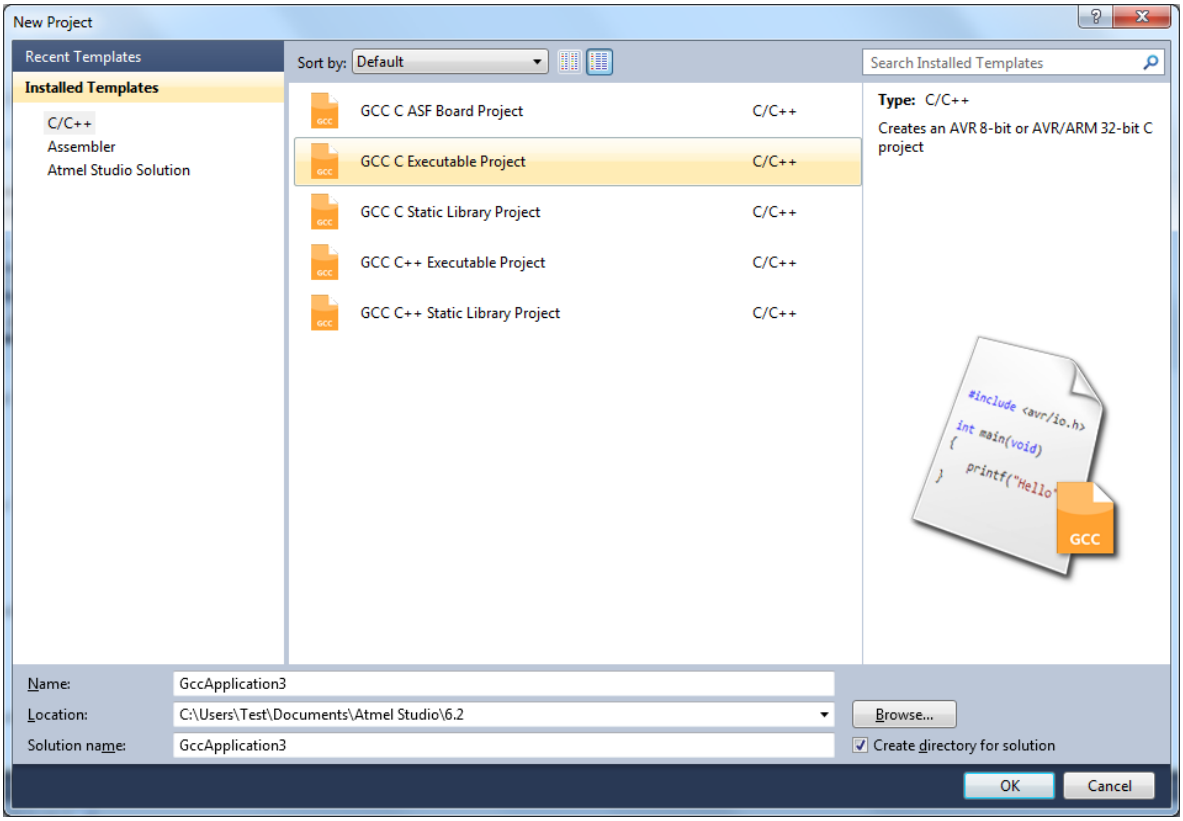
Input the project name. The project main file, which is generated automatically, will be named with the same name by default (ASM or C). If you wish, you can change this name. It is possible to check a box to create a new folder, bearing the project name. This box is unchecked by default.

You can choose to create a new solution in the **Solution** drop-down menu, or to reuse existing code. Input the solution name in the **Solution Name** field.

If you are satisfied with the project name and type, press **OK** and proceed to the debugging platform selection stage. You can also leave the platform undefined for now, but then the you will have to select the debug platform and device upon starting a debug session. See also Assembler Projects, Object File Formats

3.2.2. Starting a New GCC Project for AVR Device

1. Create a new project by selecting **New Project** from the **Project** menu. This will open the **Project Wizard**.

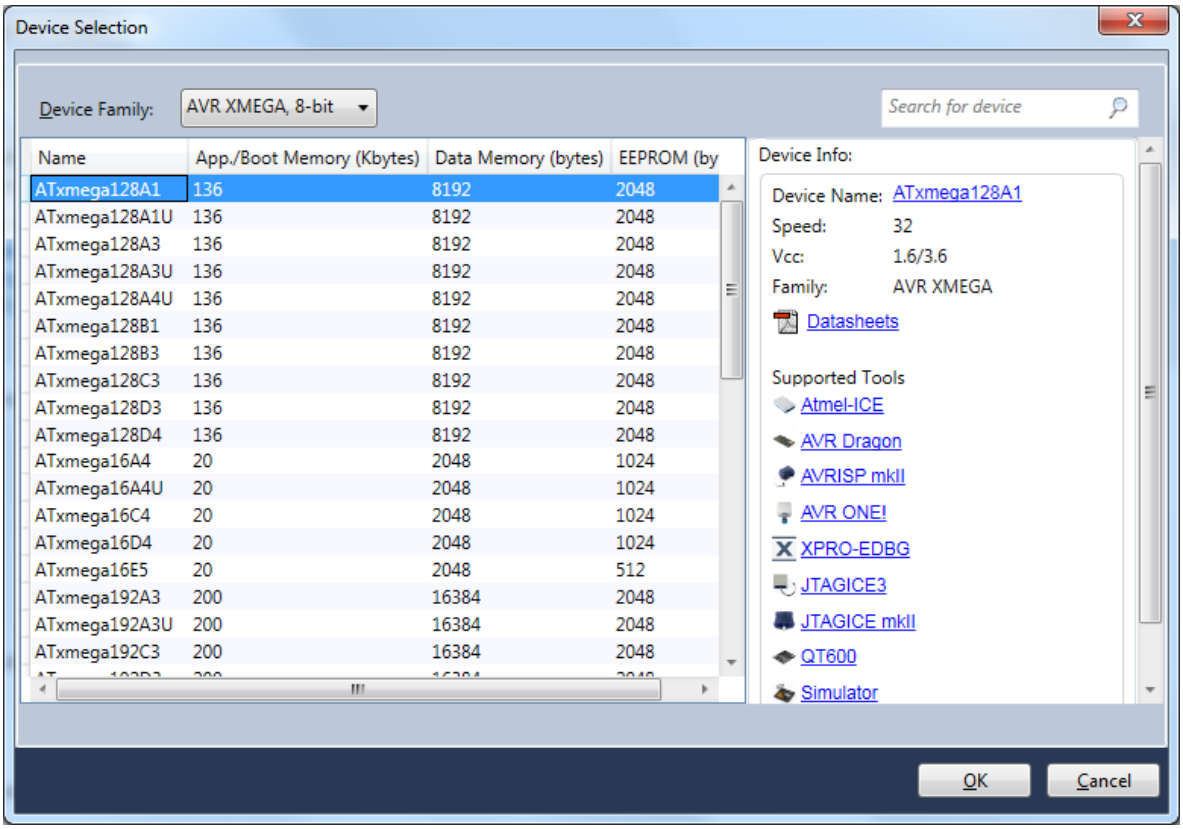


2. Select **C/C++→GCC C Executable Project** as a template, then specify a project name, select a location, and write a solution name for the project. A file with the same name as the project will be created and added to the project by default. It will contain an empty `main()` function. If you want to change the name of the initial file, just edit the main file name afterward. Press **OK** when you are satisfied with the settings.
3. Select **C/C++→GCC C Static Library Project** as a template, then specify a project name, select a location, and write a solution name for the project. This creates a Static Library (LIB) project, which is a good way to reuse code.

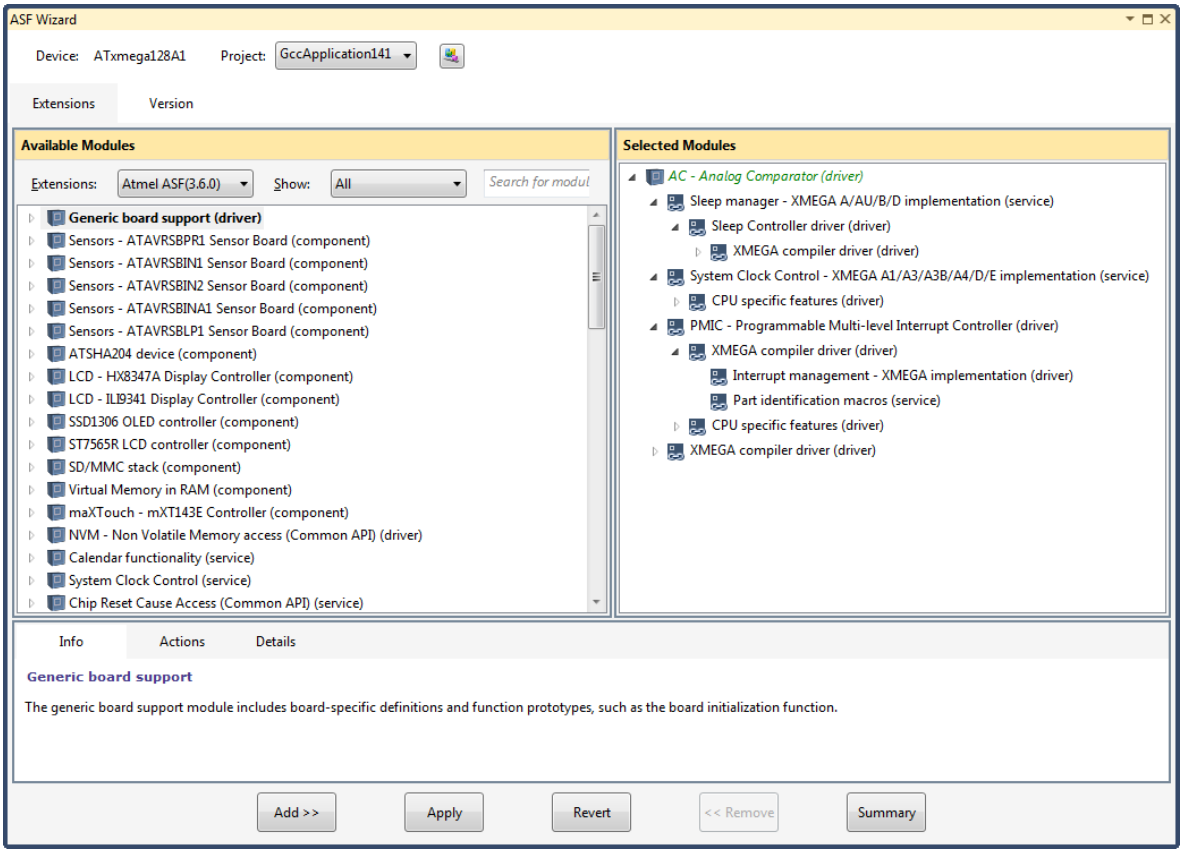
Tip:
See section Starting a New GCC Static Library Project to learn more about Static Library projects.

4. A device selection table will appear. Choose the appropriate target platform for your project. To start you can select the ATxmega128A1 device.

Figure 3-5. Device Selection



5. The project tree will be set up. Notice that the initial file created in step 2 has been added to the project node. Also, the initial file will be opened in the editor.
6. In order to facilitate applications development and verification you can also use the Driver Selection Wizard, invoked from **Project → ASF Wizard...**



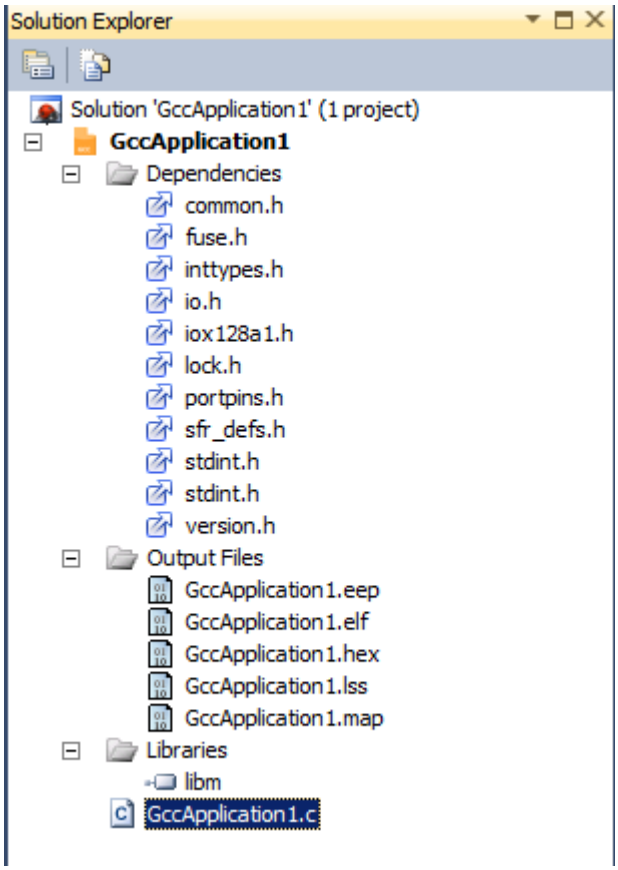
In the **ASF Wizard** you can select which Drivers, Components, and Services you would like to use in the project for current build architecture and board.

7. Now, write the following code into the open editor window.

```
#define MAXINT 200000

int main(void)
{
    unsigned int t=1000, k=0, l=5, pn=2;
    unsigned int primes[t];
    primes[0]=2;
    primes[1]=3;

    while (pn < t || primes[pn] < MAXINT)
    {
        for ( k = 0; k <= pn; k++)
        {
            if (l % primes[k] == 0)
            {
                goto otog;
            }
            else
            {
                if (k == pn)
                    primes[pn++]=l;
            }
        }
otog:
        l += 2;
    }
    return 0;
}
```

8. Build the project.

Figure 3-6. View of a GCC Project after Build Completed



Dependencies

All the included files are listed here. Double click on any file to open it in the editor.

Output Files

All output files will be displayed below this item.

Libraries

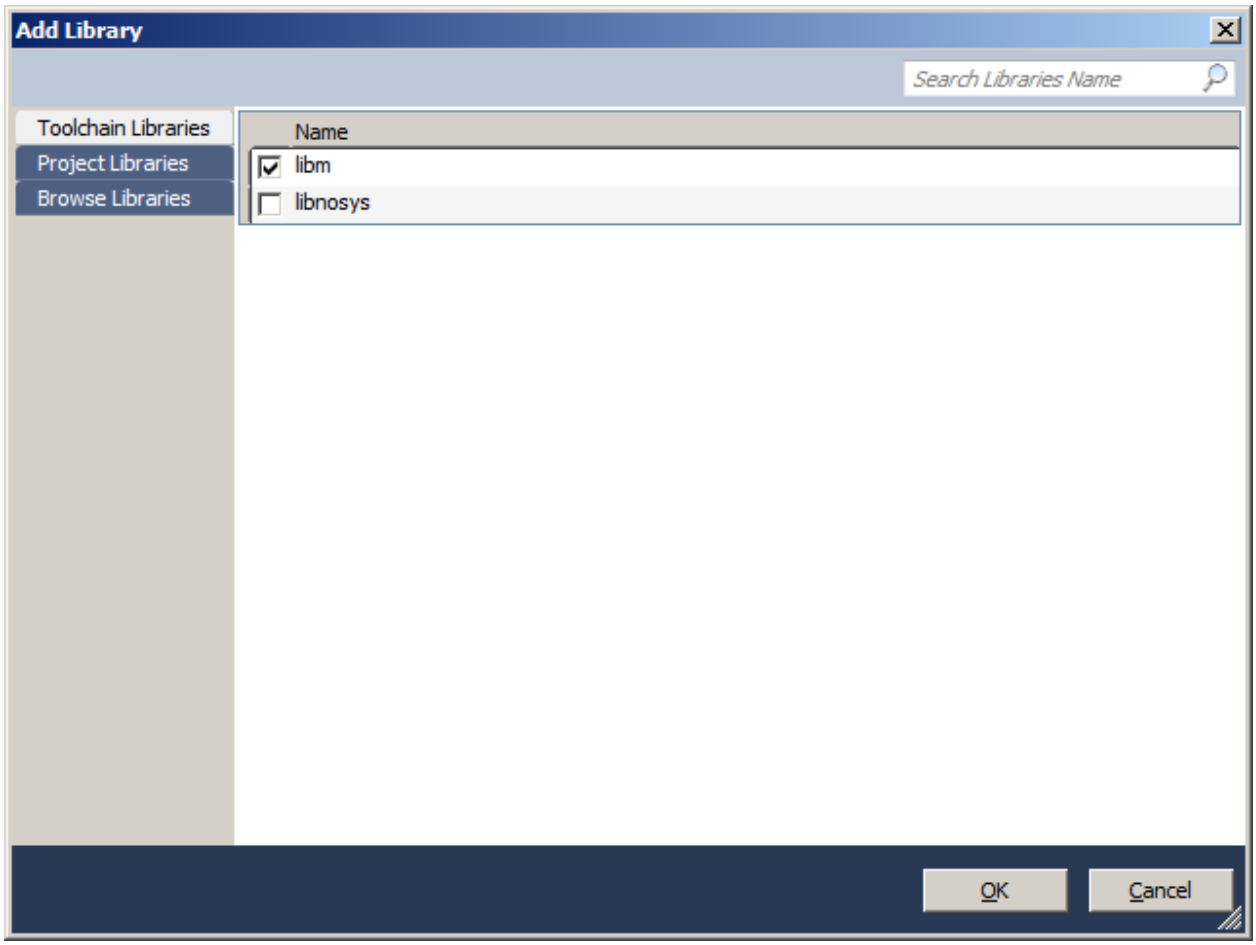
All Static Library files, Toolchain Library, and other Library Files will be displayed below this item.

Tip:
See section Library Options to know more about Library options.

3.2.3. Libraries Options

All Static Library files, Toolchain Library, and other Library Files will be displayed below this item.

Figure 3-7. Libraries



3.2.3.1. Toolchain Libraries

The toolchain libraries would be listed here.

The Library search path provided by the toolchain would be enumerated to form the library list.

3.2.3.2. Project Libraries

The projects available at the current Solution would be enumerated and the static libraries would be listed here.

3.2.3.3. Browse Libraries

You can browse for other libraries.

3.2.3.4. How to Add Project Library

Tip:
Ensure you have static library projects in the current solution.

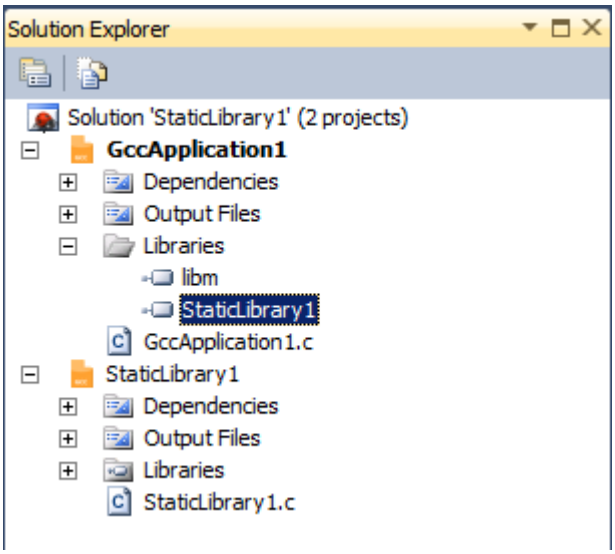
Right click on Project or Libraries Node in the project to invoke "Add Library" Wizard.

Select Project Libraries Tab; here would see the all the static libraries in current solution listed.

Select the Static Library which you would like to add.

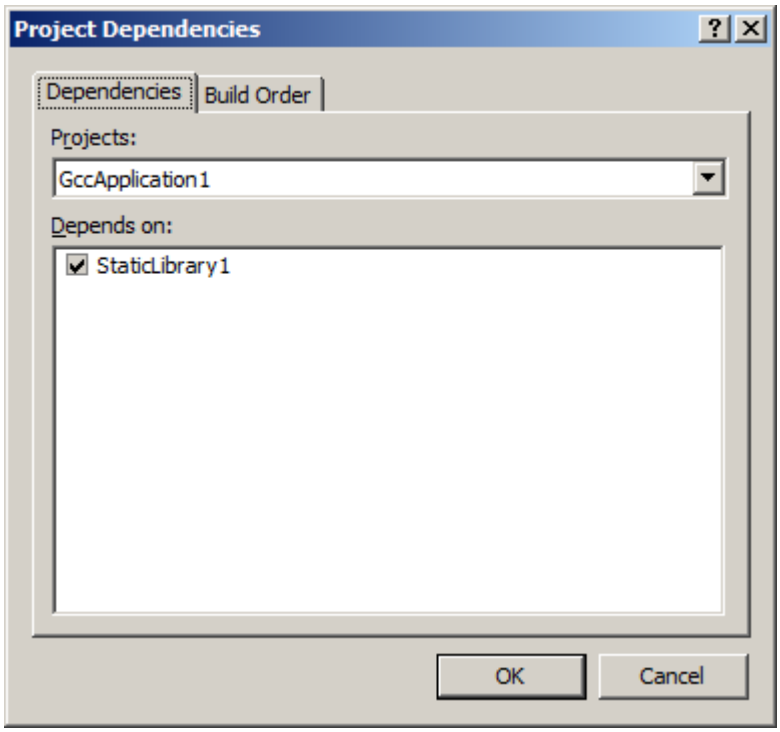
Click OK.

Figure 3-8. View of a Project after Adding Libraries



Also you will see that at **Project** → **Project Dependencies** Static Library added.

Figure 3-9. View of a Project Dependencies after Adding Libraries



3.2.3.5. How to Add Toolchain Library

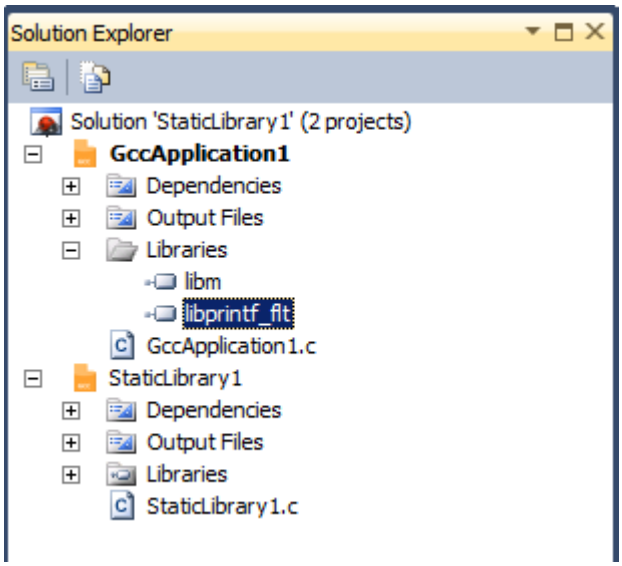
Right click on Project or Libraries Node in the project to invoke "Add Library" Wizard.

Select Toolchain Libraries Tab; here you will see the available toolchain libraries for the currently selected toolchain for project.

Select the libraries which you like to add.

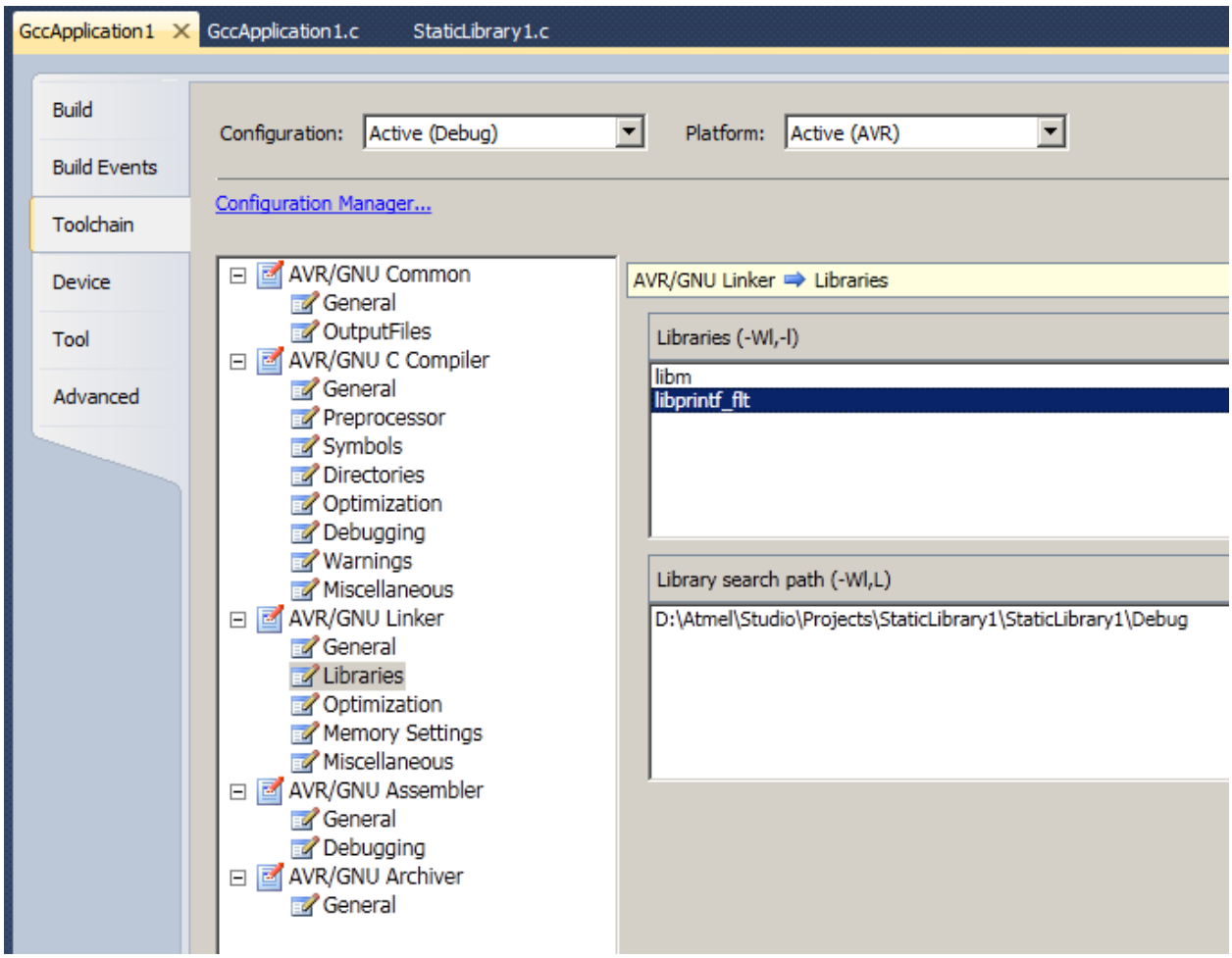
Click OK.

Figure 3-10. View of a Project after Adding Libraries



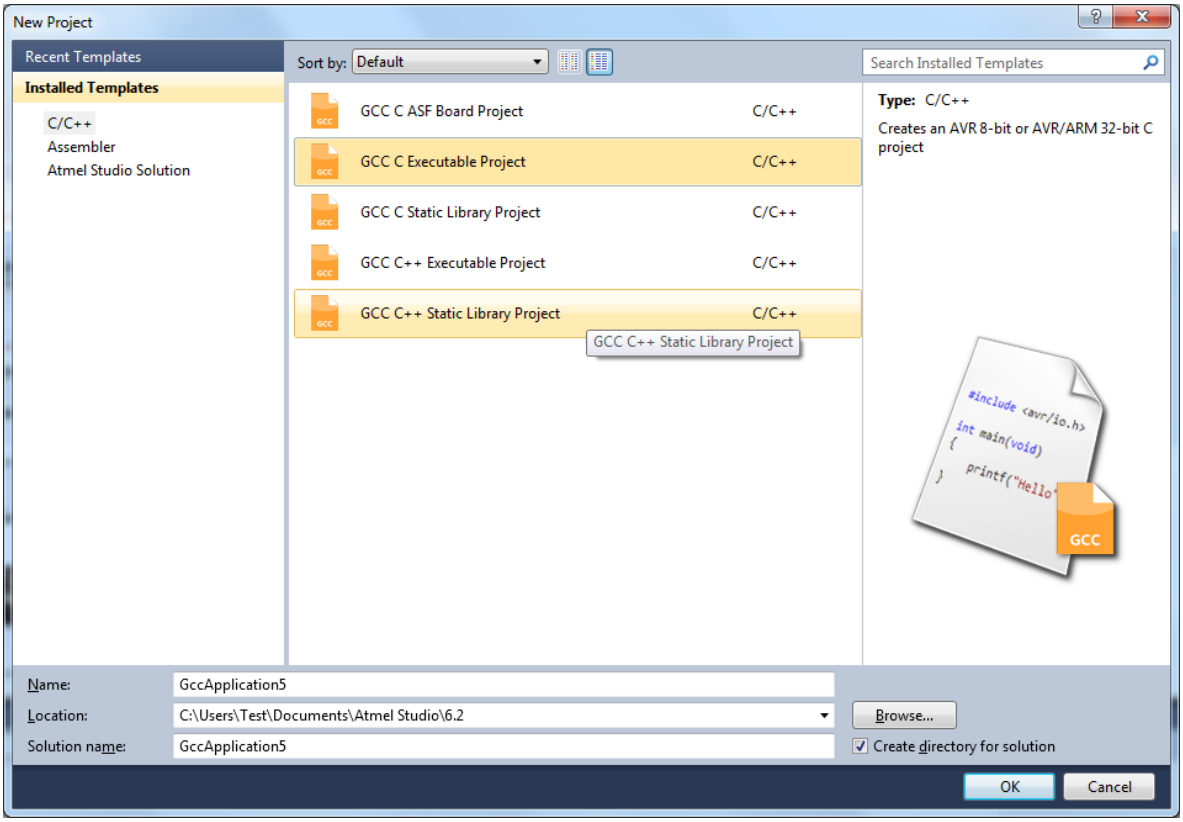
You will also be able to see the new library added in the Toolchain Linker Settings.

Figure 3-11. View of a Linker Option after Adding Libraries



3.2.4. Starting a New GCC Project for SAM (ARM) Device

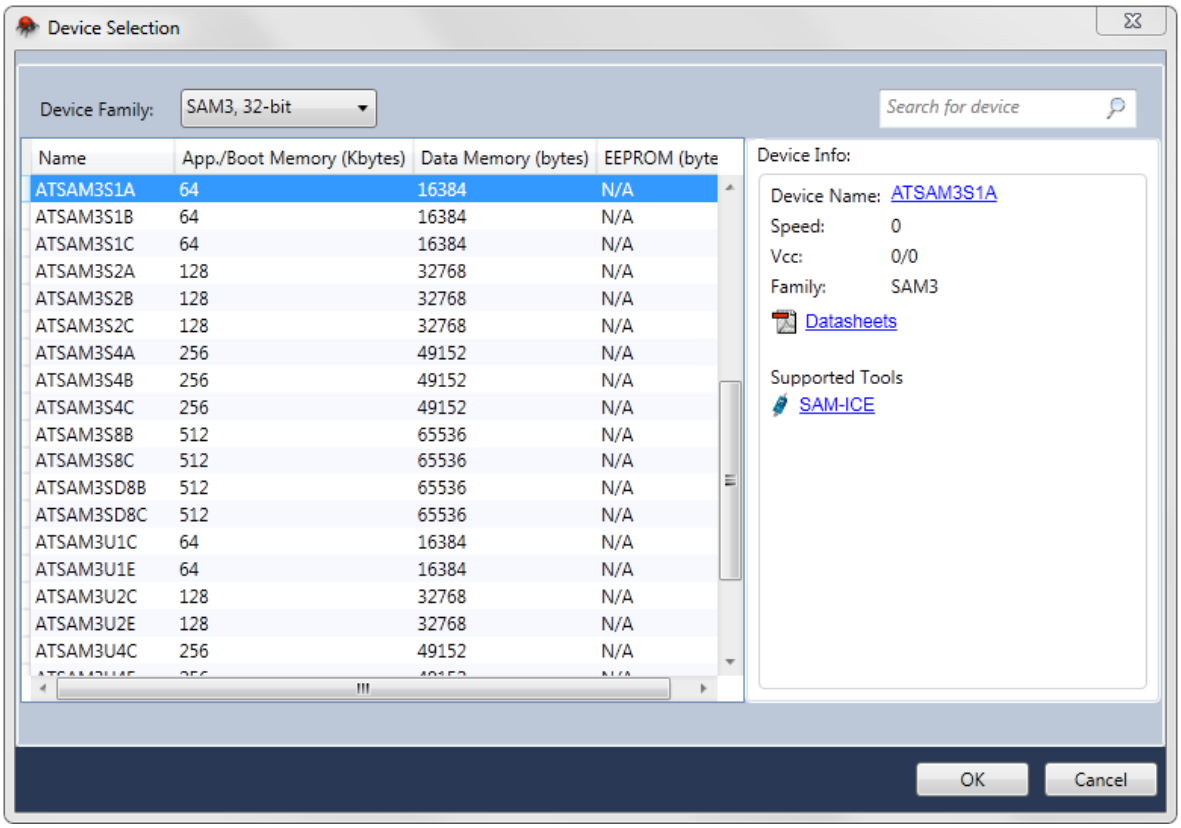
1. Create a new project by selecting **New Project** from the **Project** menu. This will open the **Project Wizard**.



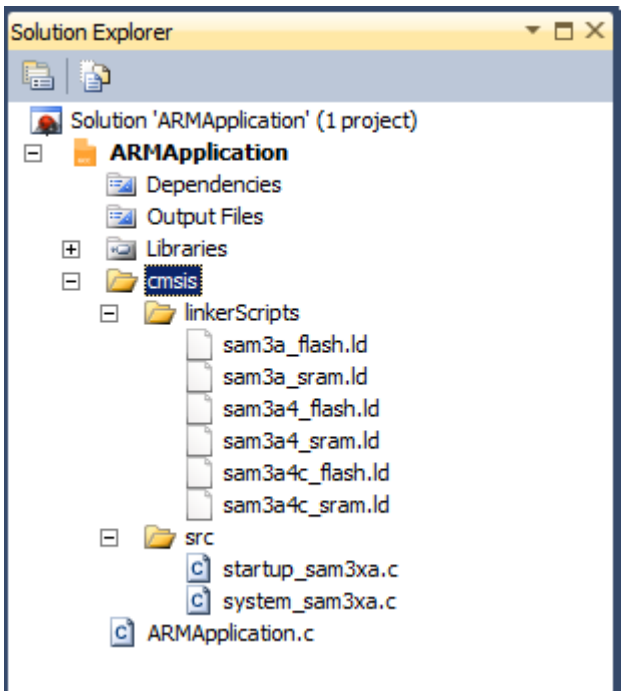
2. Select **C/C++ → GCC C Executable Project** as a template, then specify a project name, select a location, and write a solution name for the project. Some start-up files will be added to the project by default, which will contain some device specific functions and libraries. Press **OK** when you are satisfied with the settings.
3. Select **C/C++ → GCC C Static Library Project** as a template, then specify a project name, select a location, and write a solution name for the project. This creates a Static Library (LIB) project, which is a good way to reuse code.

Tip:
See section Static Library Project to learn more about Static Library projects.

4. A device selection table will appear. Choose the device family as SAM3 or SAM4 and select the target platform for your project. To start you can select the ATSAM3S1A device.

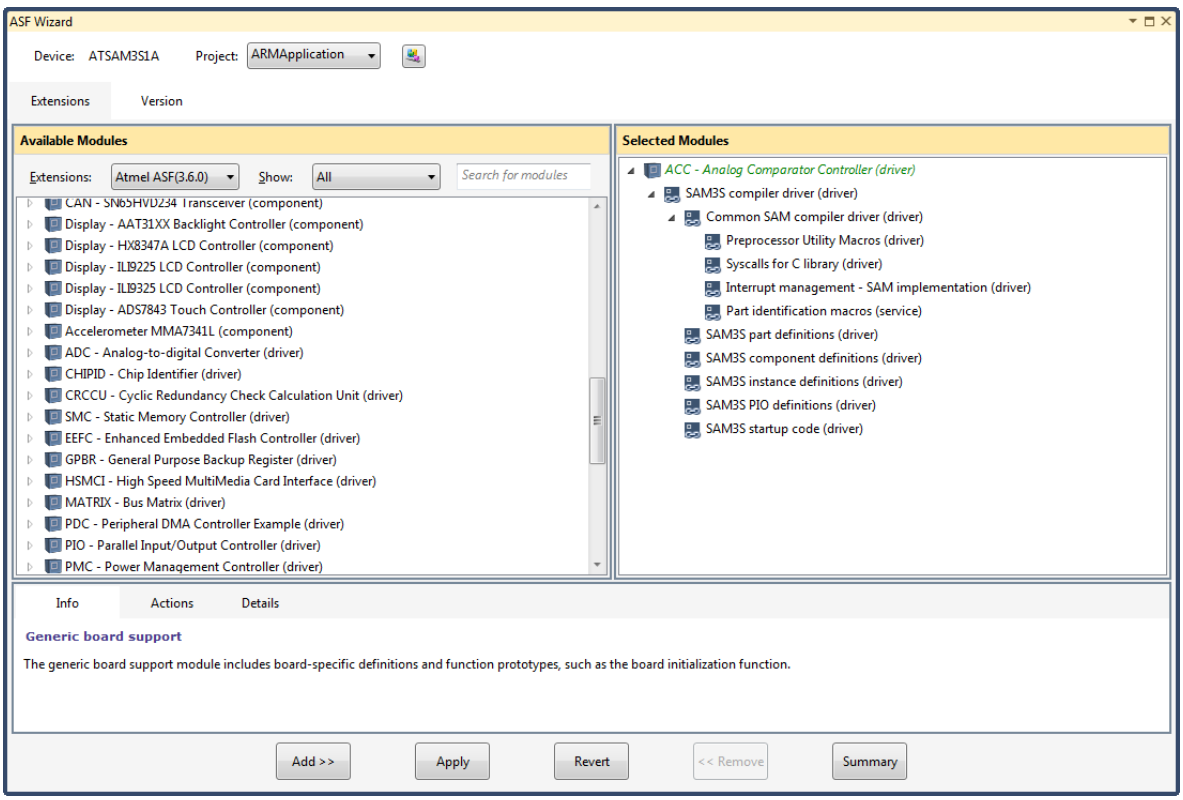


5. The project tree will be set up. Notice that the initial files created in step 2 has been added to the project node. Also, the file containing main() function will be opened in the editor.
Here is a list of files that will be created:
   - A file with the same name as the project will be created and added to the project by default. It will contain the `main()` function.
   - A startup file(startup_*.c) will be available at "cmsis\src" directory. It contains the default interrupt handlers for all the peripherals.
   - A system file(system_*.c) available at "cmsis\src" provides the system level initialization functions that are called on start-up
   - Linker scripts with appropriate sections based on the device will be created at "cmsis \LinkerScripts" directory in the project folder
   - In case if you have deleted any files in cmsis folder and want to revert it back or if you have changed the device, just right click the Project and click "CMSIS Update from Atmel" to get the appropriate files.



Note: It is recommended not to change the contents of the startup_*.c and system_*.c files unless you have no other choice. These startup, system, and linker scripts will not be created for ARM static library projects.

6. In order to facilitate applications development and verification you can also use the Driver Selection Wizard, invoked from **Project → ASF Wizard.**



In the **ASF Wizard** you can select which Drivers, Components, and Services you would like to use in the project for current build architecture and board.

7. Now, write the following code into the open editor window:

```
#define MAXINT 200000

int main(void)
{
    unsigned int t=1000, k=0, l=5, pn=2;
    unsigned int primes[t];
    primes[0]=2;
    primes[1]=3;

    while (pn < t || primes[pn] < MAXINT)
    {
        for ( k = 0; k <= pn; k++)
        {
            if (l % primes[k] == 0)
            {
                goto otog;
            }
            else
            {
                if (k == pn)
                    primes[pn++]=l;
            }
        }
otog:
        l += 2;
    }
    return 0;
}
```

8. Build the project.

3.2.5. Code Editing

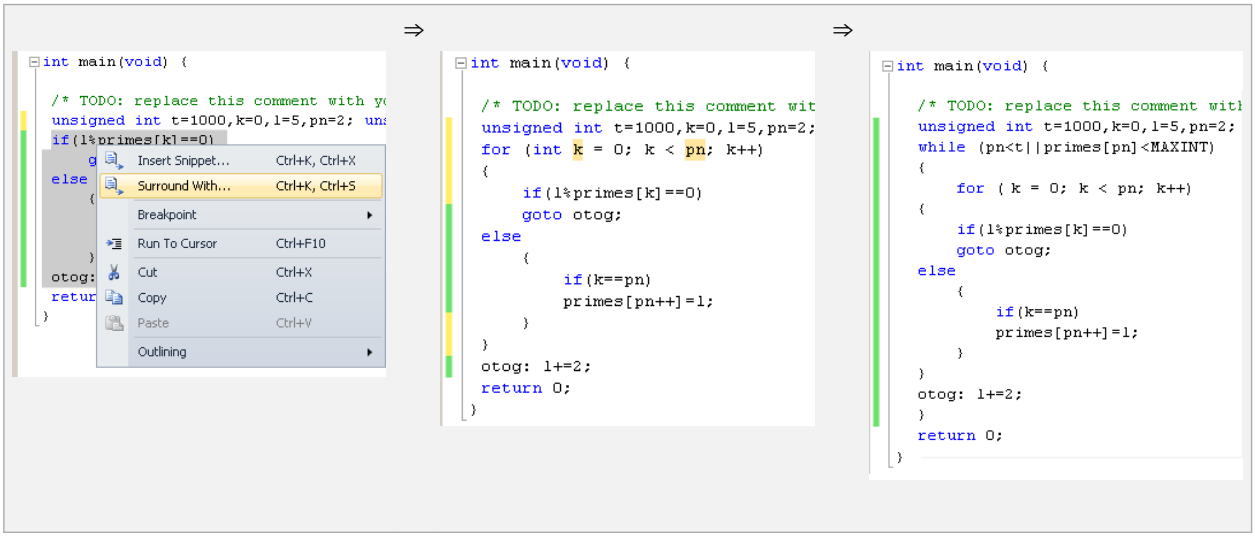
For the following part of the introduction we will reuse the same code as you have previously seen.

```
#define MAXINT 200000

int main(void)
{
    unsigned int t=1000, k=0, l=5, pn=2;
    unsigned int primes[t];
    primes[0]=2;
    primes[1]=3;

    while (pn < t || primes[pn] < MAXINT)
    {
        for ( k = 0; k <= pn; k++)
        {
            if (l % primes[k] == 0)
            {
                goto otog;
            }
            else
            {
                if (k == pn)
                    primes[pn++]=l;
            }
        }
otog:
        l += 2;
    }
    return 0;
}
```

Atmel Studio has a rich editor that is made even richer by Atmel and third-party plugins. Atmel Studio has an automatic code generation faculty for snippets of C source code. To use it select and right click the part of the code you wish to enclose in a conditional structure (like `for`,`while`,`if` … etc).

Using the code snippets you can add parts to your core source. In some snippets the variable names and exit conditions are parametric within the IDE, so as if only one instance is changed all instances within the snippet will also change, such is the case of `for` loop.

Table 3-1. Using "Surround With"



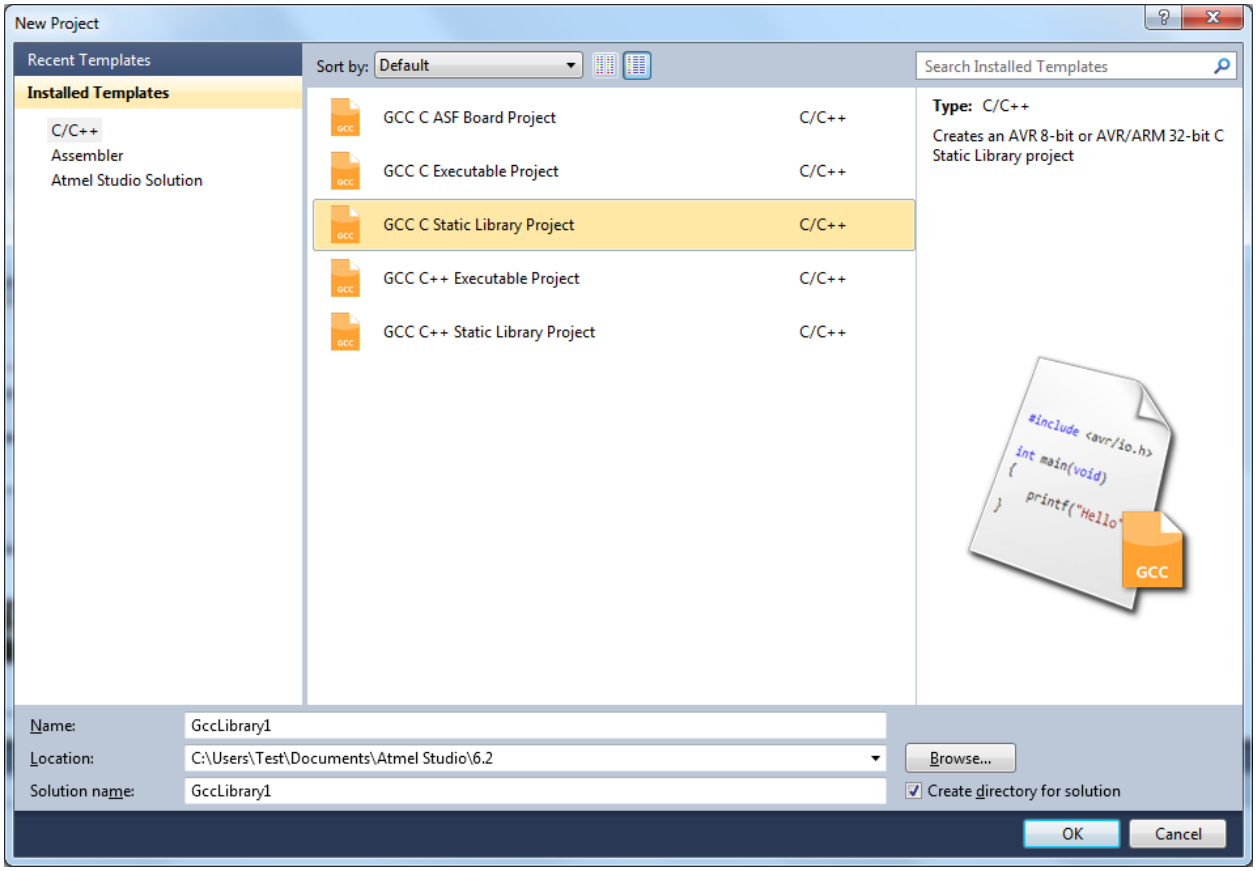
3.2.6. Starting a New GCC Static Library Project

3.2.6.1. Why Static Libraries

Static Libraries (LIB) is a good way to reuse code. Rather than re-creating the same routines/functions in all the programs, the user can write them once and reference from the applications that need the functionality.

3.2.6.2. Create New Static Library Project

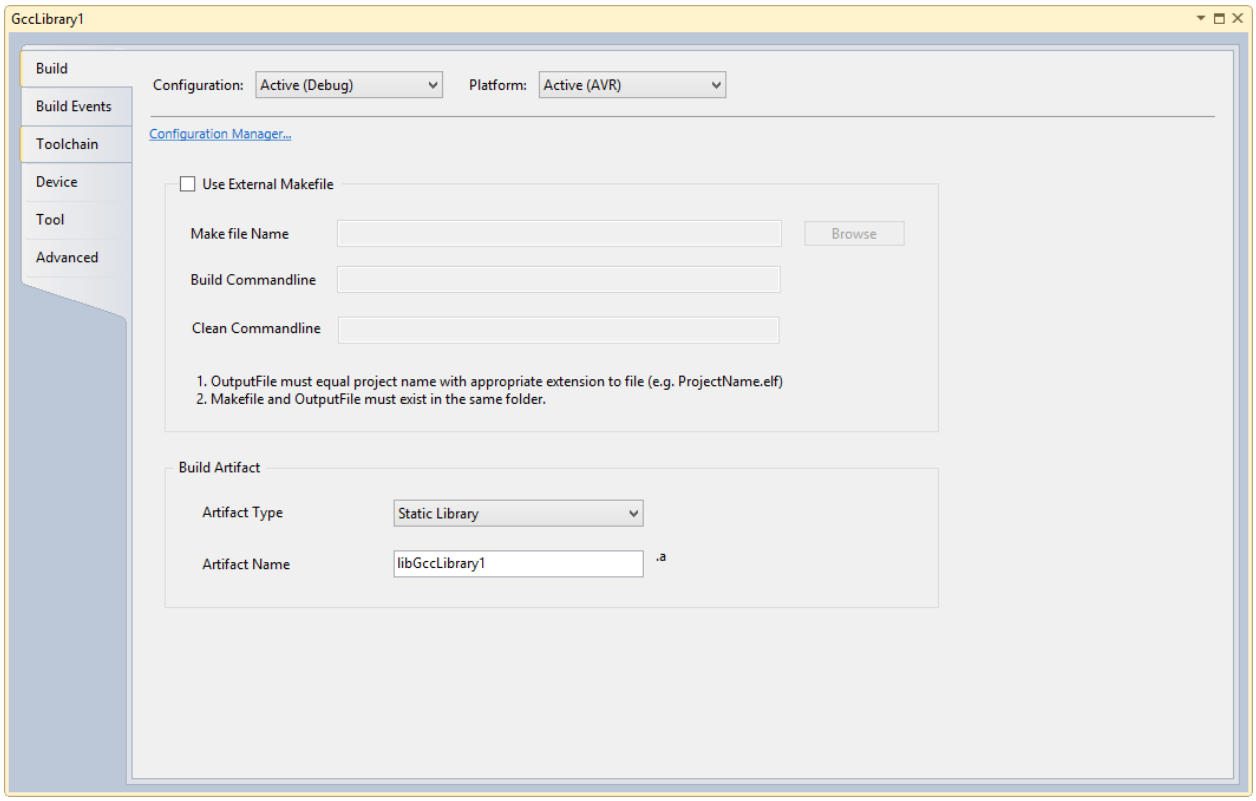
Figure 3-12. New Static Library Project



Click **OK** to create the Static Library project. A default source file with the same name as the project will be added to the solution. You may then write and compile your routines/functions. You can also add new source files or header files into the project.

Open the Project Properties on the menu **Project** → **"*Your_project_name* Properties"**. This menu item is only available when a Static Library project is open. Select the **Build** property page. Here you will see that the **Artifact Type** is selected as **Static Library**.

Figure 3-13. Static Library Build Properties



Compile the project by selecting **Build Solution** from the **Build** menu. This creates a Static Library, which can be used by other programs.

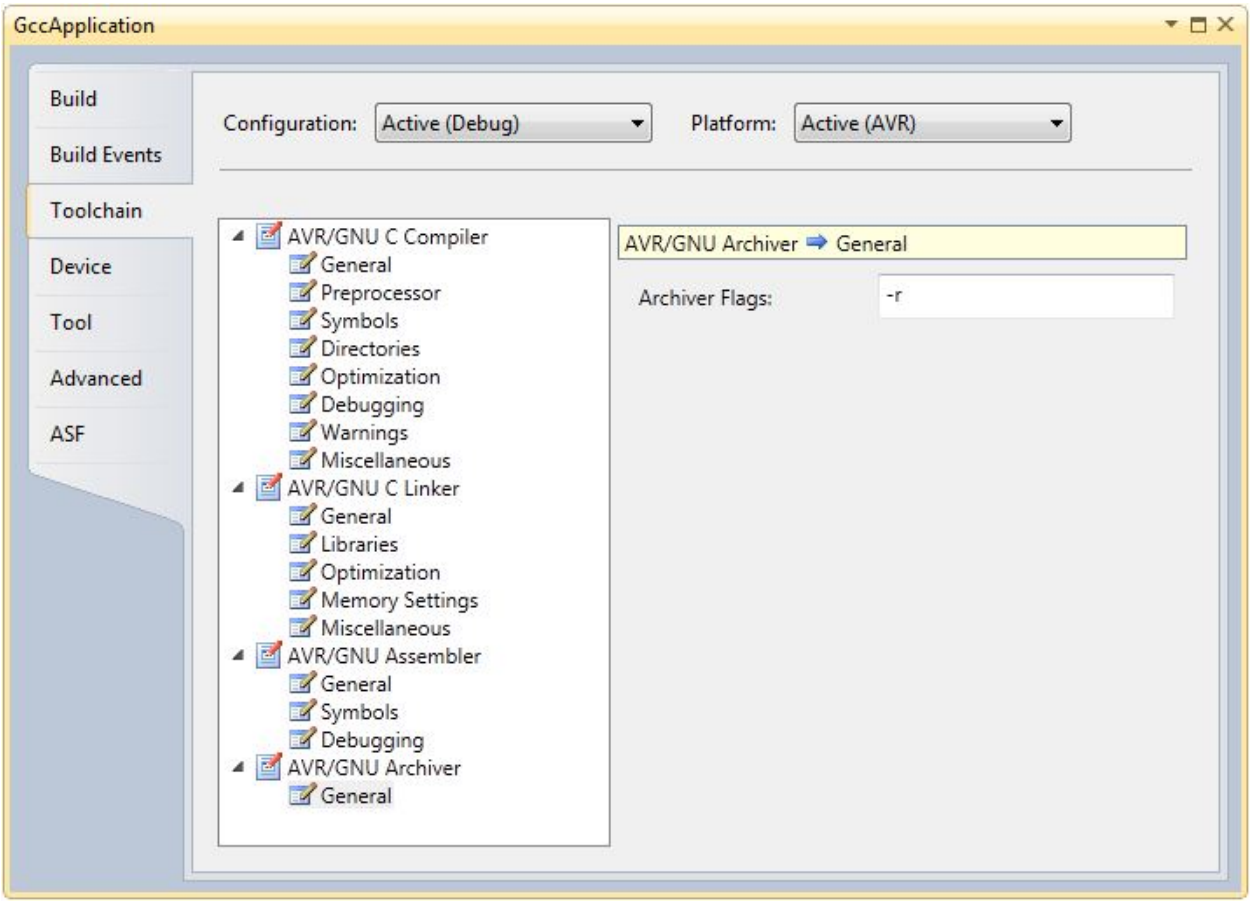
3.2.6.3. Static Library Project Options (AVR/GNU Archiver)

The AVR/GNU archiver, avr-ar, combines a collection of object files into a single archive file, also known as a library.

Open the Project Properties on the menu **Project** → **"*Your_project_name* Properties"**. This menu item is only available when a Static Library project is open. In order to configure Static Library options, click on the **Toolchain** property tab.

In the Toolchain property page, you will see **AVR/GNU Archiver** active and enabled. You may also see that the **AVR/GNU Linker** is disabled for a static library project.

Figure 3-14. AVR/GNU Archiver Setup Dialog, Command Line Shown



You can set the AVR/GNU Archiver flags at the **Archiver Flags** textbox in the above **General** options.

Now, save the project and compile by selecting **Build Solution** from the **Build** menu.

3.2.7. GCC Project Options and Configuration

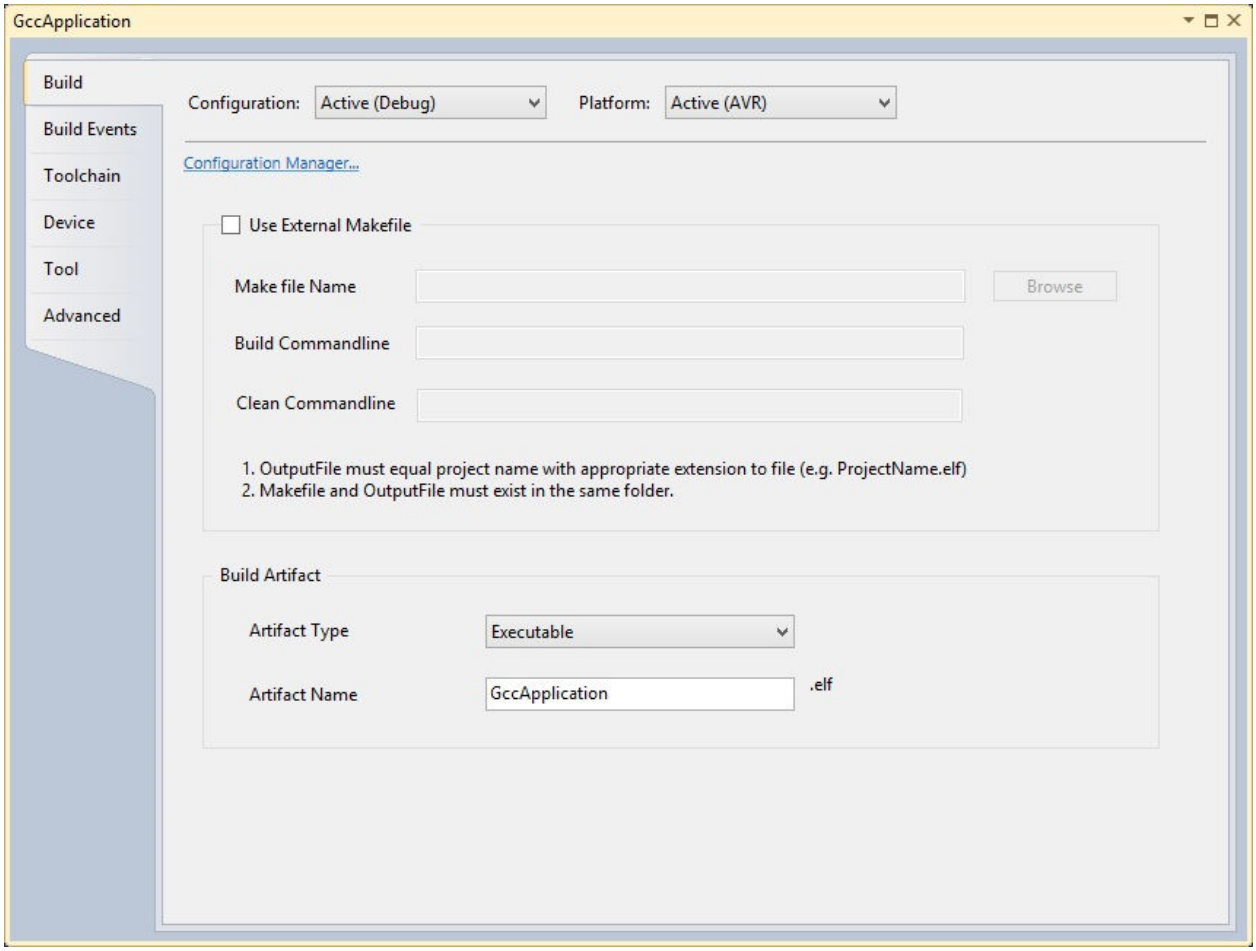
Project options and configuration can be set up by either right clicking on the Solution Explorer → **Project Properties**, or by pressing Alt Enter .

This will call up the Project properties window, it has seven tabs:

If a tab supports properties that are configuration specific, then the tab has two slide-down menus: The **Configuration** field defines the project configurations to modify. By default, two configurations are provided in each project - Debug and Release. The **Platform** field is set to AVR. If a tab supports configuration independent properties, then the **Configuration** and **Platform** fields are disabled.
Note: Use the "**Save All** (Ctrl Shift S)" from the **File** menu or toolbar to update the changes in the project file whenever changes are made.

3.2.7.1. Build Options

Figure 3-15. Build Configuration



In the Build tab page, you can configure whether you want to use an external Makefile for your project. In that case, just tick the **Use External Makefile** check box and browse to select the correct path of make file.

Build Commandline will be provided to the external makefile when build is invoked for the project. The default build target is "all".

Clean Commandline will be provided to the external makefile when clean is invoked for the project. The default clean target is "clean".

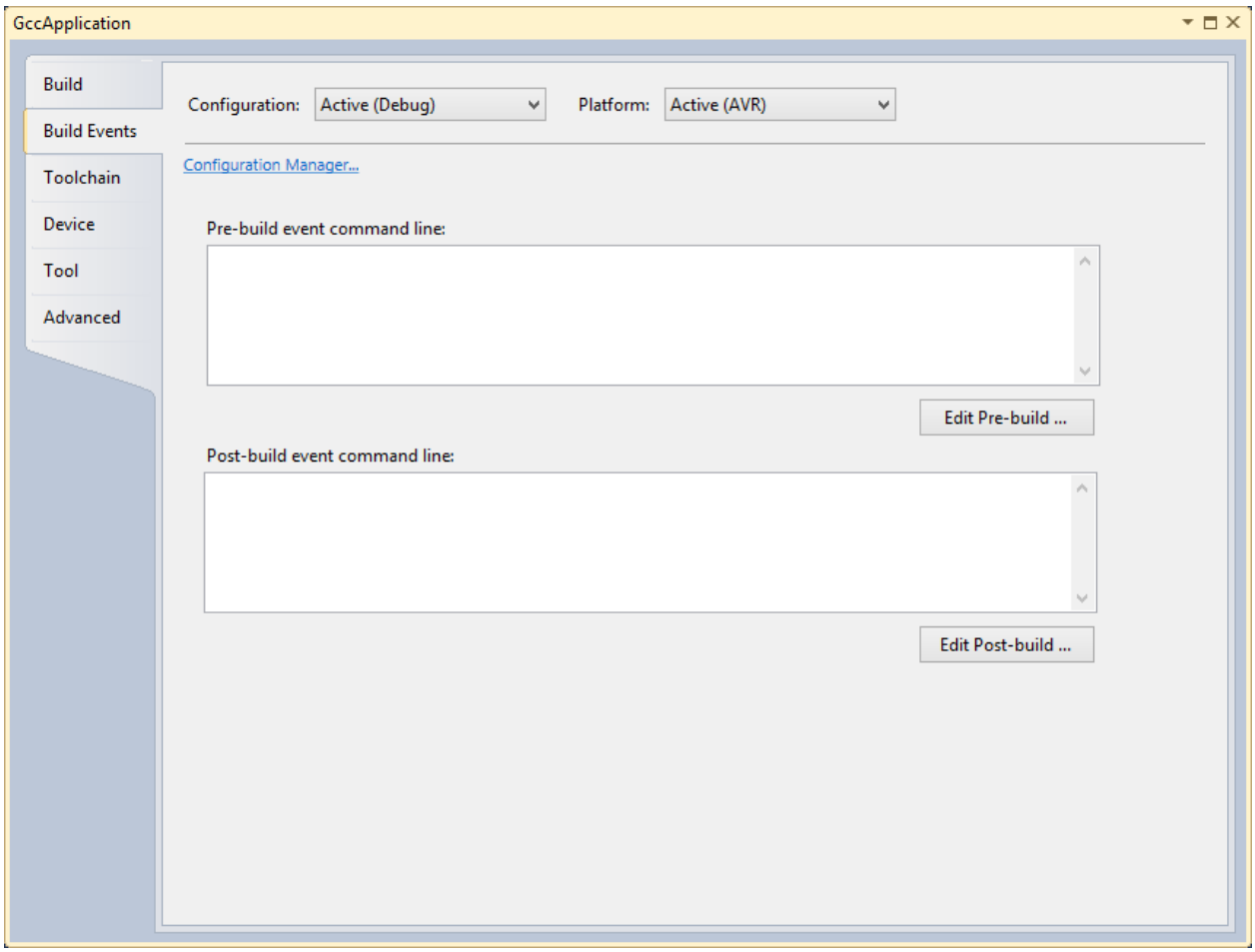
Besides the external make file configuration, you can also specify the type of application to build. The options are Executable or Static Library, which can be selected using the Artifact Type combo box.

Note: Custom makefile must fulfill those conditions:

1. Target name must equal project name.
2. Makefile and target must exist in the same folder (can be referenced with NTFS links too).

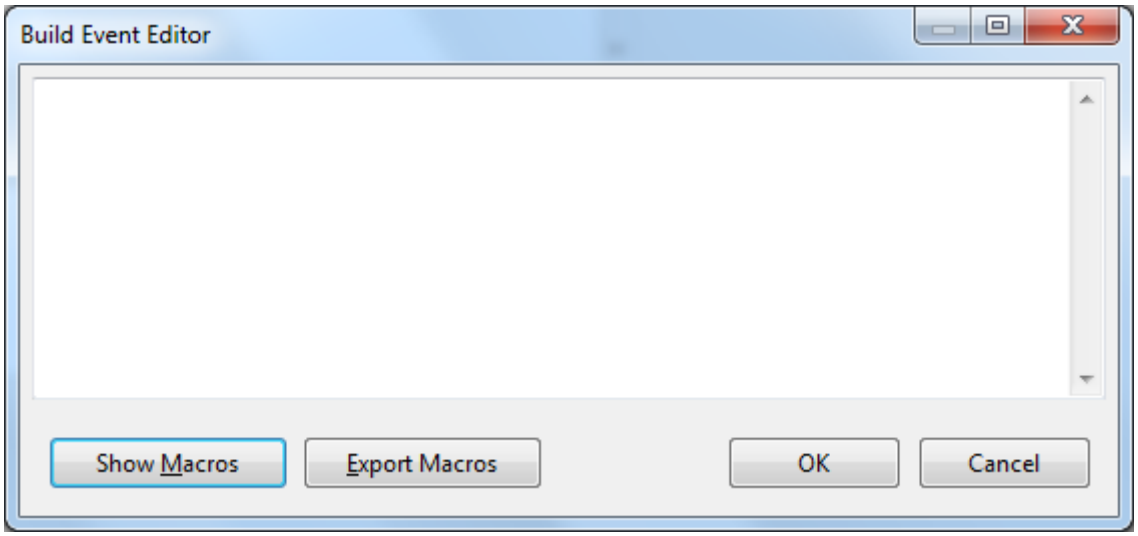
3.2.7.2. Build Events

Figure 3-16. Build Events Options



The **Build events** tab contains a list of scheduled events for each configuration, launched by the pre-build and post-build scripts. These events can be added, deleted, or modified by clicking either the **Edit pre-build...** or **Edit post-build...** buttons. Upon clicking these buttons, you should manually add your commands in the following dialog. As of the current release it is possible to use environment variables and values declared within them as a link with other available applications. In order to use that function press the **Show Macros** button.

Figure 3-17. Build Event Editor



Macros

Expands the edit box to display the list of macros/environment variables to insert in the command line edit box.

Macro Table

List the available macros/environment variables and its value. You can select only one macro at a time to insert into the command line edit box. MSBuild also provides a set of reserved properties that store information about the project file and the MSBuild binaries. These properties may also be listed in the edit box.

See Macros/environment variables below for a description which are specific to Atmel Studio.

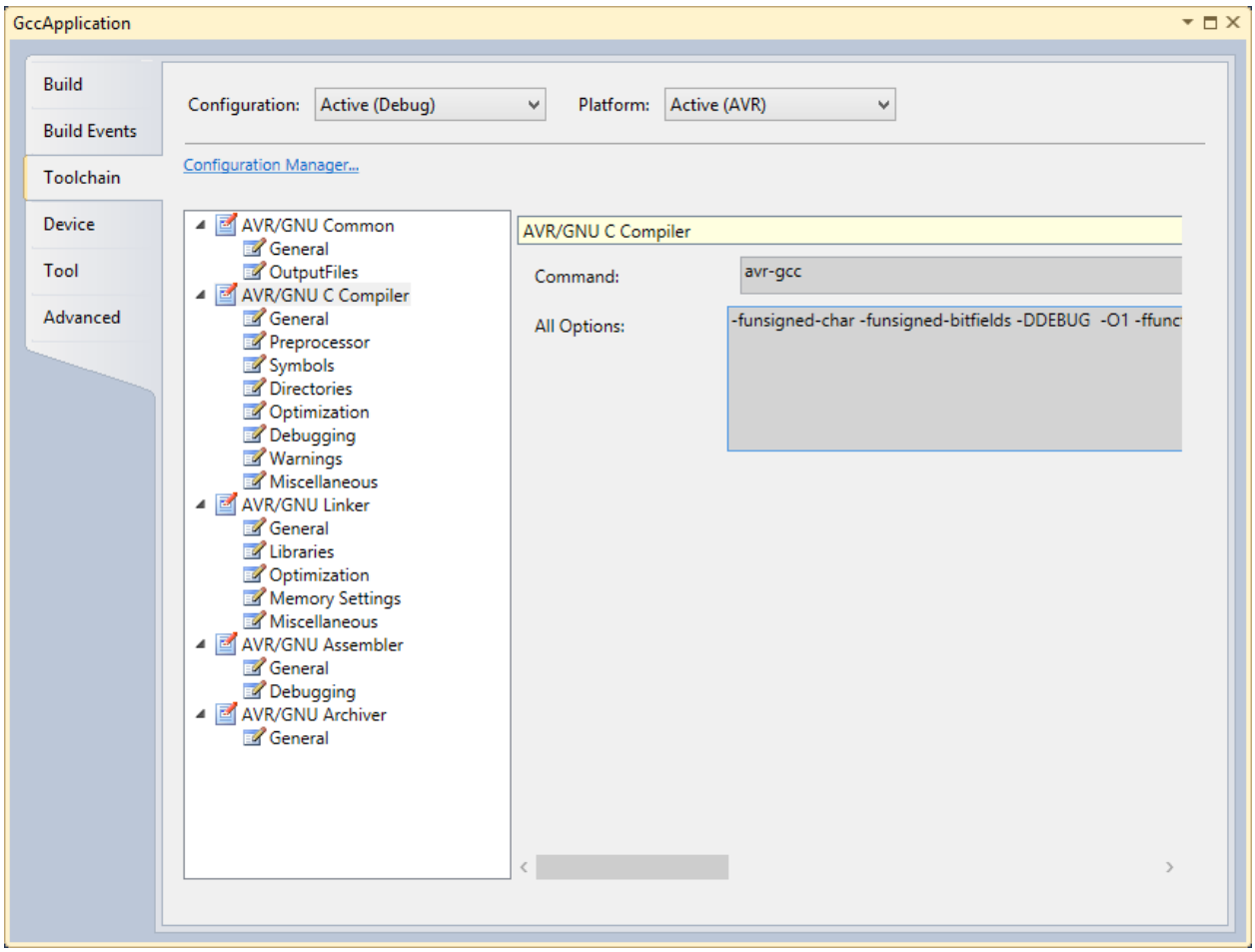
Table 3-2. Atmel Studio Build Macro Table

Macro	Description
$(AVRSTUDIO_EXE_PATH)	The installation directory of Atmel Studio (defined with drive and path)
$(SolutionDir)	The directory of the solution (defined with drive and path)
$(SolutionPath)	The absolute path name of the solution (defined with drive, path, base name, and file extension)
$(SolutionFileName)	The file name of the solution
$(SolutionName)	The base name of the solution
$(SolutionExt)	The file extension of the solution. It includes the '.' before the file extension.
$(Configuration)	The name of the current project configuration, for example, "Debug"
$(Platform)	The name of the currently targeted platform, for example, "AVR"
$(DevEnvDir)	The installation directory of Atmel Studio (defined with drive and path)
$(ProjectVersion)	The version of the project

Macro	Description
$(ProjectGuid)	A unique identifier of the project
$(avrdevice)	The name of the currently selected device
$(avrdeviceseries)	The series of the selected device. Used internally by the Atmel Studio.
$(OutputType)	Defines if the current project is an Executable or a Static Library type
$(Language)	Language of the current project; for example, C, CPP, or Assembler
$(OutputFileName)	The file name of the primary output file for the build (defined as base file name)
$(OutputFileExtension)	The file extension of the primary output file for the build. It includes the '.' before the file extension
$(OutputDirectory)	The absolute path of the output file directory
$(AssemblyName)	The assembly name of the primary output for the build
$(Name)	The base name of the project
$(RootNamespace)	The base name of the project
$(ToolchainName)	The name of the toolchain
$(ToolchainFlavour)	The name of the toolchain's compiler
Macro	Description

3.2.7.3. Compiler and Toolchain Options

Figure 3-18. Compiler and Toolchain Options



AVR GNU C Compiler Options

Table 3-3. AVR GNU C compiler Options

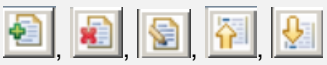
Option	Description
General options	
-mcall-prologues	Use subroutines for functions prologues and epilogues
-mno-interrupts	Change stack pointer without disabling interrupts
-funsigned-char	Default char type is unsigned
-funsigned-bitfield	Default bit field is unsigned
Preprocessor options	
-nostdinc	Do not search system directories
-F	Preprocess only
Symbols options	

Option	Description
There one can define (-D) or undefine (-U) a number of in-source symbols. New symbol declarations can be added, modified, or reordered, using the interface buttons below: • Add a new symbol. This and all following icons are reused with the same meaning in other parts of Atmel Studio interface. • Remove a symbol. • Edit symbol. • Move the symbol up in the parsing order. • Move the symbol down in the parsing order.	
Include directories	
Contains all the included header and definition directories, can be modified, using the same interface as symbols.	
Optimization options	
Optimization level (drop down menu): -O0, -O1, -O2, -O3, -Os	No optimization, optimize for speed (level 1 - 3), optimize for size
Other optimization flags (manual input form)	Here you should write optimization flags specific for the platform and your requirements
-ffunction-sections	Prepare functions for garbage collection, if a function is never used, its memory will be scrapped
-fpack-struct	Pack structure members together
-fshort-enums	Allocate only as many bytes needed by the enumerated types
-mshort-calls	Use rjmp/rcall limited range instructions on the >8K devices
Debug options	
Debug level (drop down menu): none, -g1, -g2, -g3	Specifies the level of tracing and debugging code and headers left or inserted in the source code
Other debug options (form field)	Architecture specific debug options
Warning messages output options	
-Wall	All warnings
-Werror	Escalate warnings to errors
-fsyntax-only	Check syntax only

Option	Description
-pedantic	Check conformity to GNU, raise warnings on non-standard programming practice
-pedantic-errors	Same as above, plus escalate warnings to errors
Miscellaneous options	
Other flags (form field)	Input other project-specific flags
-v	Verbose
-ansi	Support ANSI programs
-save-temps	Do not delete temporary files
Option	Description

AVR GCC Linker Options

Table 3-4. AVR GCC Linker Options

Option	Description
-Wl -nostartfiles	Do not use standard files
-Wl -nodefault	Do not use default libraries
-Wl -nostdlib	No start-up or default libraries
-Wl -s	Omit all symbol information
-Wl -static	Link statically
Libraries options	
Libraries -Wl, -l (form field)	You can add, prioritize, or edit library names here, using these buttons: 
Library search path -Wl,-L (form field)	You can add, prioritize or edit path where the linker will search for dynamically linked libraries, same interface as above
Optimization options	
-Wl, -gc-sections	Garbage collect unused sections
--rodata-writable	Put read-only data in writable spaces
-mrelax	Relax branches
Miscellaneous options	
Other linker flags (form field)	Input other project-specific flags

AVR Assembler Options

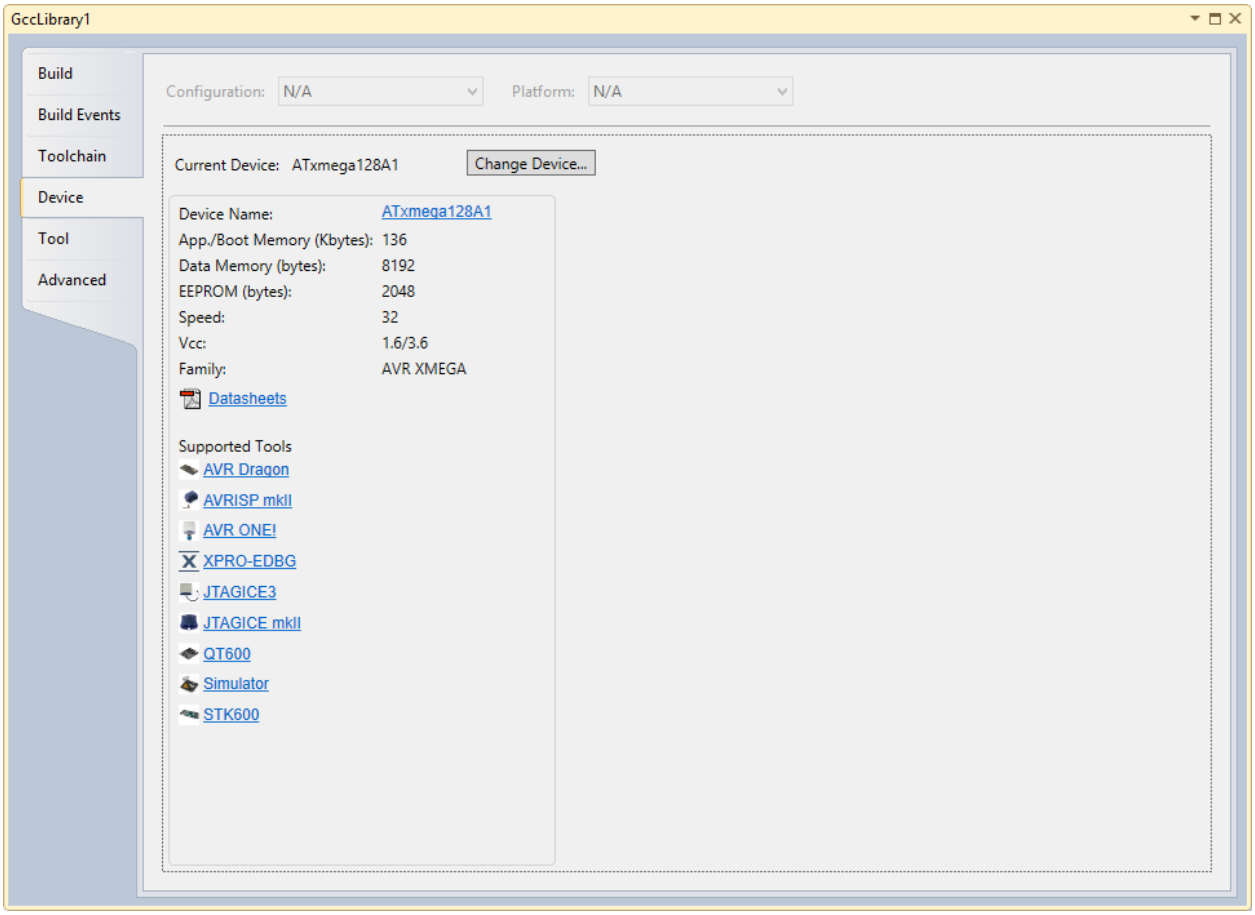
Table 3-5. AVR Assembler Options

Option	Description
Optimization options	
Assembler flags (form field)	Miscellaneous assembler flags
Include path (form field)	You can add, prioritize or edit path to the architecture and platform specific included files here
-v	Announce version in the assembler output
Debugging options	
Debugging level (drop down menu) -Wa -g1, -Wa, -g2, -Wa, -g3	Defines a level of debugging symbol and debugging source insertion

3.2.7.4. Device Options

This tab allows you to select and change the device for the current project and is similar to the device selector, see Figure 3-5.

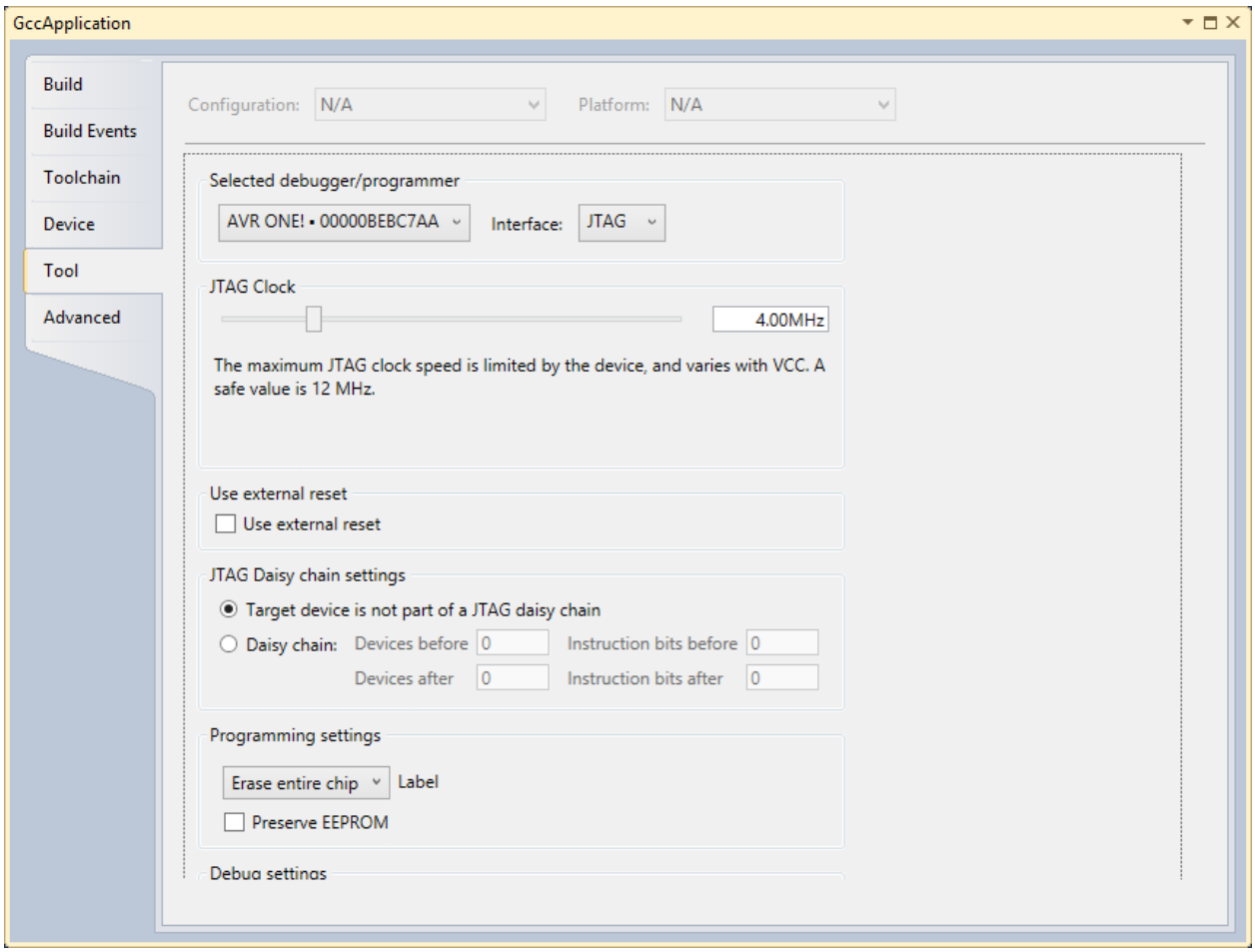
Tip:
Click on the **Device** button on the **Device and Debugger** toolbar to get to this tab quickly while editing.



3.2.7.5. Tool Options

This tab allows you to select and change the debugger platform for the current project.

Tip:
Click on the **Device** button on the **Device and Debugger** toolbar to get to this tab quickly while editing.



Select tool/debugger from the drop-down list. Current selection is shown.

Select Interface from the drop-down list. Current selection is shown.

Note: Only tools and interfaces valid for the current device selection are shown.

Further Properties are dependent on tool and interface selected.

JTAG

If you have selected JTAG as the programming interface, clock speed, use external reset - and daisy chain setting may be available. This depends on the tool and device.

JTAG clock

JTAG clock is the maximum speed the tool will try to clock the device at. The clock range is different for different tools and devices. If there are restrictions, they will be stated in a message below the clock slider.

Clock can be set to Manual (all tools), Auto (SAM-ICE only), or Adaptive (SAM-ICE only).

Use external reset

If checked, the tool will pull the external reset line low when trying to connect to the device.

JTAG daisy chain settings

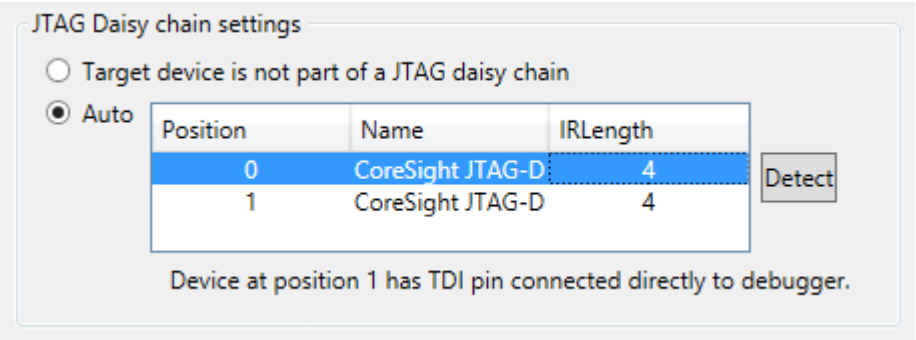
Specify the JTAG daisy chain settings relevant to the device to program.

Target is not part of a daisy chain. Select this option when the target device is not part of a daisy chain.

Daisy chain - Manual. Allows you to manually configure the JTAG daisy chain in case you are programming in a system-on-board.

- **Devices before** - specifies the number of devices preceding the target device.
- **Instruction bits before** - specifies the total size of the instruction registers of all devices, preceding the target device.
- **Devices after** - specifies the number of devices following the target device.
- **Instruction bits after** - specifies the total size of the instruction registers of all devices, following the target device.

Daisy chain-Auto. Automatically detects the devices in the JTAG daisy chain. Allows you to select the device in the JTAG daisy chain. Auto-detection is supported only for SAM devices.



PDI

The PDI interface has only one setting – the PDI clock speed.

PDI Clock is the maximum speed the tool will try to clock the device at. The clock range is different for different tools and devices. If there are restrictions, they will be stated in a message below the clock slider.

The clock can not be adjusted on all tools, so an empty **Interface settings** page will be presented.

Programming and debug settings

In the drop-down menu it is possible to specify which parts of memory that should be erased during a programming/debug cycle.

- **Skip programming** - specifies that no programming should occur. The tool will try to attach to the program already in memory.
- **Erase only program area** - specifies that only the program area of memory should be erased.
- **Erase entire chip** - specifies that the entire chip is to be erased.

The "Preserve Eeprom" option lets you decide whether EEPROM data should be written when launching a debug session. When this checkbox is checked, EEPROM data in the object file will be written to the device at the start of each debug session. The EESAVE fuse will be set and cleared accordingly.

When a device is programmed at the start of a debug session, the default behavior is to erase the content of the device (chip erase, if available). This can be changed by selecting a different option from the drop down box under "Programming settings".

Keep timers running in stop mode

When checked, the timers set in the program will continue to run even when the program breaks at breakpoint or is halted

This setting is only enabled for certain tool/interfaces.

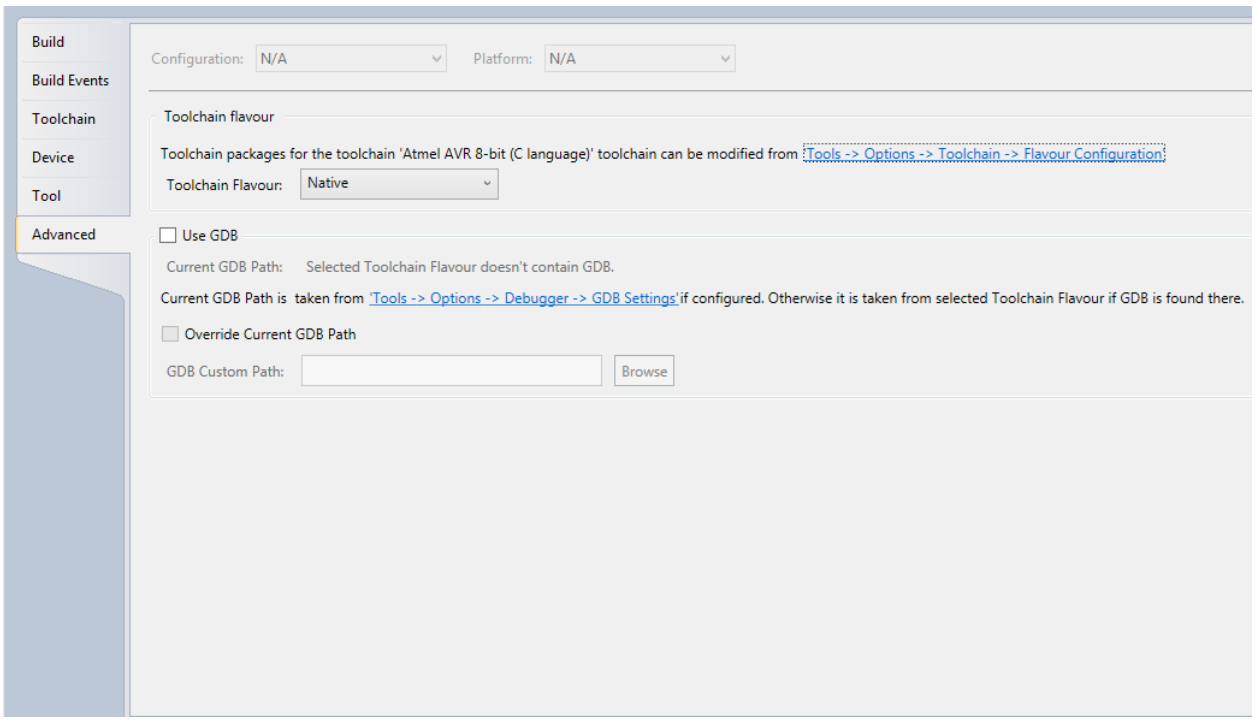
Override Vector Table Offset Register

At reset, load PC and SP from the specified address.

Note: The tool you select on this page is also used with the command **Start without Debugging**. That is why you also can select tools that do not have debugging capabilities. See Start without Debugging for more information.

3.2.7.6. Advanced Options

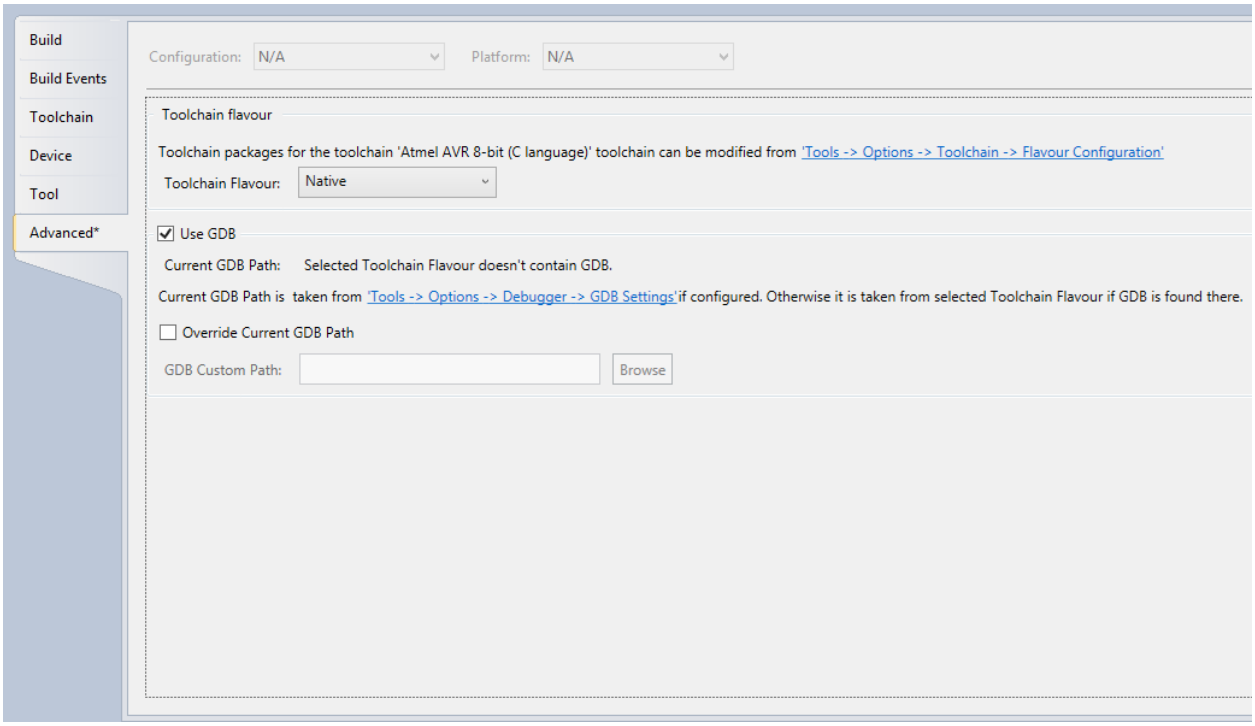
Setting ToolchainFlavour
This section allows you to set the flavour of the toolchain for the current project. Default flavour of the respective toolchain is selected. The toolchain path configured in the flavour is used for building the projects. For configuring and adding new flavours Toolchain.



Use GDB
This section allows you to select whether GDB has to be used for the current project. The Current GDB Path will be computed by the following

1. The Current GDB Path is taken from 'Tools → Options → Debugger → GDB Settings if configured GDB Settings.
2. Otherwise it is taken from selected Toolchain Flavor if GDB is found there.

The Current GDB Path will be overridden when we use the option **"Override Current GDB Path"** in the "Advanced" project property page.



Note: The option **"Use GDB"** is enabled by default for ARM based devices and also the following warning will be shown for AVR 32-bit devices if GDB is enabled.

3.2.7.7. Creating ELF Files with Other Memory Types

ELF files that contains data for all memory types can be made with the GCC compiler. The data sections are specified in code as shown in following code examples.

Creating ELF Files for tinyAVR, megaAVR, and XMEGA devices

This code shows how to add memory sections for tinyAVR, megaAVR, and XMEGA devices (except ATtiny4/5/9/10) . This example is for an ATxmega128A1 device. For other devices, it has to be modified accordingly.

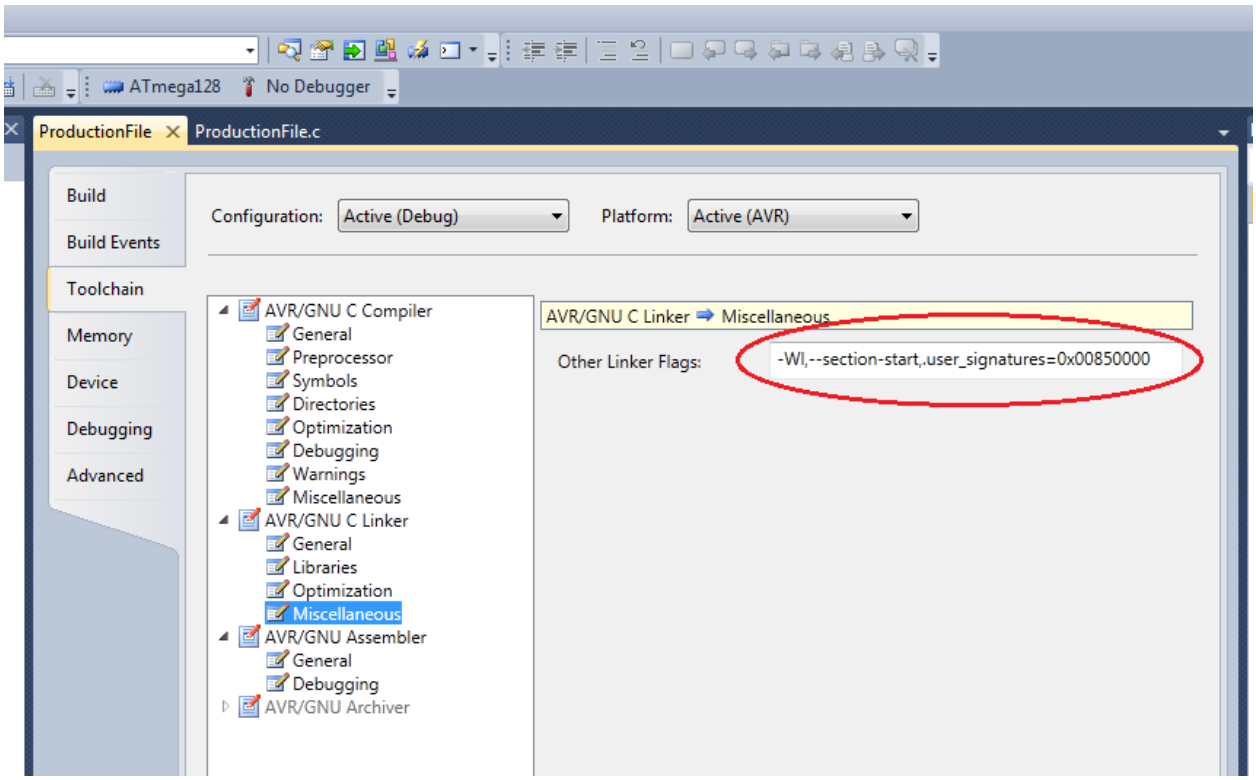
```
#include <avr/io.h>

// Example data for ATxmega128A1
const char eeprdata[] __attribute__ ((section (".eeprom"))) =
    "Hello EEPROM";
// The order of the fuse values is from low to high. 0xA2 is written to Fuse byte 0, 0x00 to
byte 1...
const uint8_t fusedata[] __attribute__ ((section (".fuse"))) =
    {0xA2, 0x00, 0xFF, 0xFF, 0xFF, 0xF5};
```

```
const uint8_t lockbits[] __attribute__ ((section (".lockbits"))) =
    {0xFC};
const char userdata[] __attribute__ ((section (".user_signatures"))) =
    "Hello User Signatures";

// Remember to set the following Toolchain project options,
// under AVR/GNU -> Miscellaneous:
//
//   -Wl,--section-start,.user_signatures=0x00850000

int main(void)
{
    while(1)
    {
        // TODO:: Please write your application code
    }
 }
```

Linker setup for User Signature section

The User Signatures section must have a specific Linker Setup, as shown below. This is necessary to get the correct address offset for the user signature section in the elf file. Other memory sections gets the correct address automatically.

Figure 3-19. Linker Setup for User Signature Section



Creating ELF Files for ATtiny4/5/9/10

This code shows how to add memory sections for ATtiny10.

```
#include <avr/io.h>

typedef struct _tagConfig
{
    unsigned char f1;
} Config;

typedef struct _tagLock
{
    unsigned char f1;
} Lock;
```

```
typedef struct _tagSig
{
    unsigned char f1;
    unsigned char f2;
    unsigned char f3;
} Signature;

Config __config __attribute__((section(".config"))) =
{
    f1 : 0xfb, // Set CKOUT
};

Lock __lock __attribute__((section(".lock"))) =
{
    f1 : 0xfc, // Further programming and verification disabled
};

Signature __sig __attribute__((section(".signature"))) =
{
    f1 : 0x03,
    f2 : 0x90,
    f3 : 0x1e,
};

int main(void)
{
    while(1)
    {
        // TODO:: Write your application code
    }
}
```

Creating ELF Files for UC3

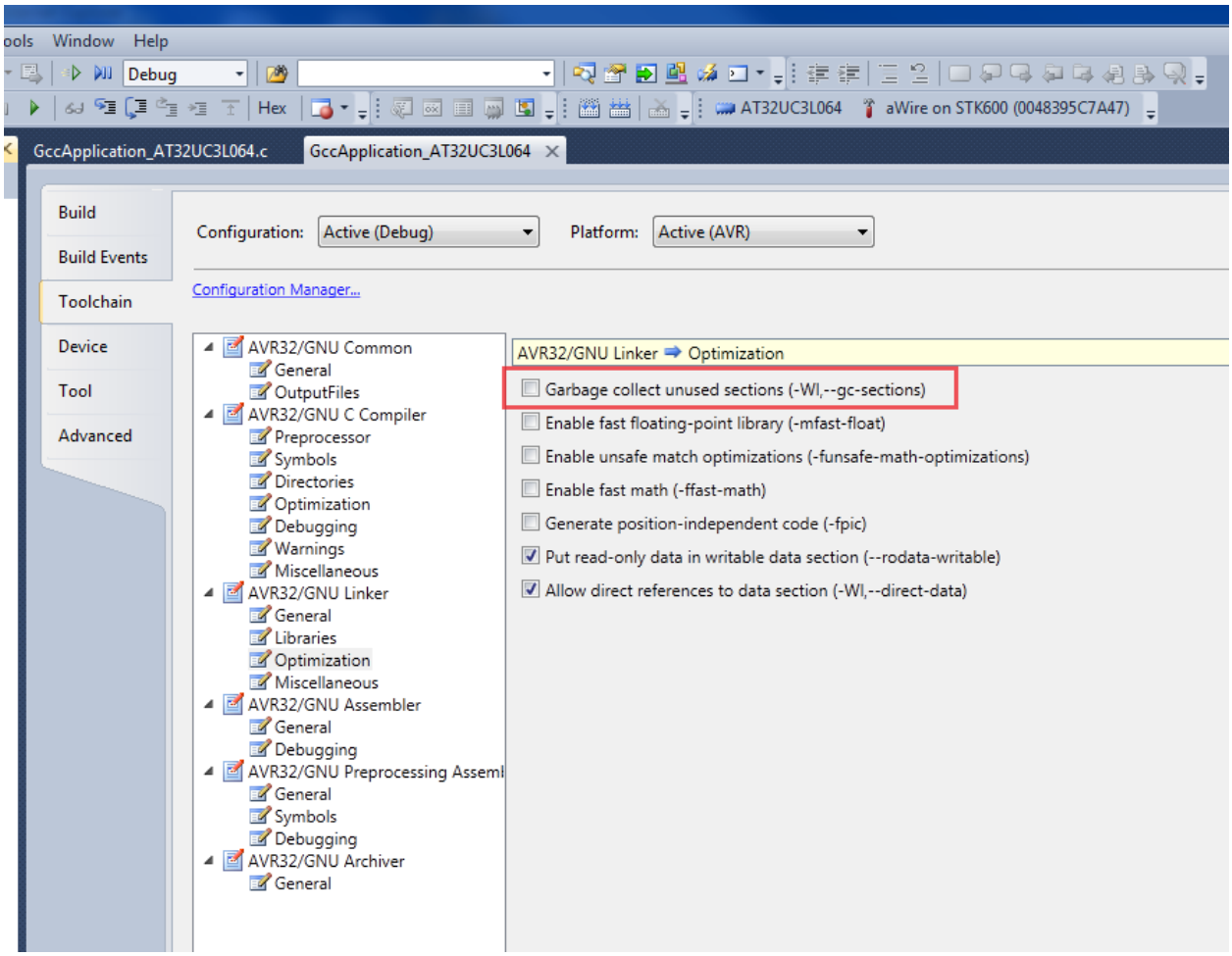
The example below shows how to add data for the user page in UC3 devices.

```
#include <avr/io.h>

const char userdata[] __attribute__((section(".userpage"))) = "Hello Page";

int main(void)
{
    while(1)
    {
        //TODO:: Write your application code
    }
}
```

Project Properties

If the memory sections are defined but not referenced in the application code, the "Garbage collect unused sections" option in Project Properties → Linker → Optimization must be unchecked. Otherwise the linker will not include the sections in the .elf file.

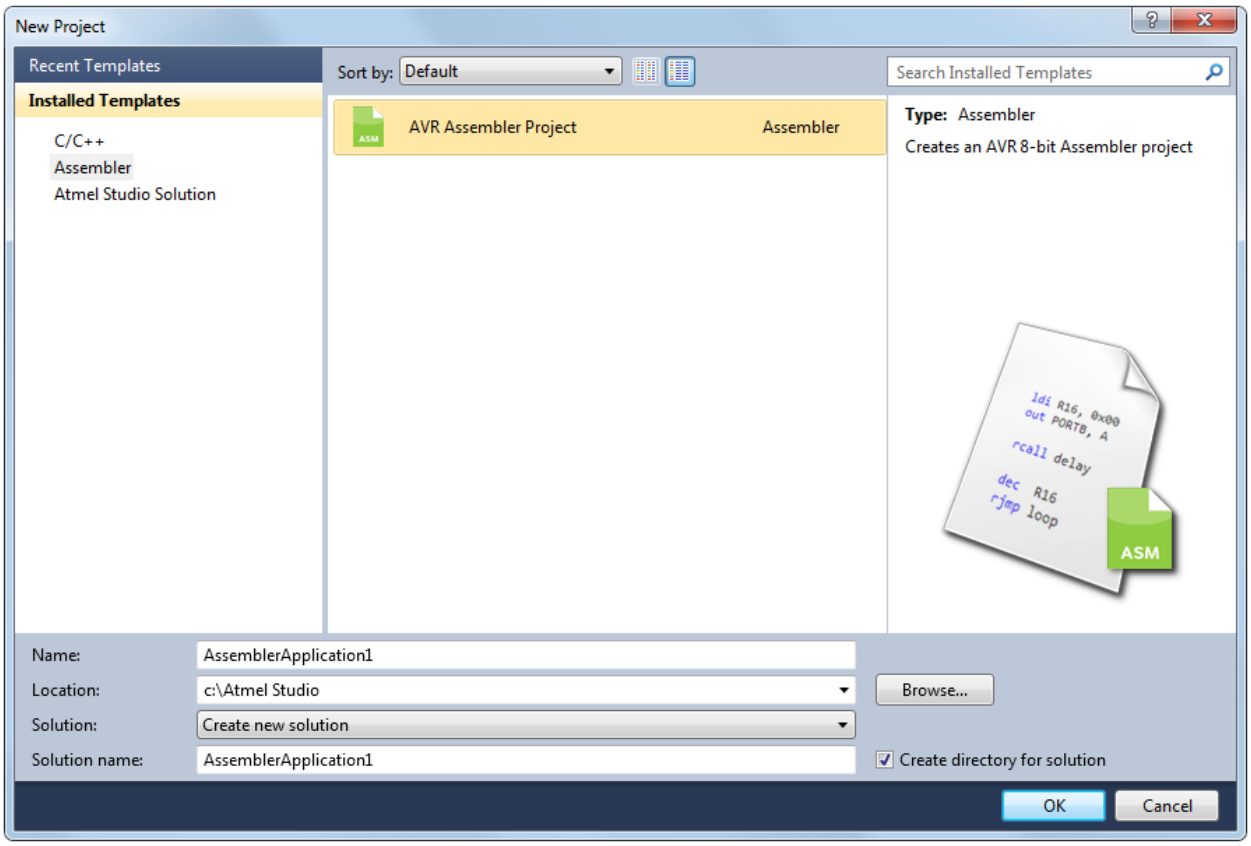
Figure 3-20. Project Properties



3.3. Assembler Projects

3.3.1. Create New Assembler Project

Figure 3-21. New Assembler Project



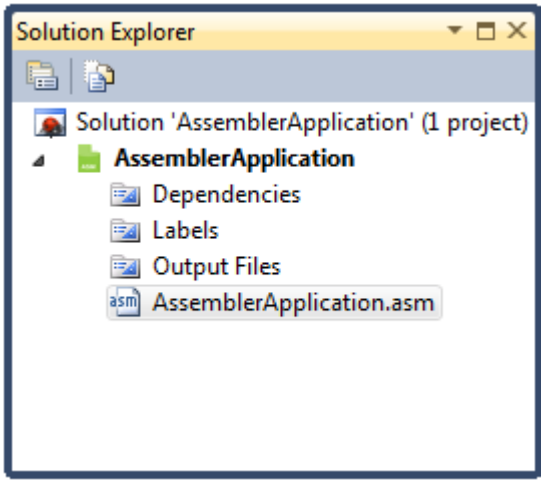
After pressing OK, you are asked to select your microcontroller.

You can try out the Assembler build and code debugging, using a simple LED-chaser code, given below. It should fit any AVR 8-bit microcontroller, simply change the port (in this case E) to your hardware.

```
start:
nop
ldi R16, 0xff
sts PORTE_DIR, r16

ldi r17, 0x80
output:
sts PORTE_OUT, r17
rol r17

ldi r16, 0x00
delay:
ldi r18, 0x00
delay1:
inc r18
brne delay1
inc r16
brne delay
break
rjmp output
```

When a new project is created or an old project is loaded, the project view will be displayed with all the project files. Files can be added, created, or removed from the project list using the context menu in the **Solution explorer** window.

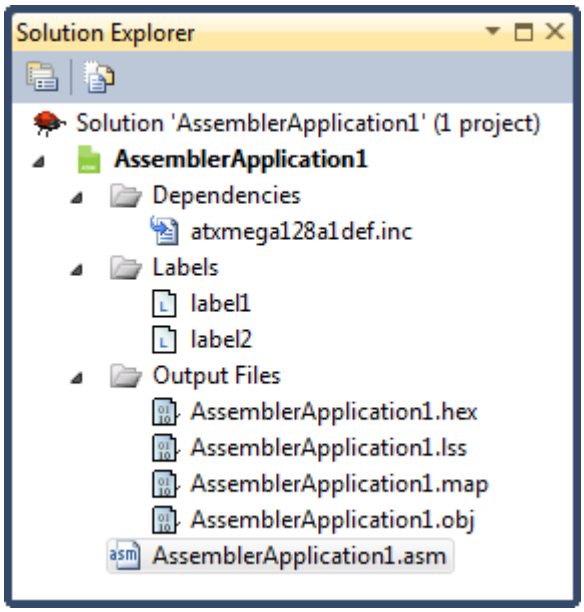
Figure 3-22. View of an Assembler Project



All the source files will be listed at the end of the list. Double click on any file to open it in the editor.

All custom include files will be listed directly under the project name item, unless you create a new folder in the project.

Figure 3-23. View of an Assembler Project after Build Completed



Dependencies: All include files are listed here. Double click on any file to open it in the editor.

Labels: All labels in your assembler program are listed here. Double click on any item to show its location in the source. A marker will point to the correct line.

Output Files: All output files will be displayed below this item.

Figure 3-24. File Context Menu

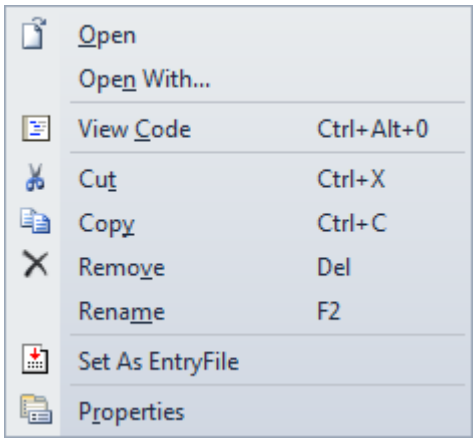


Table 3-6. File Context Menu

Menu text	Shortcut	Description
Open	Right click O	Open the selected file
Open With...	Right click n	Open selected file with another editor or tool
Cut	Ctrl X	Cut the file from current category
Copy	Ctrl C	Copy the file from current category
Remove	DEL	Remove the selected file from the project
Rename	F2	Rename the selected file
Set As EntryFile		Set the selected file as entry file
Properties	Alt ENTER	Current file properties
Menu text	Shortcut	Description

All the interface views are docked by default. You can switch between docked and undocked views by dragging windows around to a desirable location, or by dragging and dropping a window on a quick docking menu of the Visual Studio IDE. The quick docking menu will appear every time you start dragging an interface view or window.

3.3.1.1. Project Context Menu

Several build commands are available from the menu and the toolbars. There is also a context menu for the project:

Figure 3-25. Project Context Menu

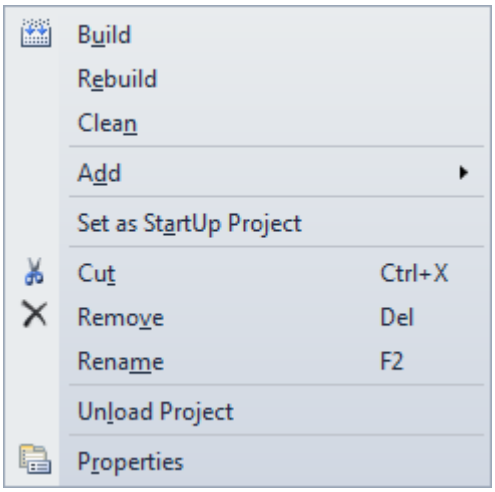


Table 3-7. Project Context Menu

Menu text	Shortcut	Description
Build	Right click+u	Build the selected project
Rebuild	Right click e	Will clean the project and build it
Clean	Right click n	Clean up and erase artifacts
Add	Ctrl Shift A / Shift Alt A (existing item)	Add new files or existing files to the project
Set as StartUp Project	Right click + a	Will set up to automatically open current project at start up
Cut	Ctrl X	Cut project to paste it as a sub-project to another solution
Remove	Del	Remove project or sub-project under cursor
Rename	F2	Rename current project
Unload Project	Right click l	Unload active project files from the IDE
Properties	Alt Enter	Project properties

3.3.2. Assembler Options

Open the options window on the menu **Project** → **"*Your_project_name* Properties"**. This menu item is only available when an assembler project is open. After opening the **Project properties** window, you will see six tabs, in order to configure assembler options click on the **Toolchain**.

Figure 3-26. Assembler Setup Dialog, Command Line Shown

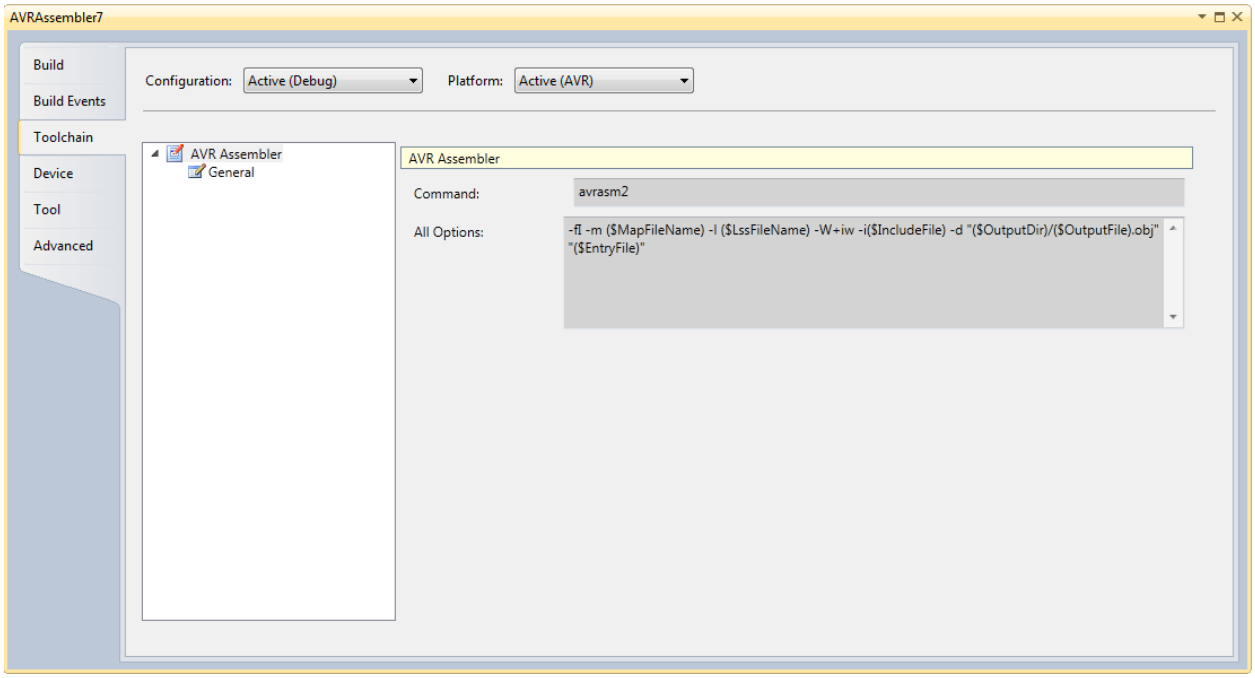
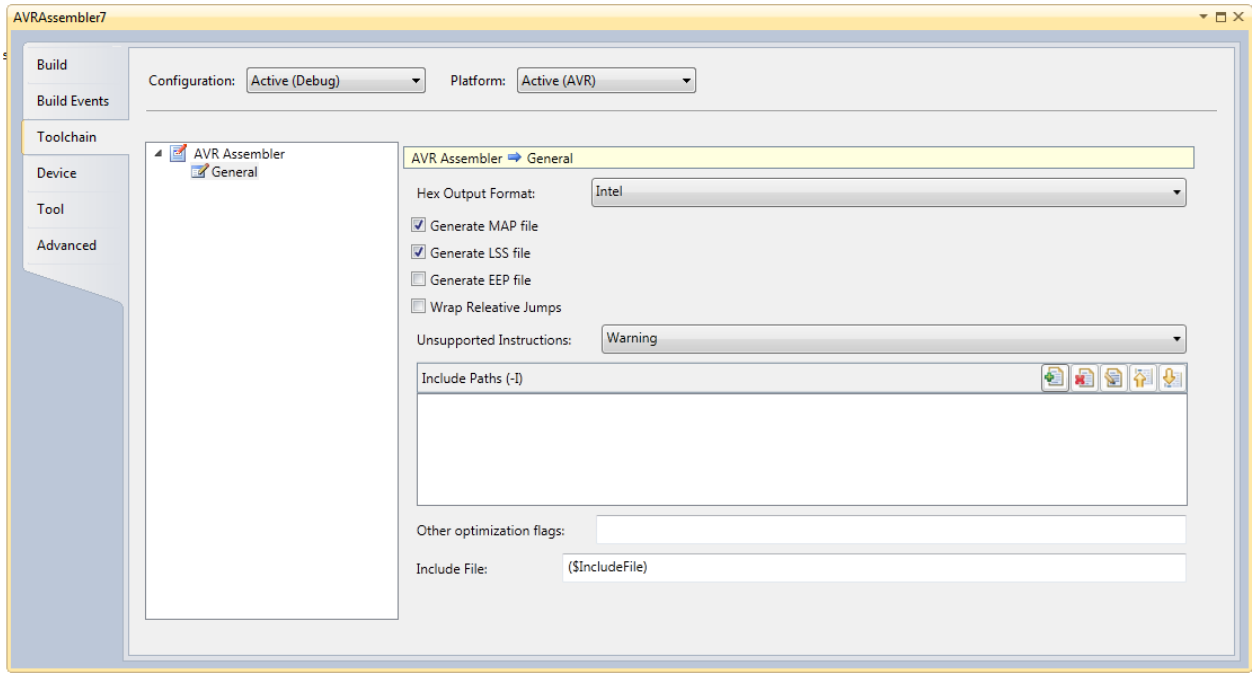


Figure 3-27. Assembler Setup Dialog, General Options Shown



3.3.2.1. Description of the Various Settings

Configuration menu allows to choose which stages of project maturity are going to be affected by the modifications to the project properties. By default Debug is the initial stage and initially active configuration. Following options are available:

1. Debug.
2. Release.
3. All configurations.

Platform menu shows compatible target platforms available for prototyping.

Hex Output format The following file formats can be selected as additional output format:

1. Intel Hex.
2. Generic Hex.
3. Motorola Hex (S-record).

Wrap relative jumps The AVR RJMP/RCALL instructions allow a 12-bit PC-relative offset, corresponding to ±2k words. For devices having 4k words (8kB) or less FLASH program memory, the Wrap option causes the assembler's offset calculation to wrap around over the addressable program memory range, enabling the entire program memory to be addressed using these instructions.

For devices with more than 4k words of program memory, using this option may cause unpredictable results and it should be turned OFF. If it is left ON, the assembler will produce a warning when wrap takes effect:

Attention:
Wrap rjmp/rcall illegal for device > 4k words - Turn off wrap option and use jmp/call.

This diagnostic is given as a warning and not an error to retain compatibility with earlier versions of the assembler, but should be treated as an error by the user. The JMP/CALL 2-word instructions take 22-bit absolute addresses and should be used instead.

Unsupported Instructions. By default, this option is set to give a warning when the assembler finds unsupported instructions for the actual device. Optionally, you can output an error.

Include Paths (-I). Additional include paths can be set here, when using third party modules or your own IP. Assembler's default include path:`\Atmel\AVR Tools\AvrAssembler2\Appnotes`.

Other optimization flags can be set to tailor optimization to your specific needs, see Assembler help for more information.

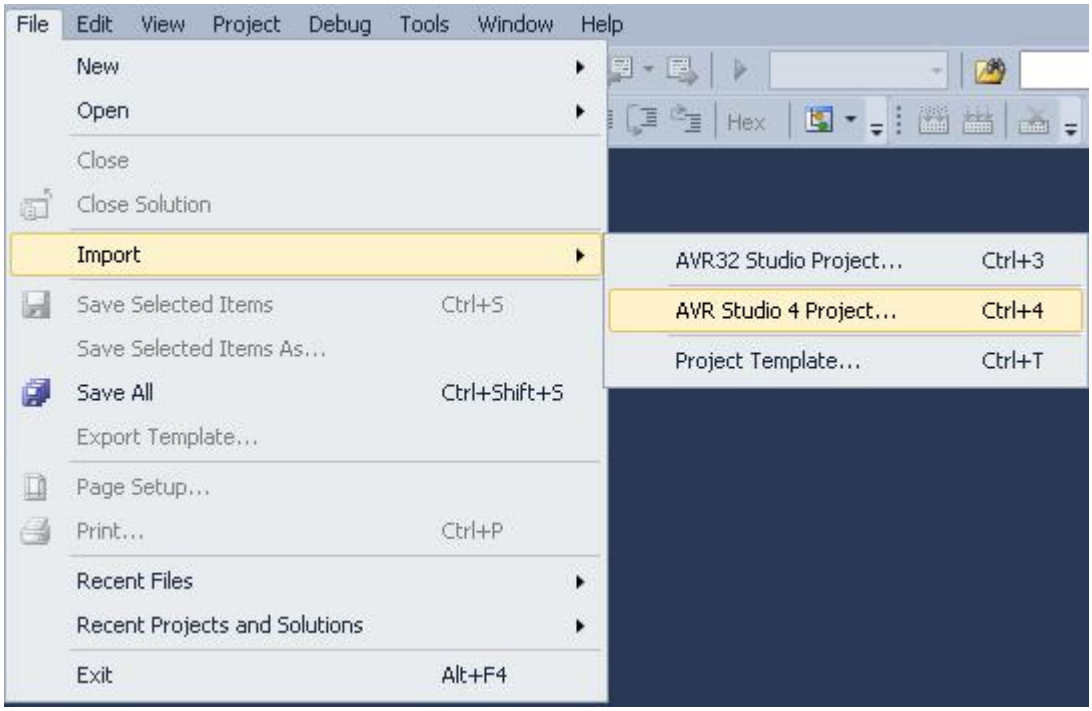
3.4. Import of Projects

3.4.1. Introduction

Atmel Studio allow import of projects from several pre-existing project sources. This section details how to import existing projects.

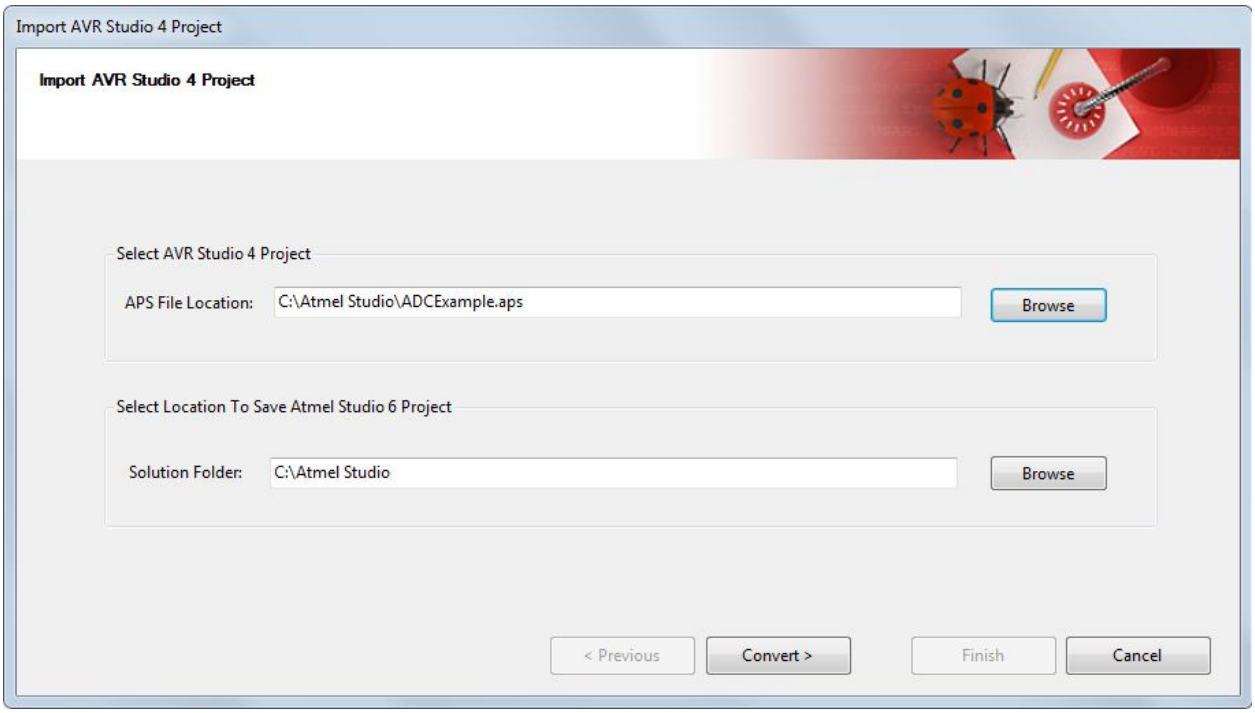
3.4.2. Import AVR Studio 4 Project

Click the menu **File** → **Import** → **AVR Studio 4 Project..** or **Ctrl+4**.



An **Import AVR Studio 4 Project** dialog will appear.

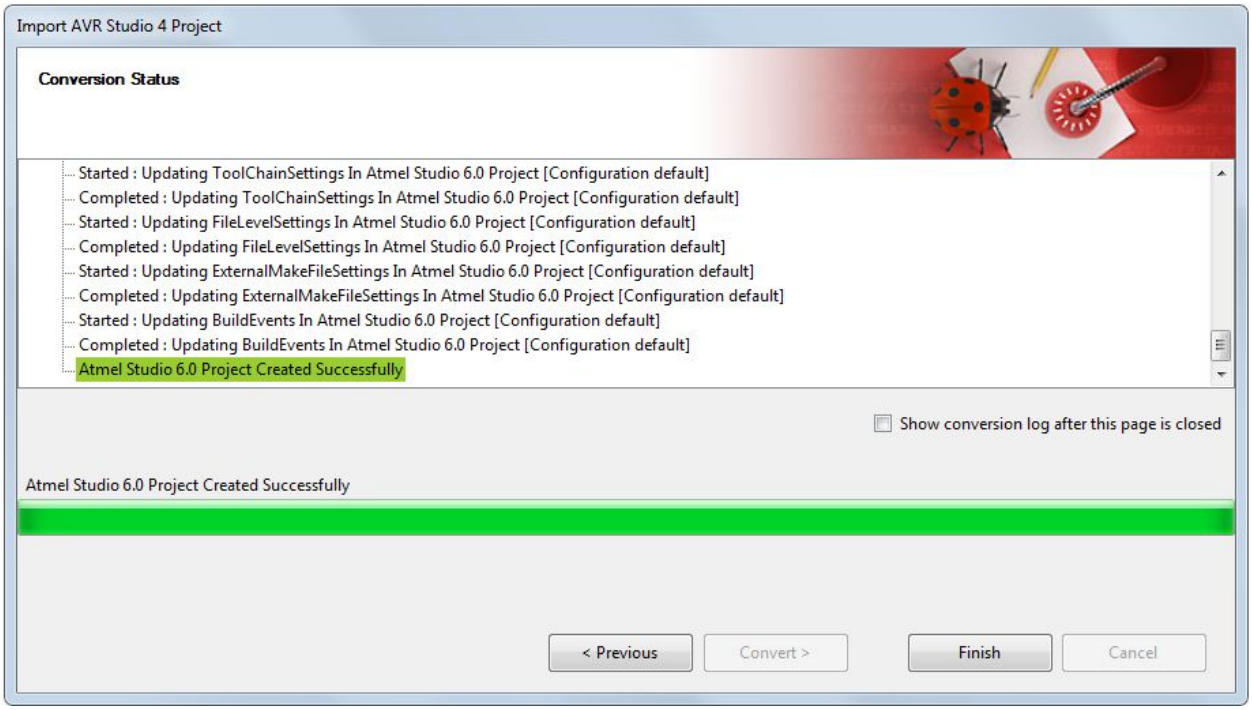
Type the name of your project or browse to the project location by clicking the Browse button of the APS File location Tab.

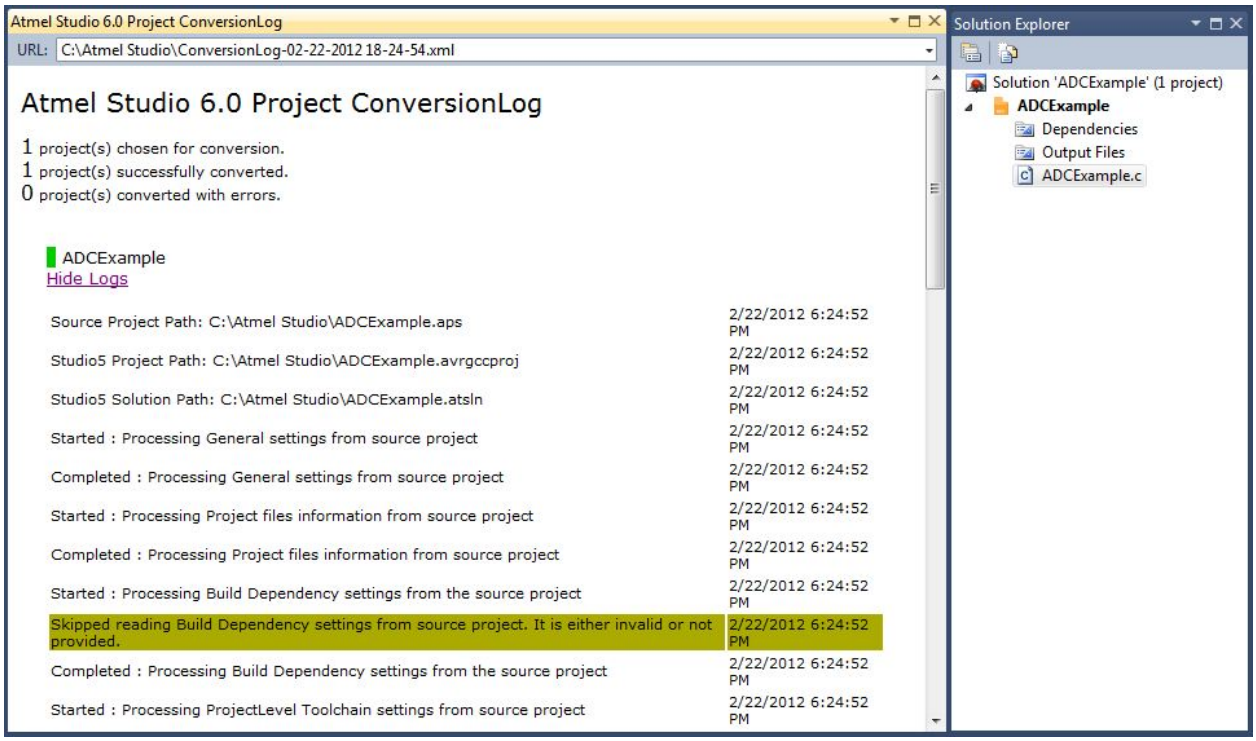


Atmel Studio will proceed with conversion also updates the progress, warnings, and errors. They will be shown in the Summary window.

Check **Show conversion log after this page is closed** to view the complete conversion log.

Click Finish to access your newly converted project.

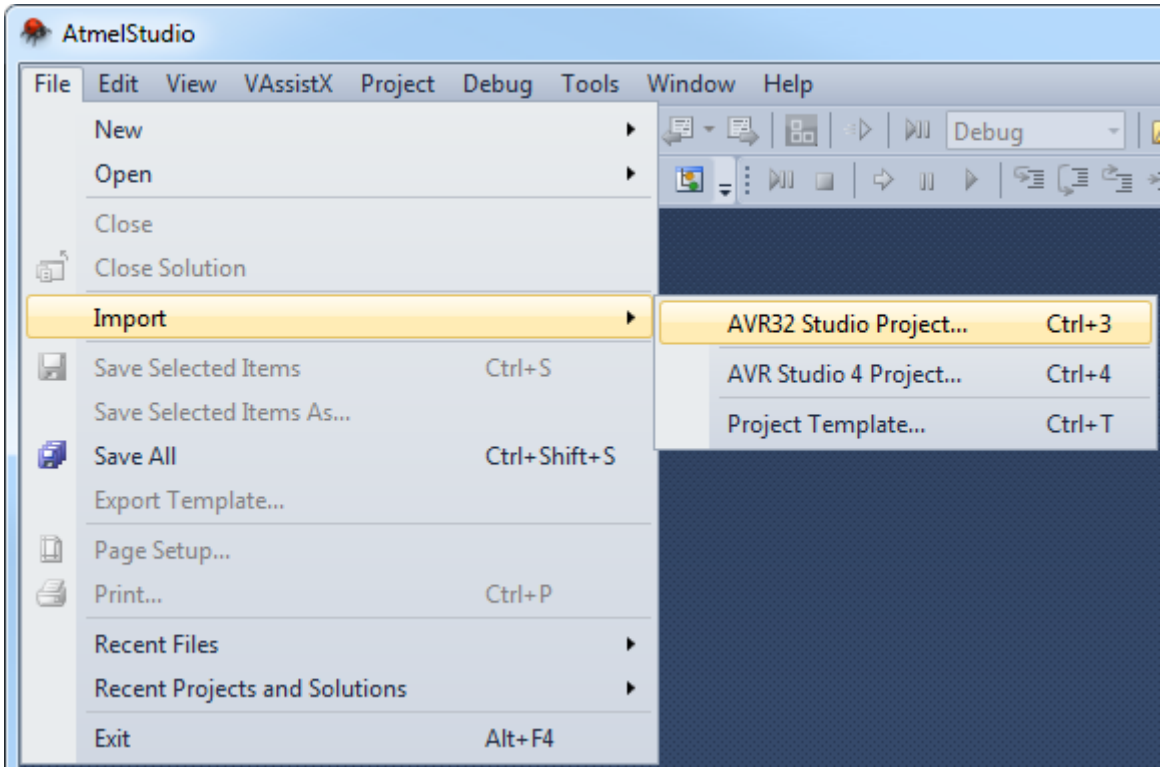




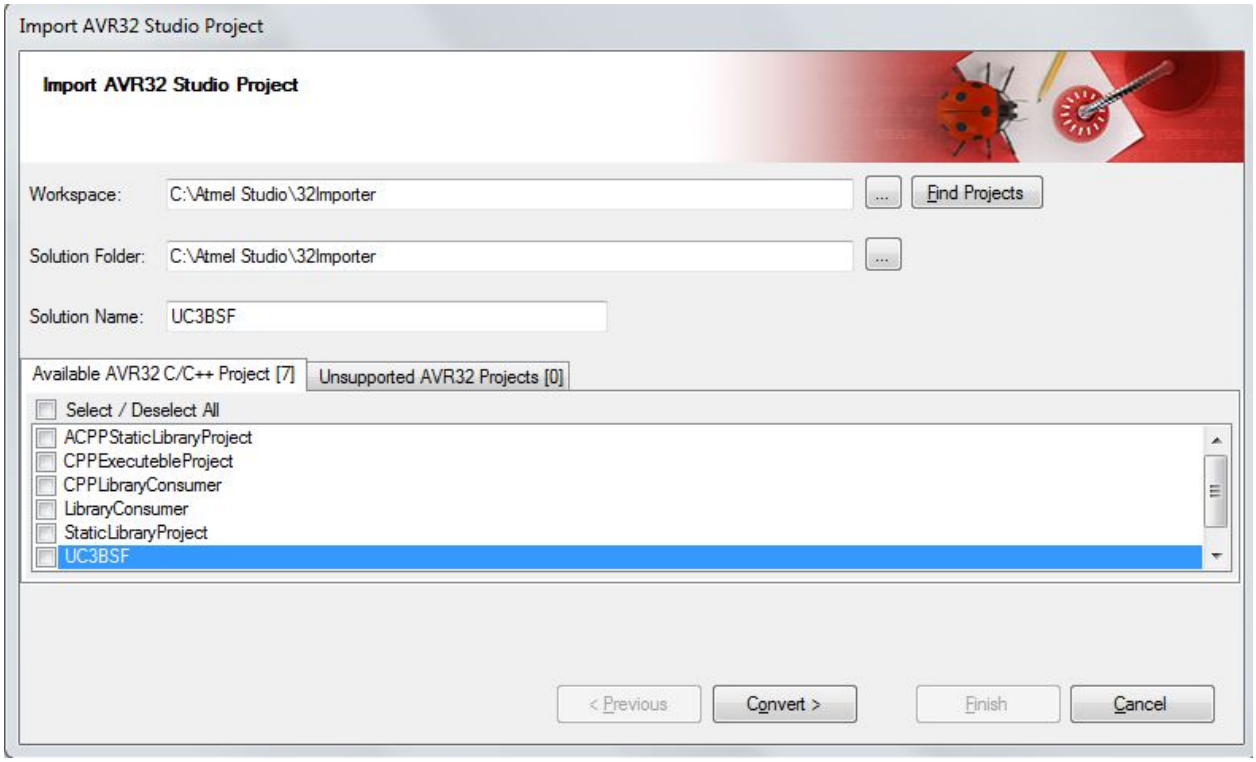
Note: Currently, conversion only adds a project file and solution file if the Solution Folder is the same as the APS File Location. No other files will be modified.

3.4.3. Import AVR 32 Studio Project

Click the menu **File** → **Import** → **AVR Studio 32 Project..** or **Ctrl+3**.

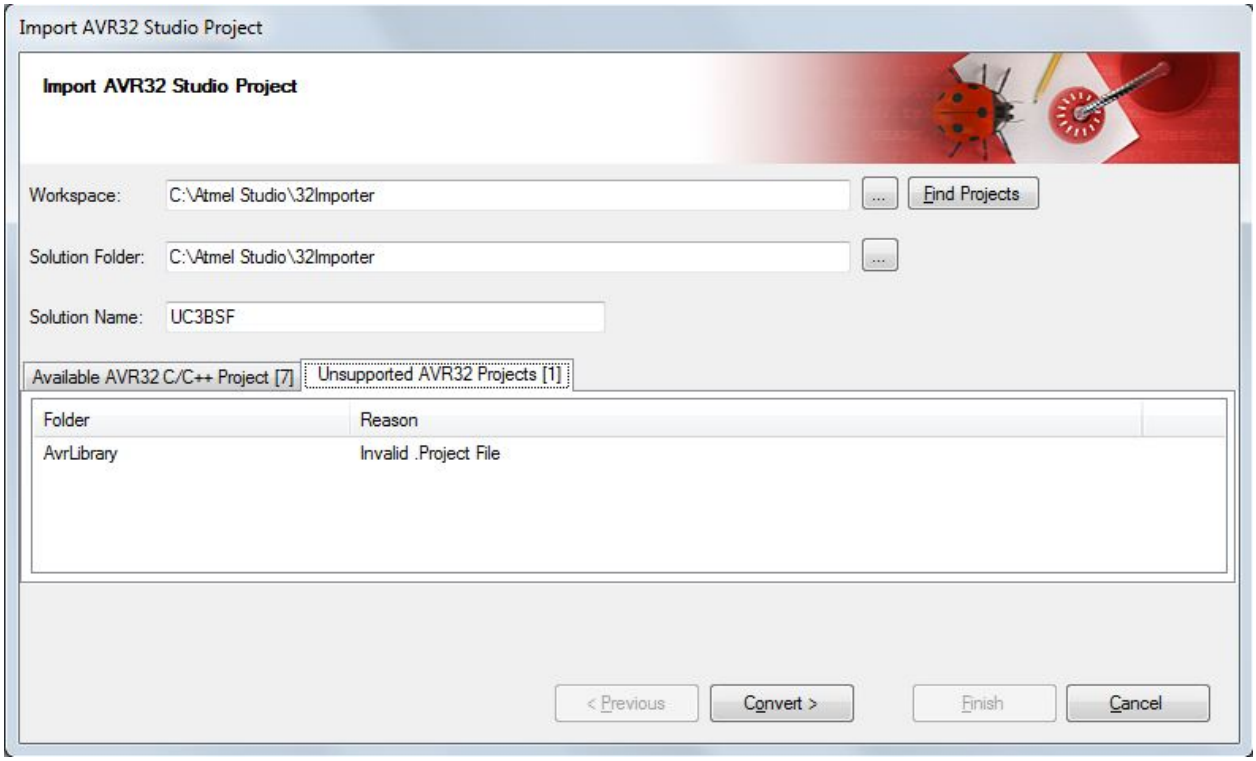


An "Import AVR Studio 32 Project" dialog will appear.



Type the name of your workspace or browse to the workspace location by clicking the **...** (Browse button) of the Workspace Tab. Click **Find Projects** to find all the project files and populate other folders available in the workspace.

The **Available AVR32 C/C++ Projects** tab will be populated with all **AVR32 C/C++ Projects** that can be imported and it will also display total number of available projects.

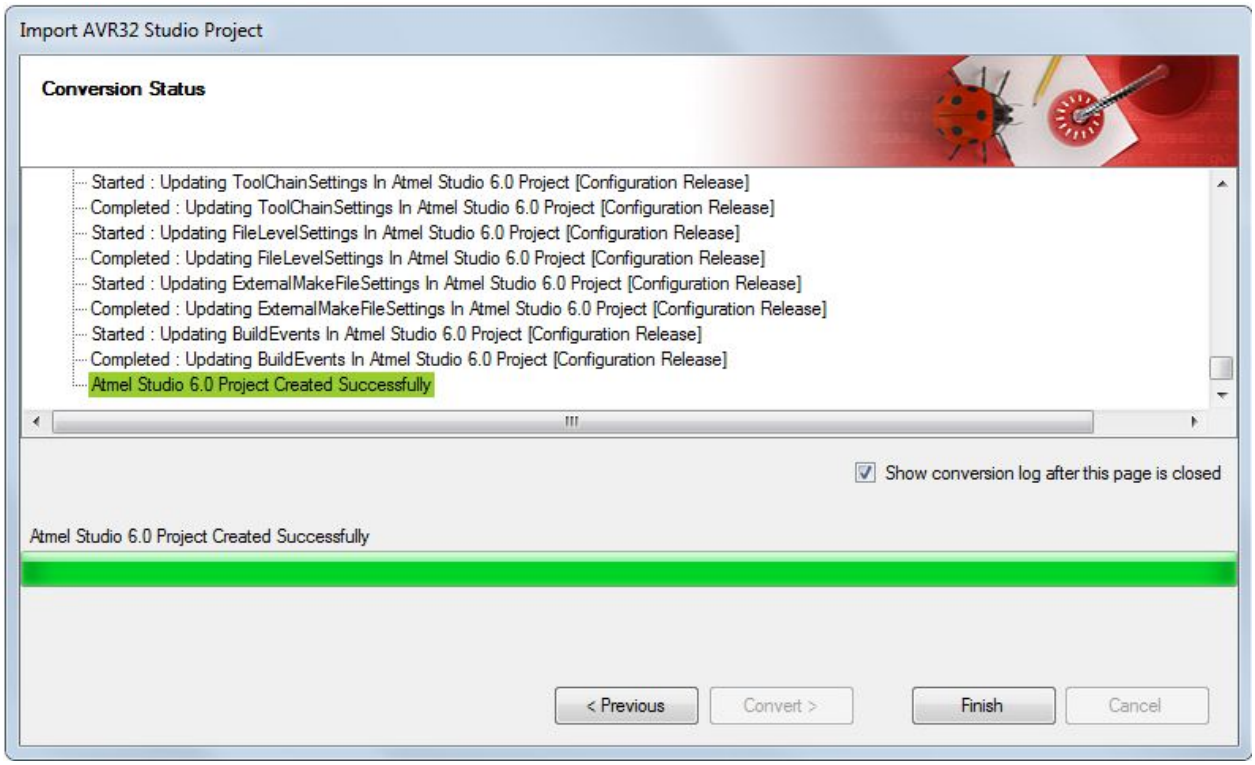


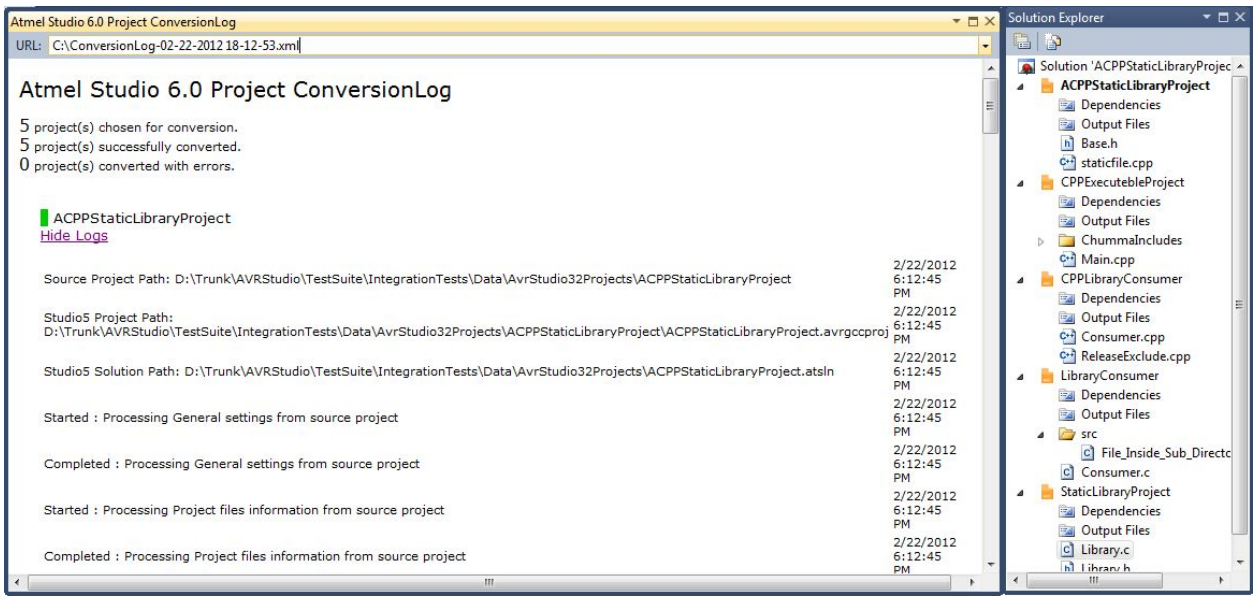
The **Invalid AVR32 Projects** tab will be populated with all **Unsupported AVR32 Projects** that can not be imported and it will also display total number of non convertible projects along with reason.

Atmel Studio will proceed with conversion also updates the progress, warnings, and errors. They will be shown in the Summary window.

Check "Show conversion log after this page is closed" to view the complete conversion log.

Click Finish to access your newly converted project.





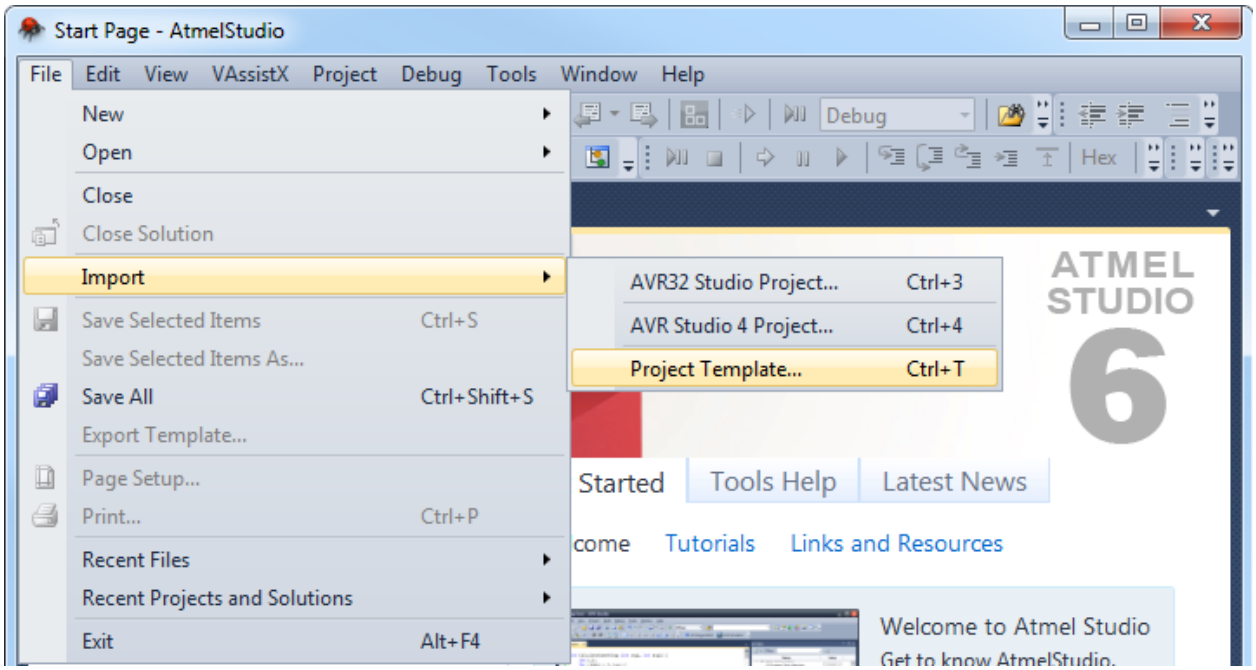
Note:

- The current version of AVR32 Importer supports AVR32 C/C++ Projects
- AP7 device family is currently not supported by Atmel Studio
- Currently, conversion only adds project files and solution file if the Solution Folder is the same as the Workspace folder. No other files will be modified.
- Pre/Post builds settings are not imported
- Automatically generate listing (*.lss) files setting is not imported

3.4.4. Import Project Template

A number of predefined project can be imported to Atmel Studio by **File → Import → Project Template..**or **Ctrl+T**

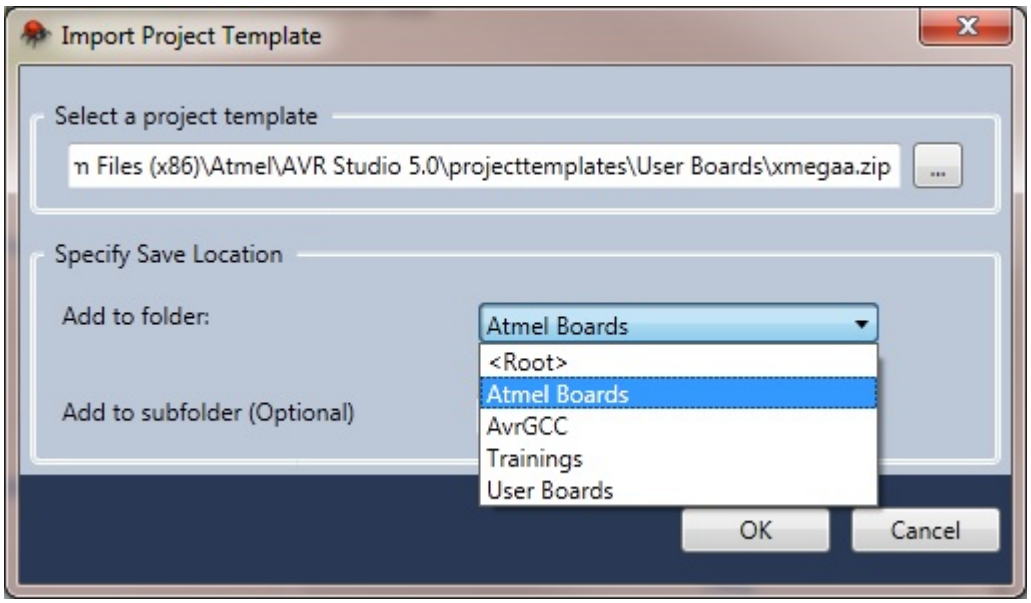
These templates provide a starting point to begin creating new projects or expanding current projects. Project templates provide the basic files needed for a particular project type, include standard assembly references, and set default project properties and compiler options.



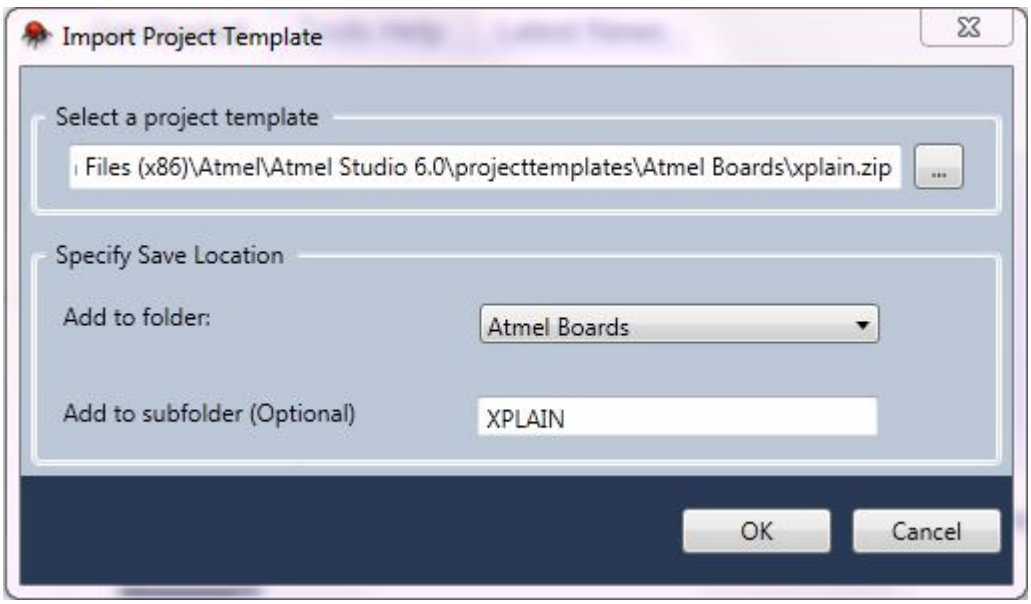
In the " Import Project Template " window specify the following:

- Specify the location of your project template
- Specify the save location. The combo box will show installed templates that are available in the **New Project → Installed Templates**.

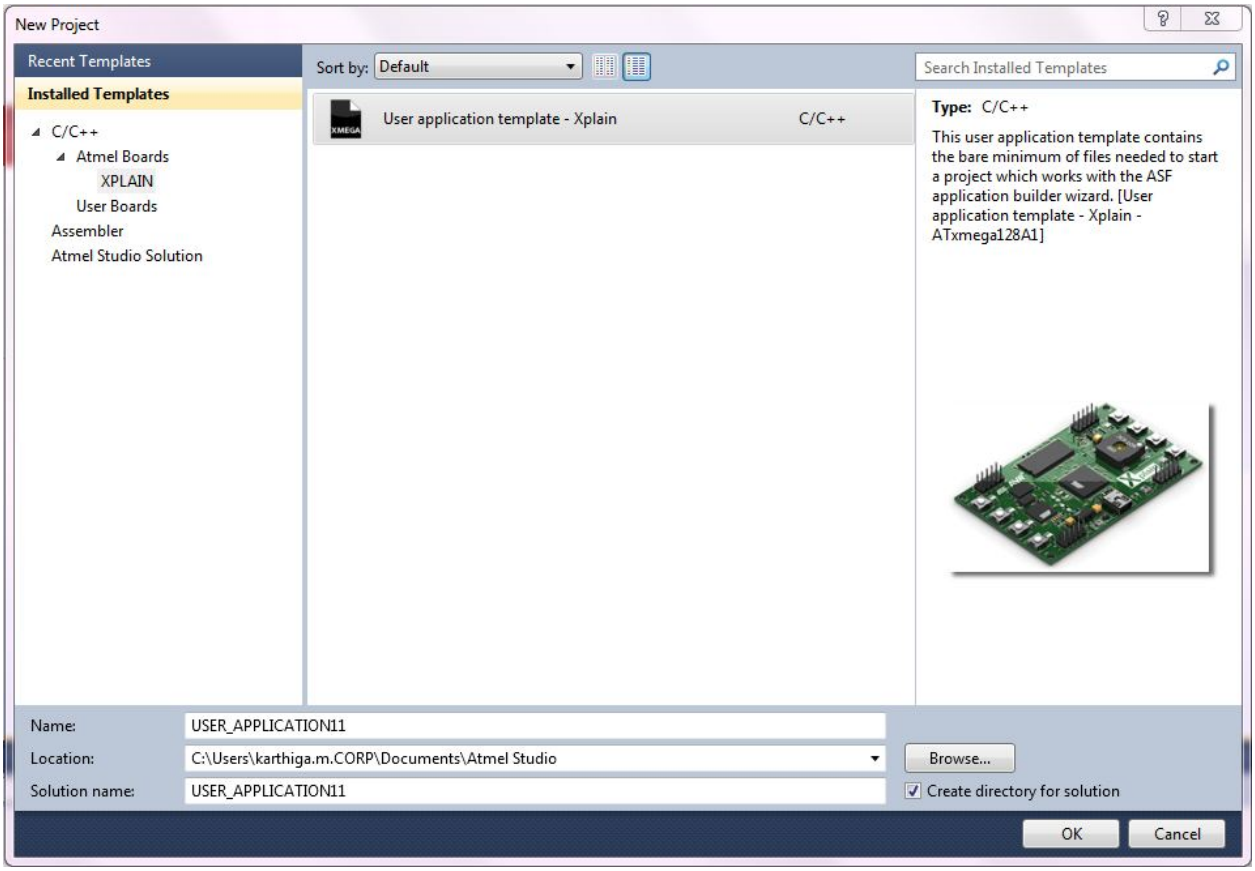
  Select any template under which you would like to add your template. You can also add your template at the root by selecting <root> in "Add to folder".



- You can create a separate folder by specifying the name of the folder under the specified "Add to Folder (Optional)", where you want to add your project template.



The resulting project template will be added to the existing installed templates and can be accessed from **File** → **New** → **Project ..** or **Ctrl+Shift+N**.



Note: "Import Project Template Importer" will work with template created for the same version.

3.5. Debug Object File in Atmel Studio

3.5.1. Introduction

Debug session requires you to load an object file which is supported by Atmel Studio. The debug file contains symbolic information for debugging.

3.5.2. Atmel Studio Supported Debug Formats

Table 3-8. Object File Formats Supported by Atmel Studio

Object file format	Extension	Description
UBROF	.d90	UBROF is an IAR proprietary format. The debug output file contains a complete set of debug information and symbols to support all type of watches. UBROF8 and earlier versions are supported. This is the default output format of IAR EW 2.29 and earlier versions. See below how to force IAR EW 3.10 and later versions to generate UBROF8.
ELF/DWARF	.elf	ELF/DWARF debug information is an open standard. The debug format supports a complete set of debug information and symbols to support all type of watches. The version of the format read by Atmel Studio is DWARF2. AVR-GCC versions configured for DWARF2 output can generate this format.
AVRCOFF	.cof	COFF is an open standard intended for 3rd party vendors creating extensions or tools supported by the Atmel Studio.
AVR Assembler format	.obj	The AVR assembler output file format contains source file info for source stepping. It is an Atmel internal format only. The `.map` file are automatically parsed to get some watch information.

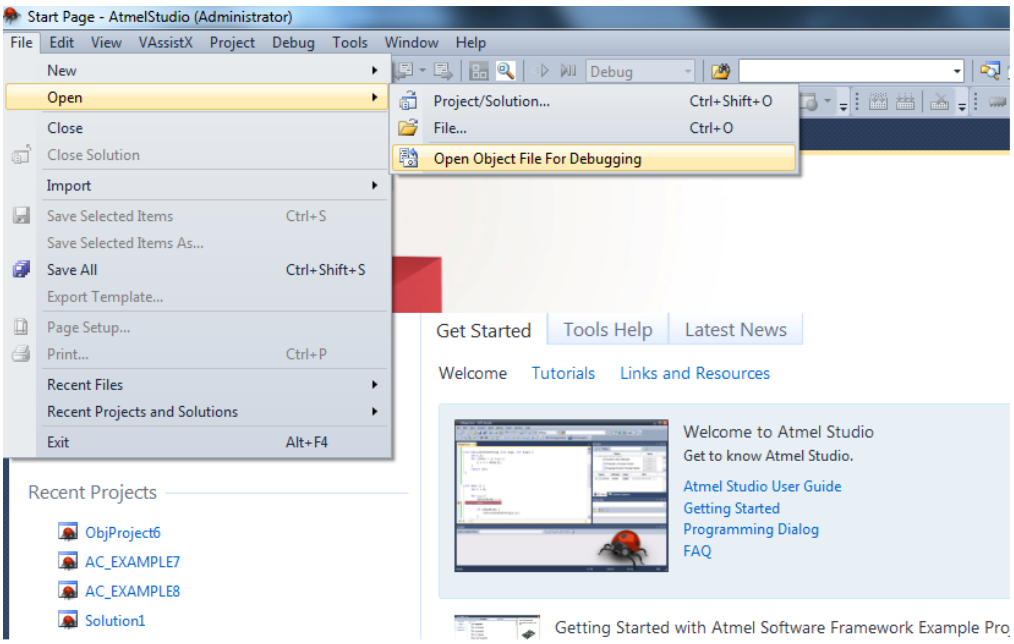
Before debugging, make sure you have set up your compiler/assembler to generate a debug file like one of the formats above. 3rd party compiler vendors should output the ELF/DWARF object file format.

3.5.3. Opening Object Debug File in Atmel Studio

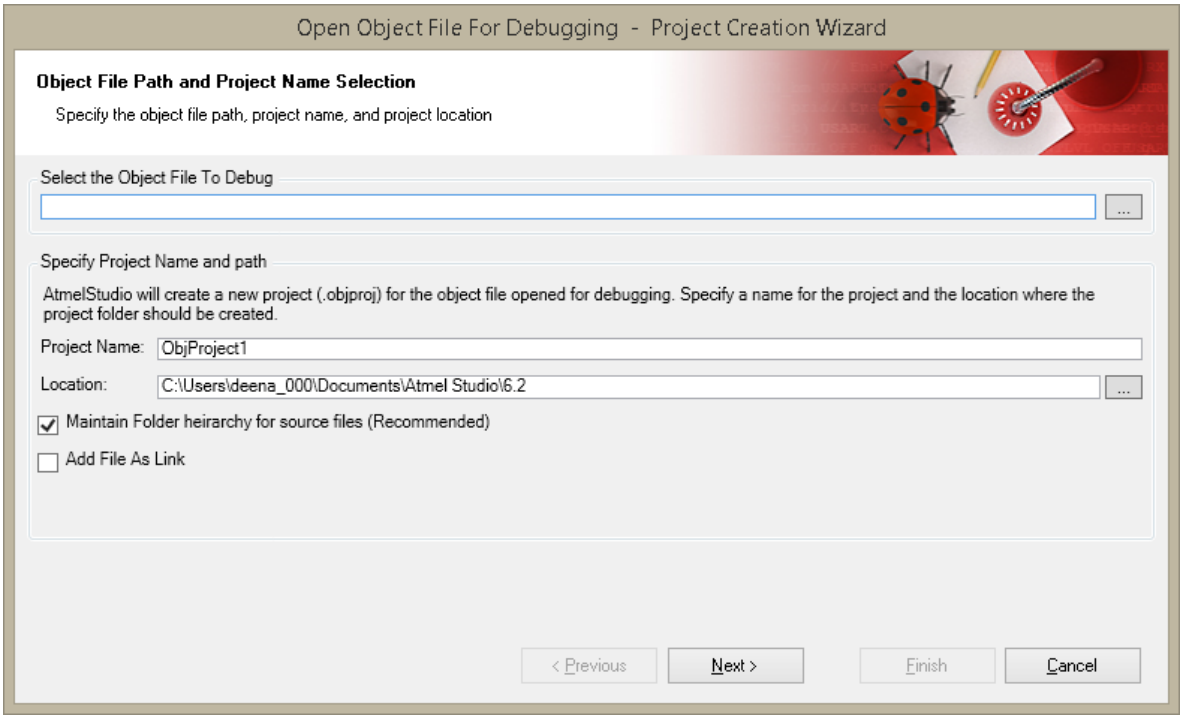
Steps to create an Object Project

- On the **File** menu, click **Open**, and then click **Open Object File For Debugging**.

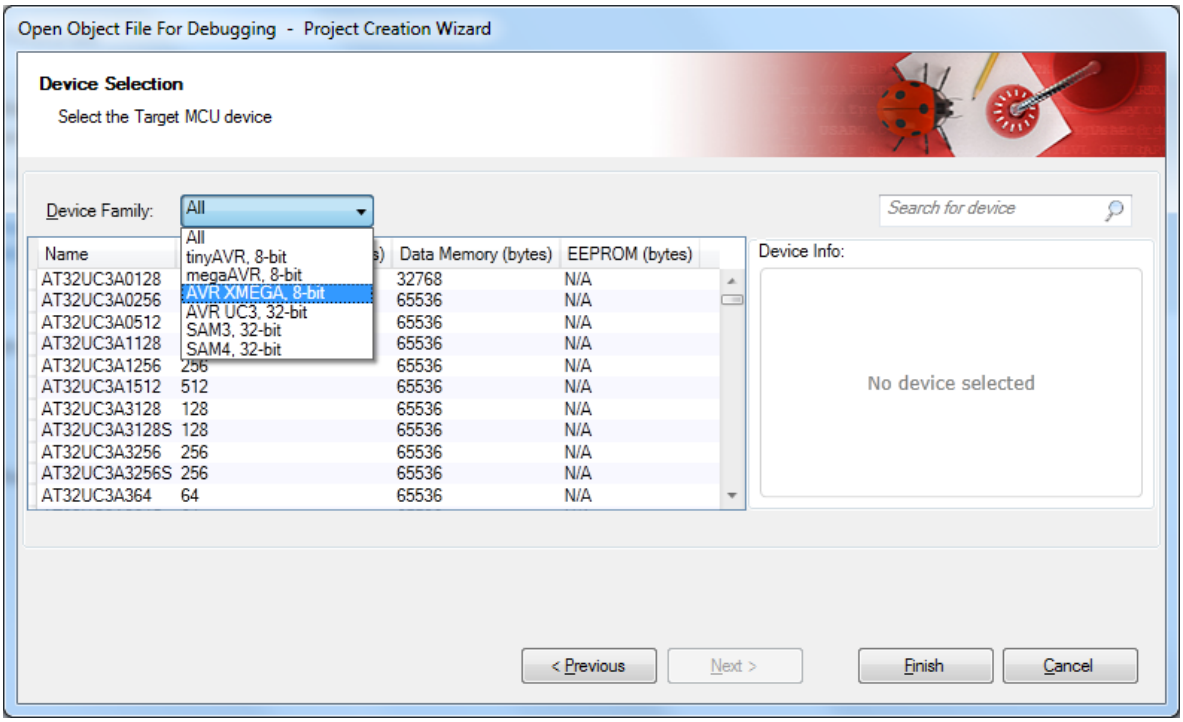
  Open Object File For Debugging wizard will appear.



- In the **Select the object file to debug**, select your object file to debug. The object file must be supported by Atmel Studio.
- In the **Project Name**, type a name for the project. Atmel Studio will suggest a name, which you can override if wanted.
- In the **Location**, select a save location. Atmel Studio will suggest a name, which you can override if wanted.
- **Maintain Folder Hierarchy for Source Files** option is selected by default which would create a similar folder structure in the Solution Explorer as that of the source project i.e. the project used to create the object file. Otherwise, all the files are added to the root folder of the project file i.e. the user would not see any folder in the Solution explorer.
- **Add File As Link option** is selected by default in which the object project shall refer the files from its original location without a local copy into the project directory. If the option is not selected, Atmel Studio would copy the files into the object project directory.

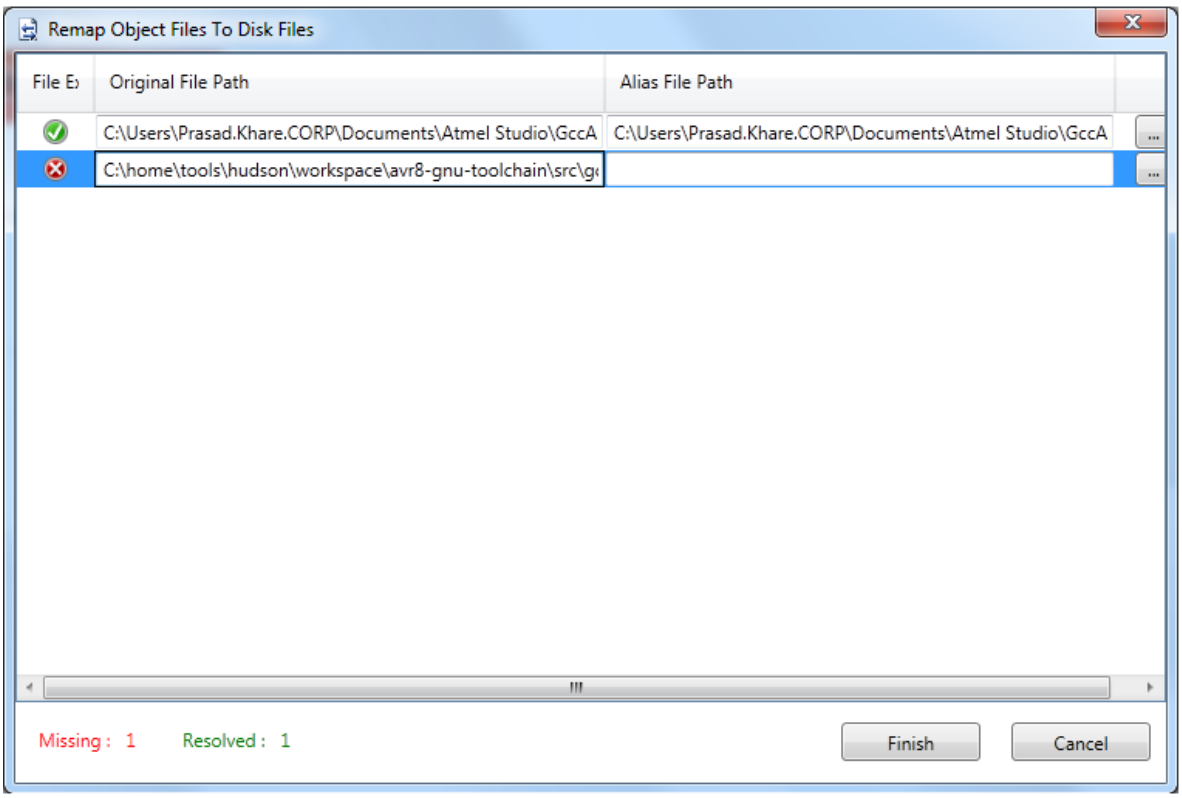


- Click **Next**. The Device Selection dialog will appear.
  - Choose the appropriate target device. The target device should be the same, which was originally chosen to create the object file.

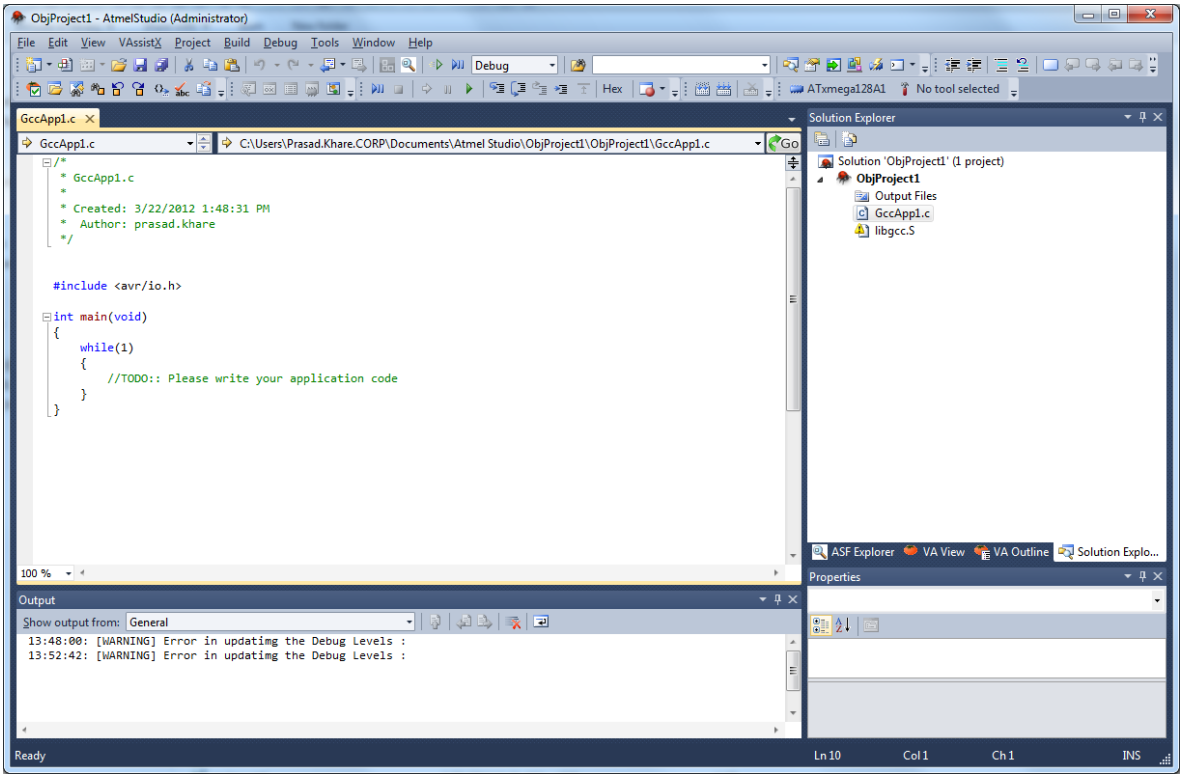


- Click **Finish**. The object project files re-mapper appears. In this particular dialog you need to remap your original files of the project using, which the elf projects was created. If files are present at their original place, it will show remapped already in dialog. If files are missing, you will have to remap it manually. Check the screen shot below.

  If the user resolves the parent folder for any original file, all other files in subsequent directory will be remapped recursively. So, it is useful for the user to remap the number of files by just remapping only one.



- Now the Object Project is Created. The files that are not remapped properly are shown in solution explorer like "libgcc.S", with warning sign . Press F5 to debug this Project.


ObjProject1 - AtmelStudio (Administrator)
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Hex
ATxmega128A1
No tool selected
GccApp1.c
GccApp1.c
C:\Users\Prasad.Khare.CORP\Documents\Atmel Studio\ObjProject1\ObjProject1\GccApp1.c
Go
/*
* GccApp1.c
*
* Created: 3/22/2012 1:48:31 PM
* Author: prasad.khare
*/
#include <avr/io.h>
int main(void)
{
while(1)
{
//TODO:: Please write your application code
}
}
100 %
Solution Explorer
Solution 'ObjProject1' (1 project)
ObjProject1
Output Files
GccApp1.c
libgcc.S
ASF Explorer
VA View
VA Outline
Solution Explo...
Properties
Output
Show output from:
General
13:48:00: [WARNING] Error in updating the Debug Levels :
13:52:42: [WARNING] Error in updating the Debug Levels :
Ready
Ln 10
Col 1
Ch 1
INS

4. Debugging

4.1. Introduction

Atmel Studio can be targeted towards the built-in Simulator, or a variety of tools (see Available Tools View), for example AVR ONE!, JTAGICE mkII or JTAGICE3 (bought separately).

4.1.1. Debug Platform Independent Debug Environment

Independent of which debug platform is running, the Atmel Studio environment will appear identical. When switching between debug platforms, all environment options are kept for the new platform. Some platforms have unique features, and new functionality/windows will appear.

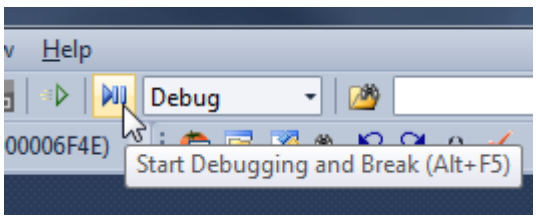
4.1.2. Differences Between Platforms

Although all debug platforms appear identical in the debug environment there will be small differences between them. A real-time emulator will be significantly faster than the simulator for large projects. An emulator will also allow debugging while the system is connected to the actual hardware environment, while the simulator only allow predefined stimulus to be applied. In the simulator, all registers are always potentially available for display, which might not be the case with an emulator.

4.2. Starting a Debug Session

To start a debug session and halt, press Alt+F5 or choose **Debug → Start Debugging and Break** from the menu, alternatively, press the toolbar button as illustrated below:

Figure 4-1. Starting a Debug Session

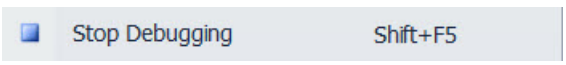


To start a debug session and keep executing, press F5 or press the toolbar button with the continue symbol, or choose **Debug → Continue** from the menu as illustrated below:

Figure 4-2. Starting a Debug Session

4.3. Ending a Debug Session

To end the debug session use the **Stop Debugging** button or keyboard shortcut Shift F5 .



4.4. Attaching to a Target

To attach a target, use the **Attach to Target** option in the **Debug** menu, or the attach icon in the debug toolbar. This causes Atmel Studio to launch a debug session on the selected target without uploading a new application or causing a reset. Once the debug session is established, the core of the target is halted and the current execution position of the target is mapped to the code in the project. This means that the state of the target is kept and is inspect-able with normal debug techniques, and the program halts in the current position. Full run control and symbolic debugging should be available after a successful attach.

Note:

- The code in the project is mapped to the content of the running target, without any possibility to verify the correctness of this mapping. This means that if the project contains code that is not on the target, then the state and run control might not reflect the truth, as variables and functions might have different code locations on the target than in the project.
- The ability to activate a debug session without resetting a target is architecture dependent. Not all architectures supports this feature.

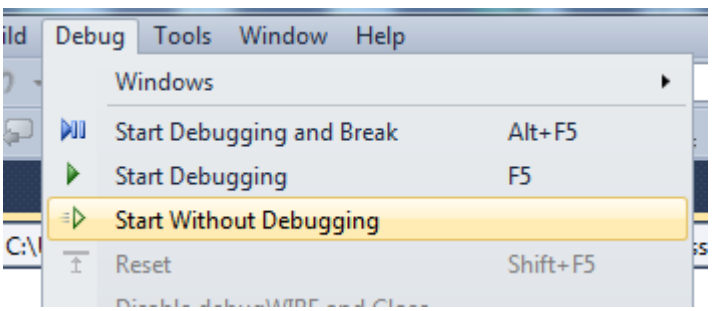
Attention:
Physically connecting a debug probe to a target might cause the target to reset, as most debug probes needs an electrical connection to the reset line of the device. Normal electrical precautions needs to be taken to avoid this.

4.5. Start without Debugging

4.5.1. One Click Programming - Program and Run

The **Start without Debugging** command is a one-click alternative to the programming dialog. Execute it by selecting **Debug → Start without Debugging** from the menu, or press the button on the toolbar.

Figure 4-3. Start without Debugging



This will build the solution (if any changes are made) and program the target device without starting a debug session.

Start without Debugging uses the tool and interface settings specified in the project options. This is different from what takes place when using the stand-alone Programming Dialog, which is not related to the project at all.

Note: Programmers and starter kits can also be used with the *Start without Debugging* command, not only debuggers.

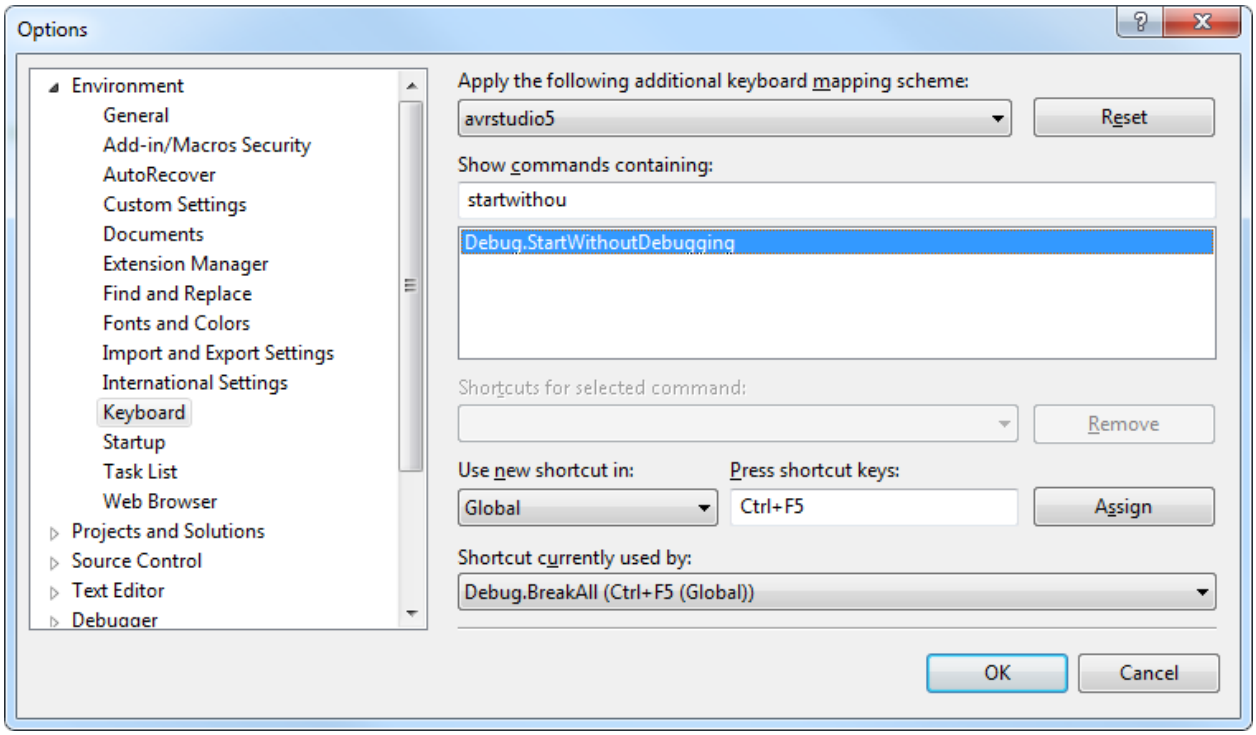
The **Start without Debugging** command will also program the EEPROM, fuses, lockbits, and user signature (XMEGA only) segments if they are present in the object file. The GCC compiler can generate ELF object format files with such segments. See Creating ELF Files with Other Memory Types for more information.

Note: The user signature is not erased by **Start without Debugging**. The programmed user signature from the ELF file will be AND-ed with the content in the device. If you want to replace the signatures with what is in the file, you must perform a **user signature erase** manually.

4.5.2. Keyboard Shortcut

By default, there is no keyboard shortcut to this function, but you might want to add one if you use it a lot. To add one, simply click the **Tools** → **Options** menu button and go to **Environment** → **Keyboard**. Start typing **startwithouttdebugging** in the **Show commands containing** input field, and select **Debug.StartWithoutDebugging** in the list. Then select the **Press shortcut keys** input field, and press the desired key combination. You can for example press **Ctrl+F5**. Note that this shortcut is already assigned to the **BreakAll** command. If you choose to override it, press the **Assign** button to assign the new keyboard shortcut. Or, you can select an unused key combination.

Figure 4-4. Add Keyboard Shortcut



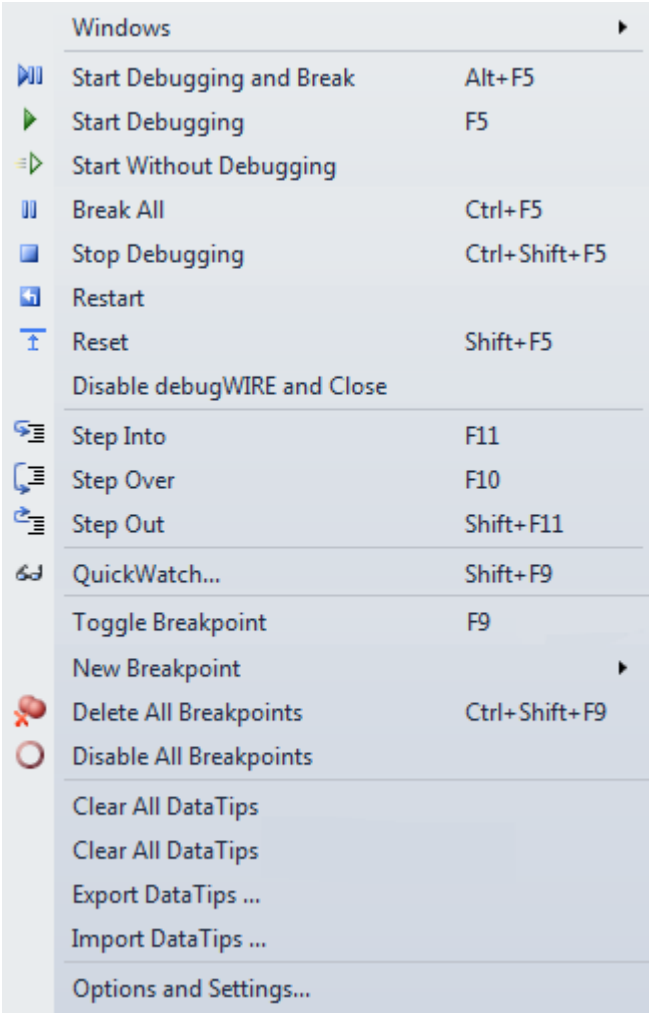
4.6. Debug Control

Several commands are available for controlling the debugger. They are available from both the **Debug** menu and several toolbars. The Atmel Studio Integrated Development Environment (IDE) has two major operating modes; *design mode* and *debug mode*. Design mode is when you are editing the source code project, while debug mode is when you debug your project. The IDE adapts to modes, and menus an toolbars changes.

Note: Some debug commands are available in design mode, some in debug mode.

- In design mode, the available debug commands are those that will start the debug session, e.g. **Start Debugging and Break**, **Start Debugging**, **Start without Debugging**.

- In debug mode, you will find commands like **Break All**, **Step Out**, and **Reset**.



Start Debugging and Break	Starts the debugger, and breaks the execution on the first statement of the program.
Start Debugging	Starts the debugger, and runs the program. In debug mode and stopped, it resumes execution.
Start Without Debugging	Programs the project without starting debugging. For details, see Start without Debugging.
Break All	Halts the debugger.
Stop Debugging	Stops and terminates the debug session, and returns to design mode.
Restart	Restarts the debugger and reloads the program.
Reset	Resets the program to the first statement.

Disable debugWire and Close	Available when debugging a device using the debugWire interface. The command disables the debugWire interface (enabling the use of the ISP interface) and terminates the debug session.
Step Into	Executes one instruction. When in disassembly level, one assembly level instruction is executed, otherwise one source level instruction is executed.
Step Over	Similar to **Step Into**, Step Over executes one instruction. However, if the instruction contains a function call/subroutine call, the function/subroutine is executed as well. If a user breakpoint is encountered during Step Over, execution is halted.
Step Out	Continue execution until the current function has completed. If a user breakpoint is encountered during Step Over, execution is halted. If a Step Out command is issued when the program is on the top level, the program will continue executing until it reaches a breakpoint or it is stopped by the user.
Quick Watch	Adds a Quick Watch for the variable or expression under the cursor. For details, see QuickWatch and Watches.
Toggle Breakpoint	Toggle the breakpoint status for the instruction where the cursor is placed. Note that this function is only available when the source window or disassembly window is the active view.
New Breakpoint	Create a new breakpoint at the location of the cursor. For more information, see Breakpoints.
Disable All Breakpoints	This function clears all set program breakpoints, including breakpoints which have been disabled.
Clear All DataTips	Clear all marked Data Tips. For more information, see DataTips.
Export Data Tips	Save all marked Data Tips to a file in Visual Studio Shell format.
Import DataTips	Load Data Tips from a Visual Studio Shell file.
Options and Settings	Debug options and settings, see Debugger.

4.7. Breakpoints

4.7.1. General Information on Breakpoints

A breakpoint tells the debugger to temporarily suspend execution of a program when a specific condition takes place, e.g. when a certain instruction is about to be executed.

Breakpoints provide a powerful tool that enables you to suspend execution where and when you need to. Rather than stepping through your code line by line or instruction by instruction, you can allow your program to run until it hits a breakpoint, and then start to debug. This speeds up the debugging process.

4.7.1.1. Breakpoint Glyphs

The source windows and the isassembly window show breakpoint locations by displaying symbols called glyphs in the left margin. The following table describes these glyphs.

If you rest the mouse on a breakpoint glyph, a breakpoint tip appears with more information. This information is especially useful for error and warning breakpoints.

Table 4-1. Breakpoint Glyphs

Glyph	Description
	Normal breakpoint. The solid glyph indicates that the breakpoint is enabled. The hollow glyph indicates that it is disabled.
	Advanced breakpoint. Active/disabled. The + sign indicates that the breakpoint has at least one advanced feature (such as condition, hit count, or filter) attached to it.
	Breakpoint error. The X indicates that the breakpoint could not be set because of an error condition.
	Breakpoint warning. The exclamation mark indicates that a breakpoint could not be set because of a temporary condition. Usually, this means that the code at the breakpoint or tracepoint location has not been loaded. It can also be seen if you attach to a process and the symbols for that process are not loaded. When the code or symbols are loaded, the breakpoint will be enabled and the glyph will change.

4.7.2. Operations with Breakpoints

4.7.2.1. To Set a Breakpoint

1. In a source window, click a line of executable code where you want to set a breakpoint. On the right-click menu, click **Breakpoint**, and then click **Insert Breakpoint**.
   —or—

   In a source window, click a line of executable code where you want to set a breakpoint.

   On the Debug menu, click **Toggle Breakpoint**.

4.7.2.2. To Set an Address Breakpoint

1. On the **Debug** menu, point to **Windows**, and then click **Disassembly** if the Disassembly window is not already visible. You need to be in a debug session for this option to be visible.
2. In the Disassembly window, click a line of code, and then click **Toggle Breakpoint** on the **Debug** menu.
   —or—

   Right-click a line of code, and then select **Insert Breakpoint** .

4.7.2.3. To Edit a Breakpoint Location

1. In the Breakpoints window, right-click a breakpoint, then click Location on the right-click menu.
   —or—

   In a source, Disassembly, or Call Stack window, right-click a line that contains a breakpoint glyph, and then click Location from Breakpoints on the right-click menu.

   Note:
   In a source window, you might have to right-click the exact character where the breakpoint is set. This is necessary if the breakpoint is set on a specific character within a line of source code.

4.7.2.4. Hit Count Keeps Track of How Many Times a Breakpoint is Hit

By default, execution breaks every time that a breakpoint is hit. You can choose to:

- Break always (the default)
- Break when the hit count equals a specified value
- Break when the hit count equals a multiple of a specified value
- Break when the hit count is greater than or equal to a specified value

If you want to keep track of the number of times a breakpoint is hit but never break execution, you can set the hit count to a very high value so that the breakpoint is never hit.

The specified hit count is retained only for the debugging session. When the debugging session ends, the hit count is reset to zero.

4.7.2.5. To Specify a Hit Count

1. In the Breakpoints window, right-click a breakpoint, and then click **Hit Count** on the right-click menu.
   —or—

   In a source, Disassembly, or Call Stack window, right-click a line that contains a breakpoint, and then click **Hit Count** from the **Breakpoints** sub menu on the right-click menu.
2. In the Hit Count dialog box, select the behavior you want from the When the breakpoint is hit list. If you choose any setting other than Break always, a text box appears next to the list. Edit the integer that appears in the text box to set the hit count you want.
3. Click OK.

4.7.2.6. To Enable or Disable a Single Breakpoint

In a source, Disassembly, or Call Stack window, right-click a line that contains a breakpoint glyph, point to Breakpoint, then click Enable Breakpoint or Disable Breakpoint.

—or—

In the Breakpoints window, select or clear the check box next to the breakpoint.

To enable or disable all breakpoints

From the **Debug** menu, click **Enable All Breakpoints**.

4.7.2.7. To Delete a Breakpoint

In the **Breakpoints** window, right-click a breakpoint, and then click **Delete** on the right-click menu.

—or—

In a source window or a **Disassembly** window, click the breakpoint glyph.

4.7.2.8. To Delete all Breakpoints

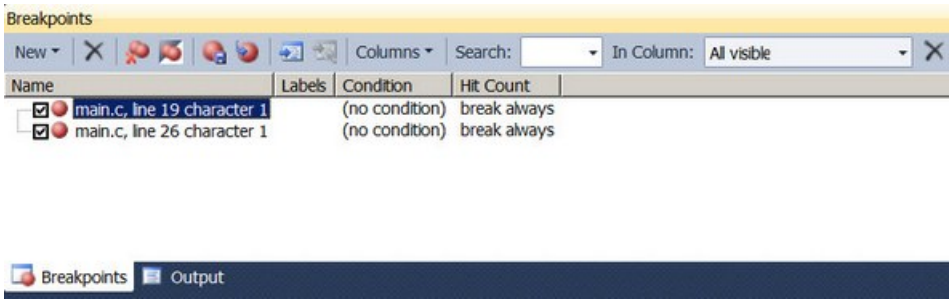
From the **Debug** menu, click **Delete All Breakpoints** .

Confirmation Prompt

When you delete all breakpoints, a prompt requesting confirmation of the action might appear, depending on options settings.

4.7.3. Breakpoint Window

Figure 4-5. Breakpoint Window



You can open the **Breakpoints** window from the **Debug** menu.

4.7.3.1. To Open the Breakpoints Window

On the Debug menu, point to Windows, and then click **Breakpoints**.

4.7.3.2. To Go to the Location of a Breakpoint

In the Breakpoints window, double-click a breakpoint.

—or—

In the Breakpoints window, right-click a breakpoint and choose **Go To Source Code** or **Go To Disassembly**.

—or—

Click a breakpoint, and then click the **Go To Source Code** or **Go To Disassembly tool**.

4.7.3.3. To Display Additional Columns

In the toolbar at the top of the **Breakpoints** window, click the **Columns** tool, and then select the name of the column you want to display.

4.7.3.4. To Export all Breakpoints that Match the Current Search Criteria

In the Breakpoints window toolbar, click the Export all breakpoints matching current search criteria icon.

1. The **Save As** dialog box appears.
2. In the **Save As** dialog box, type a name in the **File name** box.
3. This is the name of the XML file that will contain the exported breakpoints. Note the folder path shown at the top of the dialog box. To save the XML file to a different location, change the folder path shown in that box, or click **Browse Folders** to browse for a new location.
4. Click **Save**.

4.7.3.5. To Export Selected Breakpoints

1. In the **Breakpoints** window, select the breakpoints you want to export.
   To select multiple breakpoints, hold down the Ctrl key and click additional breakpoints.
2. Right-click in the breakpoints list, and choose **Export selected**.
3. The **Save As** dialog box appears.
4. In the **Save As** dialog box, type a name in the **File name** box.
   This is the name of the XML file that will contain the exported breakpoints.

   The folder path is shown at the top of the dialog box. To save the XML file to a different location, change the folder path shown in that box, or click **Browse Folders** to browse for a new location.
5. Click **Save**.

4.7.3.6. To Import Breakpoints

1. In the **Breakpoints window** toolbar, click the Import breakpoints from a file icon.
2. The **Open dialog** box appears.
3. In the **Open dialog** box, browse to the directory where your file is located, and then type the file name or select the file from the file list.
4. Click **OK**.

4.7.3.7. To View Breakpoints that Match a Specified String

In the **Search** box, perform one of the following actions:

1. Type your search string in the **Search** box and press ENTER.
   —or—
2. Click the **Search** drop-down list and select a string from the list of previous search strings.

To limit the search to a specified column, click the In **Column** drop-down list and then click the name of the column that you want to search in.

4.7.3.8. To View all Breakpoints after a Search

In the **Search** box, select and delete the search string and then press ENTER.

4.7.3.9. Breakpoint Labels

In Atmel Studio, you can use labels to help keep track of your breakpoints. A label is a name that you can attach to a breakpoint or a group of breakpoints. You can create a name for a label or you can choose from among a list of existing labels. You can attach multiple labels to each breakpoint.

Labels are useful when you want to mark a group of breakpoints that are related in some way. After you have labeled the breakpoints, you can use the search function in the Breakpoints window to find all breakpoints that have a specified label.

The search function searches all columns of information that are visible in the **Breakpoints** window. To make your labels easy to search for, avoid using label names that might conflict with strings that appear in another column. For example, if you have a source file that is named "`Program.c`", "`Program.c`" will appear in the name of any breakpoint set in that file. If you use "`Prog`" as a label name, all breakpoints set in `Program.c` will appear when you search for breakpoints labeled "`Prog`".

4.7.3.10. To Label Breakpoints

1. In the **Breakpoints** window, select one or more breakpoints.
2. To select multiple breakpoints, use the **Ctrl** key when selecting breakpoints.
3. Right-click the selected breakpoints, and then click **Edit labels**.
4. The **Edit breakpoint labels** dialog box appears.
5. Select one or more labels in the **Choose among existing labels** box.
   —or—

   Type a new label name in the **Type a new label** box, and then click **Add**.

4.7.3.11. To Search for Breakpoints that have a Specified Label

1. In the **Breakpoints** window toolbar, click the **In Column** box and select **Labels** from the drop-down list.
2. In the **Search** box, type the name of the label you want to search for and press Enter.

4.7.3.12. To Remove Labels from Breakpoints

1. In the **Breakpoints** window, select one or more breakpoints.
   Press Ctrl left-click to select multiple breakpoints.
2. Right-click the selected breakpoints, and then click **Edit labels**.

3. The **Edit breakpoint labels** dialog box appears.
4. In the **Choose among existing labels** box, clear the checkboxes for labels that you want to remove from the selected breakpoints.

4.7.3.13. To Sort the Breakpoint List by Label

1. In the **Breakpoints** window, right-click the breakpoint list.
2. Point to **Sort by** and then click **Label**.

(Optional) To change the sort order, right-click the breakpoint list again, point to **Sort by**, and then click **Sort Ascending** or **Sort Descending**.

4.8. Data Breakpoints

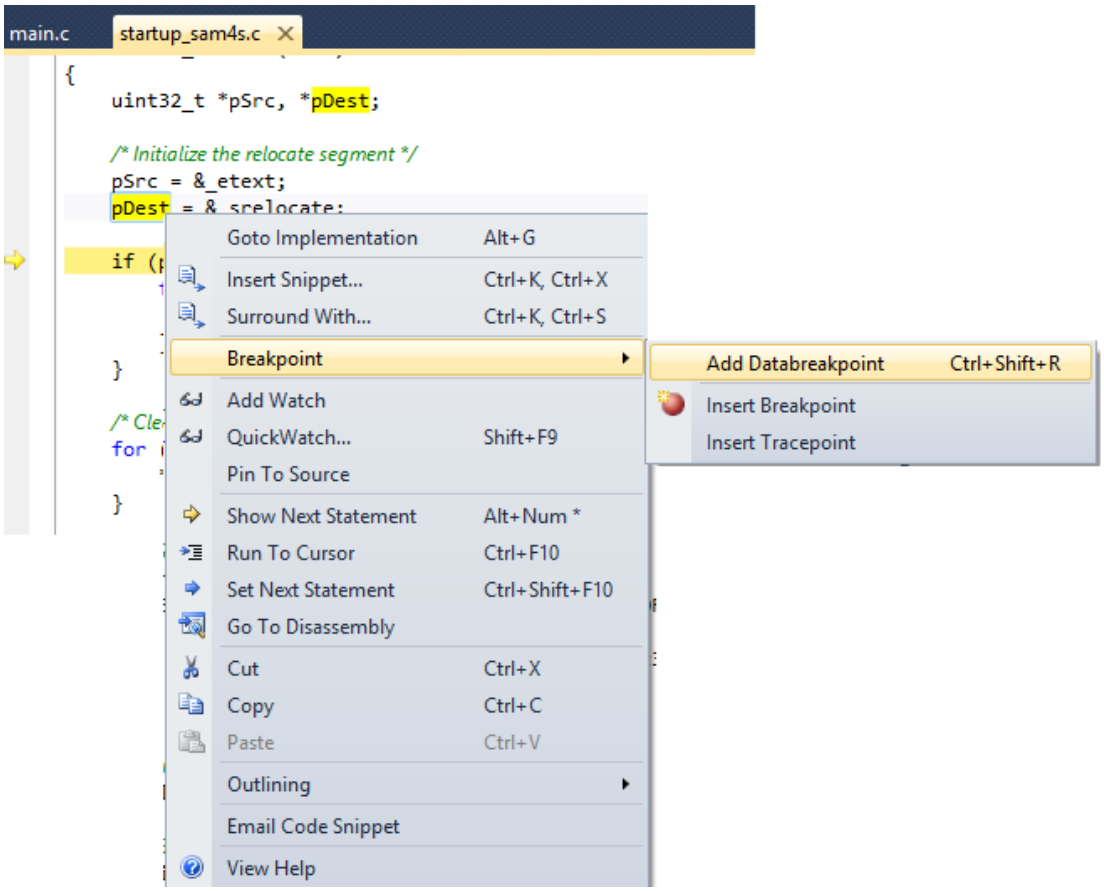
Data breakpoints allow you to break execution when the value stored at a specified memory location is accessed (read/write). In general, Data Breakpoint hardware module listens on the data address and data value lines between the CPU and the data cache and can halt the CPU, if the address and/or value meets a stored compare value. Unlike program breakpoints, data breakpoints halt on the next instruction after the load/store instruction that caused the breakpoint has completed.

4.8.1. Adding Data Breakpoint

Adding Data Breakpoint using code editor context menu

You can add Data breakpoint from code editor. Select the variable in code editor and select **Breakpoint → Add Databreakpoint** from context menu. This adds data breakpoint for a given variable address. Default access mode is write and default mask is none. You can invoke the same command using short cut key Ctrl + Shift + R .

Figure 4-6. Adding Data Breakpoint from the Context Menu



Adding Data Breakpoint from Debug menu

You can add new Data breakpoint from **Debug** → **New Breakpoint** → **New Data Breakpoint**. This opens the Data Breakpoint Configuration window.

Adding Data Breakpoint from Data Breakpoints tool window

You can add new Data breakpoint using the **New** button in Data Breakpoints tool window. This opens the Data Breakpoint Configuration window.

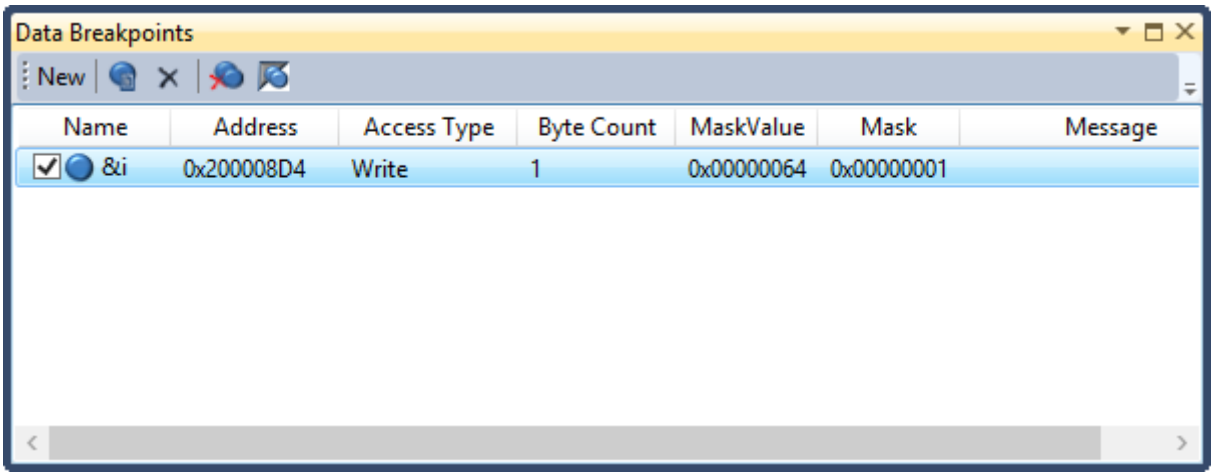
Note: You can add or modify Data breakpoint only in debug mode.

4.8.2. Data Breakpoints Window

4.8.2.1. Data Breakpoints Tool Window

The Data Breakpoint window provides the options to set data breakpoint and lists the added data breakpoints. Enable this window by choosing **Debug** → **Windows** → **Data Breakpoints**.

Figure 4-7. Data Breakpoint Window



Data Breakpoints toolbar has following elements:

- New - provides the data breakpoint configuration window to add a new data breakpoint.
- - provides the data breakpoint configuration window to edit the selected data breakpoint.
- - removes the selected data breakpoint. You can invoke the same command using the Delete key.
- - removes the all data breakpoints.
- - enable or disable all the data breakpoints.

Data Breakpoints window displays several columns related to breakpoint configuration. Some of the columns are dynamically hidden based on the breakpoints configuration. E.g.: if none of the breakpoints has Mask configured, then Mask related columns are not displayed.

Name column has three parts:

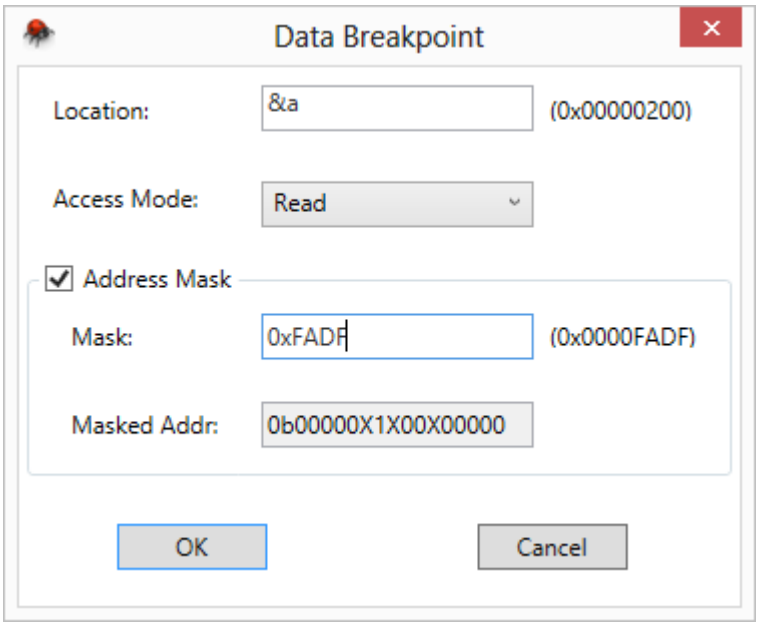
- Check box - Use Check box to enable or disable breakpoint.
- Icon - Glyph to represent the current state of the breakpoint. The following table describes these glyphs. If you rest the mouse on a breakpoint glyph, a breakpoint tip appears with more information. This information is especially useful for error and warning breakpoints.
- Text - Displays the configured location expression for breakpoint.

Table 4-2. Breakpoint Icons

Icon	Description
	Normal breakpoint. The solid glyph indicates that the breakpoint is enabled. The hollow glyph indicates that it is disabled.
	Advanced breakpoint. Active/disabled. The + sign indicates that the breakpoint has hit count attached to it.
	Tracepoint. Active/disabled. Hitting this point performs a specified action but doesn't break program execution.
	Advanced tracepoint. Active/disabled. The + sign indicates that the tracepoint has hit count attached to it.

Icon	Description
	Breakpoint or tracepoint error. The X indicates that the breakpoint or tracepoint couldn't be set because of an error condition. Check Message column for more details on error message.
	Breakpoint or tracepoint is set but with warning. Check Message column for more details on warning message.

4.8.2.2. Data Breakpoint Configuration Window for Mega



This window provides configuration options related to data breakpoint for ATmega devices. Address mask is optional.

Location

You can enter a specific address in RAM (e.g.: `0x8004`) directly or an expression that evaluates to an address in RAM (e.g.: `&x`). Make sure the expression you enter represents the address of the data to monitor.

Note: Data breakpoints on local variables can result in false hits due to reuse of stack memory. Suggestion to declare it as static for debugging purpose.

Access Mode

You can configure the breakpoint to break on specific Access Mode. Three types of access modes are supported.

- Read - Program breaks on read at specified location.
- Write (Default) - Program breaks on write at specified location.
- Read/Write - Program breaks on read or write at specified location.

Address Mask

Address Mask on Mega Data Breakpoints is optional. Use address mask to break on more than one address or a range of address on particular access.

Mask	Mask value to mask the **Location** address to define more than one address or range of address. Bits with value 1 in the mask are significant bits and 0 are don't care bits.

In general for a given address A and mask M, an address B successfully matches when:

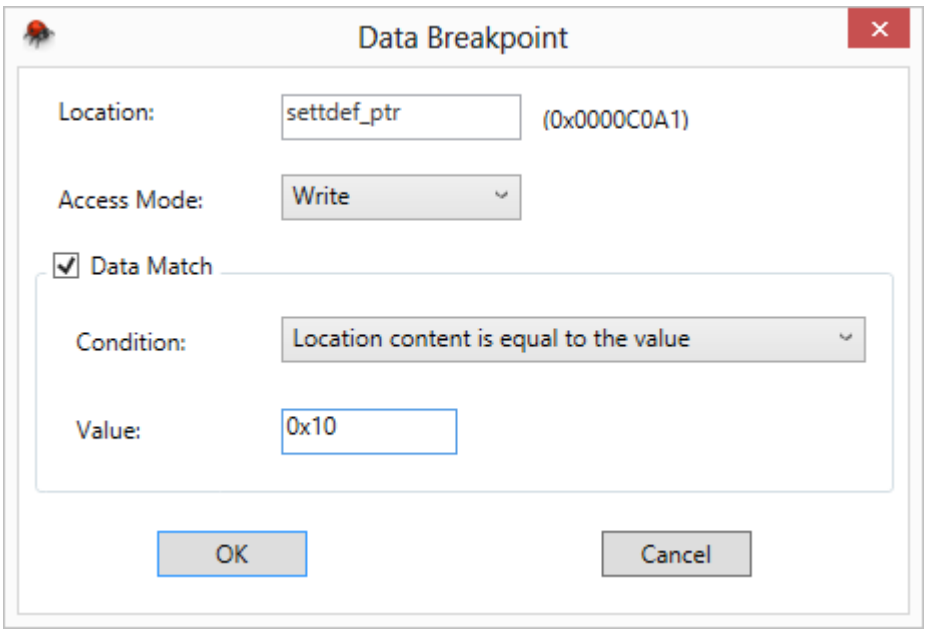
(A) & (M) == (B) & (M), where A is resolved address for the expression entered in **Location**, M is mask value entered in **Mask** and B is any address in RAM.

Masked Address This is a read only field which shows the range of matching addresses on which program can break. The masked address is shown in the binary format for simplicity. 'X' represents don't care bits, remaining bits are expected to match.

E.g. 0b000000010XX000XX means it can break as per access mode, at addresses which has all bits as per this string except X bits. In this case 0^{th}, 1^{st}, 5^{th}, and 6^{th} bit (lsb) can be anything since these bits are don't care (X).

Note: ATmega devices don't support Data Masks.

4.8.2.3. Data Breakpoint Configuration Window for XMEGA



This window provides configuration options related to data breakpoint for XMEGA devices.

Location

You can enter a specific address in RAM (e.g.: `0x8004`) directly or an expression that evaluates to an address in RAM (e.g.: `&x`). Make sure the expression you enter represents the address of the data to monitor.

Note: Data breakpoints on local variables can result in false hits due to reuse of stack memory. Suggestion to declare it as static for debugging purpose.

Access Mode

You can configure the breakpoint to break on specific Access Mode. Three types of access modes are supported.

- Read - Program breaks on read at specified location.
- Write (Default) - Program breaks on write at specified location.
- Read/Write - Program breaks on read or write at specified location.

Data Match

Use the Data Match option to configure Data Breakpoint to compare the data at specified location based on one of the following conditions:

- Location content is equal to the value
- Location content is greater than the value
- Location content is less than or equal to the value
- Location content is within the range
- Location content is outside the range
- Bits of the location is equal to the value

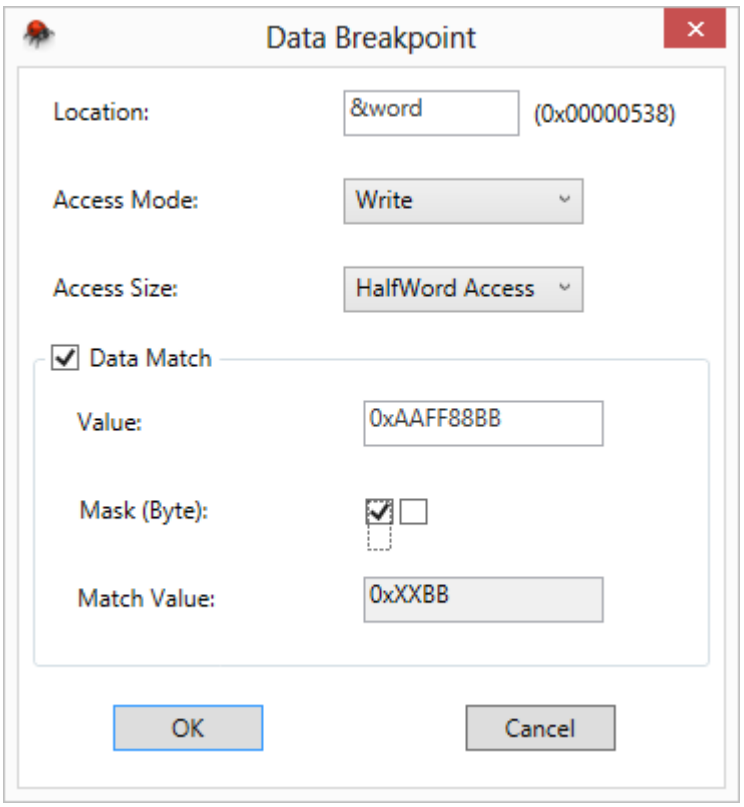
Note: Bit Mask: It is an 8-bit value where bits with '1' are significant and the bits with '0' are don't care.
In general for a given value V and bit mask M, the break event is triggered when the value in the location field VL satisfies the condition

```
(V) & (M) == (VL) & (M)
```

For example, using `Value = 0xA0` and `Bit Mask = 0xF0` will trigger a break event when the location field has any of the values between`0xA0 to 0xAF`.

Note: Condition Value field has to be of 1 byte.

4.8.2.4. Data Breakpoint Configuration Window for UC3



This window provides configuration options related to data breakpoint for UC3 devices.

Location

You can enter a specific address in RAM (e.g.: `0x8004`) directly or an expression that evaluates to an address in RAM (e.g.: `&x`). Make sure the expression you enter represents the address of the data to monitor.

Note: Data breakpoints on local variables can result in false hits due to reuse of stack memory. Suggestion to declare it as static for debugging purpose.

Access Mode

You can configure the breakpoint to break on specific Access Mode. Three types of access modes are supported.

- Read - Program breaks on read at specified location.
- Write (Default) - Program breaks on write at specified location.
- Read/Write - Program breaks on read or write at specified location.

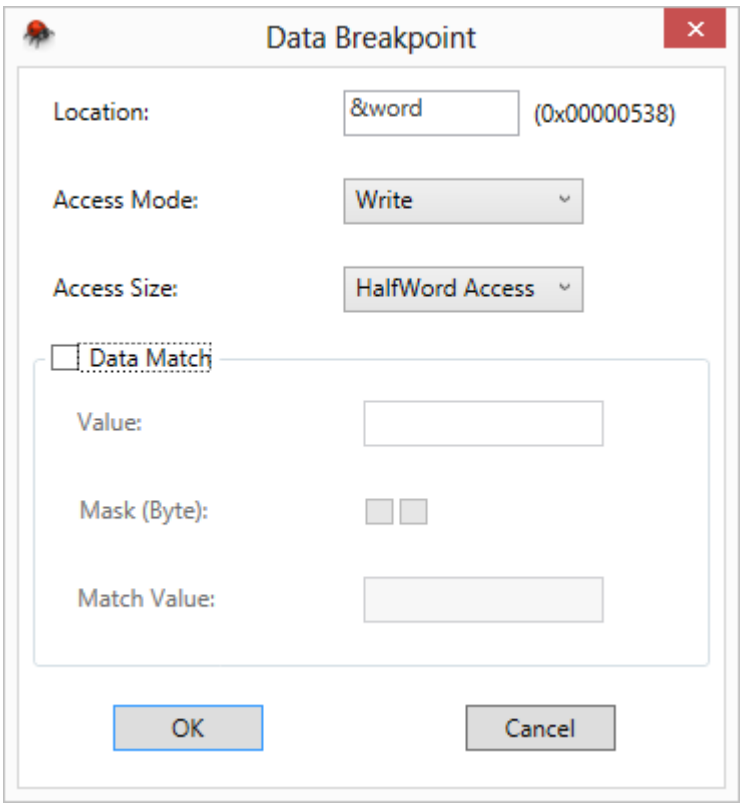
Access Size

You can configure the breakpoint to break on specific Access Size. Four types of access size are supported.

- Any Access (Default) - Program breaks on any access size at specified location.
- Byte Access - Program breaks on Byte access at specified location.
- HalfWord Access - Program breaks on HalfWord access at specified location.
- Word Access - Program breaks on Word access at specified location.

Example for setting access size

Configuration:



Code:

```
1:     int word = 0;
2:     short *halfWord = (short*)&word;
3:
```

```
4:    int main(void)
5:    {
6:      word = 0xAABBCCFF;
7:      *halfWord = 0xDDEE;
8:    }
```

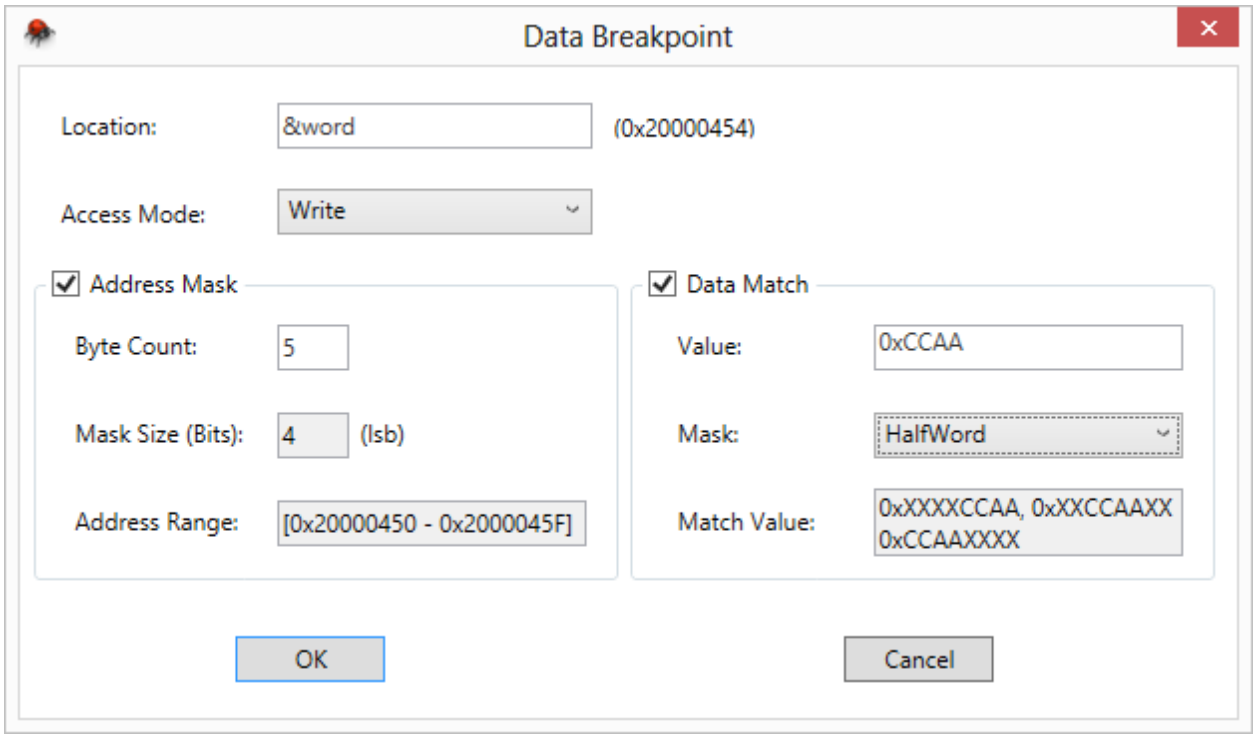
For the above configuration and code, program breaks at line eight after `halfWord` access.

Data Match

Use the **Data Match** option to configure Data Breakpoint to compare the data at specified location with a 32-bit value. Break event is triggered on a successful match.

Value	32-bit (4-byte) value to compare with data at **Location** address. The value can be decimal or hexadecimal (e.g.: `100` or `0x64`). Based on **Access Size**, respective bytes are used for data comparison. For example, if you select "HalfWord Access" as **Access Size** and enter `0xAABBCCDD` as **Value**, then only the last two bytes (`0xCCDD`) are used for data comparison. You could further refine the **Value** by specifying **Mask**.
Mask (Byte)	Each check box controls the significance of respective byte in the **Value** field. Select appropriate check box to mask specific byte in the **Value** field. The number of check boxes displayed is decided based on **Access Size**. Four check boxes (one per byte) are displayed for "Any" and "Word Access", two check boxes for "HalfWord Access" and one check box for "Byte Access".
Match Value	A read only field, which displays masked value based on **Access Size**, **Value**, and **Mask (Byte)** field. Masked byte is represented as `'XX'`, which means that byte is insignificant in data comparison.

4.8.2.5. Data Breakpoint Configuration Window for SAM



Provides the configuration options related to data breakpoint for SAM devices. Address mask and Data match are optional.

Location

You can enter a specific address (e.g.: `0x8004`) directly or an expression that evaluates to an address (e.g.: `&x`). Make sure the expression you enter represents the address of the data to monitor.

Note: Data breakpoints on local variables can result in false hits due to reuse of stack memory. Suggestion to declare it as static for debugging purpose.

Access Mode

You can configure the breakpoint to break on a specific Access Mode. Three types of access modes are supported.

- Read - Program breaks on read at specified location.
- Write (Default) - Program breaks on write at specified location.
- Read/Write - Program breaks on read or write at specified location.

Address Mask

Use this setting to define range of addresses to monitor for access.

Byte Count	You can enter a number of address locations to monitor starting at **Location** address. The actual range of monitored address can be wider than the expected range. E.g.: if the **Location** is `0x23FA` and the **Byte Count** is `5`, then the actual range of the monitored address is [`0x23F8` to `0x23FF`]. The way actual range is computed is by calculating the number of least significant bits that has to be masked in the **Location** address (`0x23FA`) in order to cover the expected range [`0x23FA` to `0x23FA + 5`]. In this case the number of bits to be masked is three. As a result the actual range [`0x23F8` to `0x23FF`] is wider than the expected range [`0x23FA` to `0x23FF`].
Mask Size	This is a read only field, which displays the number of least significant bits masked in the **Location** address. **Mask Size** is calculated based on the **Byte Count** and **Location** addresses.
Address Range	This is a read only field, which displays the actual range of address monitored for access. Range is closed interval including both minimum and maximum address.

Data Match

Use the **Data Match** option to configure Data Breakpoint to compare the data at specified location with a 32-bit value. Break event is triggered on a successful match.

Value	32-bit (4-byte) value to compare with data at the **Location** address. The value can be decimal or hexadecimal (e.g.: `100` or `0x64`). You could further refine the **Value** by specifying **Mask**.
Mask	You can use **Mask** to extract appropriate bytes from **Value** to use for data comparison. E.g.: if the **Value** is `0xAABBCCDD` and **Mask** is HalfWord, then last two bytes (`0xCCDD`) is extracted from **Value** and used for data comparison. This means data comparison would succeed for the following matches `0xXXXXCCDD`, `0xXXCCDDXX`, and `0xCCDDXXXX`, where `X` is a don't care hexadecimal digit (0 to F). Three types of **Mask**'s are supported. • Byte - Last byte extracted from **Value** is used for data comparison.

- HalfWord - Last two bytes extracted from **Value** is used for data comparison.
- Word (Default) - Whole four bytes from **Value** is used for data comparison.

Match Value This is a read only field which displays match values.

Data Match Example

Byte matching example

- **Location** = 0x200008D4
- **Value** = 0x0000CC**AA**
- **Mask** = Byte

The program breaks when Location`0x200008D4` has any of the following values:

- 0xXXXXXX**AA**
- 0xXXXX**AA**XX
- 0xXX**AA**XXXX
- 0x**AA**XXXXX

HalfWord matching example

- **Location** = 0x200008D4
- **Value** = 0x0000**CCAA**
- **Mask** = HalfWord

The program breaks when Location`0x200008D4` has any of the following values:

- 0xXXXX**CCAA**
- 0xXX**CCAA**XX
- 0x**CCAA**XXXX

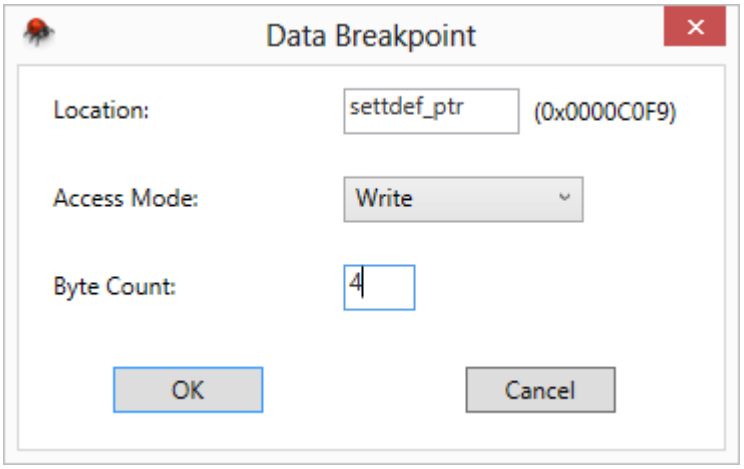
Word matching example

- **Location** = 0x200008D4
- **Value** = 0x**0000CCAA**
- **Mask** = Word

The program breaks when Location 0x200008D4 has the following value:

- 0x**00000CCAA**

4.8.2.6. Data Breakpoint Configuration Window for Simulator Tool



The above configuration window is displayed for any device architecture when a simulator tool is selected.

Location

You can enter a specific address in RAM (e.g.: `0x8004`) directly or an expression that evaluates to an address in RAM (e.g.: `&x`). Make sure the expression you enter represents the address of the data to monitor.

Note: Data breakpoints on local variables can result in false hits due to reuse of stack memory. Suggestion to declare it as static for debugging purpose.

Access Mode

You can configure the breakpoint to break on specific Access Mode. Three types of access modes are supported.

- Read - Program breaks on read at specified location.
- Write (Default) - Program breaks on write at specified location.
- Read/Write - Program breaks on read or write at specified location.

Byte Count

You can enter number of address locations to monitor starting at **Location** address.

Note:

- Simulator supports unlimited number of data breakpoints

4.8.2.7. How to: Specify a Data Breakpoint Hit Count

Hit count

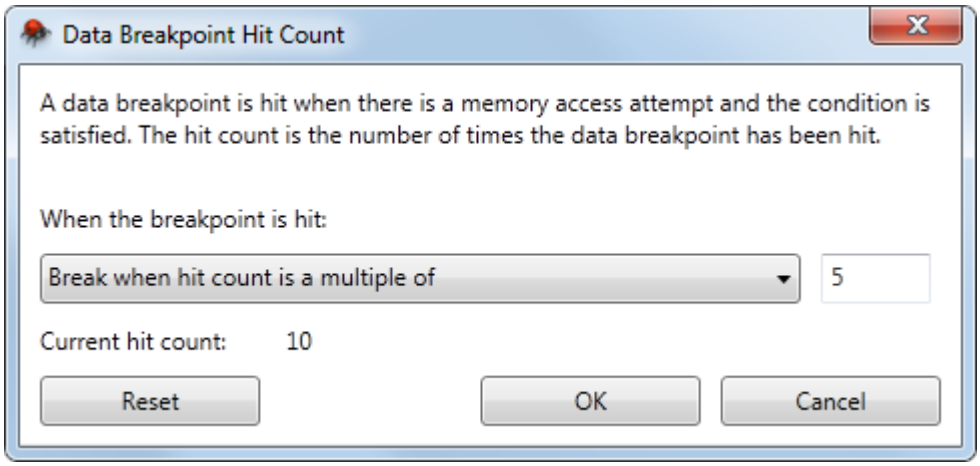
The number of times the value stored in the specified memory location is accessed (read/write).

To specify a hit count

To specify or edit the Hit Count property, you must open the Hit Count Dialog Box.

In **Data Breakpoints** window, select a breakpoint row, and then choose **Hit Count** on the context menu.

Figure 4-8. Hit Count Dialog Box



To set or modify a hit count property, use the following controls:

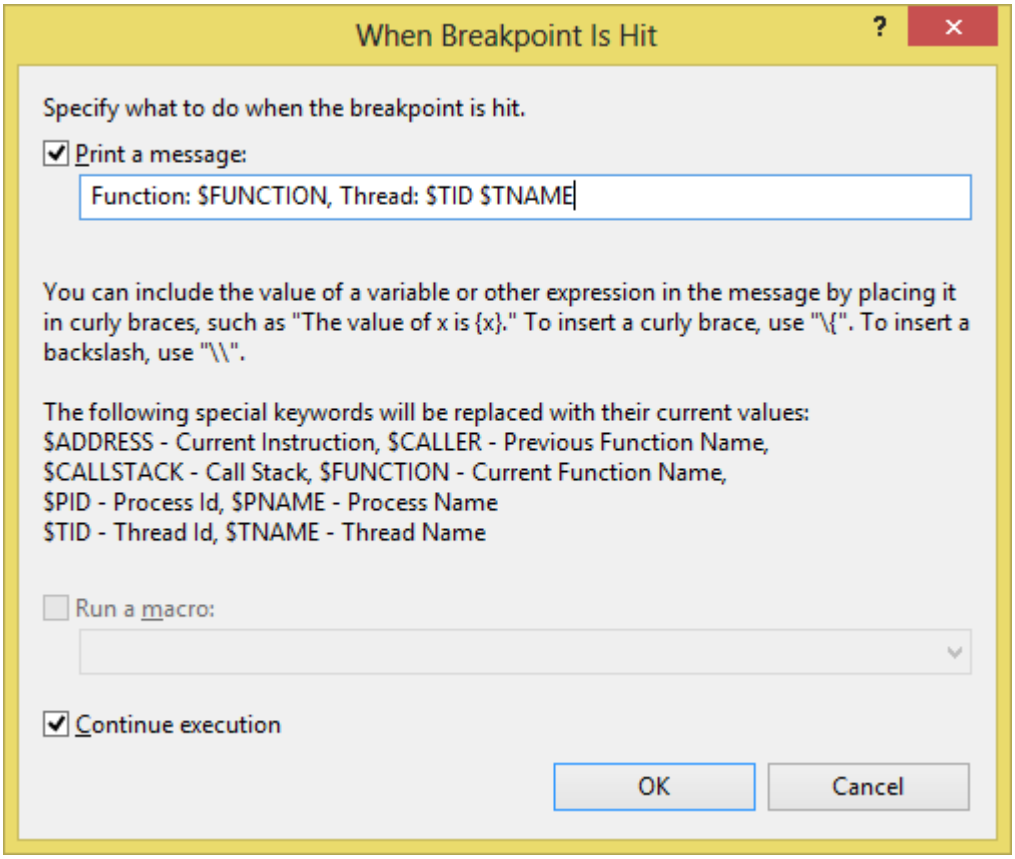
- **When the breakpoint is hit**: This setting determines how the breakpoint should behave when it is hit. You can choose to:
  - Break always (the default)
  - Break when the hit count equals a specified value
  - Break when the hit count equals a multiple of a specified value
  - Break when the hit count is greater or equal to a specified value
- **Current hit count**: This value shows the number of times the data breakpoint has been hit. A read/write data for a variable will be converted into multiple instructions, resulting in several memory access. So the data breakpoint hits multiple times for the same variable and the hit count will be updated accordingly.
- **Reset**: This button resets the value shown for the **Current hit count** to 0.

If you choose any option other than the default in **When the breakpoint is hit** list control, an edit box appears next to it. Edit the value in this edit box to set the hit count value. For example, you might choose break when hit count is equal to and enter 5. This causes execution to stop the 5th time the breakpoint is hit, not on any other hit.

4.8.2.8. When Breakpoint is Hit Dialog Box

With this dialog box you can print a message in the output window when a data breakpoint is hit.

To open the **When Breakpoint Is Hit** Dialog Box, go to the **Data Breakpoints** window, select a breakpoint row, and then choose **When Hit** on the context menu.



Specify the message

- You can use any of the keywords that are described on the **When Breakpoint Is Hit** dialog box. E.g.: Process Id : $PID.
- You can also specify expressions in the message by placing it in the curly braces, such as sum={a +b}

Specify the breakpoint behavior

To break execution when the breakpoint is hit, clear the **Continue Execution** check box. When **Continue Execution** is checked, execution is not halted. In both cases, the message is printed.

4.8.3. General Information on Data Breakpoint

- Data Breakpoint can be edited/added only in debug mode
- Local variables must always be qualified with the function name. This is also the case if the user wants to add a variable from the function that the program has stopped in.

  Data breakpoints on local variables can result in false hits due to reuse of stack memory.

Tip:
Declare local variables as static and provide the static variable's address in the location field, the address of the static variable is fixed during compilation/linking.

- Global variables are initialized with default values during the start up, the valid data breakpoint for global variables will hit in the disassembly or initialization code during the start up

- There can be several instructions to perform read/write data for a variable, for example 'int' data type can have two individual byte read/write instructions so the data breakpoint hits twice for the same variable
- Data breakpoint event can occur when the bus access happens for the specific address
- Maximum number of data breakpoint supported: (this may vary based on specific device/family refer data sheet)

Architecture	Maximum Data breakpoint supported
ATmega	• Two without Data Mask (OR) • One without Data Mask and one with Data Mask
XMEGA	• Two without Data Mask (OR) • One without Data Mask and one with Data Mask (OR) • Two with Data Mask
UC3	• Two without Data Mask (OR) • One without Data Mask and one with Data Mask (OR) • Two with Data Mask
SAM	Device dependent refer data sheet
Tiny	Does not support data breakpoint

Most of the devices conforms to the above limit.

Note: ATmega and SAM device uses multiple hardware resources when a data breakpoint with data mask is set. Hence, using data mask can reduce the number of data breakpoint that can be set.

4.8.4. Data Breakpoint Usage

4.8.4.1. Stack Overflow Detection Using Data Breakpoint

You can decide on maximum stack size for the your application and calculate the approximate end address for the stack.

In the end address set the data breakpoint for address range by applying address mask and access type as Read/Write.

Note: The above method may cause false break when heap memory tries to access the specified stack end address.

4.9. QuickWatch, Watch, Locals, and Autos Windows

The Atmel Studio debugger provides several windows, collectively known as variable windows, for displaying variable information while you are debugging.

Each variable window has a grid with three columns: **Name**, **Value**, and **Type**. The **Name** column shows the names of variables added automatically in the **Auto** and **Locals** windows.

In the Watch window, the **Name** column is where you can add your own variables or expressions. See how to watch an expression in the Debugger.

The **Value** and **Type** columns display the value and data type of the corresponding variable or expression result.

You can edit the value of a variable in the **Value** column.

The variable windows, **Autos**, **Locals**, and **Watch**, display the values of certain variables during a debugging session. The QuickWatch dialog box can also display variables. When the debugger is in break mode, you can use the variable windows to edit the values of most variables that appear in these locations.

Note: Editing floating-point values can result in minor inaccuracies because of decimal-to-binary conversion of fractional components. Even a seemingly harmless edit can result in changes to some of the least significant bits in the floating-point variable.

When an expression is evaluated in the Watch window, you might see a refresh icon. This indicates an error or out-of-date value. For more information, see How to: Refresh Watch Values.

If you want to, you can enter an expression for a value. The debugger will evaluate the expression and replace it with the resulting value. The debugger accepts most valid language expressions in a Watch window. For more information, see Expression Formatting.

If you are programming in native code, you might sometimes have to qualify the context of a variable name or an expression that contains a variable name. The context means the function, source file, and module where a variable is located. If you have to do this, you can use the context operator syntax. For more information, see Context Operator (C/C++ Language Expressions).

Evaluating some expressions can change the value of a variable or otherwise affect the state of your program. For example, evaluating the following expression changes the value of `var1` and `var2`:

```
var1 = var2++
var1 = var2++
```

Expressions that change data are said to have side effects, which can produce unexpected results if you are not aware of them. Therefore, make sure you understand the effect of an expression before you execute it.

To edit a value in a variable window

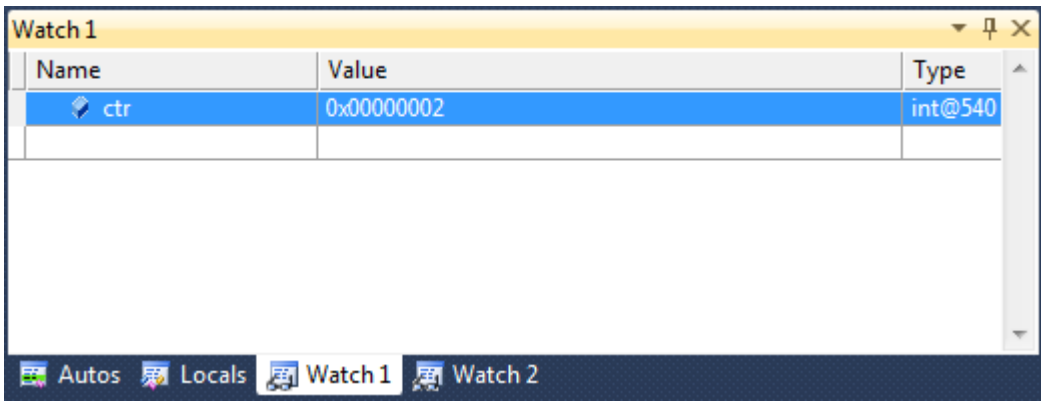
1. The debugger must be in break mode.
2. If the variable is an array or an object, a tree control appears next to the name in the Name box. In the Name column, expand the variable, if necessary, to find the element whose value you want to edit.
3. In the row you want to change, double-click the Value column.
4. Type the new value.
5. Press ENTER.

To display a variable window

On the Debug menu, choose Windows, then choose the name of the variable window you want to display (Autos, Locals, Watch, or Watch1 through Watch4).

You cannot access these menu items or display these windows in design mode. To display these menu items, the debugger must be running or in break mode.

4.9.1. Watch Window



The Watch window and QuickWatch dialog box are places where you can enter variable names and expressions that you want to watch during a debugging session.

The **QuickWatch** dialog box enables you to examine a single variable or expression at a time. It is useful for taking a quick look at one value or a larger data structure. The Watch window can store several variables and expressions that you want to view over the course of the debugging session. Atmel Studio has multiple Watch windows, which are numbered **Watch1** through **Watch4**.

A variable name is the simplest expression you can enter. If you are debugging native code, you can use register names as well as variable names. The debugger can accept much more complex expressions than that, however. For example, you could enter the following expression to find the average value of three variables:

```
(var1 + var2 + var3) / 3
```

The debugger accepts most valid language expressions in a Watch window. For more information, see Expression Formatting.

If you are programming in native code, you may sometimes need to qualify the context of a variable name or an expression containing a variable name. The context means the function, source file, and module where a variable is located. If you have to do this, you can use the context operator syntax.

Expressions that Affect the State of Your Program

Evaluating some expressions can change the value of a variable or otherwise affect the state of your program. For example, evaluating the following expression changes the value of `var1`:

```
var1 = var2
```

Expressions that change data are said to have side effects. If you enter an expression that has a side effect into the Watch window, the side effect will occur every time the expression is evaluated by the **Watch** window. This can produce unexpected results if you are unaware that the expression has side effects. An expression that is known to have side effects is only evaluated one time, when you first enter it. Subsequent evaluations are disabled. You can manually override this behavior by clicking an update icon that appears next to the value.

Unexpected side effects are frequently the result of function evaluation. For example, you could enter the following function call into the Watch window:

```
PrintFunc1(var1)

Func1(var1)
```

If you call a function from the Watch window or QuickWatch, the function you are calling might change data, creating a side effect. One way to avoid possible unexpected side effects from function evaluation is

to turn OFF automatic function evaluation in the Options dialog box. This disables automatic evaluation of newer language features, such as properties. However, it is safer.

Note: When you examine an expression in the Watch window, you might see an update icon, which resembles two green arrows, circling in opposite directions within a green circle. This is especially likely if you have turned OFF automatic function evaluation. The update icon indicates an error or out-of-date value.

The Atmel Studio debugger automatically expands common data types to show their most important elements. You add expansions for custom data types. For more information, see Displaying Custom Data Types and Visualizers.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the Tools menu. For more information, see Menus and Settings.

To evaluate an expression in the Watch window

1. In the Watch window, click an empty row in the **Name** column. The debugger must be in break mode at this point. Type or paste the variable name or expression you want to watch.
   —or—

   Drag a variable to a row in the Watch window.
2. Press ENTER.
3. The result appears in the **Value** column. If you type the name of an array or object variable, a tree control appears next to the name in the Name column. Expand or collapse the variable in the **Name** column.
4. The expression remains in the Watch window until you remove it.

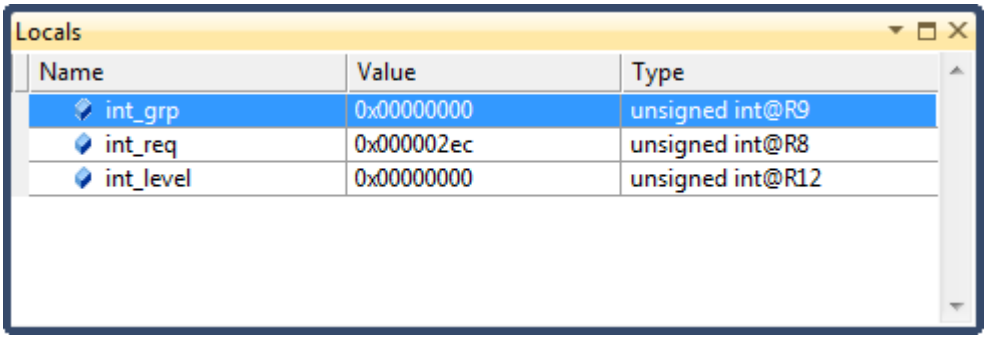
To evaluate an expression in QuickWatch

1. In the QuickWatch dialog box, type or paste the variable, register, or expression into the Expression text box.
2. Click Reevaluate or press ENTER.
3. The value appears in the **Current** value box.
4. If you type the name of an array or object variable in the Expression box, a tree control appears next to the name in the **Current** value box. Expand or collapse the variable in the **Name** column.

To reevaluate a previous expression in QuickWatch

1. In the QuickWatch dialog box, click the down arrow that appears to the right of the **Expression** box.
2. Choose one of the previous expressions from the drop-down list.
3. Click **Reevaluate**.

4.9.2. Locals Window



The Locals window displays variables local to the current context.

4.9.2.1. To Display the Locals Window

From the Debug menu, choose Windows and click Locals. (The debugger must be running or in break mode.)

4.9.2.2. To Choose an Alternative Context

The default context is the function containing the current execution location. You can choose an alternate context to display in the Locals window:

- Use the Debug Location toolbar to select the desired function, thread, or program
- Double click on an item in the Call Stack or Threads window

To view or modify information in the Locals window, the debugger must be in break mode. If you choose Continue, some information may appear in the Locals window while your program executes, but it will not be current until the next time your program breaks (in other words, it hits a breakpoint or you choose Break All from the Debug menu).

4.9.2.3. To Modify the Value of a Variable in the Locals Window

1. The debugger must be in break mode.
2. In the Locals window, select the value you want to edit by double-clicking on it or by using the TAB key.
3. Type the new value, and press ENTER.

Attention:
Editing floating-point values can result in minor inaccuracies because of decimal-to-binary conversion of fractional components. Even a seemingly harmless edit can result in changes to some of the least significant bits in the floating-point variable.

Setting Numeric Format

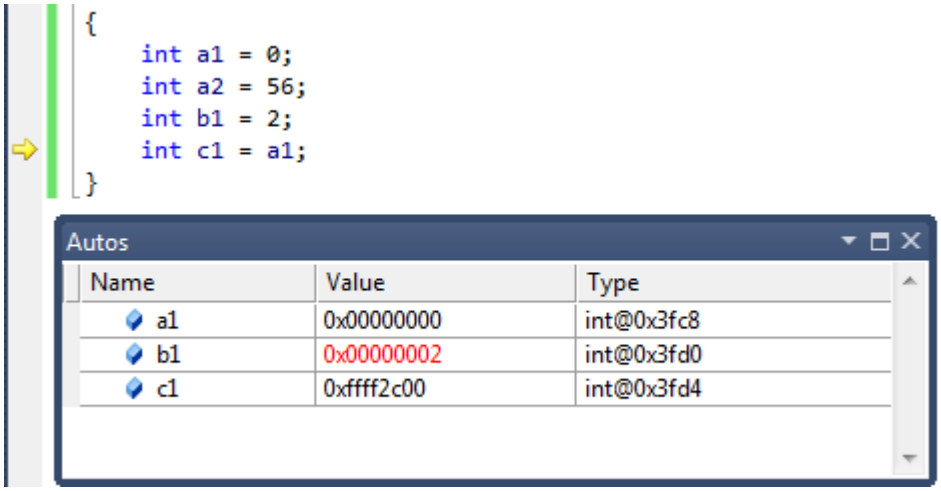
You can set the numeric format used in the debugger windows to decimal or hexadecimal. Right click inside the Locals window, and check/uncheck the **Hexadecimal display** menu item.

4.9.3. Autos Window

The Autos window displays variables used in the current statement and the previous statement.

The current statement is the statement at the current execution location (the statement that will be executed next if execution continues). The debugger identifies these variables for you automatically, hence the window name.

Structure and array variables have a tree control that you can use to display or hide the elements.



To display the Autos window

From the Debug menu, choose Windows and click Autos. (The debugger must be running or in break mode.)

4.9.3.1. To Modify the Value of a Variable in the Autos Window

1. The debugger must be in break mode.
2. Display the **Autos** window, if necessary.
3. In the **Value** column, double-click the value you want to change.
   -or-

   Single-click to select the line, then press the TAB key.
4. Type the new value, and press ENTER.

Attention:
Editing floating-point values can result in minor inaccuracies because of decimal-to-binary conversion of fractional components. Even a seemingly harmless edit can result in changes to some of the least significant bits in the floating-point variable.

4.9.3.2. Setting Numeric Format

You can set the numeric format used in the debugger windows to decimal or hexadecimal. Right click inside the Autos window, and check/uncheck the **Hexadecimal display** menu item.

4.9.4. QuickWatch and Watches

While debugging you might want to track a value of a variable or an expression. To do so you can right click at the expression under cursor and select **Add a Watch** or **Quickwatch**.

The **QuickWatch** dialog box lets you examine and evaluate variables and expressions. Because **QuickWatch** is a modal dialog box, you have to close it before you can continue to debug. You can also edit the value of a variable in **QuickWatch**. For more information on how to watch a variable, see Watch Window.

Some users might wonder why **QuickWatch** is useful. Why not add the variable or expression to the Watch window? That is possible, but if you just want to do a quick scratch calculation that involves one or

more variables, you do not want to clutter the Watch window with such calculations. That is when the **QuickWatch** dialog box is especially useful.

Another feature of the **QuickWatch** dialog box is that it is resizeable. If you want to examine the members of a large object, it is frequently easier to expand and examine the tree **QuickWatch** than it is in the Watch, Locals, or Autos window.

The **QuickWatch** dialog box does not allow you to view more than one variable or expression at a time. Also, because **QuickWatch** is a modal dialog box, you cannot perform operations such as stepping through your code while **QuickWatch** is open. If you want to do these things, use the Watch window instead.

Some expressions have side effects that change the value of a variable or otherwise change the state of your program when they are executed. Evaluating an expression in the **QuickWatch** dialog box will have the same effect as executed the expression in your code. This can produce unexpected results if you do not consider the side effects of the expression.

Note: In Atmel Studio, you can view a variable's value by placing the cursor over the variable. A small box called a DataTip appears and shows the value.

To open the QuickWatch dialog box

While in break mode, choose **QuickWatch** on the Debug menu.

To open the QuickWatch dialog box with a variable added

While in break mode, right-click a variable name in the source window name and choose **QuickWatch**. This automatically places the variable into the **QuickWatch** dialog box.

To add a QuickWatch expression to the Watch window

In the QuickWatch dialog box, click Add Watch.

Whatever expression that was displayed in the **QuickWatch** dialog box is added to the list of expressions in the Watch window. The expression will normally be added to the Watch1 window.

4.9.5. Expression Formatting

The Atmel Studio debugger includes expression evaluators that work when you enter an expression in the QuickWatch and Watches, Memory View, Watch Window, or Immediate window. The expression evaluators are also at work in the Breakpoints window and many other places in the debugger.

General Syntax:

```
Val, formatString
```

Format Specifier for values

The following tables show the format specifiers recognized by the debugger.

Table 4-3. Debug Format Specifiers for Values

Specifier	Format	Expression	Value displayed
d,i	signed decimal integer	0xF000F065, d	-268373915
u	unsigned decimal integer	0x0065, u	101
b	unsigned binary number	0xaa,b2	0b10101010
o	unsigned octal integer	0xF065, o	0170145

Specifier	Format	Expression	Value displayed
x,X	Hexadecimal integer	61541, x	0x0000F065
1,2,4,8	As a suffix specifying number of bytes for: d, i, u, o, x, X.	00406042,x2	0x0c22
s	String	0x2000, s	"Hello World"
f	signed floating point	(3./2.), f	1.500000
e	signed scientific notation	(3./2.), e	1.500000e+000
g	signed floating point or signed scientific notation, whichever is shorter	(3./2.), g	1.5
c	Single character	0x0065, c	101 'e'

Size Specifier for Pointers as Arrays

If you have a pointer to an object you want to view as an array, you can use an integer to specify the number of array elements:

`ptr,10` or `array,20`

Memory type specifier

The following memory type specifiers will force the memory reference to a specific memory type. To be used in the memory window in the address field, you should have a pointer to an object you want to view as an array. You can use an integer to specify the number of the array elements:

Table 4-4. Debug Memory Type Specifiers

Specifier	Expression	Value displayed
flash or program	Program memory	0,flash
data	Data memory	0x2000,data
sram	SRAM	0x100,sram
reg or registers	registers	1,reg
io, eeprom, fusebytes, lockbytes, signature, usersign, prodsign	Memory types with same names	

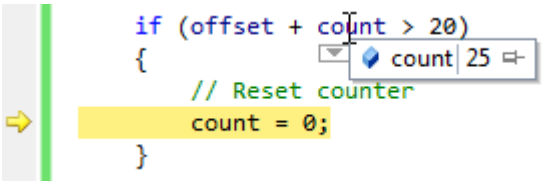
4.10. DataTips

DataTips provide a convenient way to view information about variables in your program during debugging. DataTips work only in break mode and only with variables that are in the current scope of execution.

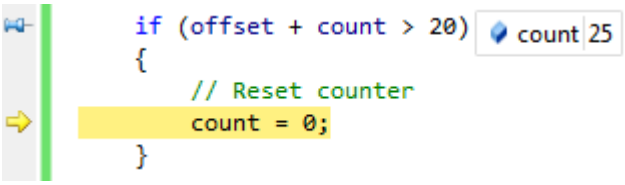
In Atmel Studio, DataTips can be pinned to a specific location in a source file, or they can float on top of all Atmel Studio windows.

To display a DataTip (in break mode only)

1. In a source window, place the mouse pointer over any variable in the current scope. A DataTip appears.



2. The DataTip disappears when you remove the mouse pointer. To pin the DataTip so that it remains open, click the Pin to source icon, or
   - Right-click on a variable, then click Pin to source



The pinned DataTip closes when the debugging session ends.

To unpin a DataTip and make it float

- In a pinned DataTip, click the Unpin from source icon

  The pin icon changes to the unpinned position. The DataTip now floats above any open windows. The floating DataTip closes when the debugging session ends.

To repin a floating DataTip

- In a DataTip, click the pin icon

  The pin icon changes to the pinned position. If the DataTip is outside a source window, the pin icon is disabled and the DataTip cannot be pinned.

To close a DataTip

- Place the mouse pointer over a DataTip, and then click the Close icon

To close all DataTips

- On the Debug menu, click Clear All DataTips

To close all DataTips for a specific file

- On the Debug menu, click Clear All DataTips Pinned to File

4.10.1. Expanding and Editing Information

You can use DataTips to expand an array, a structure, or an object to view its members. You can also edit the value of a variable from a DataTip.

To expand a variable to see its elements:

- In a DataTip, put the mouse pointer over the + sign that comes before the variable name

  The variable expands to show its elements in tree form.

When the variable is expanded, you can use the arrow keys on your keyboard to move up and down. Alternatively, you can use the mouse.

To edit the value of a variable using a DataTip

1. In a DataTip, click the value. This is disabled for read-only values.
2. Type a new value and press ENTER.

4.10.2. Making a DataTip Transparent

If you want to see the code that is behind a DataTip, you can make the DataTip temporarily transparent. This does not apply to DataTips that are pinned or floating.

To make a DataTip transparent

- In a DataTip, press CTRL

  The DataTip will remain transparent as long as you hold down the CTRL key.

4.10.3. Visualizing Complex Data Types

If a magnifying glass icon appears next to a variable name in a DataTip, one or more Visualizers are available for variables of that data type. You can use a visualizer to display the information in a more meaningful, usually graphical, manner.

To view the contents of a variable using a visualizer

- Click the magnifying glass icon to select the default visualizer for the data type

  -or-

  Click the pop-up arrow next to the visualizer to select from a list of appropriate visualizers for the data type.

A visualizer displays the information.

4.10.4. Adding Information to a Watch Window

If you want to continue to watch a variable, you can add the variable to the Watch window from a DataTip.

To add a variable to the Watch window

- Right-click a DataTip, and then click Add Watch

The variable is added to the Watch window. If you are using an edition that supports multiple Watch windows, the variable is added to Watch 1.

4.10.5. Importing and Exporting DataTips

You can export DataTips to an XML file, which can be shared with a colleague or edited using a text editor.

To Export DataTips

1. On the Debug menu, click Export DataTips.
   The Export DataTips dialog box appears.
2. Use standard file techniques to navigate to the location where you want to save the XML file, type a name for the file in the File name box, and then click OK.

To Import DataTips

1. On the Debug menu, click Import DataTips.
   The Import DataTips dialog box appears.
2. Use the dialog box to find the XML file that you want to open and click OK.

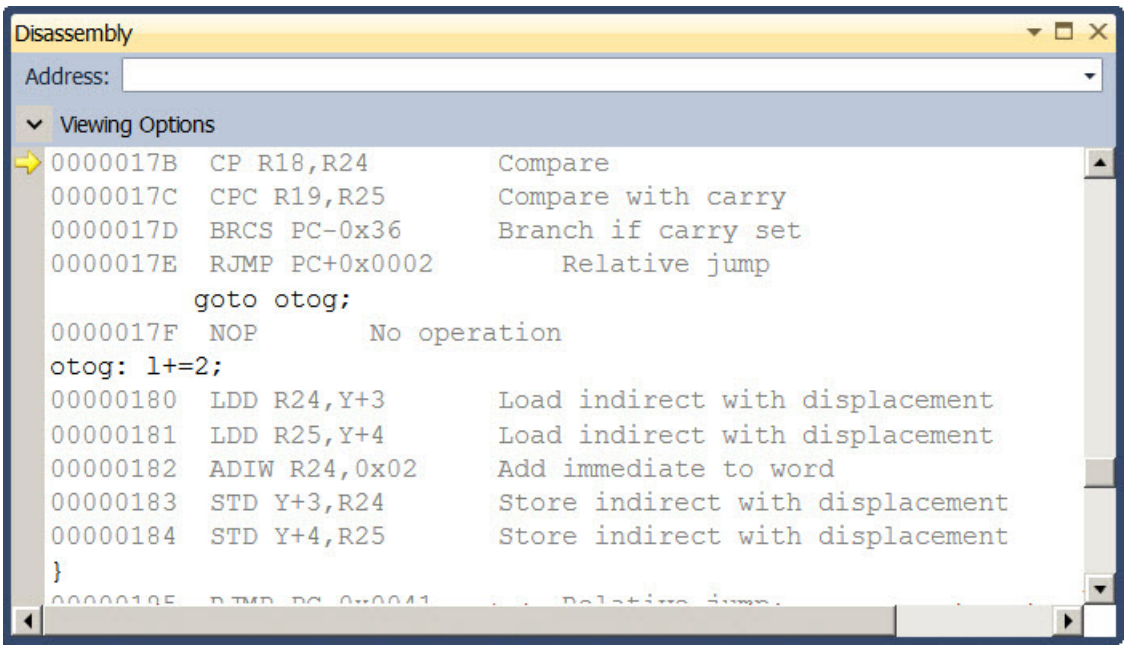
4.11. Disassembly View

The Disassembly window is only available when debugging. When any supported high level language is used, the source window is automatically displayed and the disassembly window is OFF. Enable it by choosing **Debug** → **Windows** → **Disassembly** or Ctrl Alt D during a debugging session.

The disassembly window shows your program code disassembled. Program execution and AVR instructions can be followed in this view. By right clicking inside the Disassembly window you will be able to set breakpoints, run to the position of the cursor, or go to the source code. You cannot modify the source code from the Disassembly window.

In addition to assembly instructions, the Disassembly window can show the following optional information:

- Memory address where each instruction is located. For native applications, this is the actual memory address.
- Source code from which the assembly code derives
- Code bytes byte representations of the actual machine
- Symbol names for the memory addresses
- Line numbers corresponding to the source code



Assembly-language instructions consist of mnemonics, which are abbreviations for instruction names, and symbols that represent variables, registers, and constants. Each machine-language instruction is represented by one assembly-language mnemonic, usually followed by one or more variables, registers, or constants.

Because assembly code relies heavily on processor registers or, in the case of managed code, common language runtime registers, you will often find it useful to use the Disassembly window in conjunction with the Registers window, which allows you to examine register contents.

Note: You may see inconsistencies in instructions that work on explicit addresses. This stems from the historic difference between the AVR Assembler and Assembly Language and the GCC Assembler and the assembly used on bigger computer systems. You might therefore encounter disassemblies that look like the one below.

```
    13:     asm volatile ("JMP 0x0001778A");
0000007D 0c.94.c5.bb          JMP 0x0000BBC5        Jump >
```

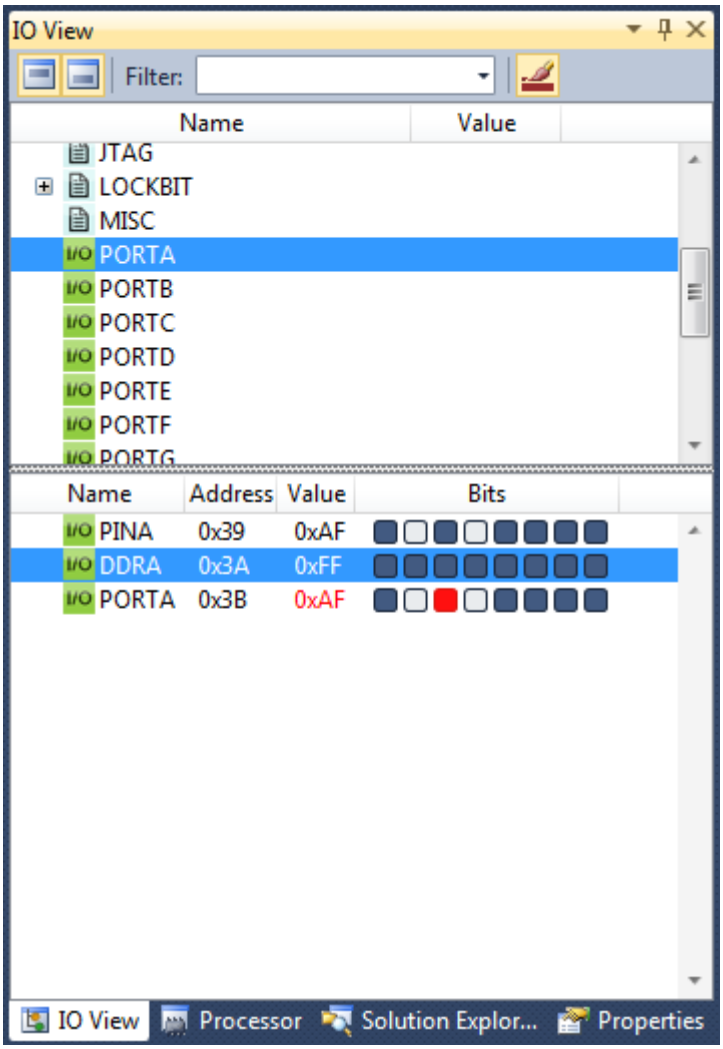
Here the assembly instruction `JMP 0x0001778A` is being assembled by the GCC Assembler, and disassembled using the built-in disassembler in Atmel Studio, which resolves the jump to `0x0000BBC5`, which is exactly half of the address in the initial assembly.

It should be noted that the addresses are always of the same dimension as the line addresses shown in the disassembly, so the code is functionally similar.

4.12. I/O View

4.12.1. About the I/O View

The purpose of the I/O View is to provide an overview of the registers of the target device for the current project. It serves as a quick reference during design, and is capable of displaying register values when the project is in debug mode. The view supports both 32- and 8-bit devices equally.



The default view of the tool window is a vertically split window with peripheral groups in the top section, and registers in the bottom section. Each peripheral typically has a set of defined settings and value enumerations, which can be displayed by expanding a register in the peripheral view (top section). The register view (bottom section) will display all registers which belong to a selected peripheral group. If no peripheral is selected, the view is empty. Each register can also be expanded to display the pre-defined value groupings which belongs to the register.

4.12.2. Using the I/O View Tool

The I/O View is confined to a single tool window in the development environment. There can only be one instance of the I/O view at a time. To open the window, select **Debug → Windows → I/O View**. When in design mode, the I/O View will be disabled for inputs. It is still possible to change the layout or filters, and to navigate the view, but no values can be set or read. To read or change a value in the registers, AVR Studio must be in debug break mode (execution paused). In this mode, all the controls of the I/O View will be enabled and values can be read and updated in the view.

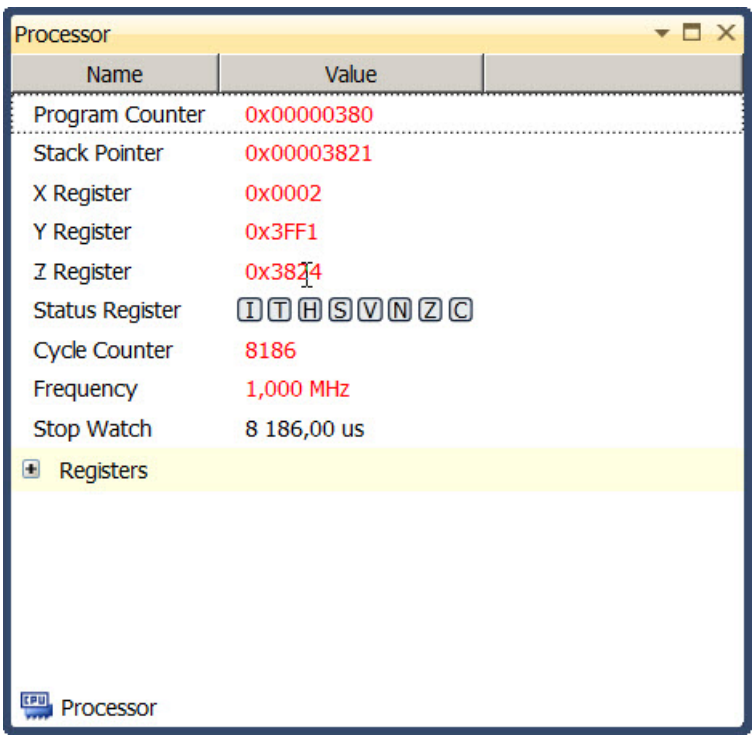
In addition to simply displaying the value of a register, the I/O View will display each bit in the register in a separate column. Bits which are set will have a dark color by default, and cleared bits will have no color (default white). To change a bit, simply click it, and the value will be toggled.

4.12.3. Editing Values and Bits in Break Mode

When the project is in debug break mode, any value can be changed by clicking the value field and writing a new value. Some values and bits cannot be modified as they are read-only, and some bits may be write-only. See the documentation for each device for more information. When a bit or value is set, it is immediately read back from the device, ensuring that the I/O View only displays actual values from the device. If a new value is set, but the I/O view does not update as expected, the register might be write-only or simply not accessible.

When a register has changed since last time it was displayed, it will indicate so with a red colored value and bits in the display. If a bit has been set since last time, it will be solid red. If it has been cleared it will simply have a red border. This feature can be toggled on or off in the toolbar.

4.13. Processor View

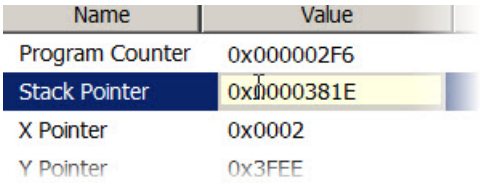


Debug → Processor View. The processor view offers a simulated or direct view of the current target device MPU or MCU. On the picture above you can see a partial list of the simulated device's ATxmega128U1 registers.

The **program counter** shows the address of the instruction being executed, the stack pointer shows the application's current stack pointer value. The X,Y, and Z registers are temporary pointers that can be used in indirect passing or retrieving arguments or objects to and from functions. **Cycle counter** counts the cycles elapsed from the simulation's start. Status register or SREG shows the currently set flags. Further on you will be able to toggle a setting for displaying the flag names.

The stop watch field allows you to make rudimentary profiling of your application. It is influenced by the frequency set in the **Frequency** field, which defines target MCU/MPU frequency, in case when the prototyping board is connected.

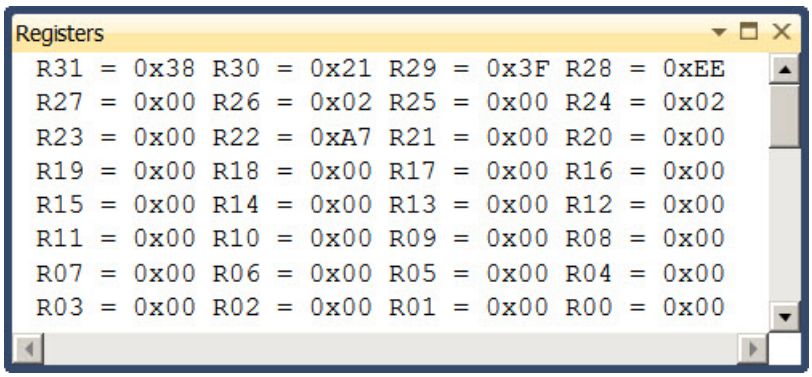
Each register can be displayed in hexadecimal, decimal, octal, and binary (flag) format by right-clicking and choosing **Display in binary**, etc., or **Display in...**. Each field can also be modified, as shown in the below image. If a field is a status or flags register, composed of a number of the one-bit flags, you can toggle individual flags by clicking on them - ☐■.



The processor view is only active in the debug mode.

4.14. Register View

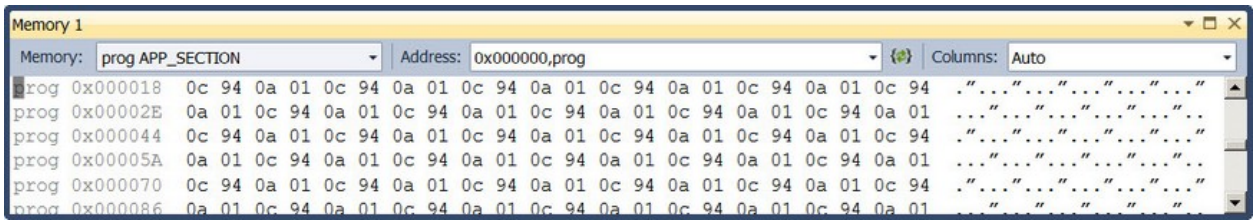
Debug → Windows → Register View or Ctrl Alt G. The register view offers a simple way to see the data and system registers of your target or simulated device. You cannot modify the registers' contents from the Register view.



4.15. Memory View

Debug → Windows → Memory view, or Ctrl Alt M n where n is the memory's number. The memory view gives you an outline of the memory. It is possible to select among the attached memories to see all the

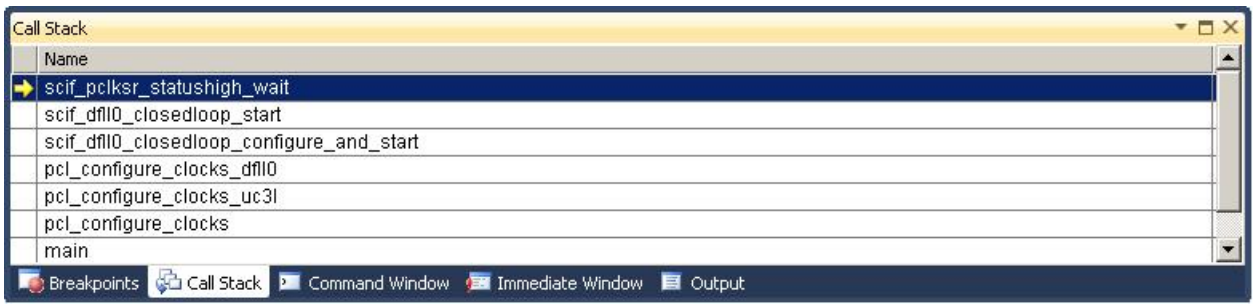
segments by switching between them in a **Segment** drop-down menu on top of the memory view. You can also specify the starting address for the memory view window in the **Address** form field on top of the memory view. In order to specify the address you can use either a normal hexadecimal entry or an expression. See Expression Formatting. The **Columns** drop-down menu allows you to specify how many byte-aligned memory columns you wish to see at one time, most often this should be left at **Auto** setting, but if you have to manually check a fixed-length type values and you know how many words or bytes those values occupy, you could align the memory view so that each row will correspond to a desired number of values.



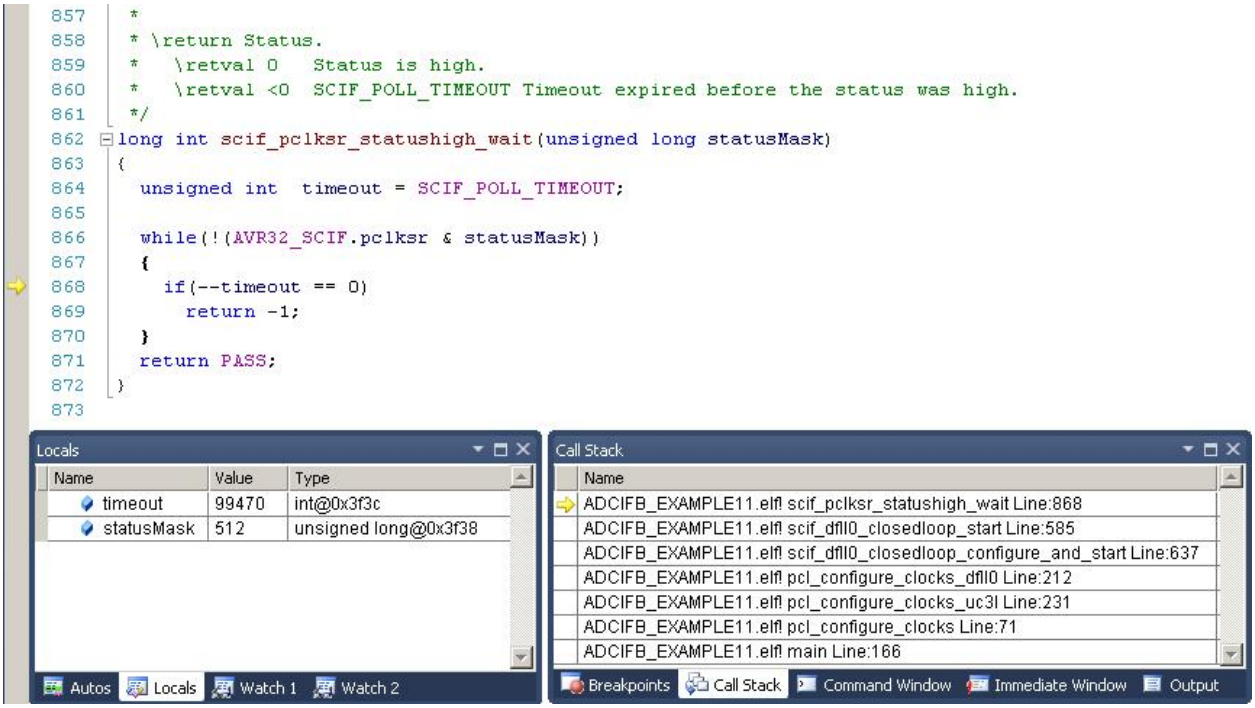
4.16. Call Stack Window

Note: Call Stack Window is currently only supported for 32-bit devices.

Call stack shows the hierarchical information of callers of the current method. By default, the Call Stack window displays the name of each function. To display Call Stack, Click the menu, Debug → Windows → Call Stack.



Along with function name, optional information such as module name, line number, etc. may also be displayed. The display of this optional information can be turned ON or OFF. To switch ON/OFF the optional information displayed, Right-click the Call Stack window and select or deselect Show <the information that you want>

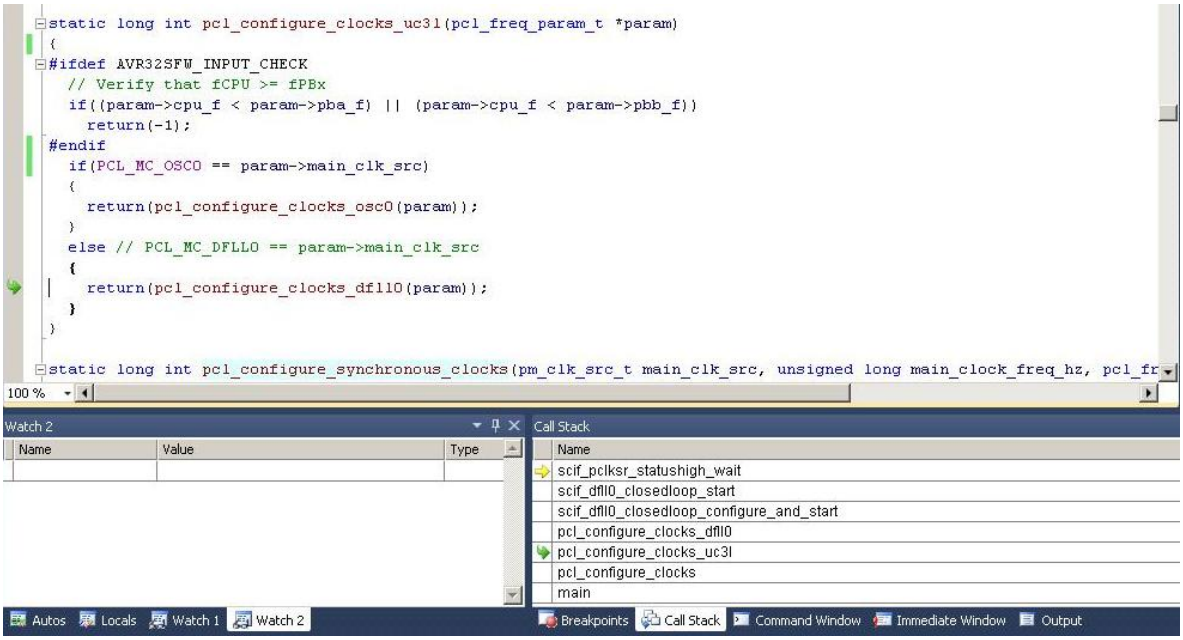


Stack frame of the current execution pointer is indicated by an yellow arrow. By default, this is the frame whose information appears in the source, Disassembly, Locals, Watch, and Auto windows. The stack frame context can also be changed to be another frame displayed in the Call Stack window.

Warning: Call Stack may not show all the call frames with Optimization levels -O1 and higher.

To switch to another stack frame

1. In the Call Stack window, right-click the frame whose code and data you want to view.
2. Select Switch to Frame.
   A green arrow with a curly tail indicates the changed stack context. The execution pointer remains in the original frame, which is still marked with the yellow arrow. If you select Step or Continue from the Debug menu, the execution will be continued from the yellow arrow, not the frame you selected.

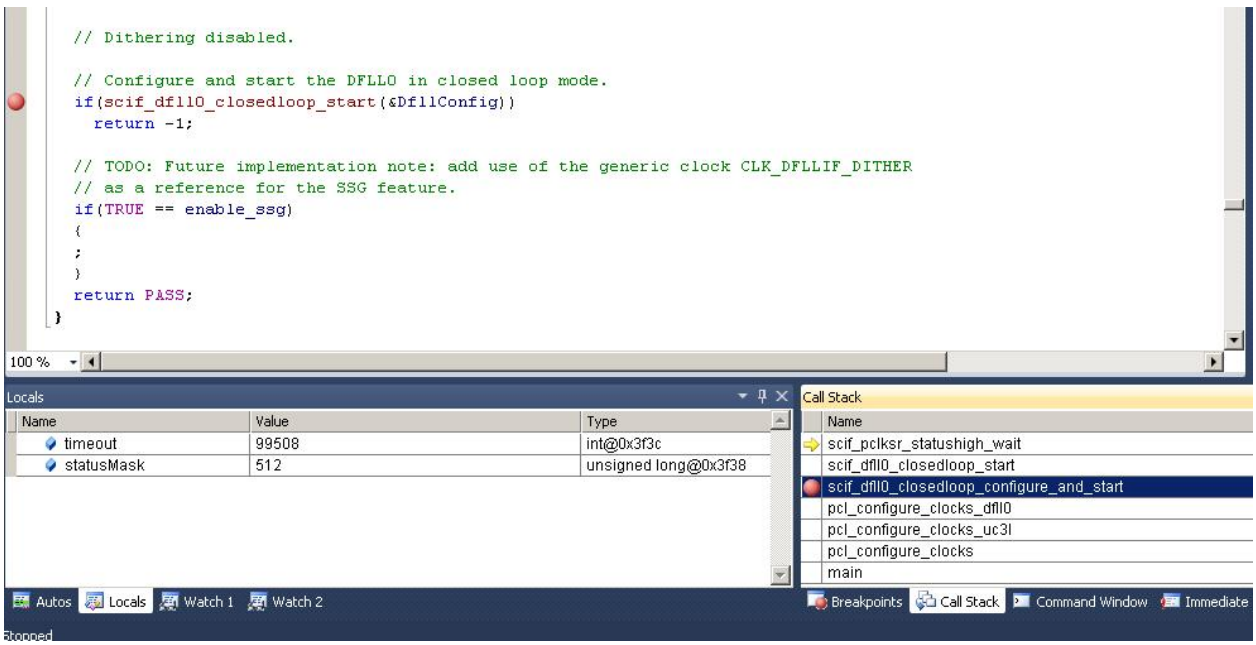


To view the source/disassembly code for a function on the call stack

1. In the Call Stack window, right-click the function whose source code you want to see and select Go To Source Code.
2. In the Call Stack window, right-click the function whose disassembly code you want to see and select Go To Disassembly.

To set a breakpoint on the exit point of a function call

In the Call Stack window, right-click the stack frame to which you would like to add the breakpoint. Select "BreakPoint → Insert Breakpoint" to add the breakpoint.



4.17. Object File Formats

While Atmel Studio uses ELF/DWARF as the preferred object file-/debug info-format, You may debug other formats. Several object file formats from various compiler vendors are supported. You can open and debug these files, but you may not be able to edit the code from within Atmel Studio. Using your own editor to edit code and Atmel Studio to debug (use the reload button), you will still have a powerful code/ debug environment.

All external debug sessions require you to load an object file supported by Atmel Studio. An object file for debugging usually contains symbolic information which is not included in a release file. The debug information enables Atmel Studio to give extended possibilities when debugging, e.g. Source file stepping and breakpoints set in high level language like C.

Precompiled object files can be opened by using the menu command **Open file → Open Object File for Debugging**. See section Debug Object File in Atmel Studio for more info.

Table 4-5. Object File Formats Supported by Atmel Studio

Object file format	Extension	Description
UBROF	.d90	UBROF is an IAR proprietary format. The debug output file contains a complete set of debug information and symbols to support all type of watches. UBROF8 and earlier versions are supported. This is the default output format of IAR EW 2.29 and earlier versions. See below how to force IAR EW 3.10 and later versions to generate UBROF8.
ELF/DWARF	.elf	ELF/DWARF debug information is an open standard. The debug format supports a complete set of debug information and symbols to support all types of watches. The version of the format read by Atmel Studio is DWARF2. AVR-GCC versions configured for DWARF2 output can generate this format.
AVRCOFF	.cof	COFF is an open standard intended for 3rd party vendors creating extensions or tools supported by the Atmel Studio.
AVR Assembler format	.obj	The AVR assembler output file format contains source file info for source stepping. It is an Atmel internal format only. The `.map` file are automatically parsed to get some watch information.

Before debugging, make sure you have set up your external compiler/assembler to generate an object file with debug information in one of the formats above.

3rd party compiler vendors should output the ELF/DWARF object file format to ensure support in Atmel Studio. Optionally you could provide an extension to have both debugging and compile support. See Contact Information for more information.

Tip:

How to generate AVR-compatible ELF file in IARW32:

In the **Project options → Output format** dialog choose **elf/dwarf**, and in the **Project options → Format variant** select **ARM-compatible "-yes"**.

Tip:

How to force IAR EW 3.10 and later versions to generate UBROF8:

By default IAR EW 3.10 and later versions output UBROF9. Currently, Atmel Studio cannot read this format. To force the debug format to UBROF8 open the project options dialog and change the **Output format** setting to **ubrof 8 (forced)**. Note that the default file name extension is changed from '`.d90`' to '`.dbg`' when selecting this option. To keep the '`.d90`' extension, click the **Override default** check button and change the extension.

4.18. Trace

In Atmel Studio, trace is provided on a plug-in basis. This means that different plugins separate from the core of Atmel Studio will be the provider of the different graphics view to visualize trace.

In the realm of trace, there are some terminology that describe the different trace sources that a device and tool combination supports. These high level source names are mapped to different architecture specific trace sources.

The following sections will describe some of the high level trace sources that might be available, and how it is mapped to the target architecture. Only a high level description of the different sources will be given, as the device specific details are available in the respective datasheet.

Note: The architecture for discovering trace capabilities in Atmel Studio is based on what the chip itself reports. This means that a debug session needs to be running so that the capabilities can be probed. This means that when activating a trace source, Atmel Studio might fail if the device does not support the source that was asked for during launch.

4.18.1. Application Output

Application output is a common name for a technique that provides what is known as a stimuli port. This implies some mean for the application running on the device to output data to a debugger that is connected.

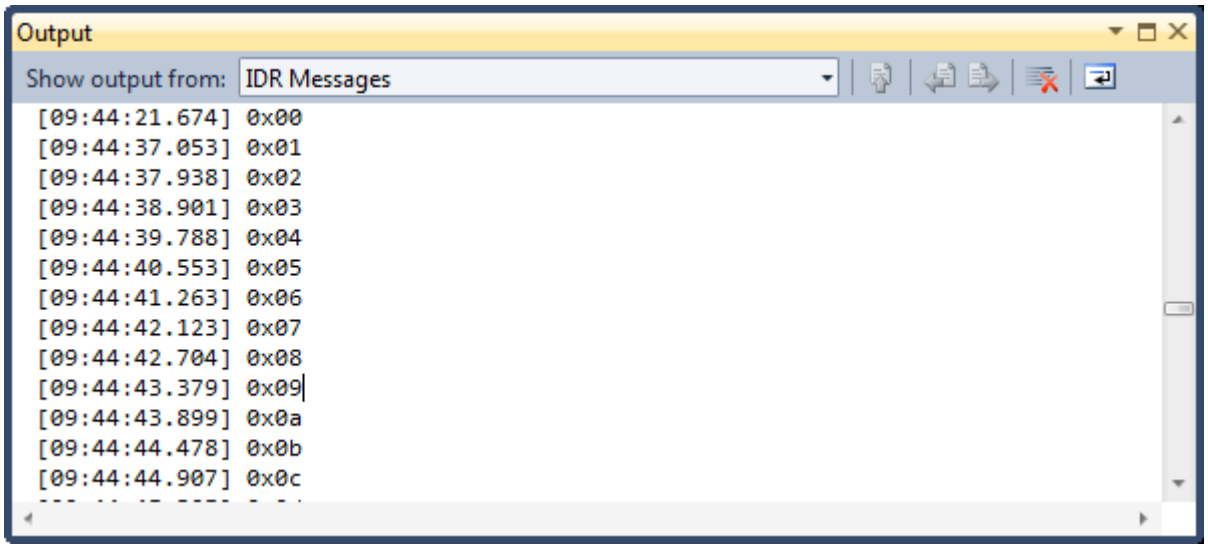
4.18.1.1. ITM

ITM is an optional part of the debug system on ARM cores. The module provides a set of registers that an application can write data to, that will be streamed out to the debugger.

4.18.1.2. IDR Events

When the application program writes a byte of data to the `OCDR` register of an AVR device while being debugged, the debugger reads this value out and displays it as IDR events in the output window as shown in the figure below. The `OCDR` register is polled at a given interval, so writing to it at a higher frequency than the one specified for the debugger will not yield reliable results. The datasheet of the device will explain how to check that a given value has been read.

Figure 4-9. IDR Events



Note that the Output window does not have the “IDR Messages” drop down if no IDR events have been sent from the debugger.

4.18.2. Program Counter Sampling

This trace source involves some sort of sample system that reads out the program counter of the device periodically. This can then be used to graph where execution time is being spent, based on some statistical average.

4.18.2.1. ARM Implementations

There are two ways of sampling the program counter on an ARM Cortex core. The first is using an optional module in the debug system that emits the program counter to the debugger without any impact on the core itself. The program counter is emitted on the SWO pin.

As not all Cortex implementation can emit the program counter, Atmel Studio also supports doing a periodical readout of the program counter while to core is running. This is possible as most Cortex devices supports readout of memory while the core is running, with a small impact on the running application as the debug system needs to access the memory bus.

4.18.2.2. AVR 32-bit Implementation

Reading the program counter on the AVR 32-bit core is possible in the same way as mentioned in ARM Implementations, as the core supports live readout of memory while the core is running.

4.18.3. Variable Watching

Watching variables are usually covered by data breakpoints, see Data Breakpoints. However, on some systems it is also possible to make a data breakpoint emit the information to the debugger without halting the core, meaning that it is possible to watch variables in applications that for instance has some sort of external timing requirement that a data breakpoint would cause to fail.

4.18.3.1. ARM Implementations

Data breakpoints on a Cortex core can be changed to emit a trace packet if the debug system implements the needed modules. This means that it is possible to get information about reads or writes to a specific memory location without an interference with the execution on the core.

As a fallback to this, it is also possible to read a memory location at a given interval while the core is executing. This will not be any specific event data, but means that if the core supports live memory readout, it is possible to sample some parts of the memory. This has a minor impact on the execution, as the debug system needs to have access to the memory bus.

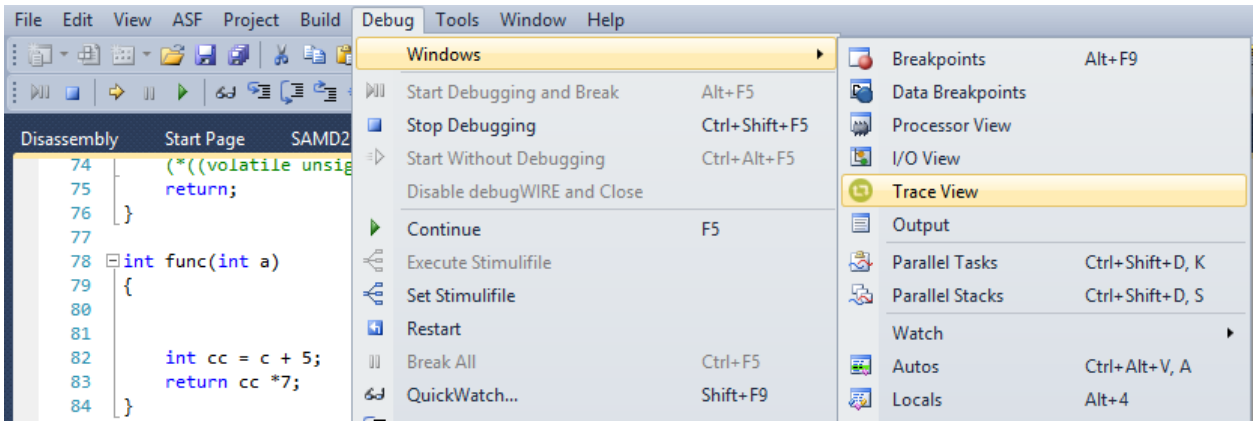
4.18.3.2. AVR 32-bit Implementation

The AVR 32-bit core also supports live readout of memory locations. This means that Atmel Studio can poll a memory location while to core is executing, giving a statistical view of how the variable is changing over time.

4.19. Trace View

The Trace View allows you to record the program counter trace when a target is running. The program counter branches are mapped with the respective source line information. It also contains coverage and statistics for the source lines executed. To open the Trace View, go to **Debug → Windows → Trace View**. To use the functionality of the Trace View, a project has to be opened in Atmel Studio.

Figure 4-10. Opening Trace View From the Menu



4.19.1. Trace View Options

The Trace View toolbar has the following elements:

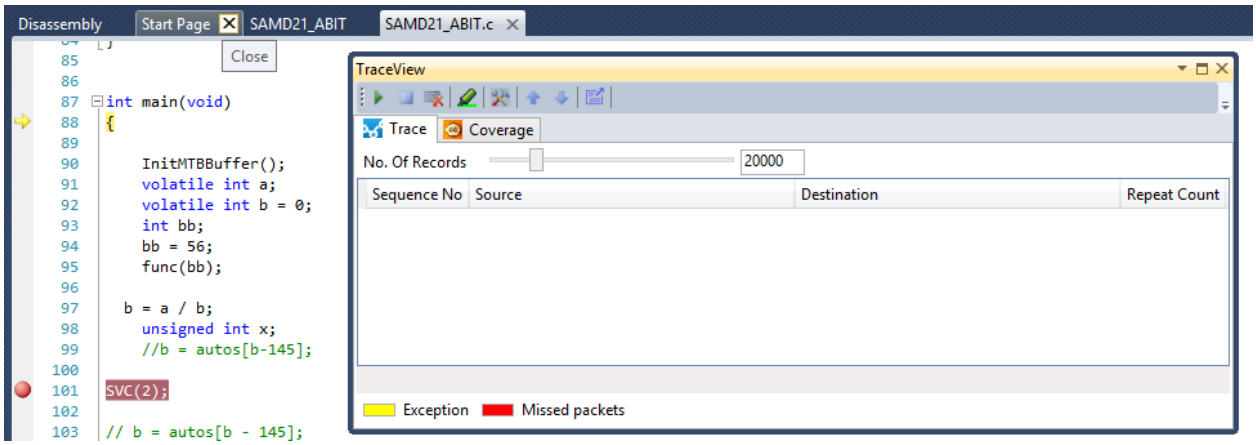
- - Starts the program trace
- - Stops the program trace
- - Clears the program trace
- - Toggles the highlighting of source code
- - Configures the device to record the program trace
- - Finds the exception record in the Trace Stack view
- - Exports coverage statistics into an xml/xslt report

4.19.1.1. Starting the Program Trace

The Program Trace can be started by clicking the play button in the Trace View Window. The start button is enabled during debug. Trace can be started and stopped any number of times in a debug session. Starting a new trace session clears all trace information of the previous trace session.

Note: A region of SRAM has to be allocated to let the device record the trace. Refer to Trace View Settings for more information.

Figure 4-11. Trace View Window



4.19.1.2. Stop Trace

Trace can be stopped in run or debug mode. The trace session will be ended automatically without user intervention when the debugging session ends.

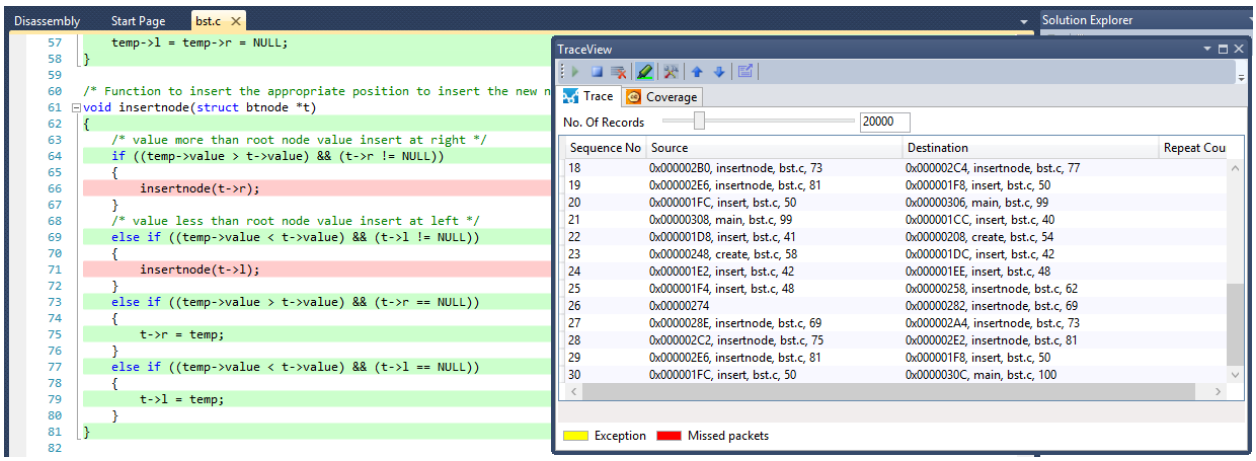
4.19.1.3. Clear Trace

The clear button clears the trace in the Trace View Window. New trace information will be logged in the same tool window with continuous sequence number. Clear can be done any number of times within a trace session. Once cleared the trace data cannot be recovered.

4.19.1.4. Highlight Source Code

Highlight is a toggle button which toggles between highlighting and non-highlighting of the source code. The source code that are covered are highlighted with a green color and remaining source lines with a red color representing the uncovered source code for the current execution.

Figure 4-12. Code Highlight



Note: Only the compilable lines are taken into consideration. For example, lines with comments and variable declaration are ignored.

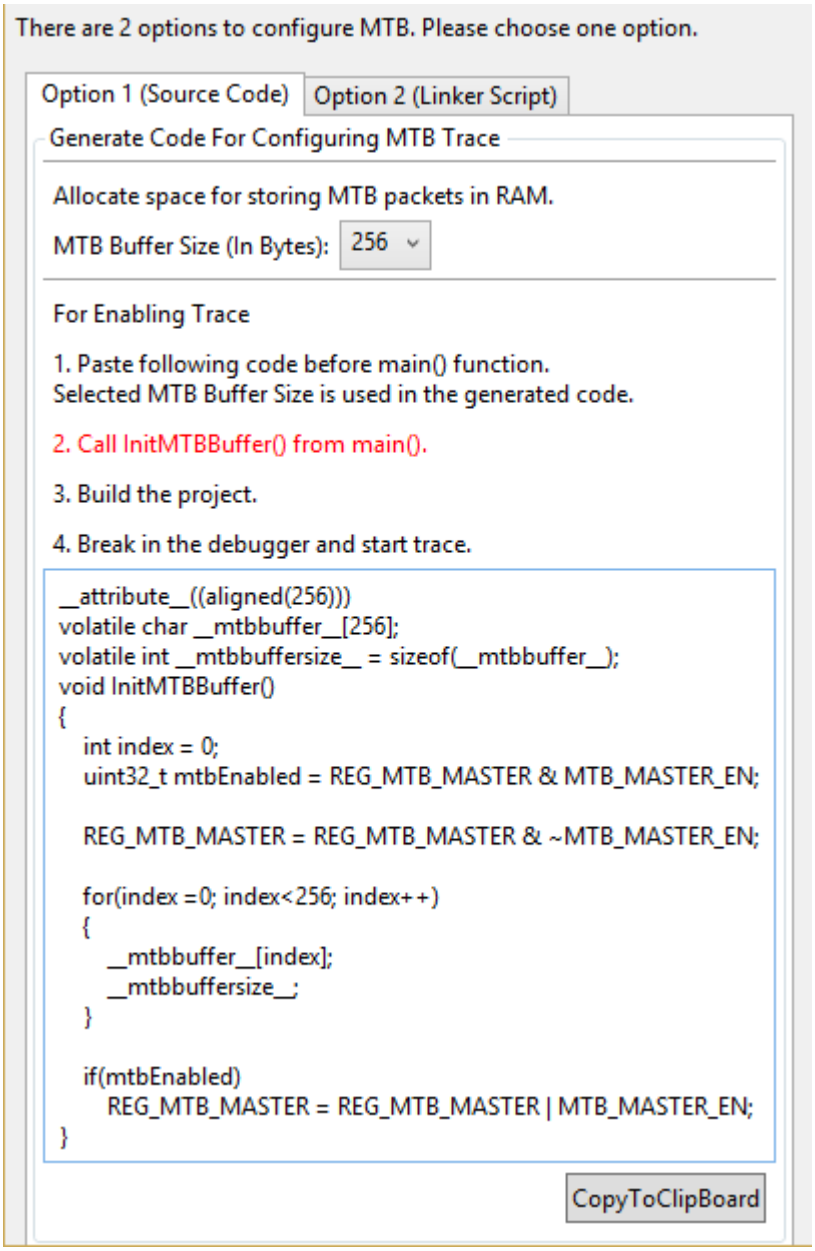
4.19.1.5. Trace View Settings

The device has to be configured to record the trace information in SRAM. The allocated size for recording the program trace can be configured from this setting. The memory can be allocated in:

- Source code, allocating a global array
- Linker scripts, reserving an amount of the memory map

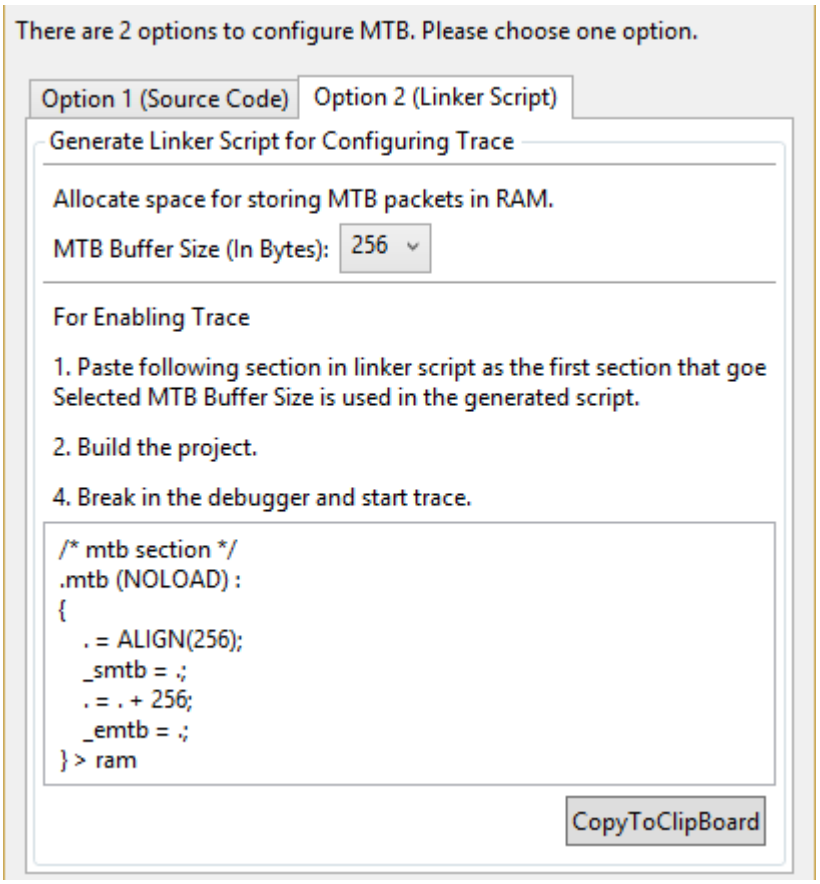
Select the memory size to be allocated for recording the program trace. Copy the snippet of code displayed in configuration window by clicking on the **CopyToClipBoard** button and paste into your source code. Follow the instructions given in the dialog to enable the tracing capability.

Figure 4-13. Trace Settings Window Through Source Code



Note: If both the linker script and the source code is configured for trace, the linker script settings takes the higher precedence over source code settings.

Figure 4-14. Trace Settings Window through Linker Script



4.19.2. Trace View Interpretation

The Trace View window contains the following items:

- Trace Stack View
- Coverage View

4.19.2.1. Trace Stack View

Trace Stack View is populated with program trace information while the target is in a running or debugging state.

Trace Stack View contains a sequence number, source and destination address, and a repeat count. A new program trace record is shown when a branching instruction happens on the target, for example as a function call or a return from a function.

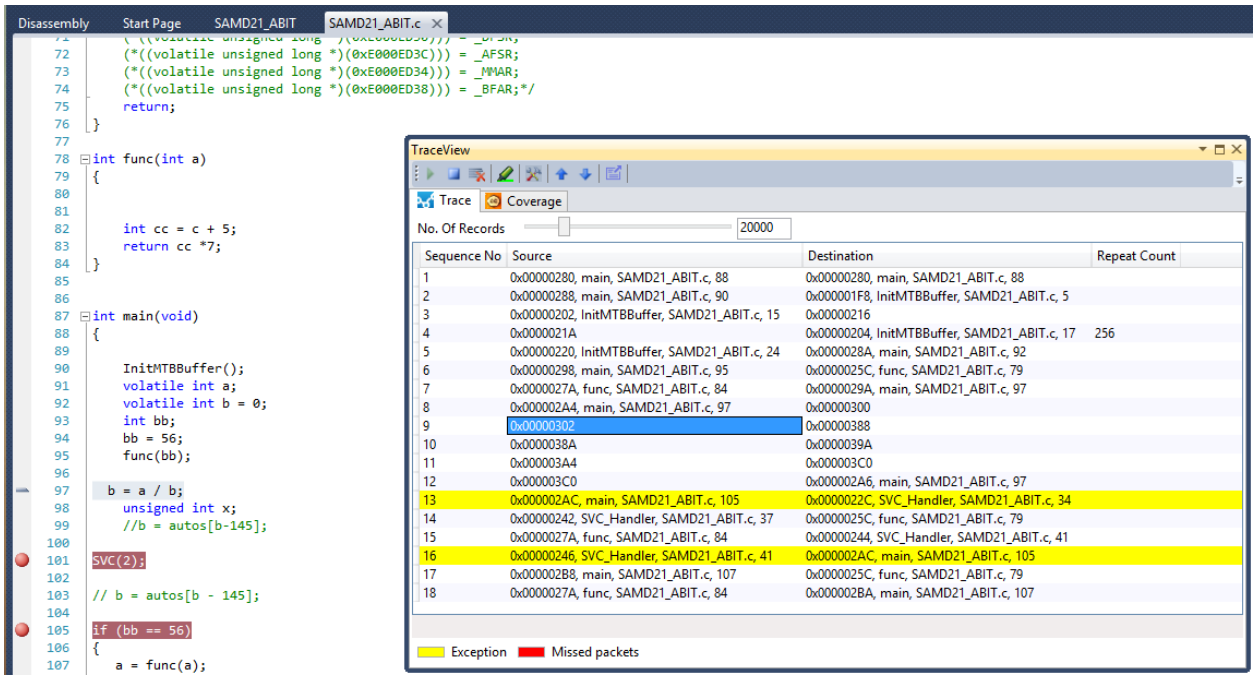
- Sequence Number - Number that keeps track of the order of the record. This number is reset for every trace session. This number will be continued without break when trace is cleared using clear button from the toolbar.
- Source - Represents the instruction/source line from which the branch happened. For example, in a function call, Source is the source line from where the function is being called.
- Destination - Represents the instruction/source line to which the branching happened. For example, in a function call, Destination is the source code of the starting line of the function.
- Repeat Count - Represents the number of times the same source and destination combination occurred consecutively. For example, if there is a delay which is logging the same packets, it will be grouped together and number of times record occurrence is termed as repeat count

The source and destination contains instruction address, function name, source file name, and line number. If the source line cannot be mapped only the instruction address is given. Double click on the source or destination navigates the cursor in the editor to the appropriate line. Navigation keys Up, Down, Left, Right, and Tab can be used to locate the source/disassembly view for the trace records.

The program trace that is shown in the TraceStack view are cut down to the latest 20000 records by default. The threshold value can be changed using the slider.

The program trace records are highlighted with a yellow color when the branching instruction was not an expected one. Unexpected branches usually happens due to some exception, and the entry and exit of the exception handler is highlighted with yellow color. The branching inside the exception are not highlighted.

Figure 4-15. Exception Record



Tip:
The next and previous exception records can be easily navigated to by using the up and down arrow buttons in the trace view window tool bar.

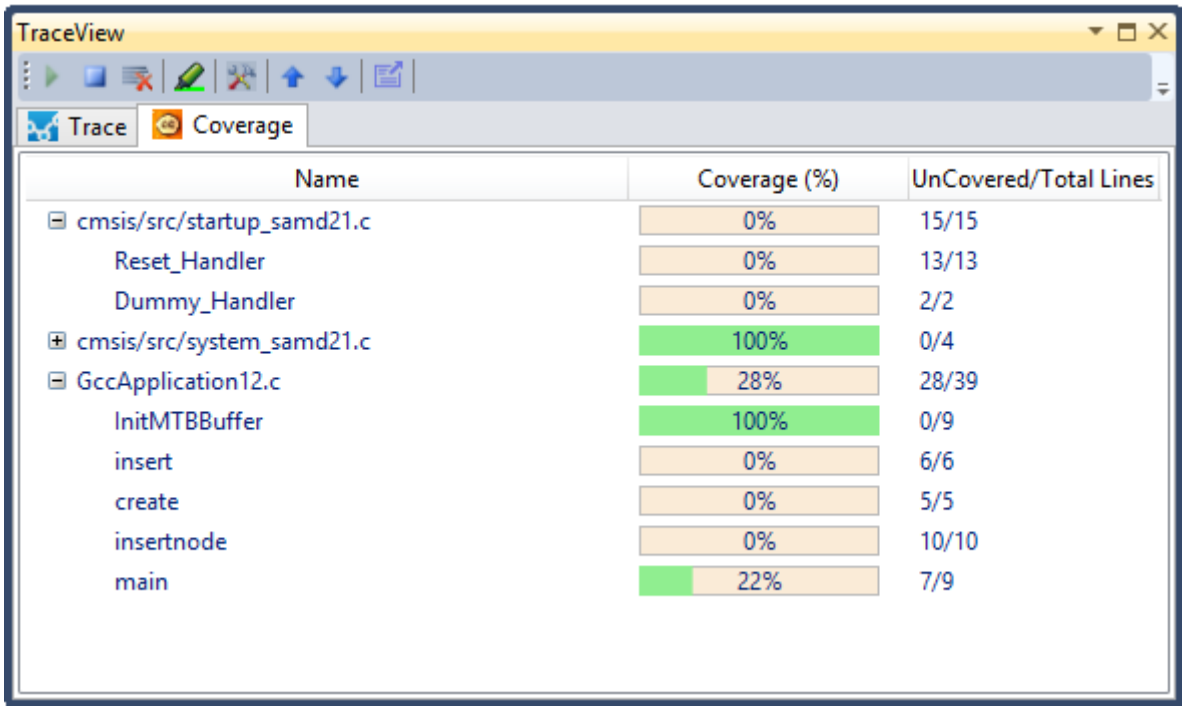
There are some exceptional cases where some program traces could be missed. In that case there will be a packet with red color which represents that there are some discontinuation of the program trace information in the sequence. Since the number of missed packets are unknown, the sequence number shown in the Trace Stack view will be continued without any break except adding a red colored packet with a sequence number for it.

Note: The disassembly view is not supported when navigation keys are used, but it is supported when the record is double clicked using mouse.

4.19.2.2. Coverage View

The coverage view shows statistics on the source covered as part of the current target execution. All the files and functions are listed in the coverage view with the information of number of lines covered or uncovered against the total number of lines in the source file.

Figure 4-16. Code Coverage



Note: Only the compilable lines are taken into consideration for the statistics. For example, lines with comments and variable declaration are not taken into account.

A coverage report can be exported. Click the export icon in the trace view toolbar to invoke the export operation.

5. Programming Dialog

5.1. Introduction

The Device Programming window (also known as the programming dialog), gives you the most low-level control of the debugging and programming tools. With it, you can program the device's different memories, fuses, and lockbits, erase memories, and write user signatures. It can also adjust some of the starter kit properties such as voltage and clock generators.

Note: If you are editing a code project in Atmel Studio and want to see the results of a compilation by downloading the code into the device, take a look at the Start without Debugging command. It is a sort of one-click programming alternative to the programming dialog. See section Start without Debugging for more information.

The programming dialog is accessible from a button on the standard toolbar or the menu **Tools** → **Device Programming**.

Figure 5-1. Device Programming Icon

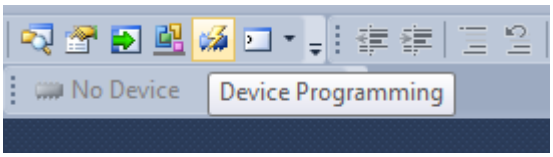


Figure 5-2. Opening Device Programming Dialog

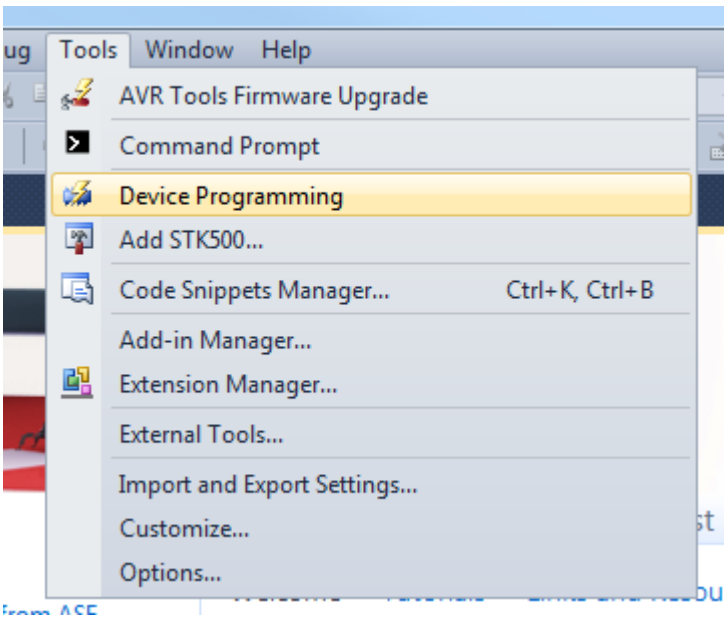
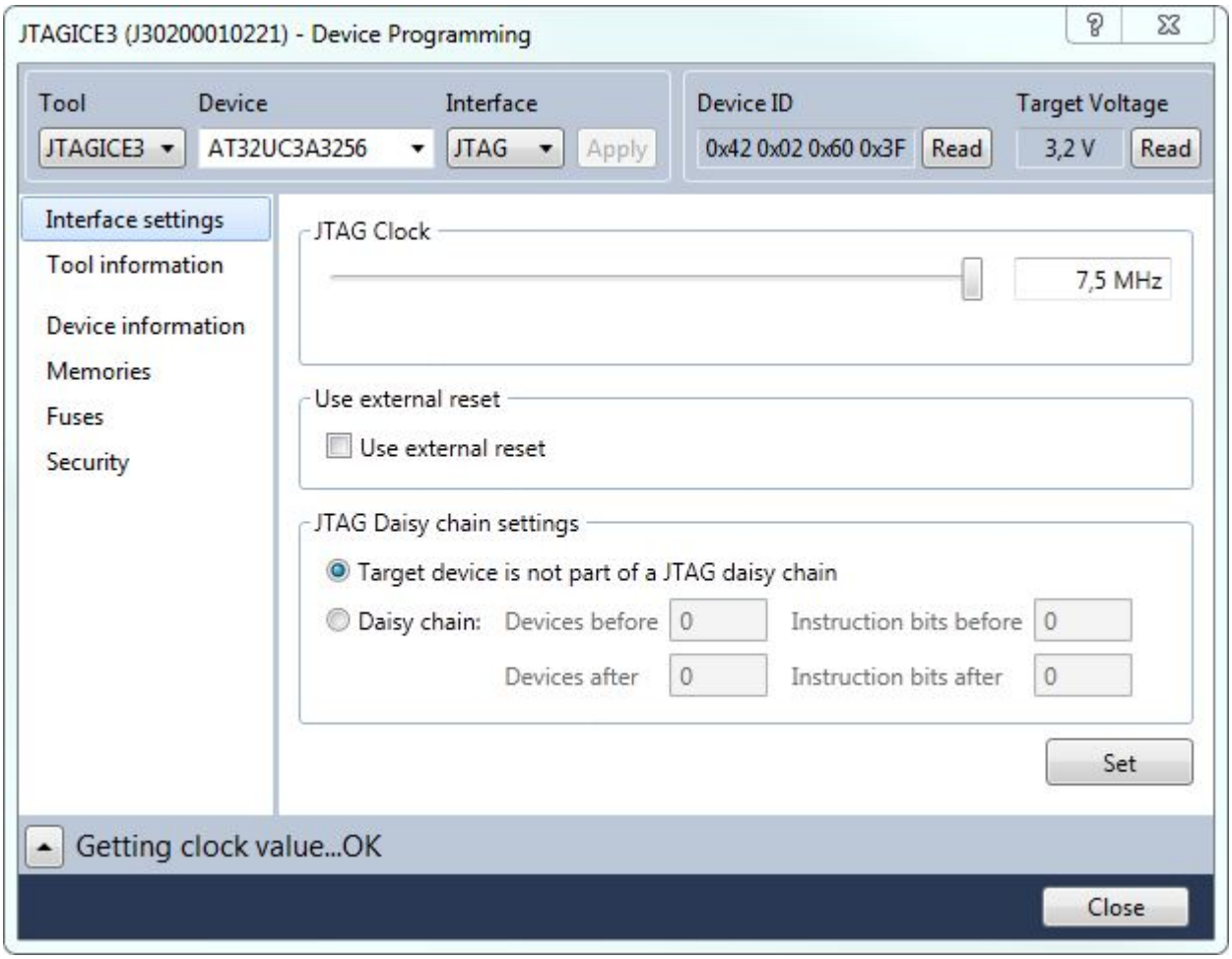


Figure 5-3. Device Programming



The programming dialog contains the following options and tabs:

Top status bar

Tool

You can choose which tool you want to use from this drop-down menu. Only tools connected to the machine are listed. Also, if a tool is used in a debug session, it will not be listed. Several tools of the same type can be connected at the same time. In order to identify them, the serial number will be shown below the tool name in the list.

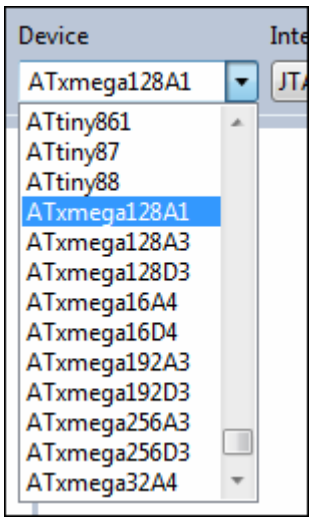
When a tool is selected, the name (and serial number) will be shown in the title bar of the Device Programming dialog.

Note: The Simulator will only offer limited support for the programming dialog features. The Simulator has no persistent memory, so you will not be able to make permanent changes to any simulated devices.

Device

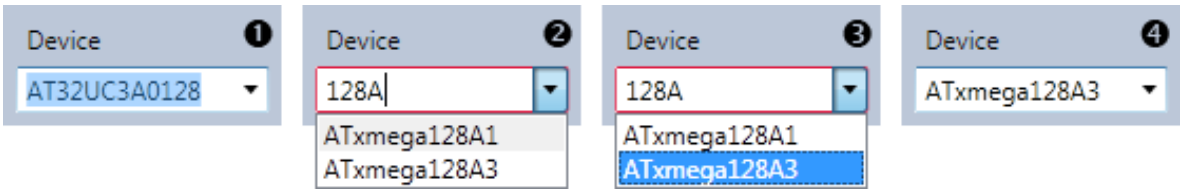
As soon as a tool is selected, the device list will show all devices supported by that tool. There are two ways to select a device:

- Select from the list



Click on the arrow. This will reveal the list of supported devices. Click to select.

- Select by typing. In this example, we will select the **ATxmega128A3**:



2.1. Double-click in the text field to select the text already present.

2.2. Start typing some part of the device's name, in this example **128A**. The list updates while you type, showing all devices containing what is typed.

2.3. Press the Arrow Down keyboard button to move the selection into the list. Use the up and down keys to navigate. Press ENTER to make a selection.

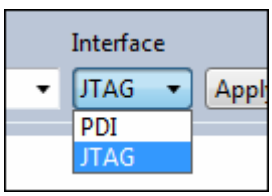
2.4. The ATxmega128A3 is now selected.

Note: A red border around the device selector indicates that the text entered is not a valid device name. Continue typing until the device name is complete, or select from the list.

Interface

When a tool and a device is selected, the interface list will show the available interfaces. Only interfaces available on both the tool and the device will appear in this menu.

Select the interface to use to program the AVR.



Apply button

When tool, device, and interface is selected, press the **Apply** button to make the selections take effect. This will establish connection to the tool. The list on the left side of the window will be updated with the relevant pages for the selected tool.

If a different tool, device, or interface is selected, the **Apply** button must be pressed again, to make the new selections take effect.

Device ID

Press the Read button to read the signature bytes from the device. The device's unique tag will appear in this field, and can be used for tool compatibility checking and to obtain help either from customer support or from the people at AVR Freaks®.

Target voltage

All tools are capable of measuring the target's operating voltage. Press the refresh button to make a new measurement.

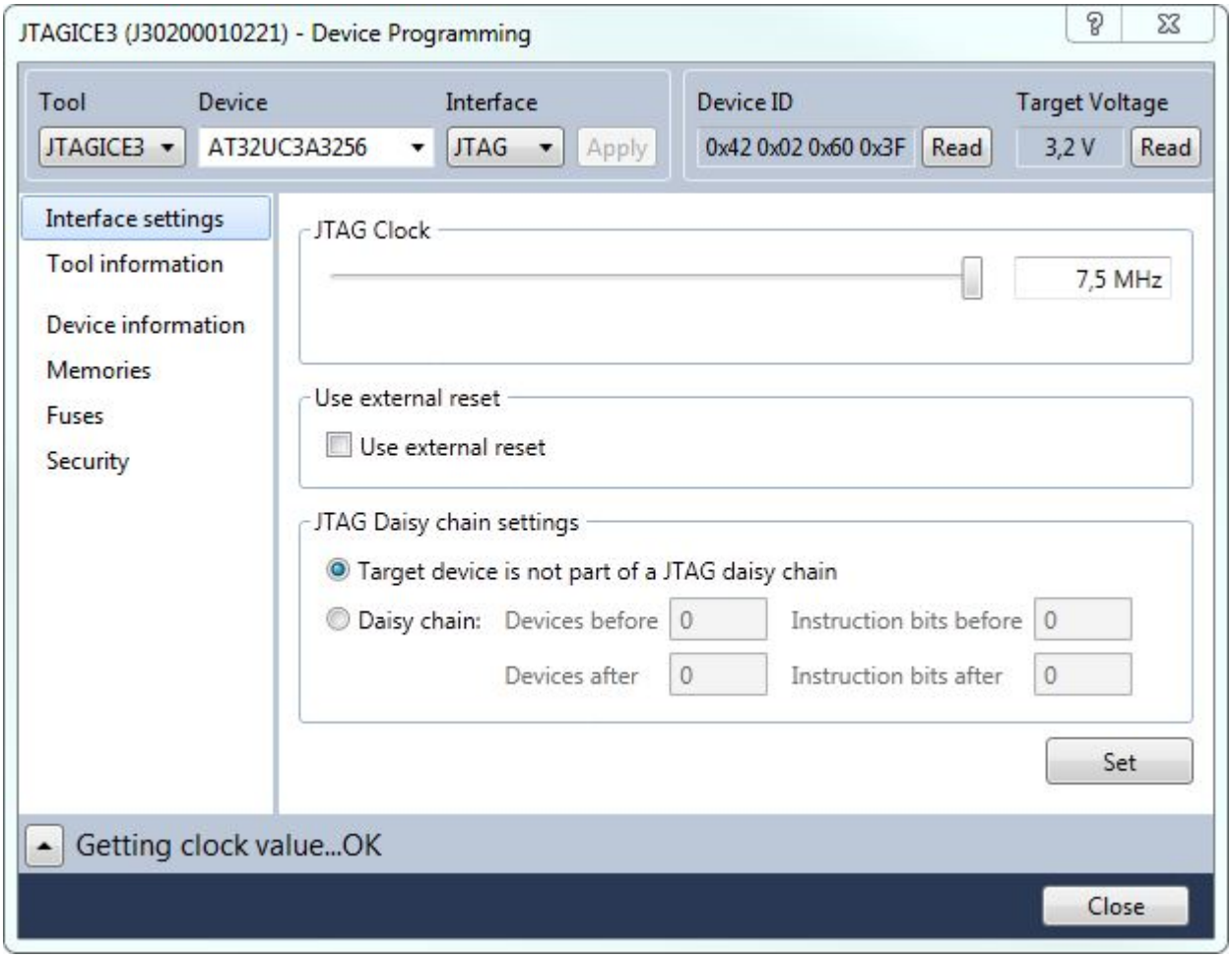
A warning message will appear if the measured voltage is outside the operating range for the selected device, and the target voltage box will turn red.

5.2. Interface Settings

The programming interfaces have different settings. Some interfaces have no settings at all, some interfaces settings are only available on some tools. This section will describe all settings, but they are not available for all tools and devices.

JTAG

If you have selected JTAG as the programming interface, clock speed, use external reset and daisy chain setting may be available. This depends on the tool and device.



JTAG Clock

JTAG clock is the maximum speed the tool will try to clock the device at. The clock range is different for different tools and devices. If there are restrictions, they will be stated in a message below the clock slider.

Use external reset

If checked the tool will pull the external reset line low when trying to connect to the device.

JTAG Daisy chain settings

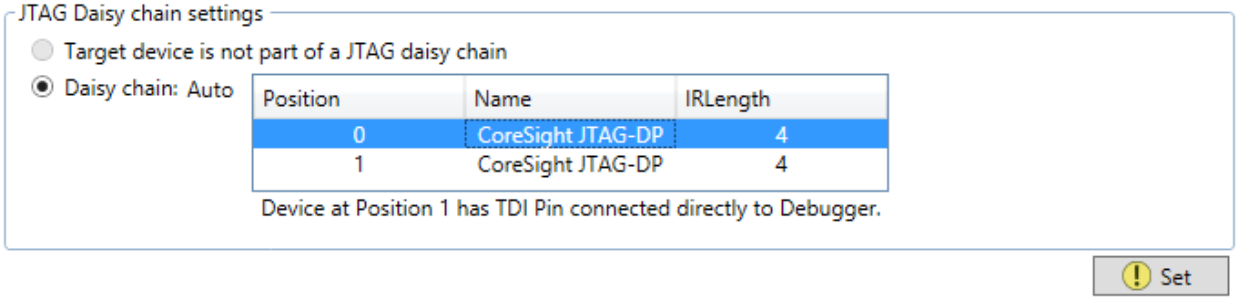
Specify the JTAG daisy chain settings relevant to the device to program.

Target is not part of a daisy chain. Select this option when the target device is not part of a daisy chain.

Daisy chain-Manual. Allows you to manually configure the JTAG daisy chain in case you are programming in a system-on-board.

- **Devices before** - specifies the number of devices preceding the target device.
- **Instruction bits before** - specifies the total size of the instruction registers of all devices, preceding the target device.
- **Devices after** - specifies the number of devices following the target device.
- **Instruction bits after** - specifies the total size of the instruction registers of all devices, following the target device.

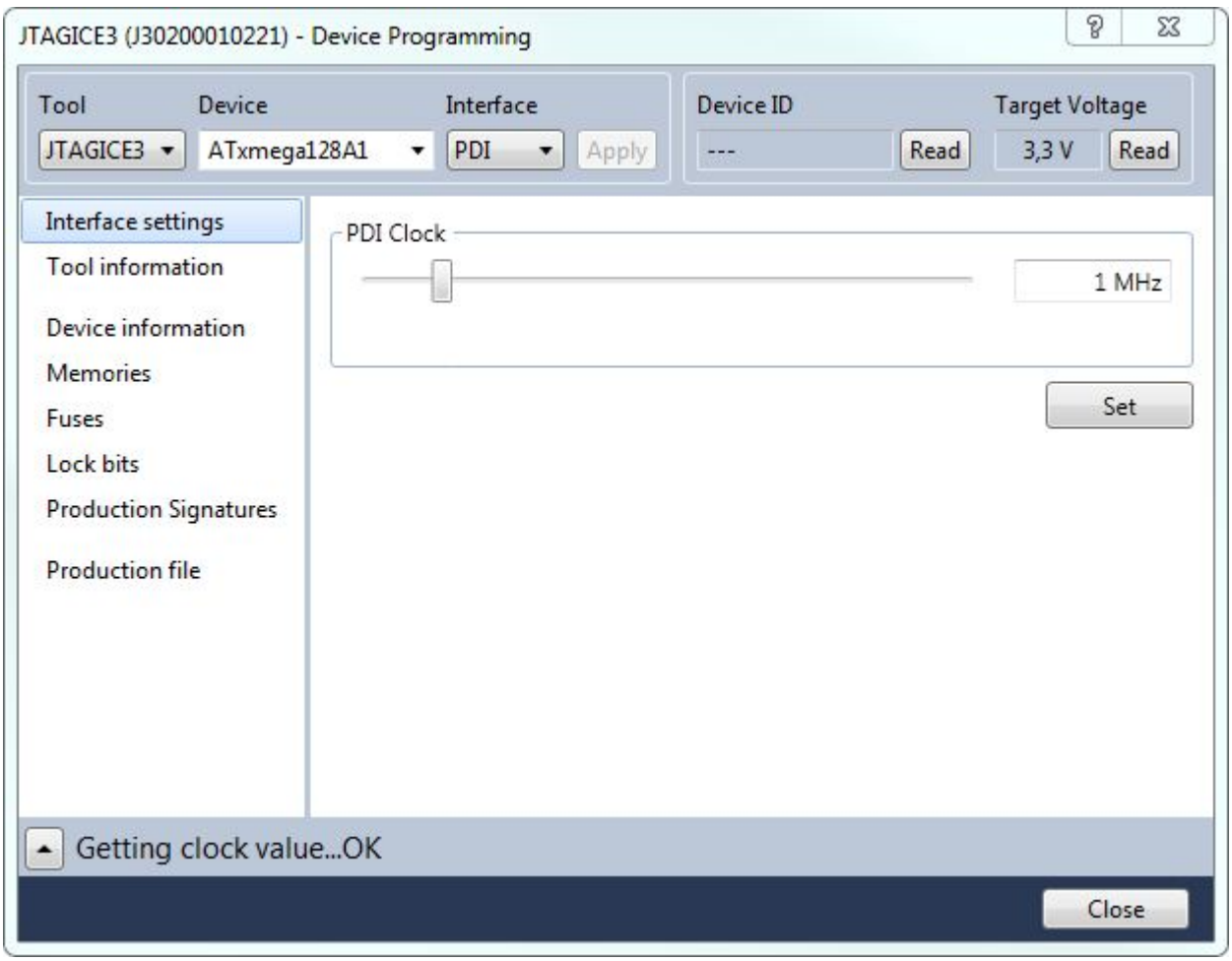
Daisy chain-Auto. Automatically detects the devices in the JTAG daisy chain. Allows you to select the device in the JTAG daisy chain. Auto-detection is supported only for SAM devices.



To accept the changes and configure the tool, press the **Set** button.

PDI

The PDI interface has only one setting – the PDI clock speed.



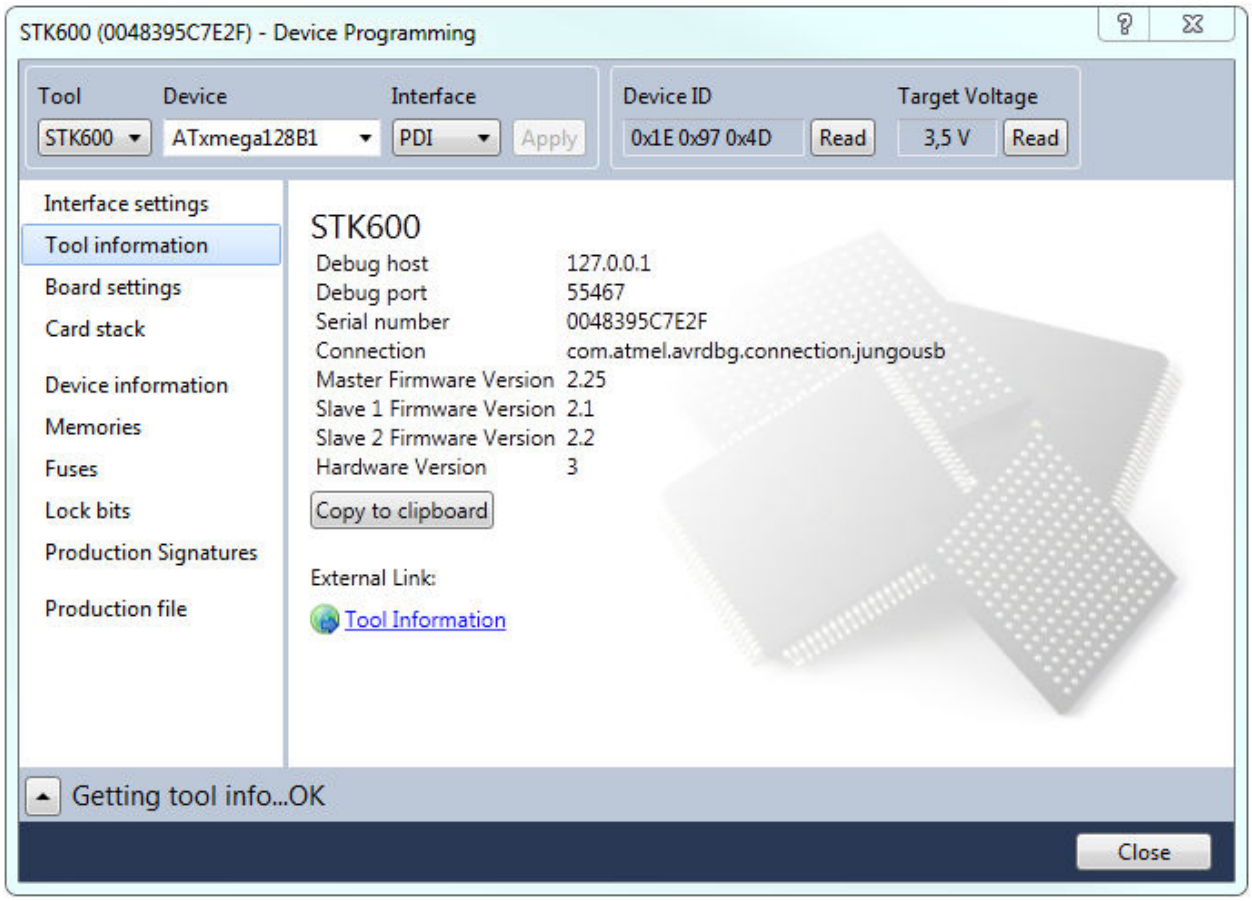
PDI Clock is the maximum speed the tool will try to clock the device at. The clock range is different for different tools and devices. If there are restrictions, they will be stated in a message below the clock slider.

To apply the changes and configure the tool, press the **Set** button.

The clock cannot be adjusted on all tools, so an empty **Interface settings** page will be presented.

5.3. Tool Information

Figure 5-4. Tool Info



The **Tool information** page contains a number of useful tool parameters.

Tool Name denotes the common name for the connected tool.

Debug Host is the debug session's host IP address for the remote debugging case. If the tool is connected to your machine, then the loopback interface IP (127.0.0.1) will show.

Debug Port is the port opened specifically for the remote debugging access to the debugging tool. The port is automatically assigned when Atmel Studio starts, and is usually 4711.

Serial number - tool serial number.

Connection - Microsoft Driver Framework Method's name used to connect the Tool on your PC.

xxx version - Firmware, hardware and FPGA file versions are listed here.

Using the link on the bottom of the dialog you can access extensive information on your tool online.

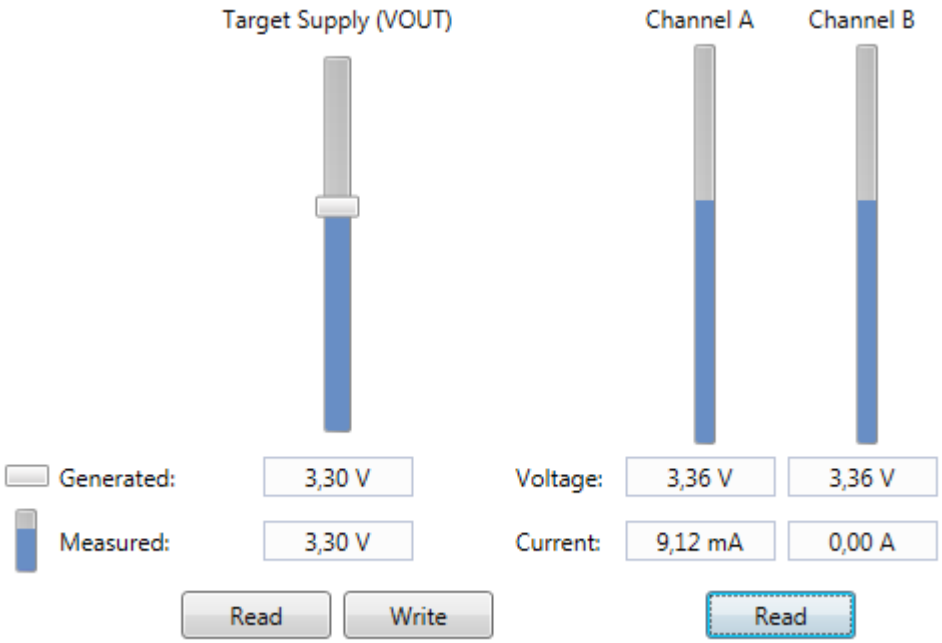
5.4. Board Settings/Tool Settings

Some tools (Power Debugger, STK500, STK600, QT600) have on-board voltage and clock generators. They can be controlled from the Board settings/Tool settings page.

5.4.1. Power Debugger

The Power Debugger has a single voltage source and two channels of voltage/current measurement.

Figure 5-5. Power Debugger Tool Settings



The voltage output (VOUT) is adjusted by the slider, or by typing a voltage in the **Generated** text boxes below the slider.

After adjusting the set-point, press the **Write** button to apply the changes. The value is then sent to the tool, and the **measured** value is read back.

Press the Read button to read both the set-point (**Generated**) and the **Measured** values from the Power Debugger.
Note: There may be slight differences between the **Generated** and the **Measured** voltages.

The output voltage range is 1.6V to 5.5V.

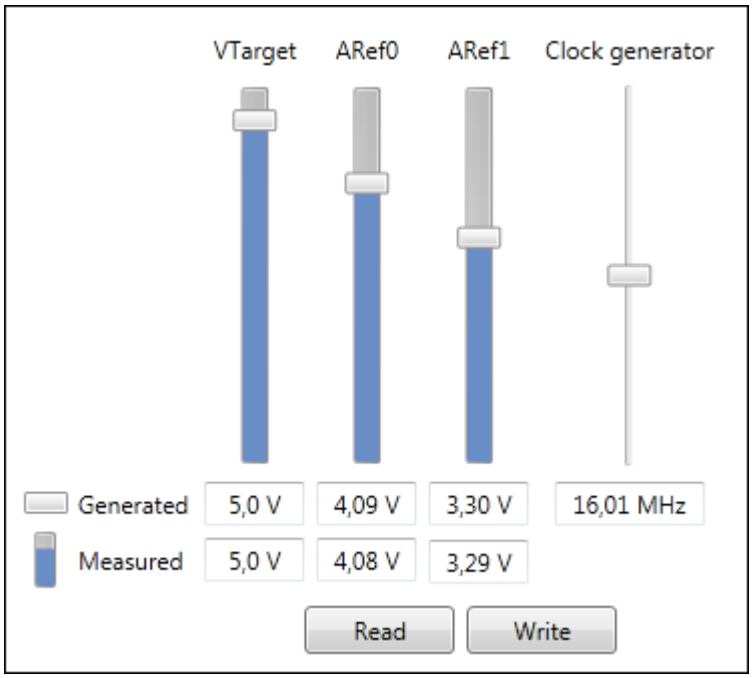
The Channel A and Channel B measurements are snapshots of analog readings taken by the Power Debugger. The tool is optimized for real-time monitoring of voltage and current, and this snapshot is thus approximate. It does not perform calibration compensation, and readings are locked in the highest-current range. For best results, use the Atmel Data Visualizer.

Note: When no load is connected to a measurement channel, non-zero measurements can be expected.

5.4.2. STK600

The STK600 has three voltage sources and one clock generator.

Figure 5-6. STK600 Board Settings



The set-points of the three voltage sources (VTG, ARef0, and ARef1) are adjusted by the means of three sliders. It is also possible to type a voltage in the **Generated** text boxes below the sliders. When you drag the sliders, the text boxes will update. And when you type a value in the text box, the slider will move.

After adjusting the set-points, press the **Write** button to apply the changes. The values are sent to the tool, and **measure** values are read back.

Measurements are shown in the **Measured** row, and shown as blue columns as part of the slider controls. The measured values cannot be edited.

Press the Read button to read both the set-point (**Generated**) and the **Measured** values from the STK600.
Note: What is the difference between the **Generated** and the **Measured** voltages? The generated voltage is the setting on the adjustable power supply, the measured voltage is the readout from the built-in volt meter. If the measured value is different from the generated voltage, this may indicate that the target circuitry draws a lot of current from the generator.

Note: If the VTARGET jumper on STK600 is not mounted, the measured voltage will be 0, unless an external voltage is applied to the VTARGET net.

The **Clock generator** is also adjusted by dragging the slider or typing into the text box below. Press the **Write** button to apply the new value.

5.4.3. QT600

The QT600 has only one setting, the VTarget voltage. This voltage can be set to five fixed voltages: 0, 1.8, 2.7, 3.3, and 5V. Press the Write button to apply the changes.

The actual VTarget value is read back automatically when pressing the **Write** button. It is also possible to read it back manually using the **Read** button.

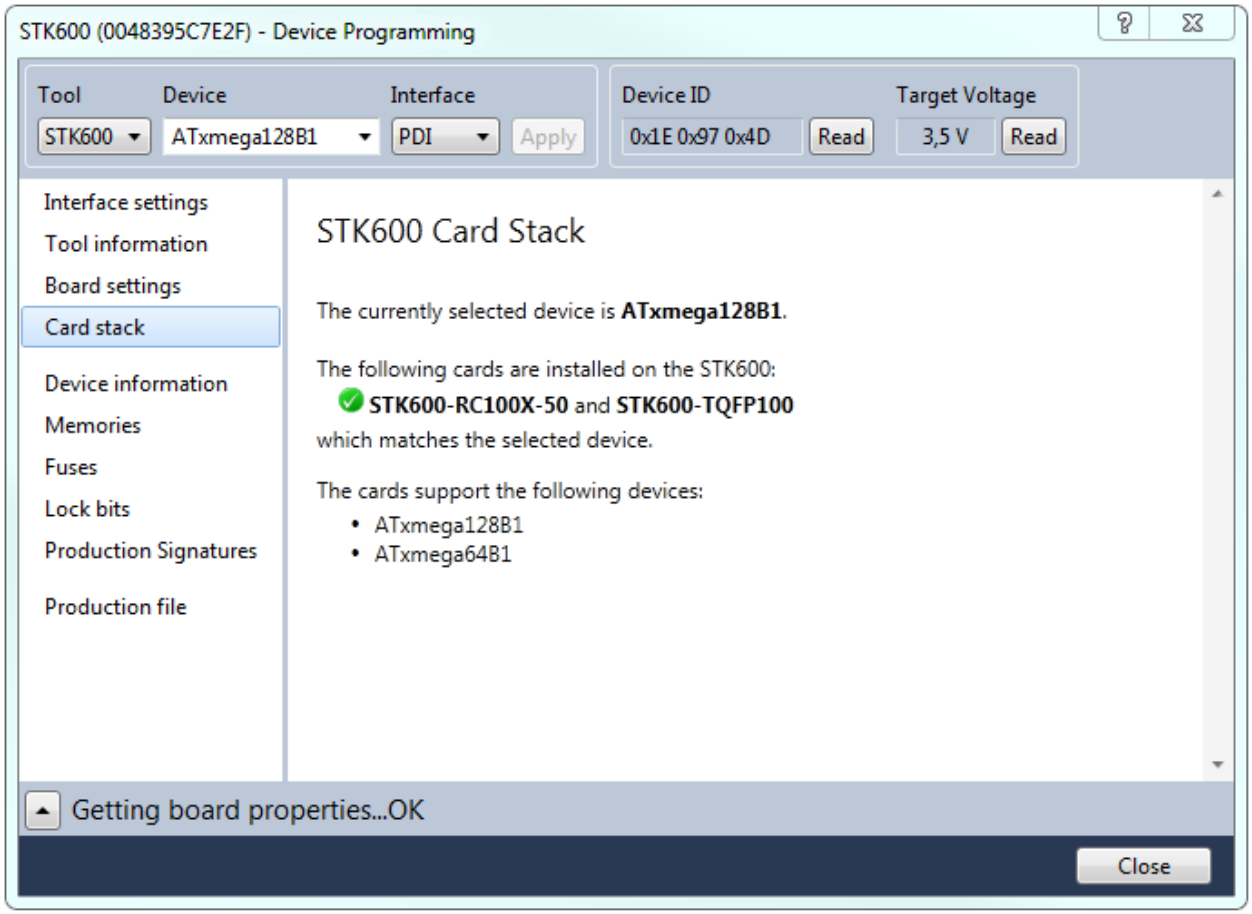
5.4.4. STK500

STK500 has settings similar to the STK600, but only one Aref voltage and combined generated/measured values.

5.5. Card Stack

The STK600 uses a combination of routing and socket cards to let all AVR devices to be mounted. Given the device, only certain combinations of routing and socket cards are valid. The Card stack page has information about this.

Figure 5-7. Card Stack

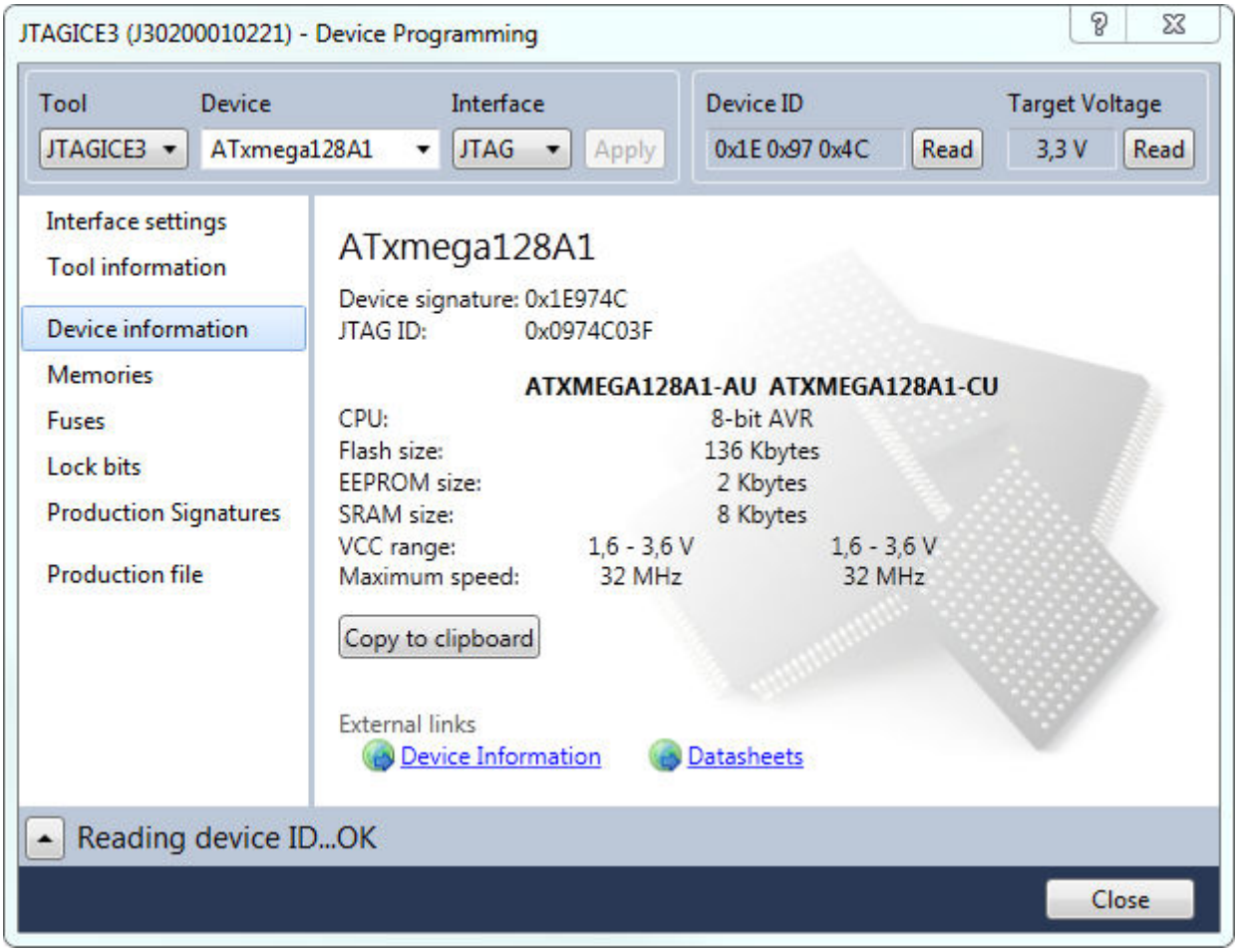


The card stack page tells which cards are mounted on the STK600 and if they support the selected device. If they do match, a list of devices supported by that card combination is listed.

If the mounted cards do not match, a list of suggested card combinations will be listed.

5.6. Device Information

Figure 5-8. Device Information



The device information page contains basic information on the selected device. When the page is accessed, it will try to read the JTAG (or device) signature from the connected device.

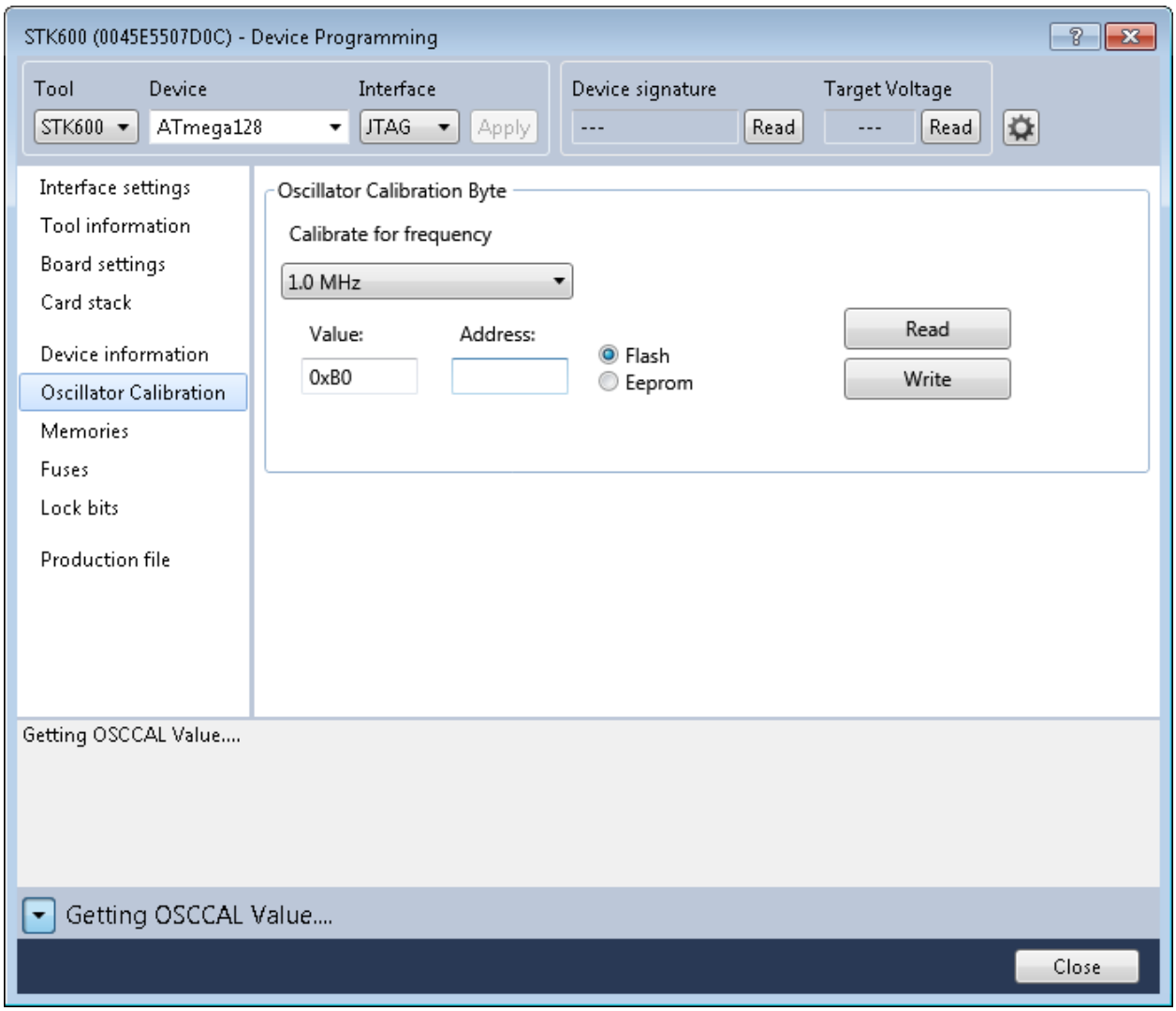
In the upper part of the dialog you can see the device name, its signature, the JTAG part identification number, and the device revision (extracted from the JTAG signature).

In the lower part of the dialog you see the device variants and characteristics of each variant. Acceptable voltage range, followed by maximum operating clock speed, and the sizes of on-chip memories.

The two links on the bottom of the dialog offer you to see a slightly more detailed device information in the purchase catalog online, or to download a complete datasheet of the target device.

5.7. Oscillator Calibration

Figure 5-9. Oscillator Calibration



Oscillator Calibration Byte(s) for ATtiny and ATmega parts

From the **Advanced** tab you can read the Oscillator Calibration Byte(s) for ATtiny and ATmega parts. The oscillator calibration byte is a value that can be written to the OSCCAL register found in selected devices, in order to tune the internal RC Oscillator to run as close to a chosen clock frequency as possible.

Program

The oscillator calibration byte is stored in the device during manufacturing and can not be erased or altered by the user. It is automatically transferred to the OSCCAL register during device start-up, or set during program initialization, depending on the device. On devices where the application sets it during program initialization, it must be transferred to FLASH or EEPROM first, using the programming dialog or the command line tools.

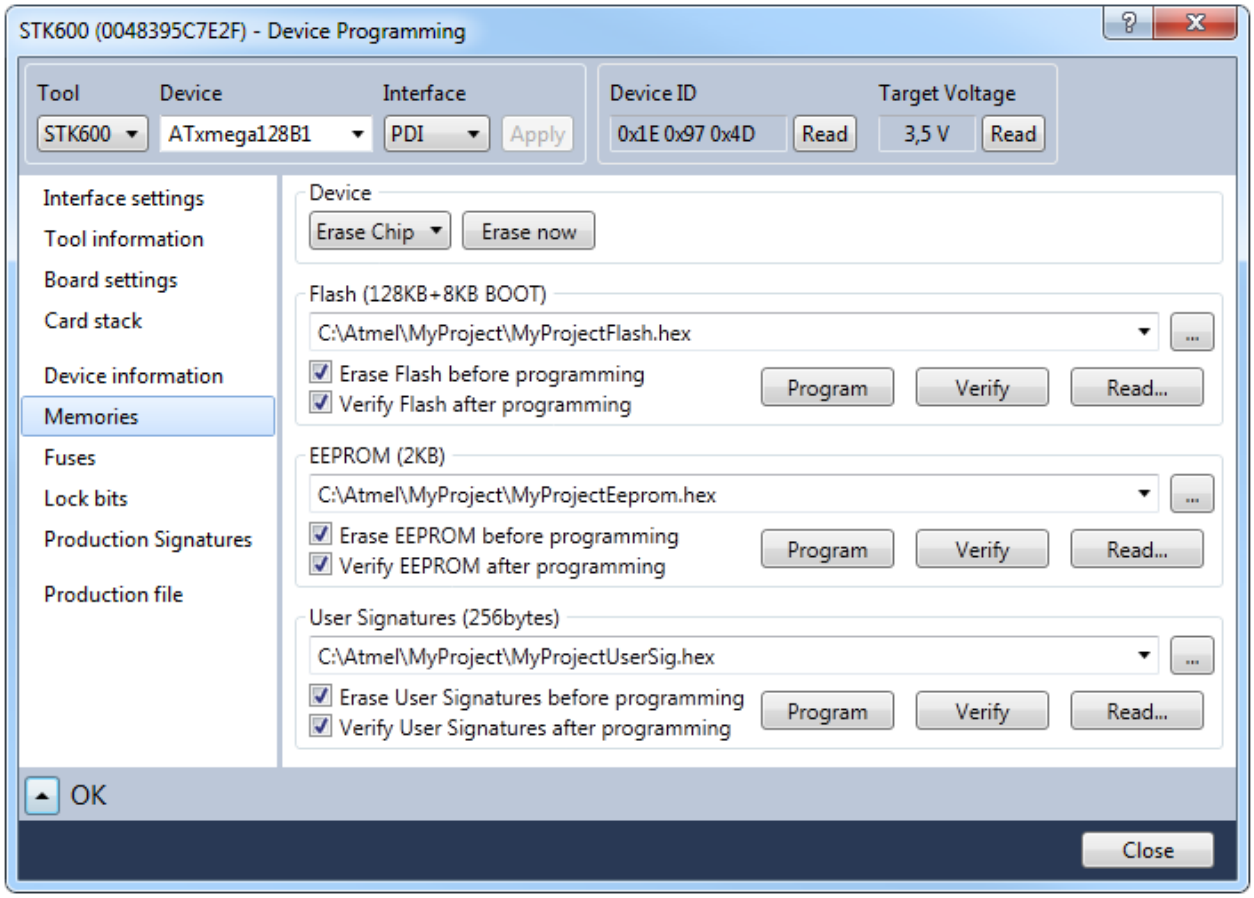
Reading and Writing the Oscillator Calibration Byte for ATtiny and ATmega parts

The calibration value is read from the storage in the device and shown in the **Value** text box by pressing the **Read** button.

The calibration byte is programmed into FLASH or EEPROM memory by pressing the **Write** button. Memory type and address must be specified first.

5.8. Memories

Figure 5-10. Memories Programming



From the **Memories** tab you can access all the programmable memories on the target device. Memory is erased by first selecting the memory type and then clicking on the **Erase** button. Selecting **Erase Chip** will erase the entire contents of the device, including FLASH, EEPROM (unless the EESAVE fuse is programmed), and lock-bits, but not Userpages if the device contains this.

Program

To program a file into the device's Flash memory, write the full path and file name in the combo box in the flash section. Or, select the file by pressing the browse button (...).

Now, press the **Program** button to program the file into the memory.

If the **Erase device before programming** check box is checked, a **chip erase operation** will be performed before the programming operation starts.

If the **Verify device after programming** check box is checked, the content will be verified after the programming operation is done.

Some devices can also be programmed through a flashloader. This is mainly an advanced technique, but it will usually give a significant speedup in the programming speed. For devices where this is supported, a

checkbox will be shown named **Program flash from RAM**. If this box is checked, the base address of the location of the flashloader needs to be given.

Verify

To verify the flash content of the device, first select the file you want to verify against. Then press the **Verify** button.

Read

The contents of the Flash memory can be read out in Intel® hexadecimal file format, using the **Read** button. Pressing the **Read** button will bring up a dialog offering you to specify where the file will be saved.

EEPROM

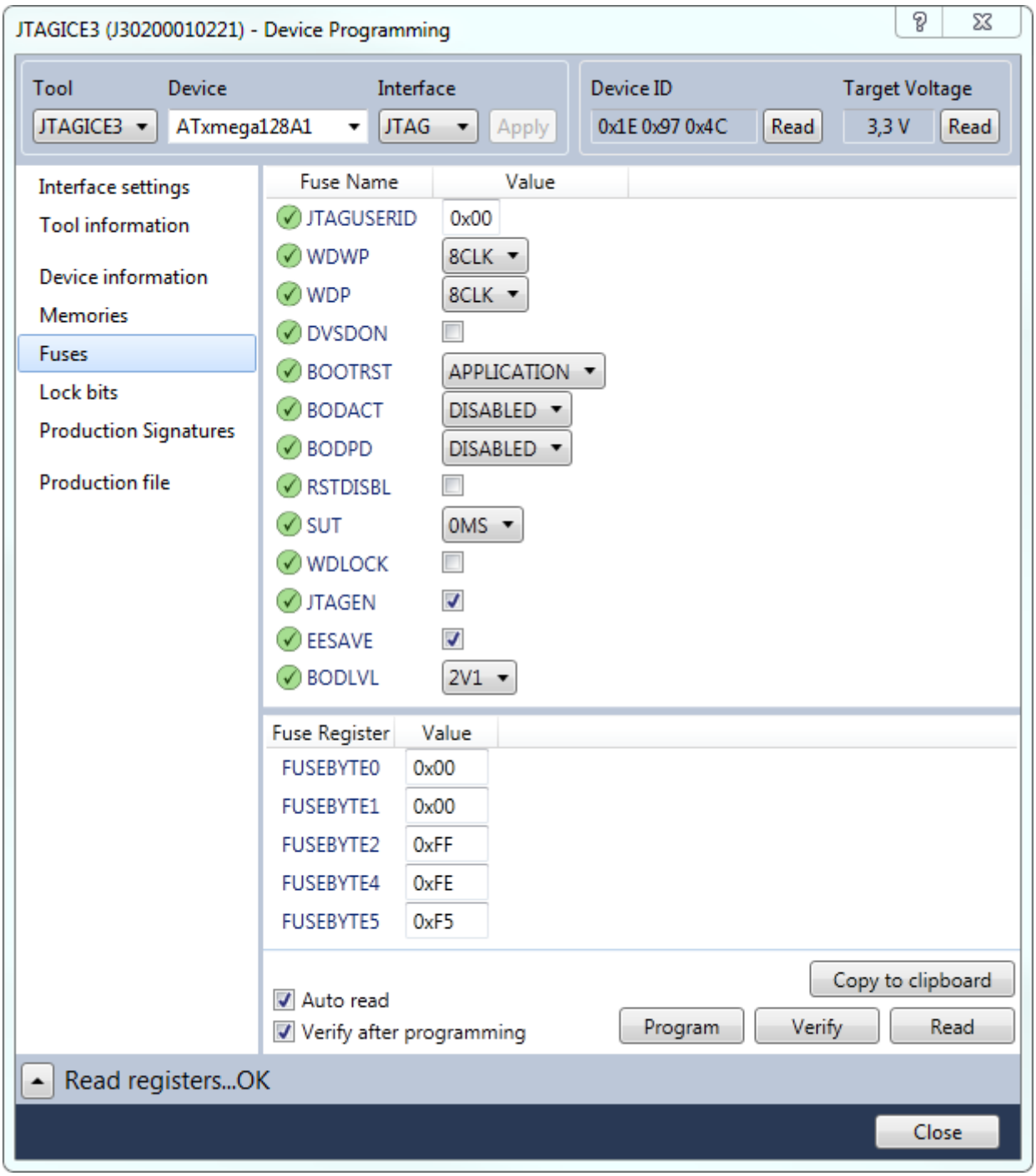
The device's EEPROM memory can be programmed in a similar way.

User Signatures

The XMEGA device's User Signature memory can be programmed the same way.

5.9. Fuse Programming

Figure 5-11. Fuse Programming



The **Fuses** page presents the fuses of the selected device.

Press the **Read** button to read the current value of the fuses, and the **Program** button to write the current fuse setting to the device. Fuse settings are presented as check boxes or as drop down lists.

Detailed information on which fuses are available in the different programming modes and their functions can be found in the device datasheet. Note that the selected fuse setting is not affected by erasing the device with a chip-erase cycle (i.e. pressing the **Chip Erase** button on the **Memories** page).

Fuse values can also be written directly into the fuse registers in the lower pane as hexadecimal values.

Auto read	If this check box is checked, the fuse settings will be read from the device each time you enter the fuse page.
Verify after programming	When this check box is checked, the settings will be verified after a programming operation is completed.

The appearance of the fuse glyph describes whether the fuse information is up to date compared to the state of the device.

the fuse value is up to date. i.e the same state as in the device.

the fuse has been modified by the user and it is not yet programmed into the device.

the fuse state is unknown, it has not been read from the device, nor modified by the user.

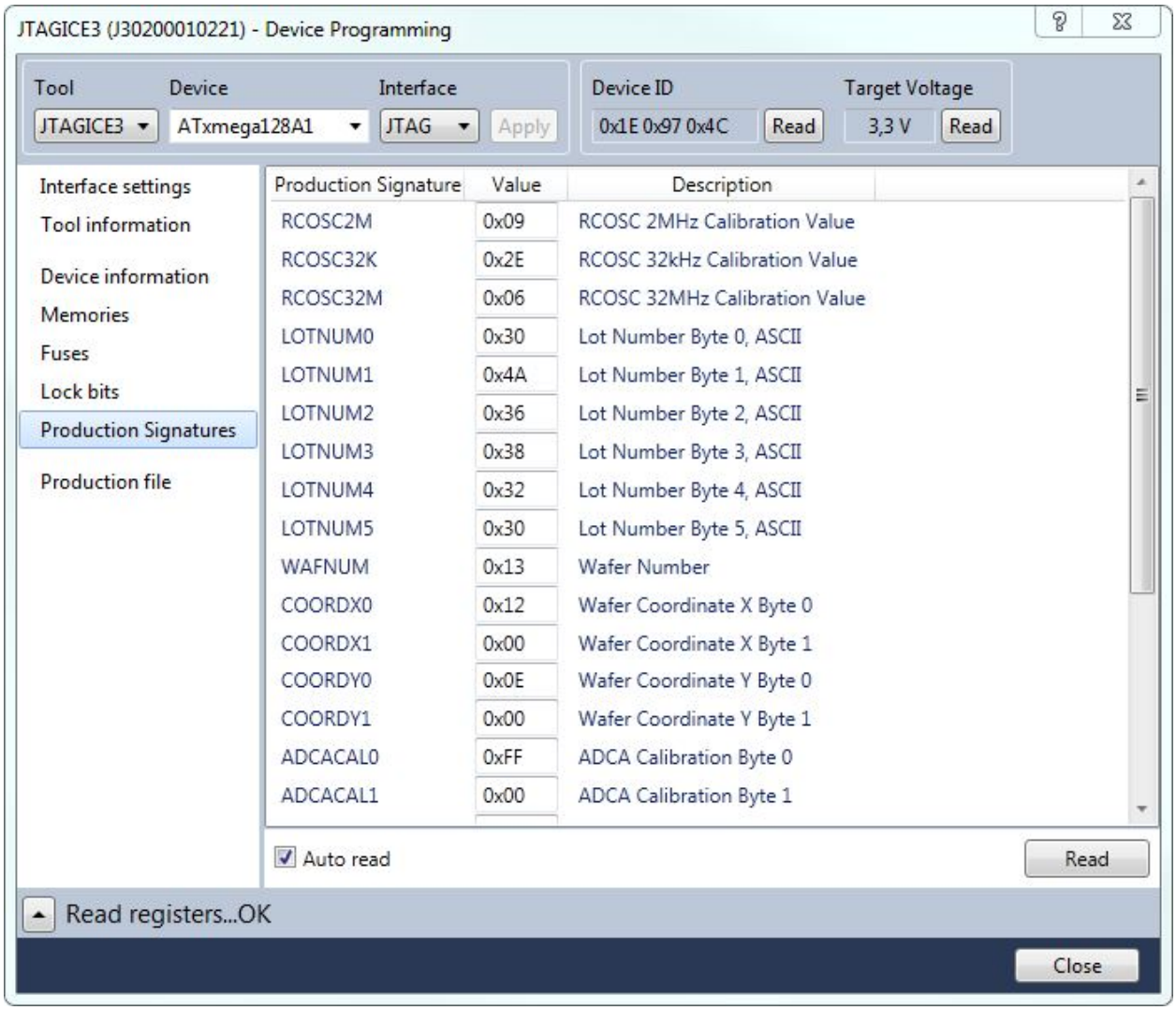
5.10. Lock Bits

The lock bit page is similar to the fuse page. For usage, see section Fuse Programming.

5.11. Production Signatures

The production signature page is only visible for AVR XMEGA devices and shows factory programmed data in the production signature row. It contains calibration data for functions such as oscillators and analog modules. The production signature row can not be written or erased.

Figure 5-12. Production Signatures

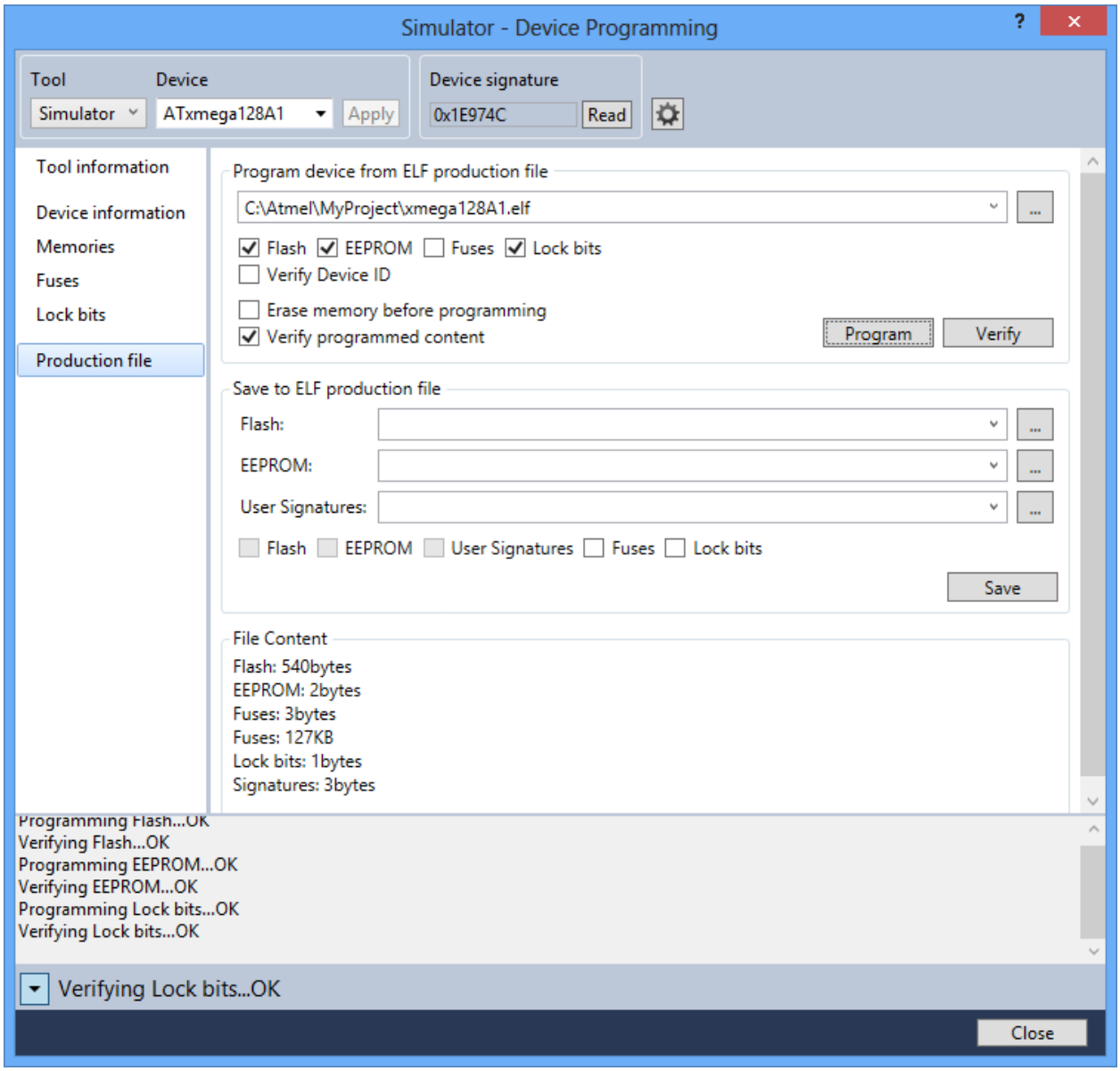


5.12. Production Files

The ELF production file format can hold the contents of both Flash, EEPROM, and User Signatures (XMEGA devices only) as well as the Fuse- Lockbit configuration in one single file. The format is based on the Executable and Linkable Format (ELF).

The production file format is currently supported for tinyAVR, megaAVR, and XMEGA. See Creating ELF Files with Other Memory Types for description on how to configure the project in order to generate such files.

Figure 5-13. Production Files Programming



Program device from ELF production file: To program your device from an ELF file, you must first select a source file by typing its full path into the combo box, or by pressing the browse button ... Depending on the contents of your file, check boxes for the different memory segments will be activated.

It is possible to select one or several of the memory segments that the ELF production file contains. You can then program and verify the device with the content of these segments in one single operation. Select which memory segments you want to program ticking off the corresponding check boxes.

Select the **Erase memory before programming** check box, if you want an erase operation to be performed before the programming operation.

Note: The *erase memory operation* will depend on the device selection. For tinyAVR and megaAVR, both Flash, EEPROM, and lockbits will be erased (chip erase) independent of which memories are selected, while for XMEGA only the selected memories will be erased.

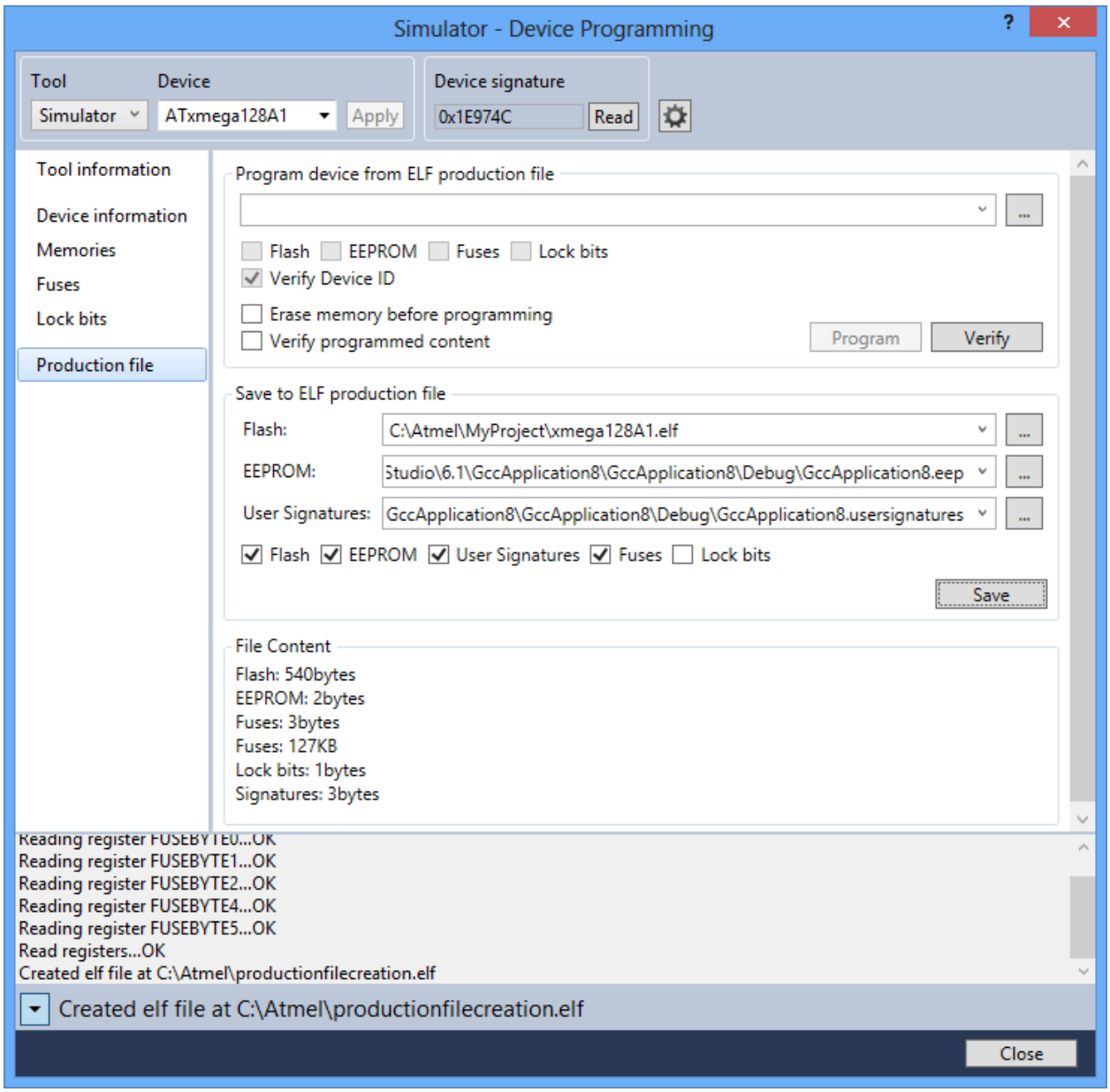
Select the **Verify device after programming** check box, if you want the contents to be verified after the programming operation is done.

Select the **Verify Device ID** check box, if you want to verify the device id stored in the file (signature bytes) with the connected device.

Now, press the **Program** button to program the file into the memory.

You can verify the contents of the device against an ELF file by pressing the **Verify** button. The verification will only verify the contents of the selected memory segments.

Figure 5-14. Production Files Creation



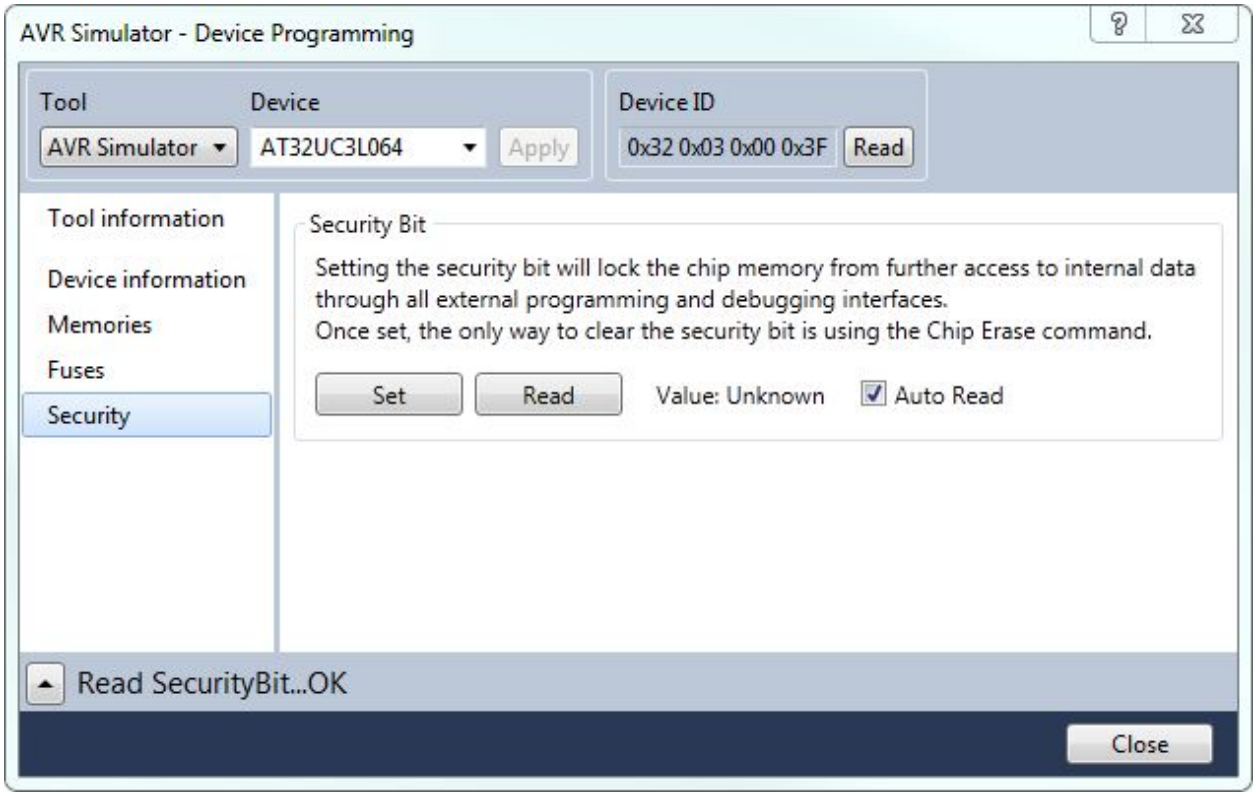
Save to ELF production file: Prior to creating the ELF file, specify the input file path for FLASH, EEPROM, and Usersignature on the production file tab. Then configure the Fuse and Lockbits on the corresponding tab and program it. The Fuse and Lockbits, which are programmed in the device will be taken as input while creating ELF file. Back on the production file tab, press the "Save" button" to generate the ELF file.

You must specify which segments are to be present in the production ELF file by ticking the corresponding check boxes.

5.13. Security

The security bit allows the entire chip to be locked from external JTAG or other debug access for code security. Once set, the *only* way to clear the security bit is through the **Chip Erase** command.

Figure 5-15. Security Page



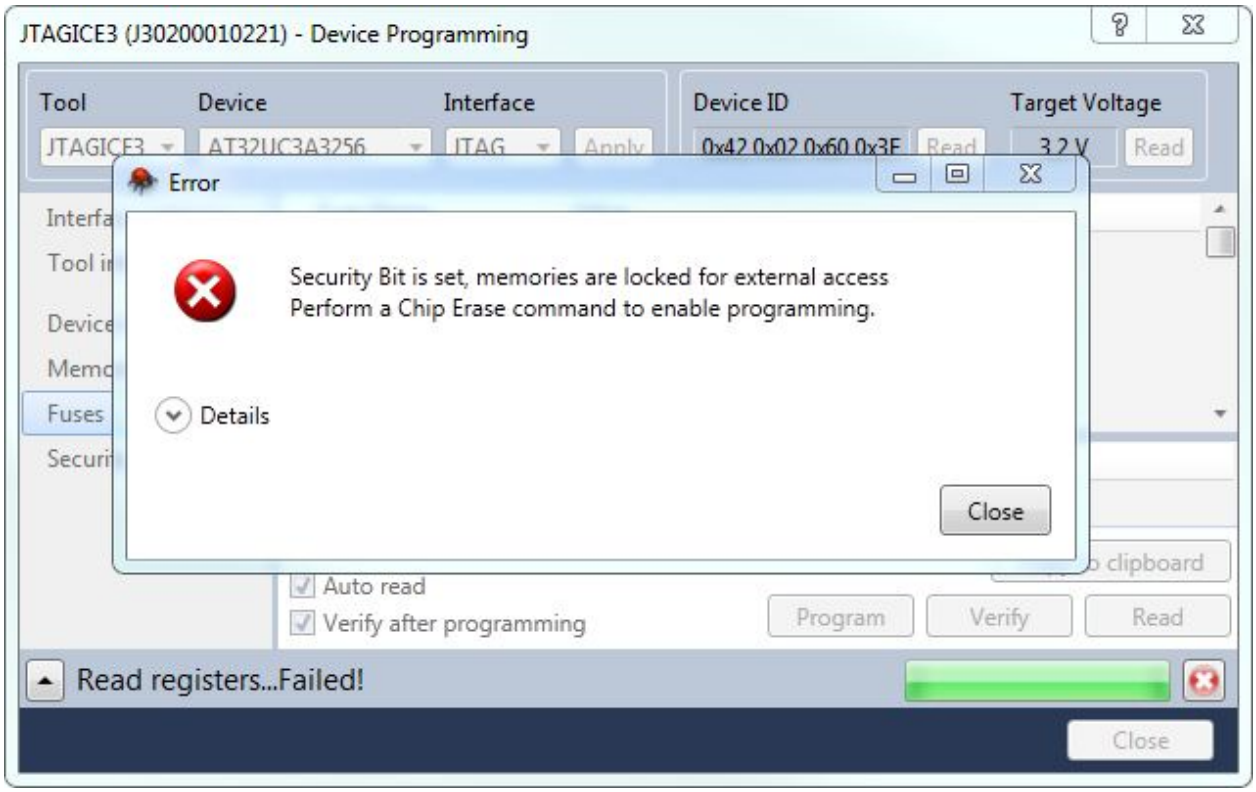
To check the state of the security bit, press the **Read** button on the **Security** page of the programming dialog. The value should now read **Cleared** or **Set**. **Set** meaning that the security bit is set, and **Cleared** meaning that it is not set. If the **Auto Read** check box is ticked off, the **Read** operation will be performed automatically when the **Security** page is opened.

To set the security bit, simply press the **Set** button on the **Security** page of the programming dialog. Now the device is locked for all further JTAG or aWire access except for the **Chip Erase** command.

Locked device

When the security bit is set, the device is locked for most external debug access. Attempts to program or read any memories or fuses, will cause an error message to appear.

Figure 5-16. Security Bit Error



To unset the security bit, issue the **Chip Erase** command. This can be done from the Memories page, see Memories.

5.14. Automatic Firmware Upgrade Detection

As mentioned in the Firmware Upgrade section, you may encounter a dialog stating that your tool's firmware is out of date when you open the **Device Programming** dialog.

6. Miscellaneous Windows

6.1. Device Pack Manager

The Device Pack Manager is used to manage the devices supported by Atmel Studio.

The Device Pack Manager is launched from **Tools** → **Device Pack Manager**.

Figure 6-1. Device Pack Manager Menu

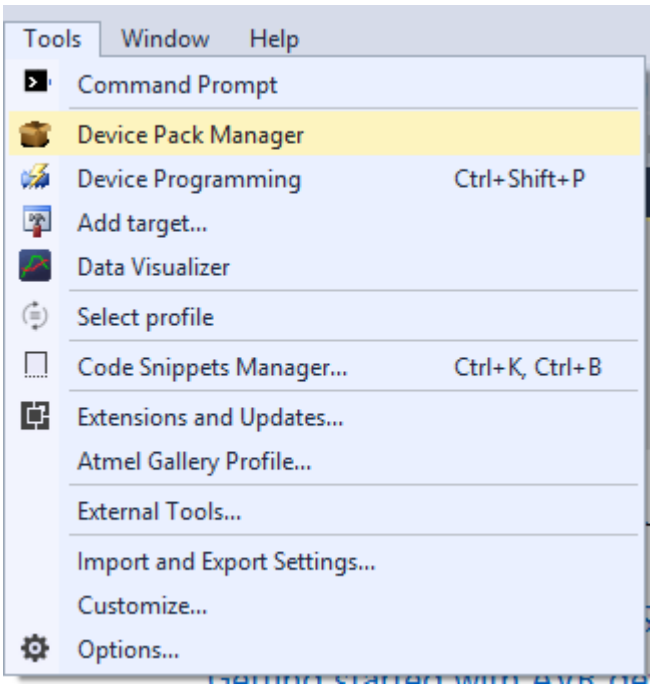
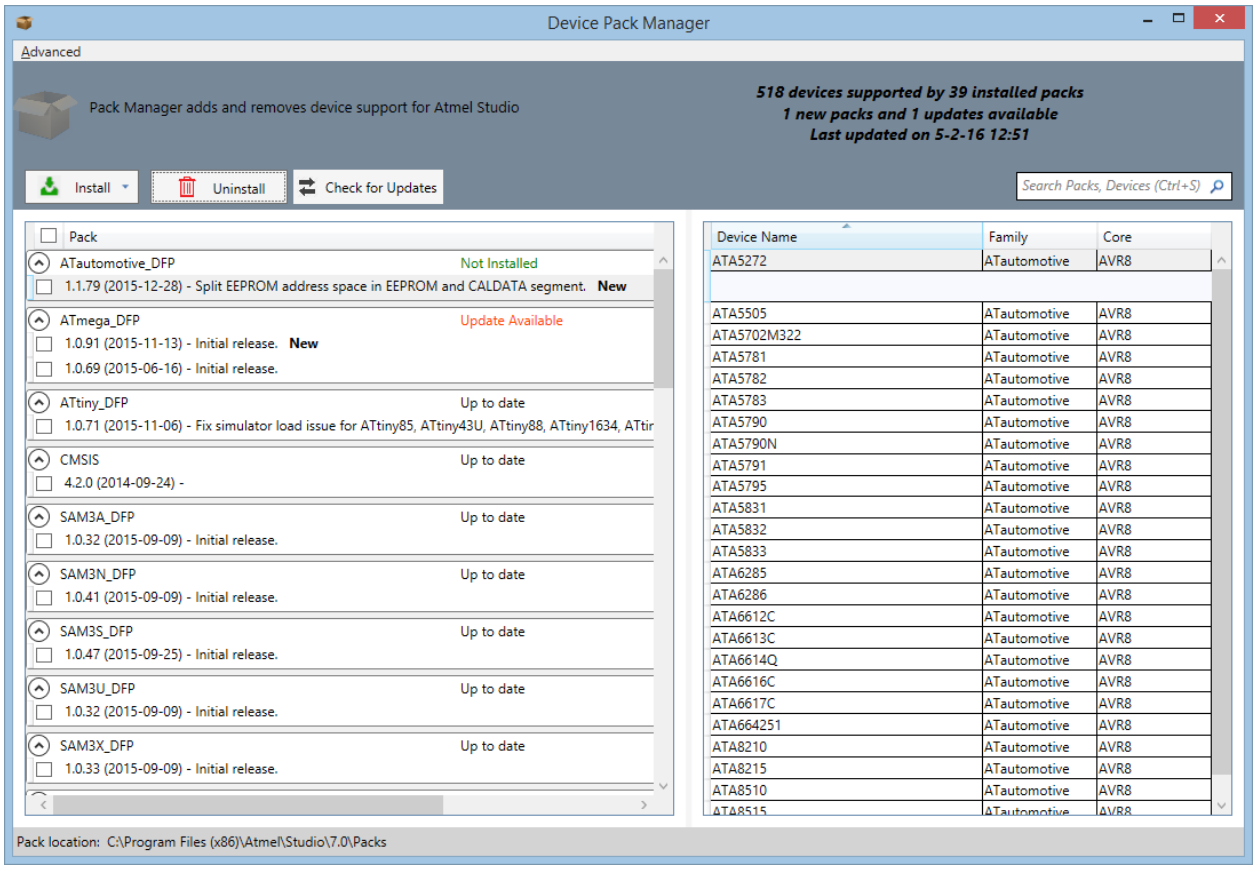


Figure 6-2. Device Pack Manager



The Device Pack Manager consists of two panes. The left pane shows the list of packs that are installed. The right pane shows the devices that are provided by the pack selected in the left pane.

Packs can have any of the following statuses:

Up to date	Pack is already up to date and latest.
Update Available	New update is available.
Not Installed	Pack is not installed, but can be downloaded.

Actions

Install selected packs	Download and install all packs that have been selected using the check-boxes besides the version.
Install all updates	Download and install all available updates.
Browse pack file	Install an already downloaded pack file.
Uninstall	Uninstalls all packs that have been selected using the check-boxes besides the version.
Check for Updates	Check for new and updated packs.
Search	The search box can be used to search after a specific pack, or a device in any of the packs.

Reset cache	Resetting the cache will re-index all installed packs. This does not uninstall or remove anything. It is in the **Advanced** menu.

Note: After installing, updating, or removing packs, Atmel Studio has to be restarted before the changes becomes visible.

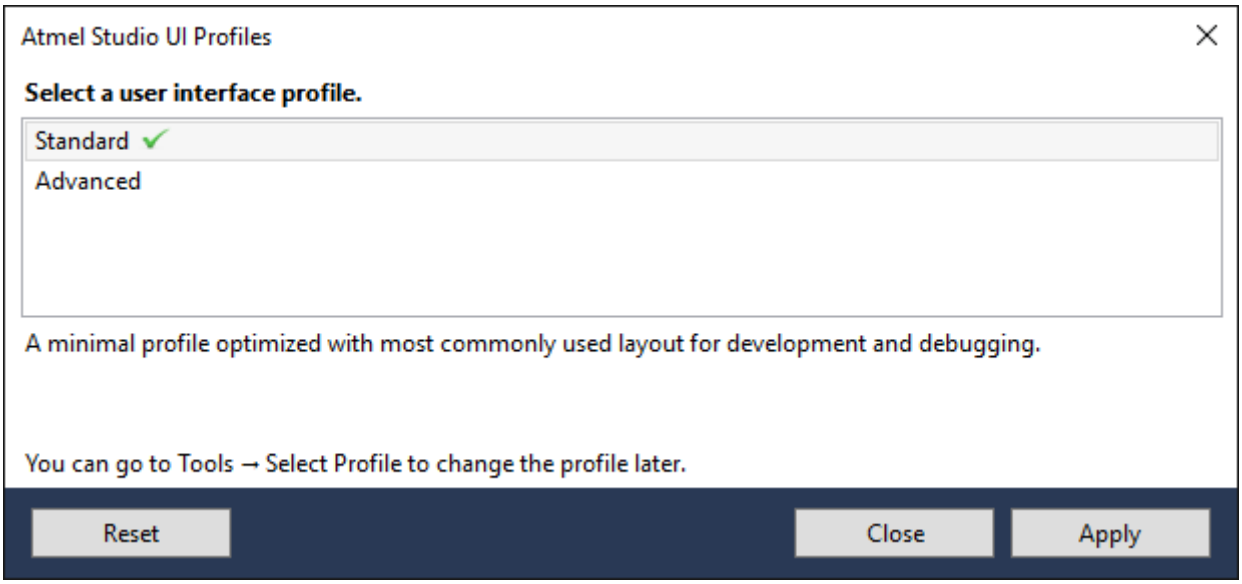
6.2. User Interface Profile Selection

Different user interface profiles targeted for different use is available in Atmel Studio.

The user interface profile controls the visibility of menus, window layouts, toolbars, context menus, and other elements of Atmel Studio. The following modes are available:

Standard	The default profile. Includes the most used windows and menus.
Advanced	The profile used in previous versions of Atmel Studio. This profile includes advanced debugging and re-factoring tools.

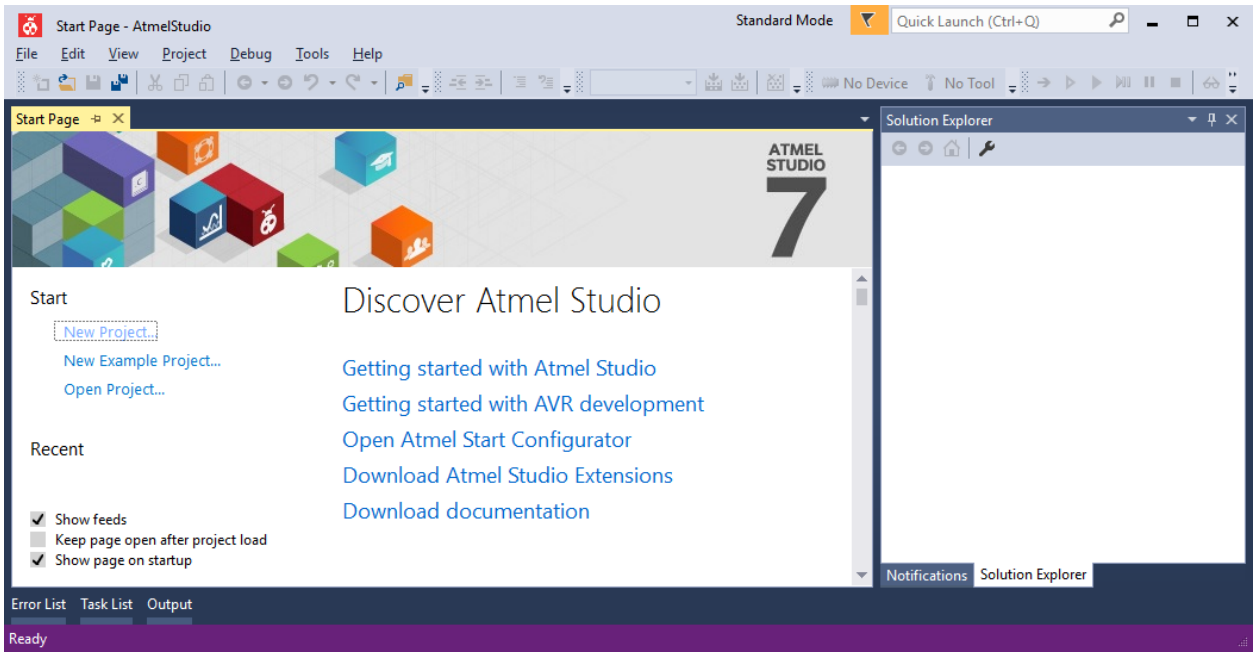
Figure 6-3. Profile Selection



The profile selection window is shown the first time Atmel Studio is started. Selecting a profile in the list will show a description of the profile. Clicking the **Apply** button applies the profile to Atmel Studio.

The profile can be changed at any time by navigating to **Tools** → **Select Profile**, or by clicking the profile name that is displayed in the top right corner of Atmel Studio.

Figure 6-4. Selected Profile



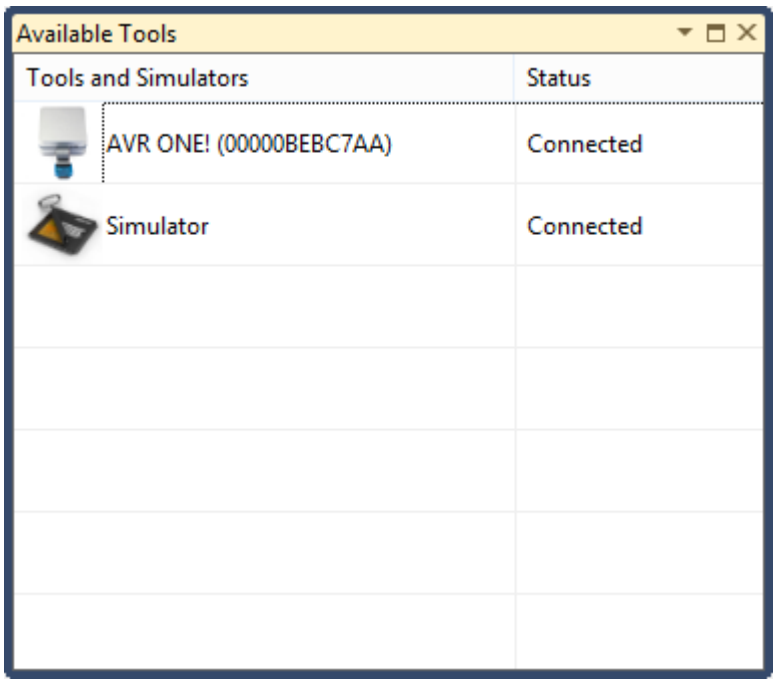
When switching profiles, any changes done to the active profile is saved. Going back to the previous profile will restore the changes as well as the profile.

Using the **Reset** option discards any changes saved to the profile and restores it to the default profile.

6.3. Available Tools View

6.3.1. Introduction

The **Available Tools** view (**View** → **Available Atmel Tools**) contains a list of all connected tools such as programmers, debuggers and starter kits. The Simulator is always present. Other tools will show up when they are connected to the PC.



6.3.2. Tool Actions

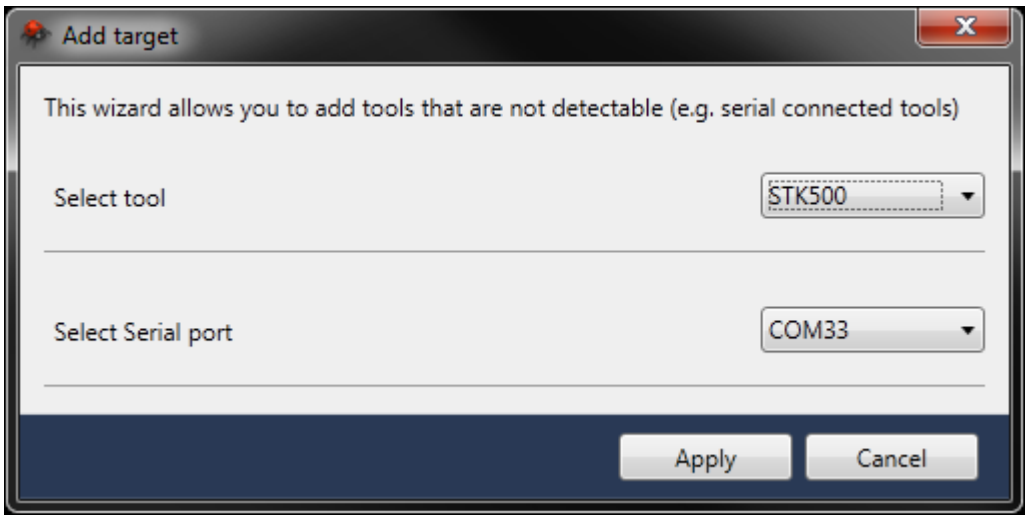
The following actions can be selected by right clicking tools in the **Available Tools** view:

Device Programming	Opens the Device Programming window with the tool preselected.
Self test	Some tools are capable of performing a self test. Follow the displayed instructions.
Add target	Adds a tool to the list of available tools that is not auto-detectable. See Add a Non-detectable Tool for more information.
Upgrade	Starts the firmware upgrade tool with the selected tool.
Show Info Window	Shows the Tool Info window. Not all tools supports this feature. See Tool Info Window for more information.

6.3.3. Add a Non-detectable Tool

The STK500 does not have a USB connection, and cannot be automatically detected by Atmel Studio. So it must be added to the list of available tools before it can be used by the **Device Programming** window.

To add an STK500, right click inside the Available Tools view, select **Add target** and select the STK500 as the tool and the COM port your STK500 will be connected to.



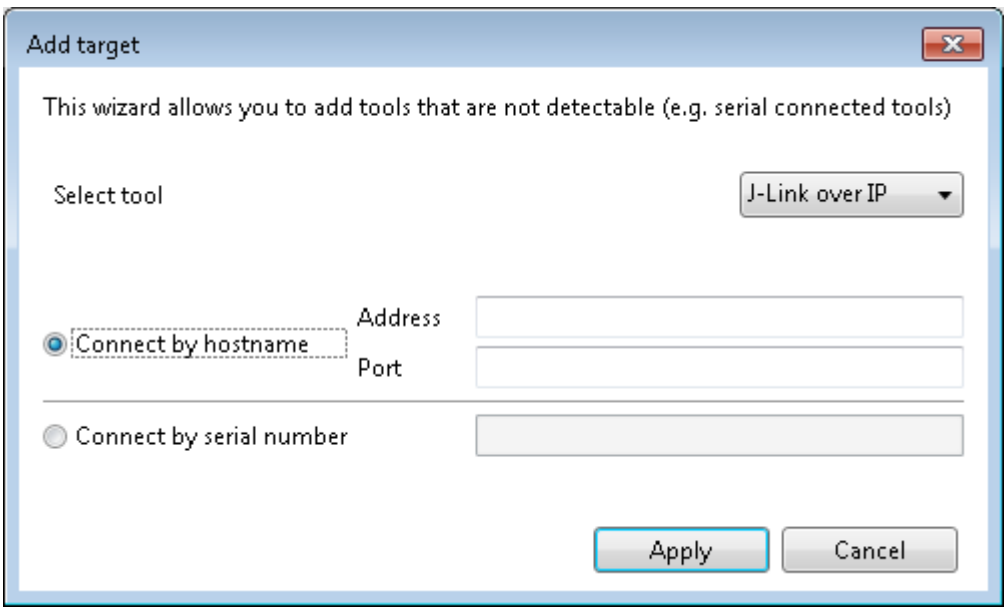
Press the **Apply** button, and the STK500 will be displayed in the list of available tools.

Note: An STK500 that has been added will be visible in the Available Tools view event even if no STK500 is connected to the specified COM port.
If you want to remove STK500 from the list, you can right click on it and select **Remove** from the context menu.

6.3.3.1. Add J-Link over IP

In the **Add target** dialog, it is possible to add a remote Segger J-Link debug probe. Both using a debug probe with built-in ethernet such as the J-Link PRO[3] and any other Segger probe by using the J-Link Remote Server software[4].

Figure 6-5. Add J-Link over IP



To add a debug probe that is connected to a J-Link Remote Server, choose **Connect by hostname** and enter the IP address or the hostname of the computer running the J-Link Remote Server. If the J-Link

[3] See https://www.segger.com/jlink-pro.html
[4] See https://www.segger.com/jlink-remoteserver.html

Remote server is running on a non-standard port[5] then the port also needs to be entered. If the J-Link Remote Server is running on the default port, the port can be left empty.

To add a debug probe that has built-in ethernet, choose **Connect by serial number** in the **Add target** dialog, and enter the serial number of the debug probe.

6.4. Tool Info Window

The **Tools Info** window shows information about connected tools. At the moment, only the Xplained Pro series is supported.

[5] The standard port of the J-Link Remote Server is port 19020

Figure 6-6. The Tool Info Window

When a tool is connected, the window will open. It has a short description about the tool, an image of the tool, and a section of links to then user guide, relevant datasheets on the internet, etc.

There is also a table with technical details about the tool, such as firmware version, serial number, etc.

6.4.1. Xplained Pro Kits

The Xplained Pro family of boards supports a range of expansion boards. When an Xplained Pro board is connected, the **Tool Info** window will show a list on the left side of the window, containing the main board, and all connected expansions. Click on the main board and the expansion to see details about the different boards.

6.4.2. Disable the Tools Info Window

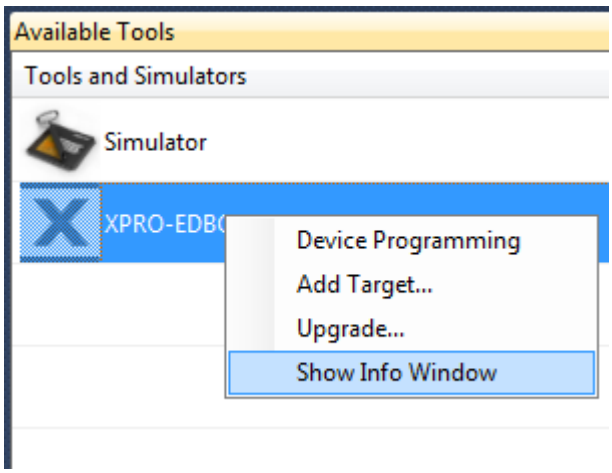
By deselecting the **Show page on connect** check box, the window will not automatically open when Atmel Studio is open and you connect the kit.

This feature works on a per-tool basis, which means you can select for every tool you have, if they should show the **Tool Info** window when connected.

6.4.3. Manually Showing the Window

If you want to see the **Tool Info** window again after it has been closed, you can right-click on the tool in the **Available Tools view**, and select **Show Info Window**.

Figure 6-7. Show Tool Info Window



See also Available Tools View.

6.5. Firmware Upgrade

6.5.1. Introduction

Atmel Studio will include the latest firmware for all Atmel tools. New firmware may provide support for new devices and bugfixes.

6.5.2. Automatic Upgrade

Atmel Studio will automatically upgrade the tool's firmware when needed. A potential firmware upgrade is triggered once you start using a tool. Examples: the first time you launch a debug session or the first time you select the tool in the Device Programming dialog.

The tool cannot be used by Atmel Studio if the user chooses not to upgrade.

You can also check for firmware upgrades by using the **Available Tools** view (**View → Available Atmel Tools**). Right click on a tool and select **Upgrade**.

For a description on how to do manual upgrade, downgrade and upgrade with a custom firmware image, see Manual Upgrade.

6.5.3. Manual Upgrade

Atmel Studio includes a command line utility called `atfw.exe` which can be used to do manual upgrade of most Atmel tools. `atfw.exe` is installed in the `atbackend` subfolder.

`atfw.exe` can be used to:

- Perform upgrade from a script
- Upgrade using a custom firmware file
- Read out firmware version

For details on how to upgrade using this utility, execute `atfw.exe -h`.

Note: If a tool is locked in firmware upgrade mode, and normal reset does not restore normal operation, a forced firmware upgrade should reset the tool to a working state.
To do a firmware upgrade on a tool already in upgrade mode, invoke `atfw` the same way as normal firmware upgrade. Some warnings may be displayed as the tool is unable to switch the tool to upgrade mode, but should proceed with the upgrade.

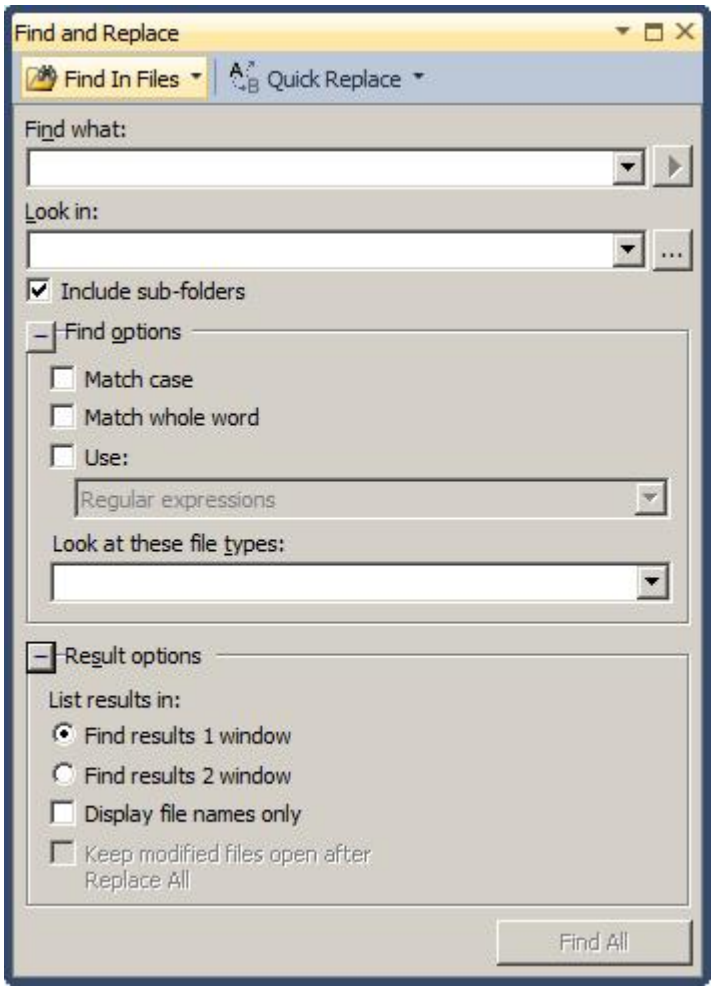
If a tool listing is done, the tool will have a name that is related to the mode it is in. `atfw` should however be invoked with the tool name as it is presented to the user in normal operation.

6.6. Find and Replace Window

You can use the Find and Replace window to search for text strings, expressions, or entity names within the code of your documents. To access this window, from the Edit menu, click Find and Replace, and then select one of the options listed.

The Find and Replace window contains a toolbar with two drop-downs, one for find operations and one for replace operations. When you select an operation, the corresponding options for the operation are displayed. You can search and replace in one or more files or an entire solution for text, code, or symbols.

Figure 6-8. Find and Replace



Quick Find allows you to search the code of one or more open documents for a string or expression. The selection moves from match to match, allowing you to review each match in its surrounding context.

Note: The matches found are not listed in the **Find Results** window.

You can use any of the following methods to display **Quick Find** in the **Find and Replace** window.

To display Quick Find

1. On the **Edit** menu, expand **Find and Replace**.
2. Choose **Quick Find**.
   -or-

   If the **Find and Replace** window is already open, on the toolbar, click the triangular **View** button on the left drop-down and then choose **Quick Find**.

Quick Find can search through a document either forward or backward from the insertion point. The search automatically continues past the end or start of the document into the unsearched portion. A message appears when the entire document has been searched.

Find what

These controls allow you to specify the string or expression that will be matched.

Reuse one of the last 20 search strings by selecting it from this drop-down list, or type a new text string or expression to find.

Table 6-1. Quick Find

Option	Description
[string with wildcards]	If you want to use wildcards such as asterisks (*) and question marks (?) in your search string, select the **Use** check box under Find options and then choose **Wildcards**.
[regular expression]	To instruct the search engine to expect regular expressions, select the **Use** check box under Find options and then choose **Regular expressions**.

Expression Builder

This triangular button next to the **Find what** field becomes available when the Use check box is selected in **Find** options and **Regular Expressions** appears in the drop-down list. Click this button to display a list of wildcards or regular expressions, depending upon the **Use** option selected. Choosing any item from this list adds it into the **Find what** string.

Find Next

Click this button to find the next instance of the **Find what** string within the search scope chosen in **Look in**.

Bookmark All

Click this button to display blue bookmarks at the left edge of the code editor to indicate each line where an instance of the **Find what** string occurs.

Look in

The option chosen from the Look in drop-down list determines whether Quick Find searches only in currently active files.

Look in

Select a predefined search scope from this list.

Table 6-2. Look in Scopes

Option	Description
Selection	This option is available when text is selected in the code editor. Searches only the selected text in the currently active document.
<Current Block>	The name of this option indicates the location of the insertion point in the code editor. Searches within the current procedure, module, paragraph, or code block.
Current Document	This option is available when a document is open in an editor. Searches only the active document for the **Find what** string.
Current Window	This option is available when a searchable tool window, such as the View in Browser window, has focus. Searches all content displayed in this window for the Find what string.

Option	Description
All Open Documents	Searches all files currently open for editing as if they were one document. When the starting point of the search is reached in the current file, the search automatically moves to the next file and continues until the last open file has been searched for the Find what string.
Current Project	Searches all files in the current project as if they were one document. When the starting point of the search is reached in one file, the search continues in the next until the last file in the project has been searched.

Find options

You can expand or collapse the Find options section. The following options can be selected or cleared:

Match case

Only displays instances of the Find what string that are matched both by content and by case. For example, a search for "MyObject" with Match case selected will return "MyObject" but not "myobject" or "MYOBJECT".

Match whole word

Only displays instances of the Find what string that are matched in complete words. For example, a search for "MyObject" will return "MyObject" but not "CMyObject" or "MyObjectC".

Search up

When selected, files are searched from the insertion point to the top of the file.

Search hidden text

When selected, the search will also include concealed and collapsed text, such as the metadata of a design-time control; a hidden region of an outlined document; or a collapsed class or method.

Use

Indicates how to interpret special characters entered in the Find what or Replace with text boxes. The options include:

Table 6-3. Search with Special Characters

Option	Description
Wildcards	Special characters such as asterisks (*) and question marks (?) represent one or more characters. For a list, see Wildcards (Visual Studio).
Regular Expressions	Special notations define patterns of text to match. For a list, see Regular Expressions (Visual Studio).

Toolbar

A toolbar, with two drop-downs, appears at the top of the **Find and Replace** window. These drop-downs allow you to choose the type of search or replace you intend to perform and changes the options displayed in the window to match.

Table 6-4. Find and Replace Toolbar

Drop-down	View menu
Find (left drop-down)	**Quick Find** **Find in Files** **Find Symbol**
Replace (right drop-down)	**Quick Replace** **Replace in Files**

Figure 6-9. Find Results

Figure 6-10. Find Symbol Results

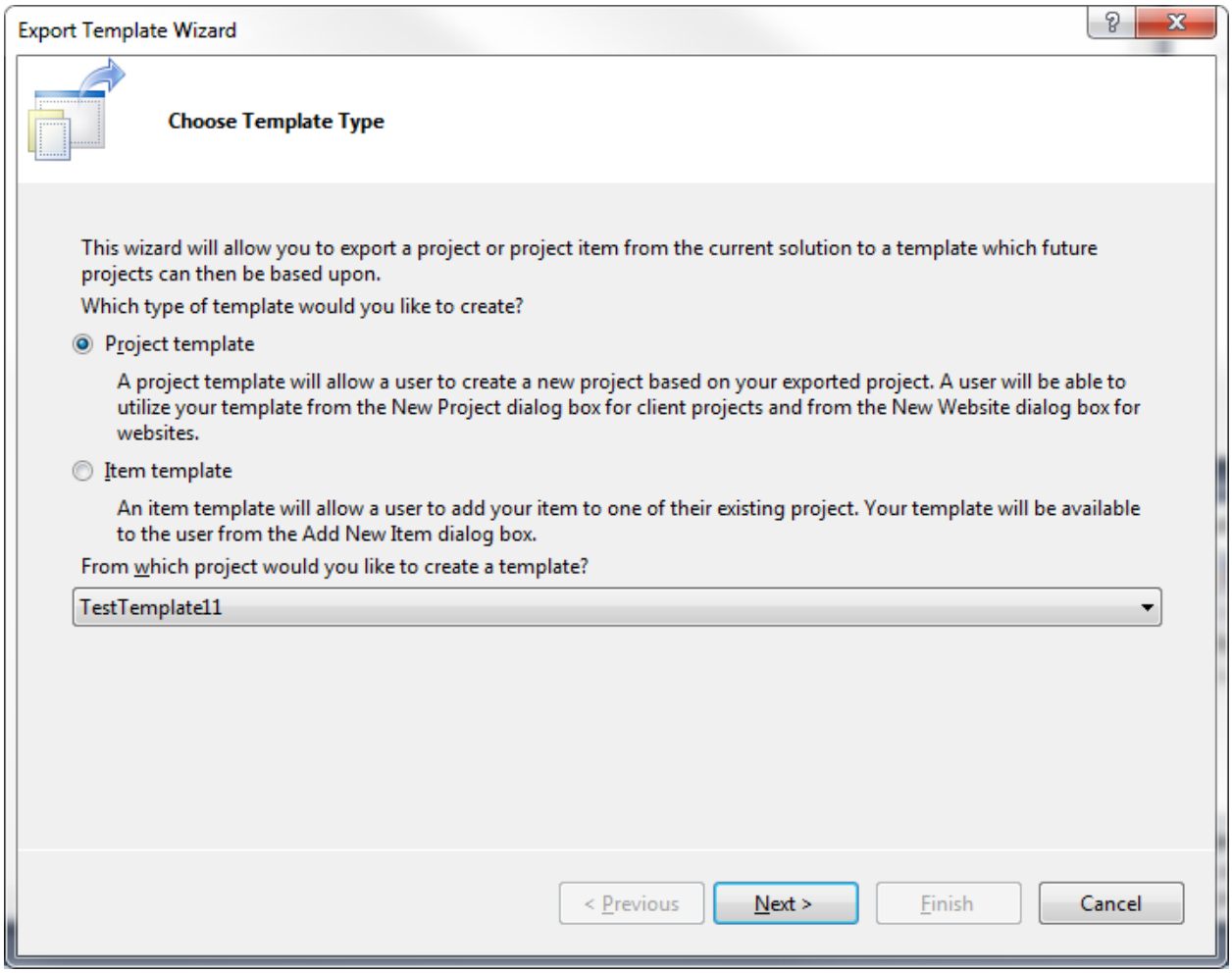
6.7. Export Template Wizard

Atmel Studio project and item templates provide reusable and customizable project and item stubs that accelerate the development process because users do not have to create new projects and items from scratch.

Note: This functionality is inherited from Microsoft Visual Studio® and the documentation from Microsoft goes beyond what is mentioned in this section. See MSDN for in-depth information.

Open the Export Template Wizard by clicking **File** → **Export Template...**. This opens the Export Template Wizard shown in the figure below.

Figure 6-11. Export Template Wizard...



6.7.1. Project Template

A Project template is a template that contains a whole project. This template can be redistributed to other users to ease setup of a default project. The code, which is to be template, can contain parameters that are substituted on creation. See Default Template Parameters for information on this.

The template wizard is mostly self explanatory, and on completion the created template will be available in the **File → File → New Project...** dialog.

6.7.2. Item Template

An Item template is a template that contains a single file or collection of files. The code which is to be templated can contain parameters that are substituted on creation. See Default Template Parameters for information on this.

The template wizard is mostly self explanatory, and on completion the created template will be available as a file type when files are added to the project.

6.7.3. Template Parameters

All templates support parameter substitution to enable replacement of key parameters, such as class names and namespaces, when the template is instantiated. These parameters are replaced by the template wizard that runs in the background when a user clicks OK in the New Project or Add New Item dialog boxes.

6.7.3.1. Declaring and Enabling Template Parameters

Template parameters are declared in the format `$parameter$`.

6.7.3.2. Default Template Parameters

The table below lists the reserved template parameters that can be used by any template.

Note: Template parameters are case-sensitive.

Table 6-5. Template Parameters

Parameter	Description
$itemname$	The name provided by the user in the Add New Item dialog box
$machinename$	The current computer name
$projectname$	The name provided by the user in the New Project dialog box
$registeredorganization$	The registry key value from `HKLM\Software\Microsoft\Windows NT \CurrentVersion\RegisteredOrganization`
$safeitemname$	The name provided by the user in the Add New Item dialog box, with all unsafe characters and spaces removed
$safeprojectname$	The name provided by the user in the New Project dialog box, with all unsafe characters and spaces removed
$time$	The current time in the format DD/MM/YYYY 00:00:00
$userdomain$	The current user domain
$username$	The current user name
$year$	The current year in the format YYYY
$guid[1-10]$	A GUID used to replace the project GUID in a project file. You can specify up to 10 unique GUIDs (for example, guid1)

6.7.3.3. Custom Template Parameters

You can use the `CustomParameter` element in your .vstemplate file to add new parameters to a template.

1. Locate the `TemplateContent` element in the .vstemplate file for the template.
2. Add a `CustomParameters` element and one or more `CustomParameter` child elements as children of the TemplateContent element.

   Figure 6-12. Adding Custom Parameters

```
<TemplateContent>
...
<CustomParameters>
    <CustomParameter Name="$MyParameter1$" Value="MyValue2"/>
    <CustomParameter Name="$MyParameter2$" Value="MyValue2"/>
</CustomParameters>
</TemplateContent>
```

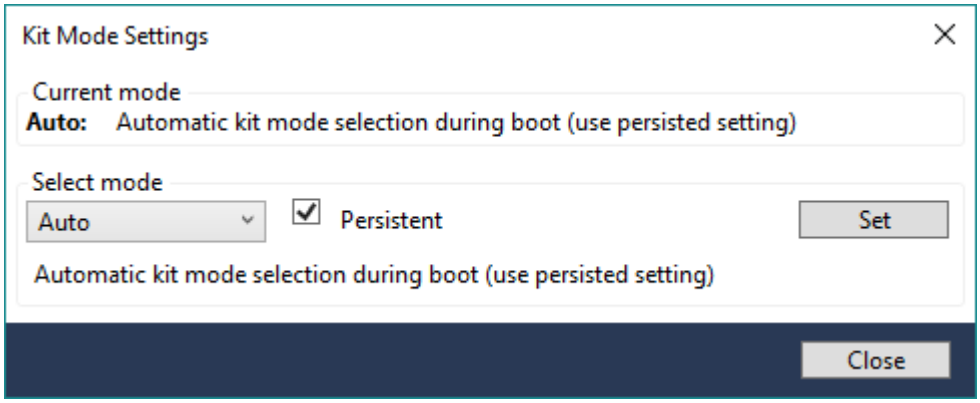
3. Use the parameter in one or more of the code files in the template as shown in Default Template Parameters.

More information on this can be found on MSDN.

6.8. Kit Mode Setting

Some kits operate with different modes. This window can be used to change the mode.

Figure 6-13. Kit Mode Settings



Some examples of the choices that can be made are listed in the following table.

Select mode	Persistent	Resulting mode
Mass Storage	Yes	Auto, enumerating as a Mass Storage Device kit
DGI		Auto, enumerating as a DGI kit
Mass Storage	No	Mass Storage, enumerating once as a Mass Storage Device kit before returning to the previous mode
DGI		DGI, enumerating once as a DGI kit before return to the previous mode

Note: When the persistent mode is used, the kit will reboot into Auto mode, since the persistent choice changes the kit default.

7. Atmel GNU Toolchains

Atmel GNU Toolchains are a set of standalone command line programs used to create applications for Atmel SAM and Atmel AVR microcontrollers.

7.1. GNU Compiler Collection (GCC)

The GNU Compiler Collection is used by Atmel Studio at the build stage. The architecture specific versions of the GNU Compiler Collection supports c-code compilation, assembly and linking of C and C++.

The AVR GNU compiler collection is distributed under the terms of the GNU General Public License, http://www.gnu.org/licenses/gpl.html. A copy of this license is also found in the installation folder of Atmel Studio .

7.2. ARM Compiler and Toolchain Options: GUI

To get help about ARM GCC Toolchain, you can do the following:

- For general information about GCC, visit the official GNU GCC web site
- Alternatively, you can write `arm-none-eabi-gcc --help` and see the explanation of some of the parameters in the command output

This section illustrates the GUI options that are available for the ARM GNU Toolchain in Atmel Studio.

Figure 7-1. ARM GNU Toolchain Properties

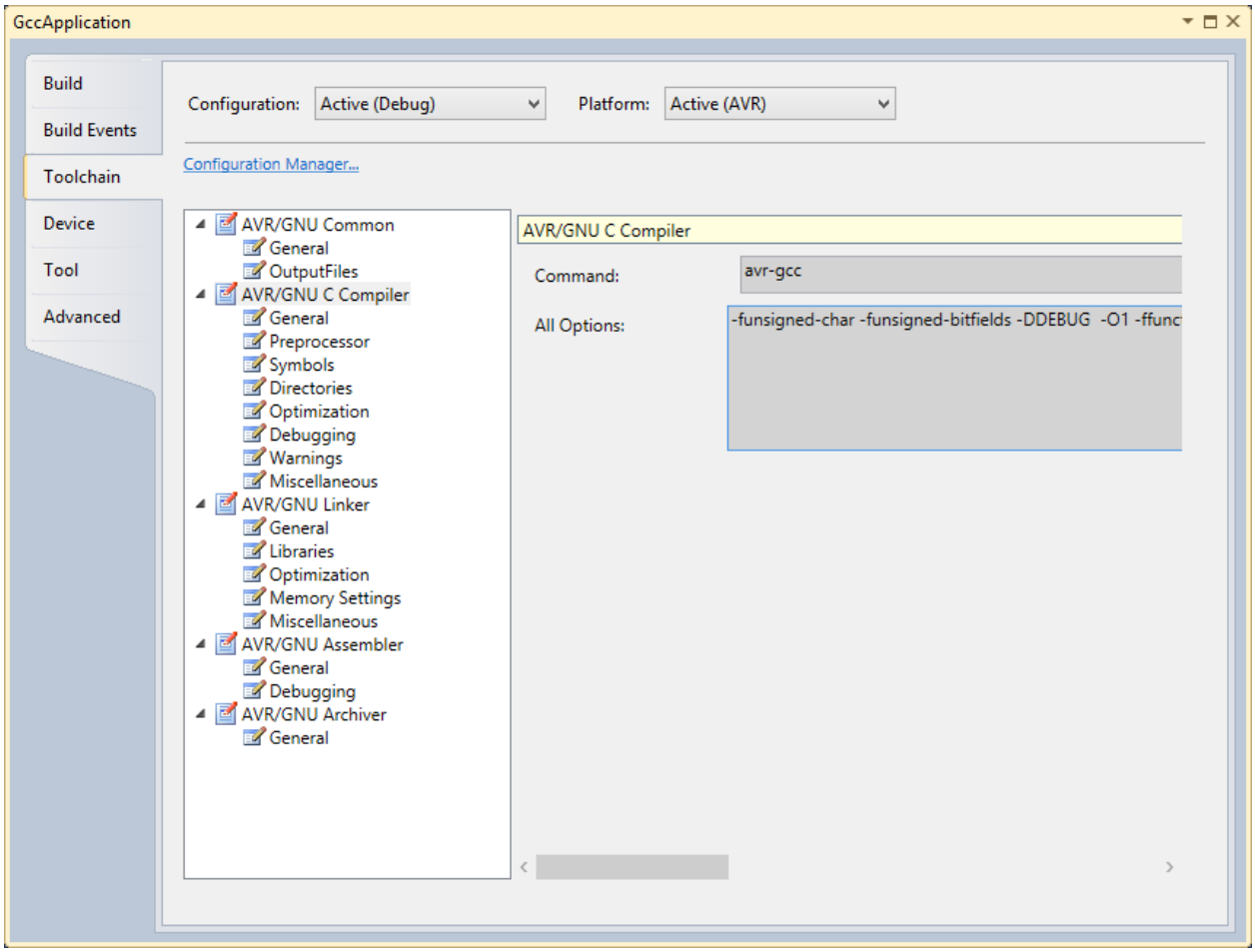


Table 7-1. ARM GNU Common Options

Option	Description
Thumb(-mthumb)/Arm(-marm)	Switch between Arm and Thumb processor mode

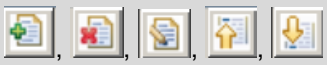
Table 7-2. ARM GNU C Compiler Options

Option	Description
Preprocessor options	
-nostdinc	Do not search system include directories
-E	Preprocess only; Do not compile, Assemble or link
Symbols options	

Option	Description
There one can define (-D) or undefine (-U) a number of in-source symbols. New symbol declarations can be added, modified, or reordered, using the interface buttons below: • Add a new symbol. This and all following icons are reused with the same meaning in other parts of Atmel Studio interface. • Remove a symbol. • Edit symbol. • Move the symbol up in the parsing order. • Move the symbol down in the parsing order.	
Include directories	
Default Include Path	Enabling this option will add the include path that are specific for the selected SAM device
Contains all the included header and definition directories, can be modified, using the same interface as symbols	
Optimization options	
Optimization level (drop down menu): -O0, -O1, -O2, -O3, -Os	No optimization, optimize for speed (level 1 - 3), optimize for size
Other optimization flags (manual input form)	Here you should write optimization flags specific for the platform and your requirements
-ffunction-sections	Place each function into its own section
-funsafe-math-optimizations	Enable unsafe match optimizations
-ffast-math	Enable fast math
-fpic	Generate position independent code
Debug options	
Debug level (drop down menu): none, -g1, -g2, -g3	Specifies the level of tracing and debugging code and headers left or inserted in the source code
Other debug options (form field)	Architecture specific debug options

Option	Description
-pg	Generate gprof information
-p	Generate prof information
Warning messages output options	
-Wall	All warnings
-Werror	Treat all warnings as errors
-fsyntax-only	Check syntax only
-pedantic	Check conformity to GNU, raise warnings on non-standard programming practice
-pedantic-errors	Same as above, plus escalate warnings to errors
-w	Inhibits all warnings
Miscellaneous options	
Other flags (form field)	Input other project-specific flags
-v	Verbose (Display the programs invoked by the compiler)
-ansi	Support ANSI programs
-save-temps	Do not delete intermediate files
Option	Description

Table 7-3. ARM GCC Linker Options

Option	Description
-Wl -nostartfiles	Do not use standard files
-Wl -nodefault	Do not use default libraries
-Wl -nostdlib	No start-up or default libraries
-Wl -s	Omit all symbol information
-Wl -static	Link statically
-Map	Generates Map file
Libraries options	
Libraries -Wl, -l (form field)	You can add, prioritize or edit library names here, using those buttons: 
Library search path -Wl, -L (form field)	You can add, prioritize or edit path where the linker will search for dynamically linked libraries, same interface as above

Option	Description
Optimization options	
-Wl, -gc-sections	Garbage collect unused sections
-funsafe-math-optimizations	Enable unsafe math optimizations
-ffast-math	Enable fast math
-fpic	Generate position independent code
Miscellaneous options	
Other linker flags (form field)	Input other project-specific flags
Option	Description

Linker Scripts

- In "`linker->miscellaneous->linker flags`" (`$LinkerScript_FLASH`) is added by default. It will be replaced with the appropriate (device_name)_flash.ld file during Build. Similarly (`$LinkerScript_SRAM`) will be replaced with the appropriate (device_name)_sram.ld file.
- You can always override the default flash linker scripts by replacing (`$LinkerScript_FLASH`) or (`$LinkerScript_SRAM`) with your custom linker script option –`T"custom_linker_script.ld"`.

Note: These device specific linker scripts will be available in the "ProjectFolder/Linkerscripts" directory. In case of changing the device after project creation, Atmel Studio will automatically add the correct linker scripts for the selected device.

ARM Assembler Options

Table 7-4. Arm Assembler Options

Option	Description
Optimization options	
Assembler flags (form field)	Miscellaneous assembler flags
Include path (form field)	You can add, prioritize or edit path to the architecture and platform specific included files here
-v	Announce version in the assembler output
-W	Suppress Warnings
Debugging options	
Debugging level (drop down menu) None , (-g).	Enable debugging symbols and debugging source insertion
Option	Description

ARM Preprocessing Assembler Options

Table 7-5. ARM Preprocessing Assembler Options

Option	Description
Optimization options	
Assembler flags (form field)	Miscellaneous assembler flags
Include path (form field)	You can add, prioritize or edit path to the architecture and platform specific included files here
-v	Announce version in the assembler output
-W	Suppress Warnings
Debugging options	
Debugging level (drop down menu) None , -Wa -g.	Enables debugging symbols and debugging source insertion
Option	Description

7.3. ARM GNU Toolchain Options

7.3.1. ARM/GNU Common Options

- `Thumb(-mthumb)/Arm(-marm)`

  Allows you to select the processor mode.

7.3.2. Compiler Options

7.3.2.1. Preprocessor

- `-nostdinc`

  Do not search the standard system directories for header files. Only the directories you have specified with -I options (and the directory of the current file, if appropriate) are searched.

- `-E`

  Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output. Input files which don't require preprocessing are ignored.

7.3.2.2. Symbols

- `-D`
  - -D *name*

    Predefine name as a macro, with definition 1.

    Eg:
  - -D *name=value*

    Predefine *name* as a macro, with definition *value*. The contents of definition are tokenized and processed as if they appeared during translation phase three in a #define directive. In particular, the definition will be truncated by embedded newline characters.
- `-U`

Cancel any previous definition of name, either built in or provided with a -D option.

-D and -U options are processed in the order they are given on the command line. All -imacros file and -include file options are processed after all -D and -U options.

7.3.2.3. Directories

- `-I dir`

  Add the directory dir to the list of directories to be searched for header files. Directories named by -I are searched before the standard system include directories. If the directory dir is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated .

7.3.2.4. Optimization

- There is a general switch '-O<optimization_level>' which specifies the level of optimization used when generating the code:
  - `-Os`

    Signal that the generated code should be optimized for code size. The compiler will not care about the execution performance of the generated code.
  - `-O0`

    No optimization. GCC will generate code that is easy to debug but slower and larger than with the incremental optimization levels outlined below.
  - `-O1 or -O`

    This will optimize the code for both speed and size. Most statements will be executed in the same order as in the C/C++ code and most variables can be found in the generated code. This makes the code quite suitable for debugging. This is default.
  - `-O2`

    Turn on most optimizations in GCC except for some optimizations that might drastically increase code size. This also enables instruction scheduling, which allows instructions to be shuffled around to minimize CPU stall cycles because of data hazards and dependencies, for CPU architectures that might benefit from this. Overall this option makes the code quite small and fast, but hard to debug.
  - `-O3`

    Turn on some extra performance optimizations that might drastically increase code size but increase performance compared to the -O2 and -O1 optimization levels. This includes performing function inlining
- Other optimization options
  - `-ffunction-sections`
  - `-fdata-sections`

    Place each function or data item into its own section in the output file if the target supports arbitrary sections. The name of the function or the name of the data item determines the section's name in the output file.

    Only use these options when there are significant benefits from doing so. When you specify these options, the assembler and linker will create larger object and executable files and will also be slower.
  - `-funroll-loops`

Perform loop unrolling when iteration count is known. If code size is not a concern then some extra performance might be obtained by making gcc unroll loops by using the ‘-funroll-loops’’ switch in addition to the ‘-O3’ switch.

7.3.2.5. Debugging

- `-g level` (Debugging level)
  - `-g1`

    It produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers.
  - `-g2`

    It is the default debugging level.
  - `-g3`

    It includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use -g3.

7.3.2.6. Warnings

- `-Wall`

  Show all warnings.
- `-Werror`

  Show warnings as errors.
- `-fsyntax-only`

  Check the code for syntax errors, but don't do anything beyond that.
- `-pedantic`

  Issue all the warnings demanded by strict ISO C, reject all programs that use forbidden extensions, and some other programs that do not follow ISO C. Valid ISO C programs should compile properly with or without this option (though a rare few will require -ansi or a -std option specifying the required version of ISO C). However, without this option, certain GNU extensions and traditional C features are supported as well. With this option, they are rejected.
- `-pedantic-errors`

  Pedantic warnings are produced as errors.
- `-w`

  Inhibit all warning messages.

7.3.2.7. Miscellaneous

- `-v`

  Verbose option. It prints (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.
- `-ansi`

  Support ANSI programs. This turns off certain features of GCC that are incompatible with ISO C90 (when compiling C code). For the C compiler, it disables recognition of C++ style // comments as well as the inline keyword. The -ansi option does not cause non-ISO programs to be rejected gratuitously. For that, -pedantic is required in addition to -ansi.

7.3.3. Linker Options

7.3.3.1. General

- `-Wl,option`

  Pass `option` as an option to the linker. If `option` contains commas, it is split into multiple options at the commas. You can use this syntax to pass an argument to the option. For example, `-Wl,-Map,output.map' passes `-Map output.map' to the linker.

- `-Wl, -nostartfiles`

  Do not use the standard system startup files when linking. The standard system libraries are used normally, unless `-nostdlib` or `-nodefaultlibs` is used.

- `-Wl,-nodefault`

  Do not use the standard system libraries when linking. Only the libraries you specify will be passed to the linker, options specifying linkage of the system libraries, such as `-static-libgcc` or `-shared-libgcc`, will be ignored. The standard start-up files are used normally, unless `-nostartfiles` is used. The compiler may generate calls to `memcmp, memset, memcpy and memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

- `-Wl,-nostdlib`

  Do not use the standard system start-up files or libraries when linking.

  One of the standard libraries bypassed by `-nostdlib` and `-nodefaultlibs` is `libgcc.a`, a library of internal subroutines that GCC uses to overcome shortcomings of particular machines, or special needs for some languages. In most cases, you need libgcc.a even when you want to avoid other standard libraries. In other words, when you specify `-nostdlib or -nodefaultlibs` you should usually specify -lgcc as well. This ensures that you have no unresolved references to internal GCC library subroutines.

- `-Wl,-s`

  Remove all symbol table and relocation information from the executable.

- `-Wl,-static`

  On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.

- `-Wl,-Map`

  Generates Map file.

7.3.3.2. Libraries

- `-Wl,-llibrary`

  Search the library named l*ibrary* when linking.

  It makes a difference where in the command you write this option; the linker searches and processes libraries and object files in the order they are specified. Thus, foo.o -lz bar.o searches library z after file foo.o but before bar.o.

  The linker searches a standard list of directories for the library, which is actually a file named liblibrary.a. The linker then uses this file as if it had been specified precisely by name.

- `-Wl, Ldir`

  Add directory dir to the list of directories to be searched for -l.

7.3.3.3. Optimization

- `-Wl, --gc-sections`

  Garbage collect unused sections.

  Enable garbage collection of unused input sections. It is ignored on targets that do not support this option. The default behavior (of not performing this garbage collection) can be restored by specifying `--no-gc-sections' on the command line. `--gc-sections' decides which input sections are used by examining symbols and relocations. The section containing the entry symbol and all sections containing symbols undefined on the command-line will be kept, as will sections containing symbols referenced by dynamic objects.

- `-funsafe-math-optimizations`

  Enable unsafe math optimizations.

- `-ffast-math`

  Enable fast math

- `-fpic`

  Generate position independent code.

7.3.4. Assembler Options

- `-I`

  Use this option to add a path to the list of directories as searches for files specified in .include directives (see .include). You may use -I as many times as necessary to include a variety of paths. The current working directory is always searched first; after that, as searches any `-I' directories in the same order as they were specified (left to right) on the command line.

- `-v`

  Announce version.

- `Debugging(-g)`

  Use this option to enable the debug level.

7.3.5. Preprocessing Assembler Options

- `-I`

  Use this option to add a path to the list of directories as searches for files specified in .include directives (see .include). You may use -I as many times as necessary to include a variety of paths. The current working directory is always searched first; after that, as searches any `-I' directories in the same order as they were specified (left to right) on the command line.

- `-v`

  Announce version.

- `Debugging(Wa,-g)`

  Use this option to enable the debug level.

7.3.6. Archiver Options

- `-r`

  Replace existing or insert new file(s) into the archive.

7.4. Binutils

The following ARM GNU Binutils are available:

- arm-none-eabi-ld - GNU linker.
- arm-none-eabi-as - GNU assembler.
- arm-none-eabi-addr2line - Converts addresses into filenames and line numbers.
- arm-none-eabi-ar - A utility for creating, modifying and extracting from archives.
- arm-none-eabi-c++filt - Filter to demangle encoded C++ symbols.
- arm-none-eabi-nm - Lists symbols from object files.
- arm-none-eabi-objcopy - Copies and translates object files.
- arm-none-eabi-objdump - Displays information from object files.
- arm-none-eabi-ranlib - Generates an index to the contents of an archive.
- arm-none-eabi-readelf - Displays information from any ELF format object file.
- arm-none-eabi-size - Lists the section sizes of an object or archive file.
- arm-none-eabi-strings - Lists printable strings from files.
- arm-none-eabi-strip - Discards symbols.

For more information about each util, use the built in help command: `<util name here> --help`.

7.5. AVR Compiler and Toolchain Options: GUI

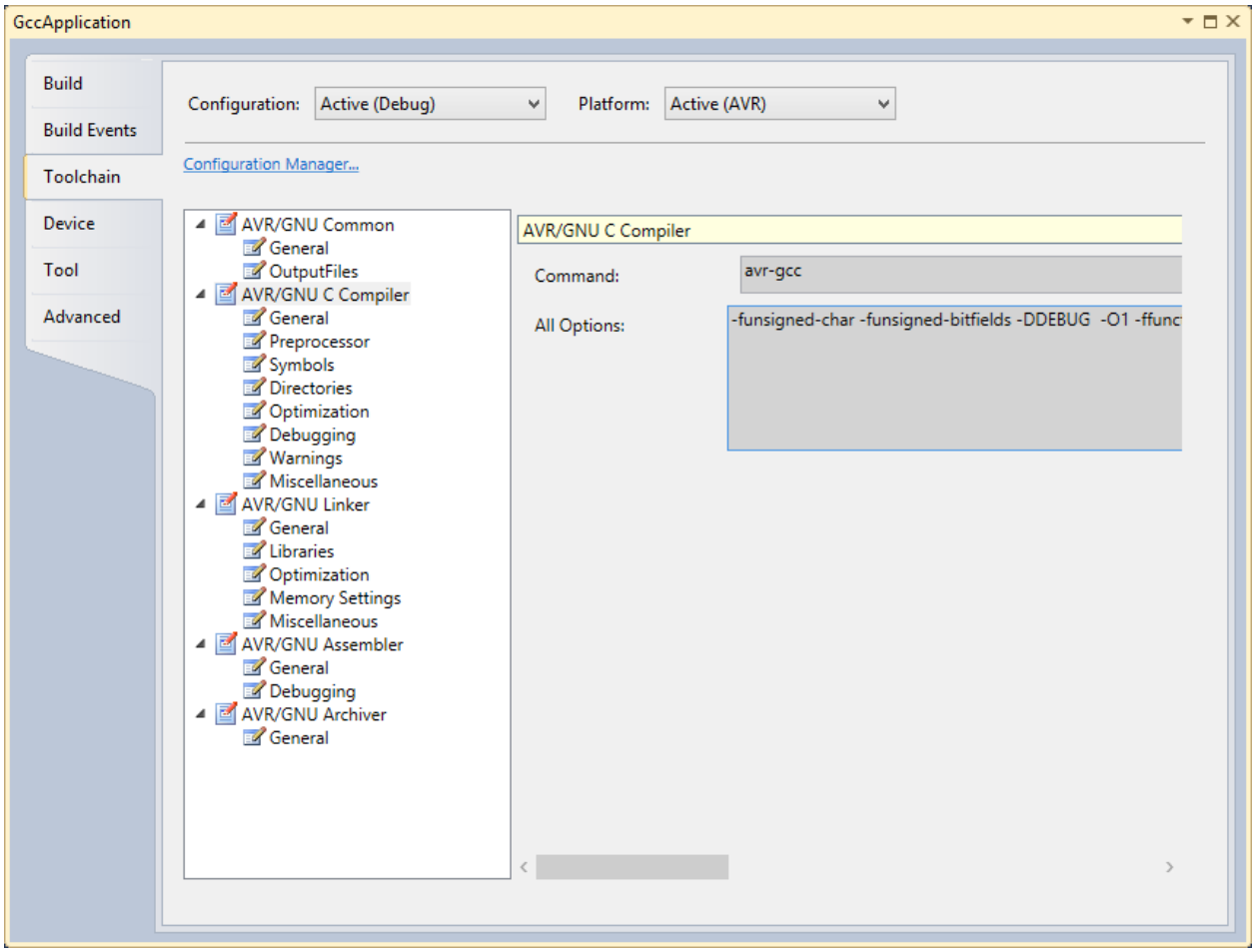
To get help about AVR GNU toolchain, you can do the following:

- For information about avr32-gcc usage in Atmel Studio and general parameters consult the GCC Project Options and Configuration section
- The API reference for the AVR libc implementation can be found here

  The API Alphabetical index can be consulted here
- For general information about GCC, visit the official GNU GCC web site
- Alternatively you can write `avr32-gcc --help` and see explanations on some of the parameters in the command output

This section illustrates the GUI options that are available for the AVR GNU toolchain from the Atmel Studio frontend.

Figure 7-2. AVR GNU Toolchain Options



AVR GNU C Compiler Options

Table 7-6. AVR GNU C Compiler Options

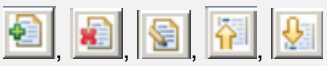
Option	Description
General options	
-mcall-prologues	Use subroutines for functions prologues and epilogues
-mno-interrupts	Change stack pointer without disabling interrupts
-funsigned-char	Default char type is unsigned
-funsigned-bitfield	Default bit field is unsigned
Preprocessor options	
-nostdinc	Do not search system directories
-E	Preprocess only
Symbols options	

Option	Description
There one can define (-D) or undefine (-U) a number of in-source symbols. New symbol declarations can be added, modified, or reordered, using the interface buttons below: • Add a new symbol. This and all following icons are reused with the same meaning in other parts of Atmel Studio interface. • Remove a symbol. • Edit symbol. • Move the symbol up in the parsing order. • Move the symbol down in the parsing order.	
Include directories	
Contains all the included header and definition directories, can be modified, using the same interface as symbols.	
Optimization options	
Optimization level (drop down menu): -O0, -O1, -O2, -O3, -Os	No optimization, optimize for speed (level 1 - 3), optimize for size
Other optimization flags (manual input form)	Here you should write optimization flags specific for the platform and your requirements
-ffunction-sections	Prepare functions for garbage collection, if a function is never used, its memory will be scrapped
-fpack-struct	Pack structure members together
-fshort-enums	Allocate only as many bytes needed by the enumerated types
-mshort-calls	Use rjmp/rcall limited range instructions on the >8K devices
Debug options	
Debug level (drop down menu): none, -g1, -g2, -g3	Specifies the level of tracing and debugging code and headers left or inserted in the source code
Other debug options (form field)	Architecture specific debug options

Option	Description
Warning messages output options	
-Wall	All warnings
-Werror	Escalate warnings to errors
-fsyntax-only	Check syntax only
-pedantic	Check conformity to GNU, raise warnings on non-standard programming practice
-pedantic-errors	Same as above, plus escalate warnings to errors
Miscellaneous options	
Other flags (form field)	Input other project-specific flags
-v	Verbose
-ansi	Support ANSI programs
-save-temps	Do not delete temporary files

AVR GCC Linker Options

Table 7-7. AVR GCC Linker Options

Option	Description
-Wl -nostartfiles	Do not use standard files
-Wl -nodefault	Do not use default libraries
-Wl -nostdlib	No start-up or default libraries
-Wl -s	Omit all symbol information
-Wl -static	Link statically
Libraries options	
Libraries -Wl, -l (form field)	You can add, prioritize or edit library names here, using those buttons: 
Library search path -Wl,-L (form field)	You can add, prioritize or edit path where the linker will search for dynamically linked libraries, same interface as above
Optimization options	
-Wl, -gc-sections	Garbage collect unused sections
--rodata-writable	Put read-only data in writable spaces
-mrelax	Relax branches

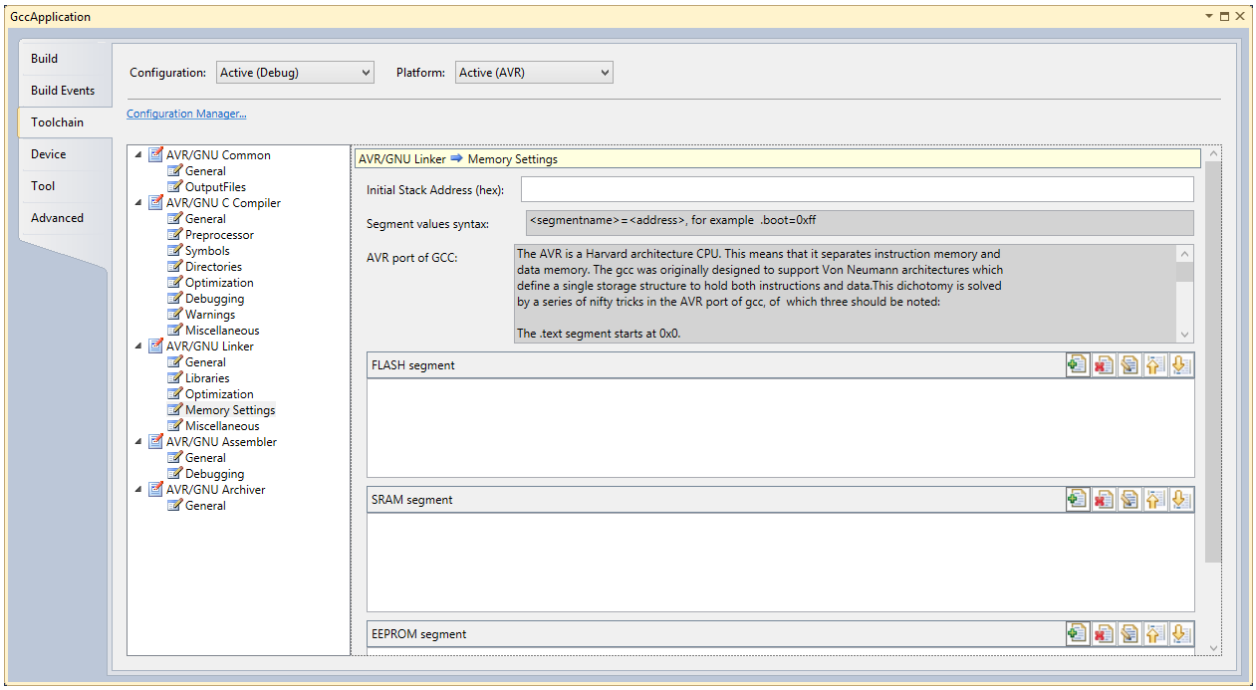
Option	Description
Miscellaneous options	
Other linker flags (form field)	Input other project-specific flags

Memory Settings

Displays a dialog where it is possible to configure memory segments. **(Syntax for specifying segment values: <segmentname> = <address>, for example boot=0xff)**

The address must be given as a hexadecimal number prefixed with 0x. It is interpreted as a word address for flash memory and a byte addresse for SRAM and EEPROM memory.

Figure 7-3. Memory settings



Notes about the AVR port of gcc

The AVR is a Harvard architecture CPU. This means that it separates instruction memory and data memory. The gcc was originally designed to support Von Neumann architectures which define a single storage structure to hold both instructions and data. This dichotomy is solved by a series of nifty tricks in the AVR port of gcc, of which three should be noted:

- The .text segment starts at 0x0
- The .data segment starts at 0x800000
- The .eeprom segment starts at 0x810000

These peculiarities have been abstracted away by the GUI , but users will see the truth when building projects with relocated segments.

A relocation definition for flash will be passed to the GNU linker via avr-gcc as the option:

- `-Wl,-section-start=bootloader=0x1fc00`

Note that the address has been multiplied by 2 to get the byte address.

A relocation definition for the .data section will be passed as:

- `-Wl,-section-start=anewdatasegment=0x800`

AVR Assembler Options

Table 7-8. AVR Assembler Options

Option	Description
Optimization options	
Assembler flags (form field)	Miscellaneous assembler flags
Include path (form field)	You can add, prioritize or edit path to the architecture and platform specific included files here
-v	Announce version in the assembler output
Debugging options	
Debugging (drop down menu) None, -Wa -g	Enables debugging symbol and debugging source insertion

7.6. Commonly Used Options

7.6.1. Compiler Options

7.6.1.1. General

- `-funsigned-char`

  Each kind of machine has a default for what char should be. It is either like unsigned char by default or like signed char by default. This option says that the default char type is unsigned.
- `-funsigned-bitfields`

  These options control whether a bit-field is signed or unsigned, when the declaration does not use either signed or unsigned. This options says that the default bitfield type is unsigned.

7.6.1.2. Preprocessor

- `-nostdinc`

  Do not search the standard system directories for header files. Only the directories you have specified with -I options (and the directory of the current file, if appropriate) are searched.
- `-E`

  Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output. Input files which don't require preprocessing are ignored.

7.6.1.3. Symbols

- `-D`
  - -D *name*

    Predefine name as a macro, with definition 1.

    E.g.:
  - -D *name=value*

Predefine *name* as a macro, with definition *value*. The contents of definition are tokenized and processed as if they appeared during translation phase three in a #define directive. In particular, the definition will be truncated by embedded newline characters.

- `-U`

  Cancel any previous definition of name, either built in or provided with a -D option.

-D and -U options are processed in the order they are given on the command line. All -imacros file and -include file options are processed after all -D and -U options.

7.6.1.4. Directories

- `-I dir`

  Add the directory dir to the list of directories to be searched for header files. Directories named by -I are searched before the standard system include directories. If the directory dir is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated .

7.6.1.5. Optimization

- There is a general switch '-O<optimization_level>' which specifies the level of optimization used when generating the code:
  - `-Os`

    Signal that the generated code should be optimized for code size. The compiler will not care about the execution performance of the generated code.
  - `-O0`

    No optimization. This is the default. GCC will generate code that is easy to debug but slower and larger than with the incremental optimization levels outlined below.
  - `-O1 or -O`

    This will optimize the code for both speed and size. Most statements will be executed in the same order as in the C/C++ code and most variables can be found in the generated code. This makes the code quite suitable for debugging.
  - `-O2`

    Turn on most optimizations in GCC except for some optimizations that might drastically increase code size. This also enables instruction scheduling, which allows instructions to be shuffled around to minimize CPU stall cycles because of data hazards and dependencies, for CPU architectures that might benefit from this. Overall this option makes the code quite small and fast, but hard to debug.
  - `-O3`

    Turn on some extra performance optimizations that might drastically increase code size but increase performance compared to the -O2 and -O1 optimization levels. This includes performing function inlining
- Other optimization options
  - `-ffunction-sections`
  - `-fdata-sections`

    Place each function or data item into its own section in the output file if the target supports arbitrary sections. The name of the function or the name of the data item determines the section's name in the output file.

Only use these options when there are significant benefits from doing so. When you specify these options, the assembler and linker will create larger object and executable files and will also be slower.

- `-funroll-loops`

  If code size is not a concern then some extra performance might be obtained by making gcc unroll loops by using the '-funroll-loops" switch in addition to the '-O3' switch.

7.6.1.6. Debugging

- `-g level` (Debugging level)
  - `-g1`

    It produces minimal information, enough for making back-traces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers.
  - `-g2`

    It is the default debugging level.
  - `-g3`

    It includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use -g3.

7.6.1.7. Warnings

- `-Wall`

  Show all warnings.
- `-Werror`

  Show warnings as errors.
- `-fsyntax-only`

  Check the code for syntax errors, but don't do anything beyond that.
- `-pedantic`

  Issue all the warnings demanded by strict ISO C, reject all programs that use forbidden extensions, and some other programs that do not follow ISO C. Valid ISO C programs should compile properly with or without this option (though a rare few will require -ansi or a -std option specifying the required version of ISO C). However, without this option, certain GNU extensions and traditional C features are supported as well. With this option, they are rejected.
- `-pedantic-errors`

  Pedantic warnings are produced as errors.
- `-w`

  Inhibit all warning messages.

7.6.1.8. Miscellaneous

- `-v`

  Verbose option. It prints (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.
- `-ansi`

  Support ANSI programs. This turns off certain features of GCC that are incompatible with ISO C90 (when compiling C code). For the C compiler, it disables recognition of C++ style // comments as

well as the inline keyword. The -ansi option does not cause non-ISO programs to be rejected gratuitously. For that, -pedantic is required in addition to -ansi.

7.6.2. Linker Options

7.6.2.1. General

- `-Wl,option`

  Pass `option` as an option to the linker. If `option` contains commas, it is split into multiple options at the commas. You can use this syntax to pass an argument to the option. For example, `-Wl,-Map,output.map' passes `-Map output.map' to the linker.

- `-Wl, -nostartfiles`

  Do not use the standard system startup files when linking. The standard system libraries are used normally, unless `-nostdlib` or `-nodefaultlibs` is used.

- `-Wl,-nodefault`

  Do not use the standard system libraries when linking. Only the libraries you specify will be passed to the linker, options specifying linkage of the system libraries, such as `-static-libgcc` or `-shared-libgcc`, will be ignored. The standard start-up files are normally used, unless `-nostartfiles` is used. The compiler may generate calls to `memcmp, memset, memcpy, and memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

- `-Wl,-nostdlib`

  Do not use the standard system start-up files or libraries when linking.

  One of the standard libraries bypassed by `-nostdlib` and `-nodefaultlibs` is `libgcc.a`, a library of internal subroutines that GCC uses to overcome shortcomings of particular machines, or special needs for some languages. In most cases, you need libgcc.a even when you want to avoid other standard libraries. In other words, when you specify `-nostdlib or -nodefaultlibs` you should usually specify -lgcc as well. This ensures that you have no unresolved references to internal GCC library subroutines.

- `-Wl,-s`

  Remove all symbol table and relocation information from the executable.

- `-Wl,-static`

  On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.

7.6.2.2. Libraries

- `-Wl,-llibrary`

  Search the library named l*ibrary* when linking.

  It makes a difference where in the command you write this option; the linker searches and processes libraries and object files in the order they are specified. Thus, foo.o -lz bar.o searches library z after file foo.o but before bar.o.

  The linker searches a standard list of directories for the library, which is actually a file named liblibrary.a. The linker then uses this file as if it had been specified precisely by name.

- `-Wl, Ldir`

  Add directory dir to the list of directories to be searched for -l.

7.6.2.3. Optimization

- `-Wl, --gc-sections`

  Garbage collect unused sections.

  Enable garbage collection of unused input sections. It is ignored on targets that do not support this option. The default behavior (of not performing this garbage collection) can be restored by specifying `--no-gc-sections' on the command line. `--gc-sections' decides which input sections are used by examining symbols and relocations. The section containing the entry symbol and all sections containing symbols undefined on the command-line will be kept, as will sections containing symbols referenced by dynamic objects.

- `--rodata-writable`

  Put read-only data in writable data section.

7.6.3. Assembler Options

- `-I`

  Use this option to add a path to the list of directories as searches for files specified in .include directives (see .include). You may use -I as many times as necessary to include a variety of paths. The current working directory is always searched first; after that, as searches any `-I' directories in the same order as they were specified (left to right) on the command line.

- `-v`

  Announce version.

7.7. 8-bit Specific AVR GCC Command Line Options

This section describes the options specific to AVR 8-bit Toolchain.

7.7.1. AVR C Compiler

7.7.1.1. General

- `-mcall-prologues`

  Functions prologues/epilogues are expanded as call to appropriate subroutines. Code size will be smaller.

- `-mno-interrupts`

  Change the stack pointer without disabling interrupts. Generated code is not compatible with hardware interrupts. Code size will be smaller.

- `-mno-tablejump`

  Do not generate table jump instructions (removed from gcc 4.5.1 coz same as -fno-jump-tables).

- `-msize`

  Output instruction sizes to the asm file (removed from avr-gcc coz same as using -dp switch which prints the instruction length).

7.7.1.2. Optimization

- `-fpack-struct`

  Without a value specified, pack all structure members together without holes. When a value is specified (which must be a small power of two), pack structure members according to this value, representing the maximum alignment (that is, objects with default alignment requirements larger than this will be output potentially unaligned at the next fitting location).

- `-fshort-enums`

  Allocate to an enum type only as many bytes as it needs for the declared range of possible values. Specifically, the enum type will be equivalent to the smallest integer type which has enough room.
- `-mshort-calls`

  Use rjmp/rcall (limited range) on >8K devices.

7.7.1.3. Miscellaneous

- `-save-temps`

  Do not delete temporary files. Store the usual "temporary" intermediate files permanently; place them in the current directory and name them based on the source file. Thus, compiling foo.c with -c -save-temps would produce files foo.i and foo.s, as well as foo.o. This creates a preprocessed foo.i output file even though the compiler now normally uses an integrated preprocessor.

7.7.2. AVR C Linker

7.7.2.1. Optimization

- `-mrelax`

  Relax branches. Linker relaxing is enabled in the linker by passing the '—relax' option to the linker. Using GCC as a frontend for the linker, this option is automatically passed to the linker when using '-O2' or '-O3' or explicitly using the '-mrelax' option. When this option is used, GCC outputs pseudo instructions like lda.w, call etc. The linker can then, if the input file is tagged as relaxable, convert a pseudo instruction into the best possible instruction with regards to the final symbol address.

7.8. 32-bit Specific AVR GCC Command Line Options

7.8.1. Optimization

- `-mfast-float`

  The switch, causes fast, non-ieee compliant versions of some of the optimized AVR 32-bit floating-point library functions to be used. This switch is by default enabled if the '-ffast-math' switch is used.
- `-funsafe-math-optimizations`

  Allow optimizations for floating-point arithmetic that (a) assume that arguments and results are valid and (b) may violate IEEE or ANSI standards. When used at link-time, it may include libraries or start-up files that change the default FPU control word or other similar optimizations. This option is not turned ON by any '-O' option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications. Enables '-fno-signed-zeros', '-fno-trapping-math', '-fassociative-math' and '-freciprocal-math'.
- `-ffast-math`

  This option causes the preprocessor macro __FAST_MATH__ to be defined. This option is not turned on by any '-O' option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications. It sets '-fno-math-errno', '-funsafe-math-optimizations', '-ffinite-math-only', '-fno-rounding-math', '-fno-signaling-nans' and '-fcx-limited-range'.
- `-fpic`

Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that '-fpic' does not work; in that case, recompile with '-fPIC' instead. (These maximums are 8k on the SPARC and 32k on the m68k and RS/6000. The 386 has no such limit.) Position-independent code requires special support, and therefore works only on certain machines. For the 386, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent. When this flag is set, the macros __pic__ and __PIC__ are defined to 1.

- `-mno-init-got`

  Do not initialize GOT register before using it when compiling PIC code.

- `-masm-addr-pseudos`

  This option is enabled by default and causes GCC to output the pseudo instructions call and lda.w for calling direct functions and loading symbol addresses respectively. It can be turned OFF by specifying the switch '-mno-asm-addr-pseudos'. The advantage of using these pseudo-instructions is that the linker can optimize these instructions at link time if linker relaxing is enabled. The '-mrelax' option can be passed to GCC to signal to the assembler that it should generate a relaxable object file.

- `-mforce-double-align`

  Force double-word alignment for double-word memory accesses.

- `-mimm-in-const-pool`

  When GCC needs to move immediate values not suitable for a single move instruction into a register, it has two possible choices; it can put the constant into the code somewhere near the current instruction (the constant pool) and then use a single load instruction to load the value or it can use two immediate instruction for loading the value directly without using a memory load. If a load from the code memory is faster than executing two simple one-cycle immediate instructions, then putting these immediate values into the constant pool will be most optimal for speed. This is often true for MCU architectures implementing an instruction cache, whereas architectures with code executing from internal flash will probably need several cycles for loading values from code memory. By default GCC will use the constant pool for AVR 32-bit products with an instruction cache and two immediate instructions for flash-based MCUs. This can be overridden by using the option '-mimm-in-const-pool' or its negated option '-mno-imm-in-const-pool'.

- `-muse-rodata-sections`

  By default GCC will output read-only data into the code (.text) section. If the code memory is slow it might be more optimal for performance to put read-only data into another faster memory, if available. This can be done by specifying the switch '-muse-rodata-section', which makes GCC put read-only data into the .rodata section. Then the linker file can specify where the content of the .rodata section should be placed. For systems running code from flash this might however mean that the read-only data must be placed in flash and then copied over to another memory at start-up, which means that extra memory usage is required with this scheme.

7.8.2. Debugging

- `-pg`

  Generate extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking.

- `-p`

  Generate extra code to write profile information suitable for the analysis program prof. You must use this option when compiling the source files you want data about, and you must also use it when linking.

7.8.3. AVR32 C Linker

7.8.3.1. Optimization

- `-mfast-float`

  Enable fast floating-point library. Enabled by default if the -funsafe-math-optimizations switch is specified.

- `-funsafe-math-optimizations`

  Allow optimizations for floating-point arithmetic that (a) assume that arguments and results are valid and (b) may violate IEEE or ANSI standards. When used at link-time, it may include libraries or start-up files that change the default FPU control word or other similar optimizations. This option is not turned on by any '-O' option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications. Enables '-fno-signed-zeros', '-fno-trapping-math', '-fassociative-math', and '-freciprocal-math'. The default is '-fno-unsafe-math-optimizations'.

- `-ffast-math`

  This option causes the preprocessor macro __FAST_MATH__ to be defined. This option is not turned on by any '-O' option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications. It sets '-fno-math-errno', '-funsafe-math-optimizations', '-ffinite-math-only', '-fno-rounding-math', '-fno-signaling-nans', and '-fcx-limited-range'.

- `-fpic`

  Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that '-fpic' does not work; in that case, recompile with '-fPIC' instead. (These maximums are 8k on the SPARC and 32k on the m68k and RS/6000. The 386 has no such limit.) Position-independent code requires special support, and therefore works only on certain machines. For the 386, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent. When this flag is set, the macros __pic__ and __PIC__ are defined to 1.

- `-Wl,--direct-data`

  Allow direct data references when optimizing. To enable the linker to convert an lda.w into an immediate move instruction, i.e. linker relaxing, the option '—direct-data' must be given to the linker.

7.8.3.2. Miscellaneous

- `-Xlinker[option]`

  Pass option as an option to the linker. You can use this to supply system-specific linker options which GCC does not know how to recognize. If you want to pass an option that takes a separate argument, you must use -Xlinker twice, once for the option and once for the argument. For example, to pass -assert definitions, you must write `-Xlinker -assert -Xlinker definitions'. It does not

work to write -Xlinker "-assert definitions", because this passes the entire string as a single argument, which is not what the linker expects. When using the GNU linker, it is usually more convenient to pass arguments to linker options using the option=value syntax than as separate arguments. For example, you can specify `-Xlinker -Map=output.map' rather than `-Xlinker -Map -Xlinker output.map'. Other linkers may not support this syntax for command-line options.

7.9. Binutils

The following AVR 32-bit GNU Binutils are available:

- avr32-ld- GNU linker.
- avr32-as - GNU assembler.
- avr32-addr2line - Converts addresses into file-names and line numbers.
- avr32-ar - A utility for creating, modifying and extracting from archives.
- avr32-c++filt - Filter to demangle encoded C++ symbols.
- avr32-nm - Lists symbols from object files.
- avr32-objcopy - Copies and translates object files.
- avr32-objdump - Displays information from object files.
- avr32-ranlib - Generates an index to the contents of an archive.
- avr32-readelf - Displays information from any ELF format object file.
- avr32-size - Lists the section sizes of an object or archive file.
- avr32-strings - Lists printable strings from files.
- avr32-strip - Discards symbols.

For more information about each util, use the built in help command: `avr32-<util name here> --help`.

- For general information about GNU Assembler (GAS), GNU linker and other binutils, visit the official GNU Binutils web site.

8. Extending Atmel Studio

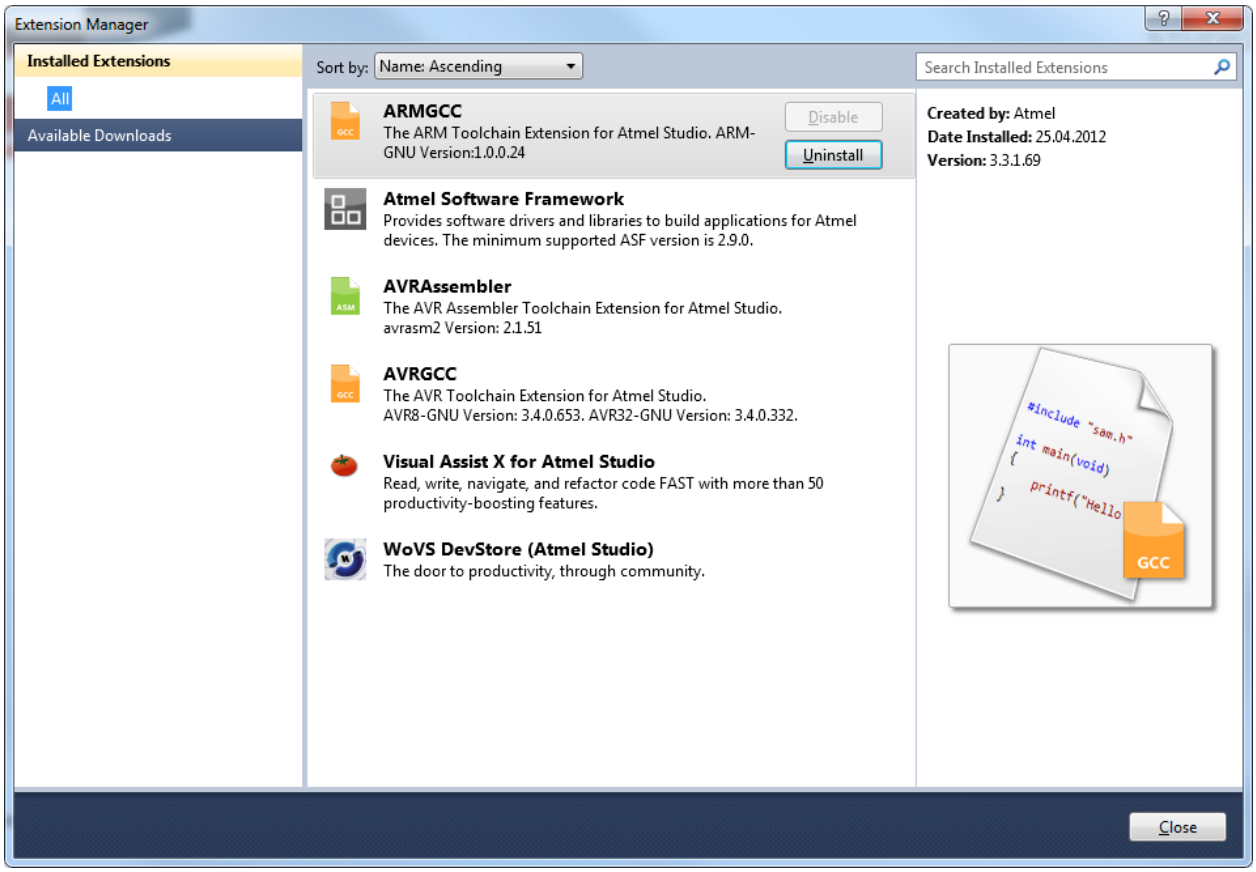
Atmel Studio includes a tool named Extension Manager that lets you add, remove, enable, and disable Atmel Studio extensions. To open Extension Manager, on the Tools menu, click **Extension Manager.**

Extension developers are advised to uninstall previous versions of extensions in progress, and uninstall or disable potentially conflicting extensions to prevent conflicts during development.

8.1. Extension Manager UI

The Extension Manager window is divided into three panes. The left pane lets you select by group: installed extensions and new extensions from the online gallery.

Figure 8-1. Extension Manager



The extensions are displayed in the middle pane. You can sort the list by name or author from the combobox above the list.

When you select an extension in the middle pane, information about it appears in the right pane. Extension installed by the current user can be uninstalled or disabled, extensions distributed with Atmel Studio cannot be changed.

The Extension Manager window also includes a search box. Depending on the selection in the left pane, you can search installed extensions, the online gallery, or available updates.

Online Gallery Extension Manager can install extensions from the Atmel Studio Gallery. These extensions may be packages, templates, or other components that add functionality to Atmel Studio.

To get started with the extension manager check the Installing New Extensions in Atmel Studio.

Extension Types

Extension Manager supports extensions in the VSIX package format, which may include project templates, item templates, toolbox items, Managed Extension Framework (MEF) components, and VSPackages. Extension Manager can also download and install MSI-based extensions, but it cannot enable or disable them. Atmel Studio Gallery contains both VSIX and MSI extensions.

Dependency Handling

If a user tries to install an extension that has dependencies, the installer verifies that those dependencies are already installed. If they are not installed, Extension Manager shows the user a list of dependencies that must be installed before the extension can be installed.

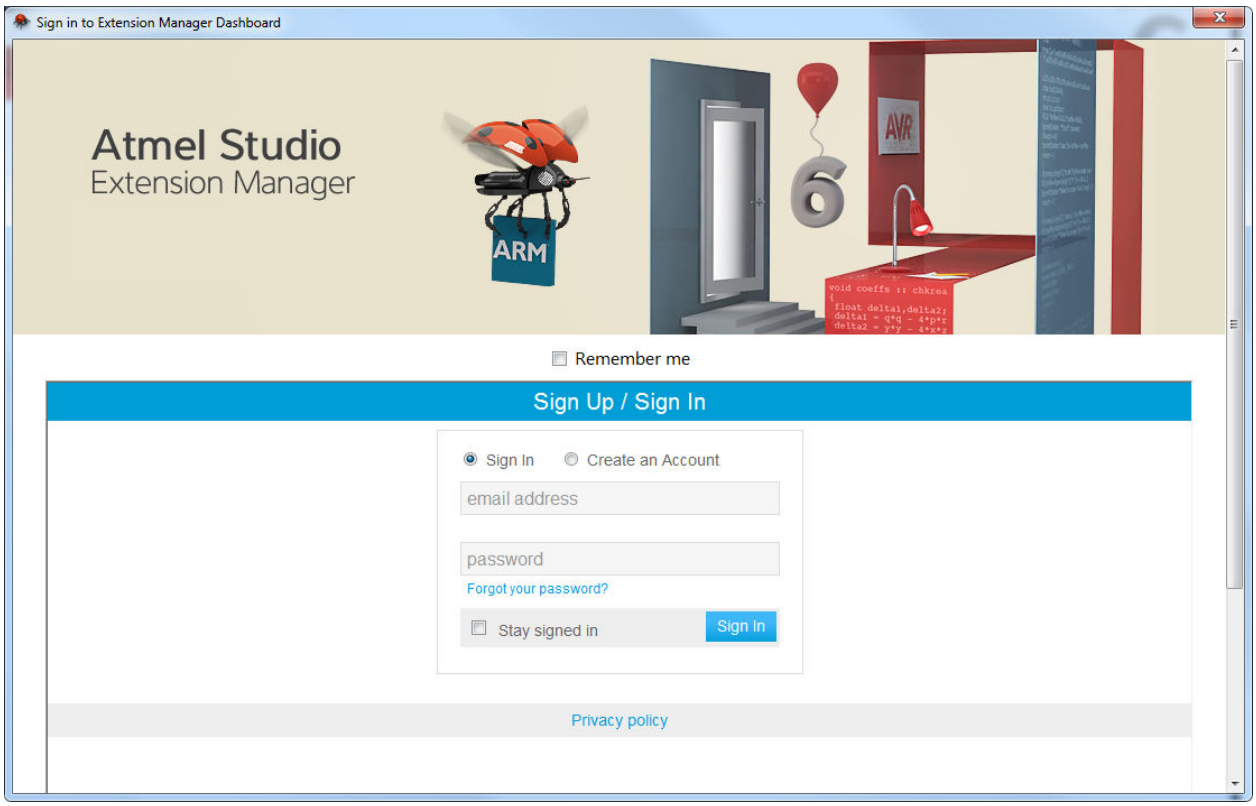
Installing Without Using Extension Manager

Extensions that have been packaged in .vsix files may be available in locations other than the Atmel Studio Gallery. Extension Manager cannot detect these files. However, you can install a .vsix file by double-clicking it and then following the setup instructions. When the extension is installed, you can use Extension Manager to enable it, disable it, or remove it.

8.2. Registering at Atmel Extension Gallery

In order to download extensions, registering at the Atmel Extension Gallery is required. The first time Updates are accessed or a download is invoked, this login screen is displayed:

Figure 8-2. Extension Manager Registration

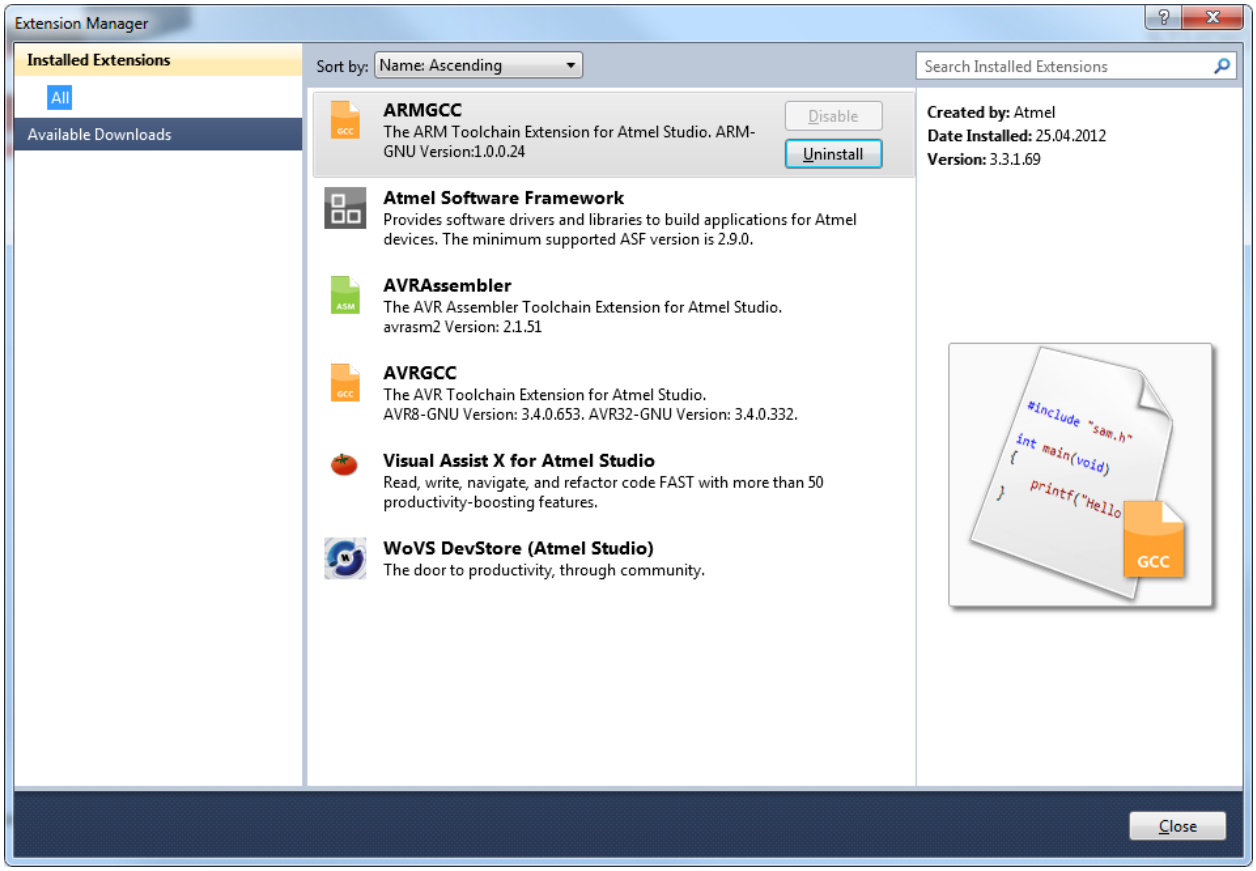


Follow the instructions on the screen.

8.3. Installing New Extensions in Atmel Studio

Step 1

Figure 8-3. Extension Manager

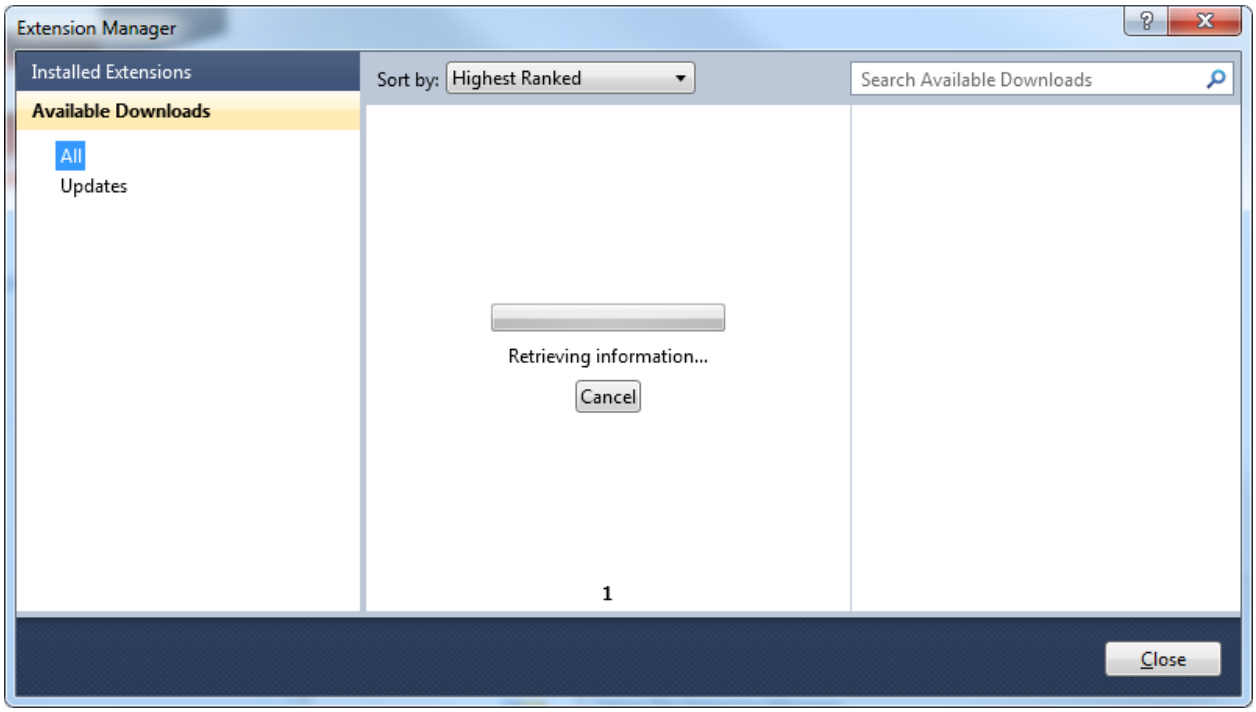


Opening the extension manager window will show extensions installed.

In order to find and install a new extension, click the Available Downloads tab on the left pane.

Step 2

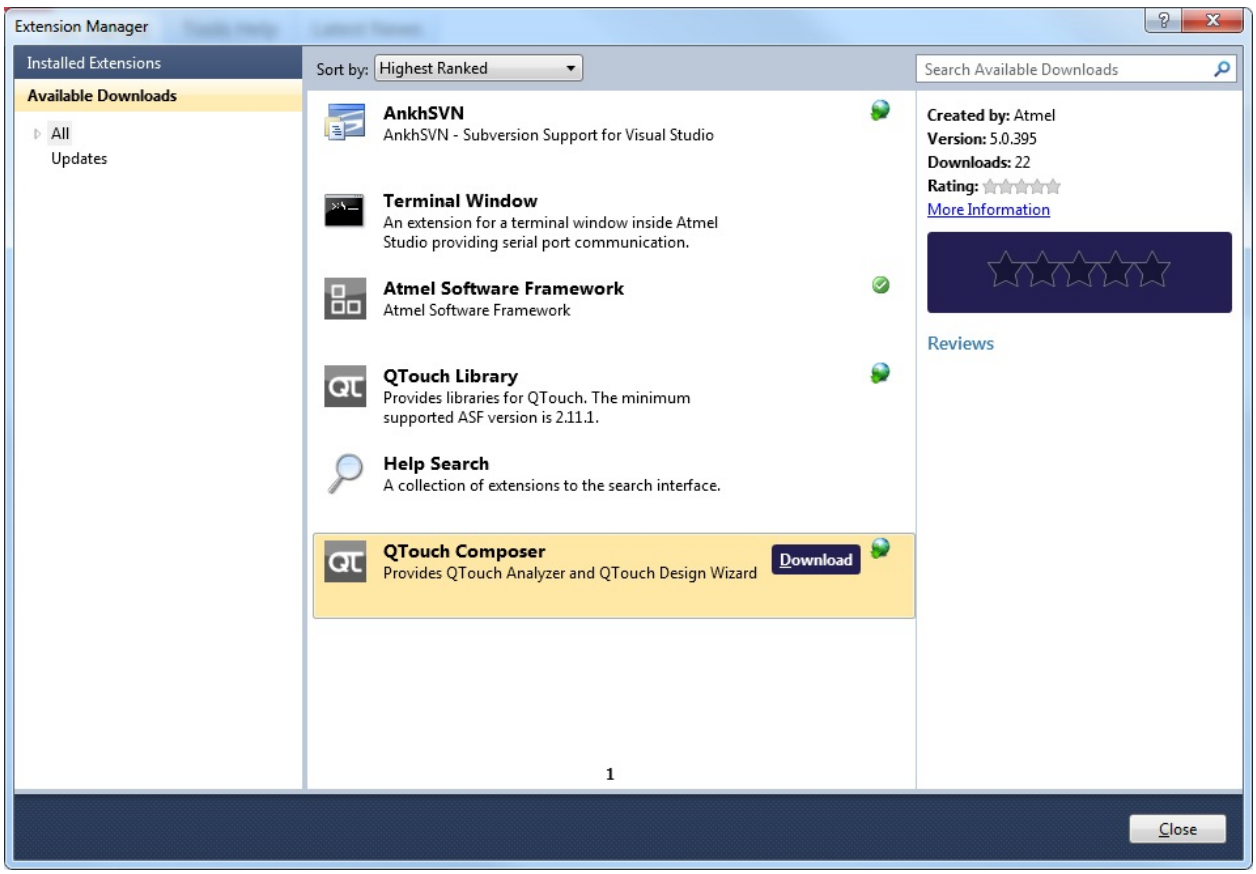
Figure 8-4. Retrieving List of Extensions



Updating the available extension list will take some time.

Step 3

Figure 8-5. List of Extensions

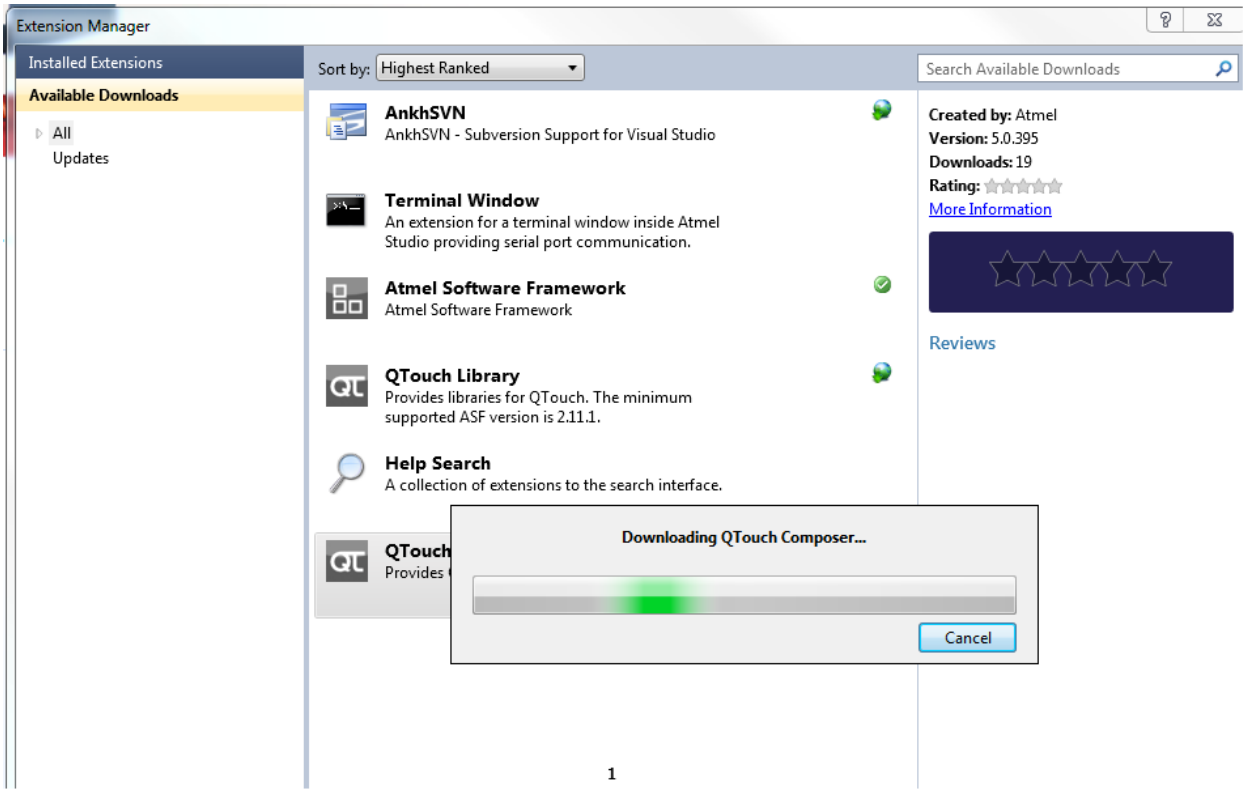


A green check mark identifies already installed extensions.

Select QTouch Composer and press Download. If you have not previously registered as a user in the Extension Gallery you will be taken to the Registering at Atmel Extension Gallery at this point.

.

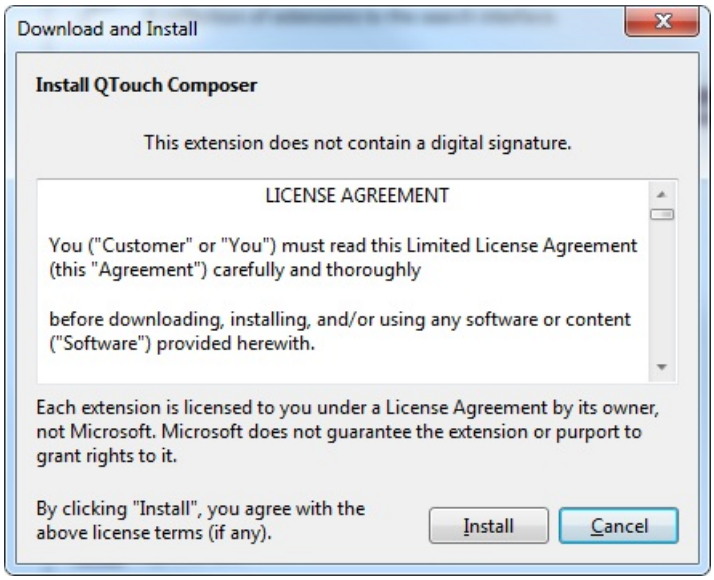
Figure 8-6. Extension Download Progression



Download will start as indicated in the status bar of Atmel Studio. If the extension is distributed as a standalone installer you will be asked for location to save the file. Downloading can take several minutes for large files. A dialog with a running bar is displayed during download. Not that download can take a long time for large extensions. Press Cancel to abort the download.

Step 4

Figure 8-7. Extension License



A license agreement will appear for you to read, most of the times, when you install a new extension.

Read it carefully and install only the extensions you really need as most of the extensions' authors do not take liability in a possible malfunction resulting from installation of mutually incompatible extensions and collateral damages, for example if extension security is breached.

Step 5

Once the extension is downloaded a message in the lower status bar will appear.

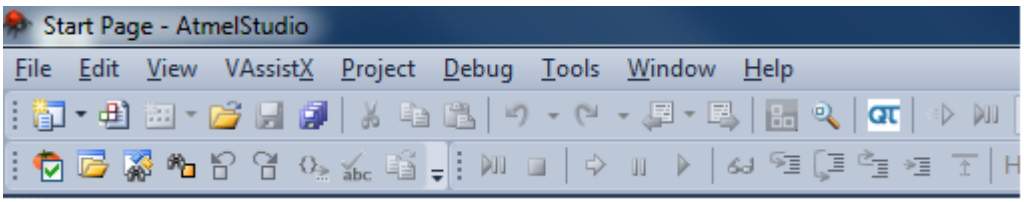
Figure 8-8. Extension Manager Restart Warning

Click the **Restart Now** button to restart the IDE immediately, otherwise if you plan to restart it later - click the **Close** button.

If you have an unsaved project you will be requested to save the changes you made, before restarting.

Step 6

Figure 8-9. QTouch Composer Button



After restarting Atmel Studio, a new button is added for starting QTouch Composer.

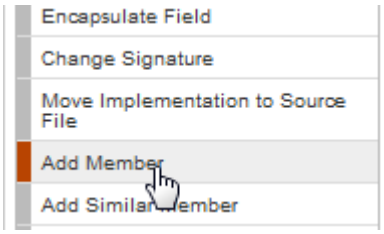
8.4. Visual Assist

The Atmel Studio comes with a preinstalled extension - the Visual Assist from WholeTomato Software.

The documentation on Visual Assist is available from several sources:

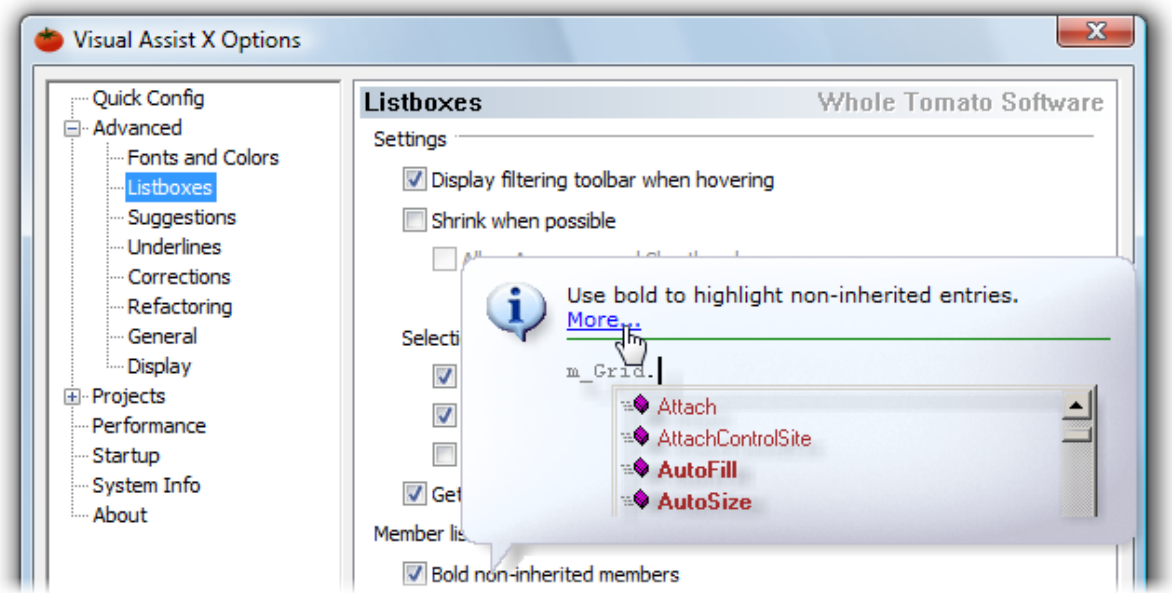
- Go to the www.wholetomato.com. Click in the left hand menu to browse documentation by feature.

Figure 8-10. WholeTomato Software Documentation

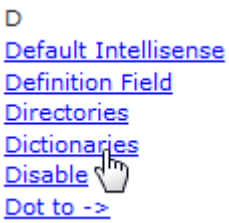


- Jump directly to relevant documentation using hyperlinks in the Visual Assist options dialog

Figure 8-11. Visual Assist Options



- Click terms in the Glossary



8.5. Overview of QTouch Composer and Library

The Atmel QTouch Composer and library allows you to easily and seamlessly develop capacitive touch functionality for your application. This simplifies the design process by tying together the tools required to edit the code in Atmel Studio and tune the touch design. QTouch Composer, formerly called QTouch Studio, is fully integrated in Atmel Studio 6 as an extension. QTouch Library is a software framework extension to Atmel studio, which allows you to add touch functionality on various Atmel devices.

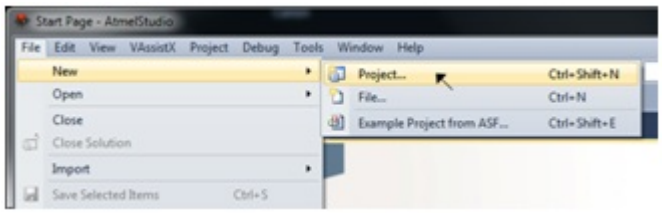
8.5.1. Installation

1. Start Atmel Studio.
2. Go to Tools → Extension Manager → Online Gallery.
3. Select QTouch Library and click “Download” and then install it.
4. Select QTouch Composer and click “Download” and then install it.
5. Click “Restart Now” button in the Extension manager window.
6. After starting Atmel Studio, go to Tools → Extension Manager. Check QTouch library and QTouch composer are listed and status is enabled.

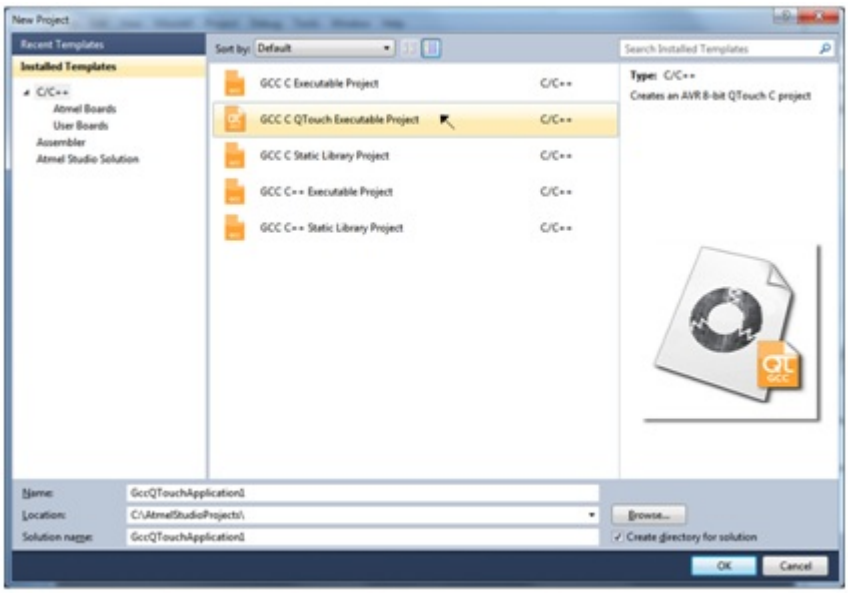
8.5.2. Overview of QTouch Project Builder

QTouch Project builder will guide you through all steps from selecting device and touch sensors to automatically generate a complete touch project.

1. Start Atmel Studio.
2. Open the File menu. Click on "New → Project".



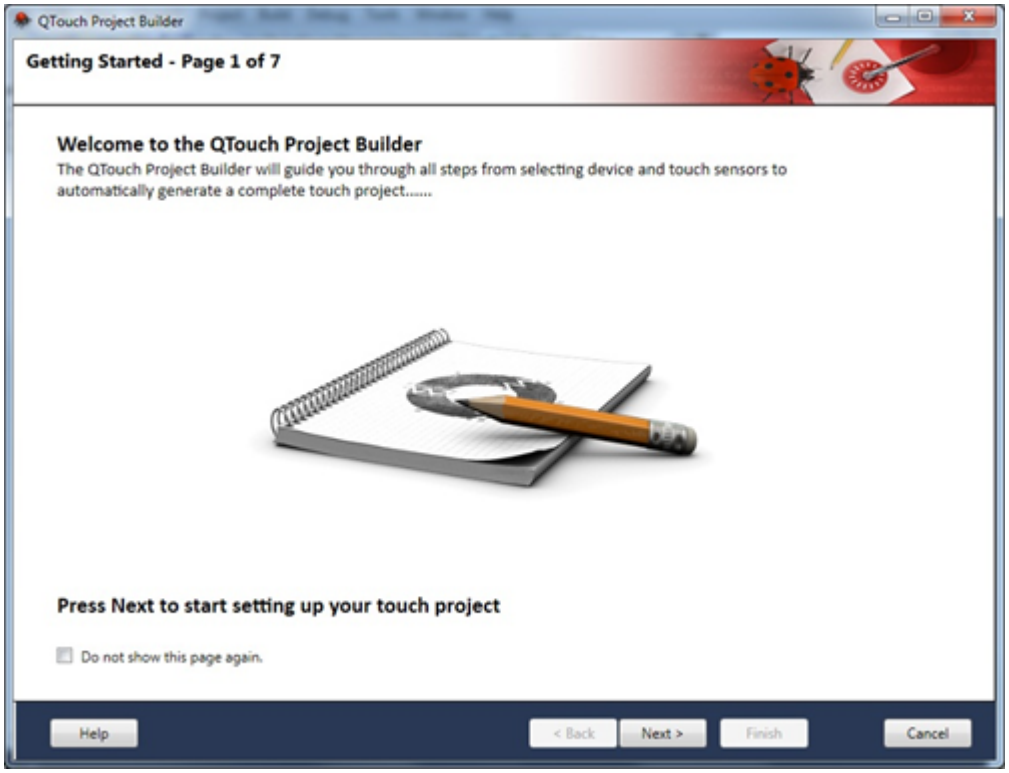
3. The “New Project” dialog is opened. Select “GCC C QTouch Executable Project” in the New Project dialog.



Enter the following details in the “New Project” dialog and click on the button OK.

- Name of the project
- Location of the project and solution
- Name of the solution

4. The “QTouch Project Builder” wizard is opened, which will guide through the steps involved in creating a project.



Sample QTouch Project creation video here.

8.5.3. Overview of QTouch Analyzer

QTouch Analyzer reads and interprets touch data sent from QTouch kit into different views. The Analyzer is separated into Kit View, Kit/Sensor Properties, Sensor Data, Trace View, Power View, and Graph view. When touch kit is connected and Atmel Studio is opened, QTouch Analyzer window opens up and connection information is updated.

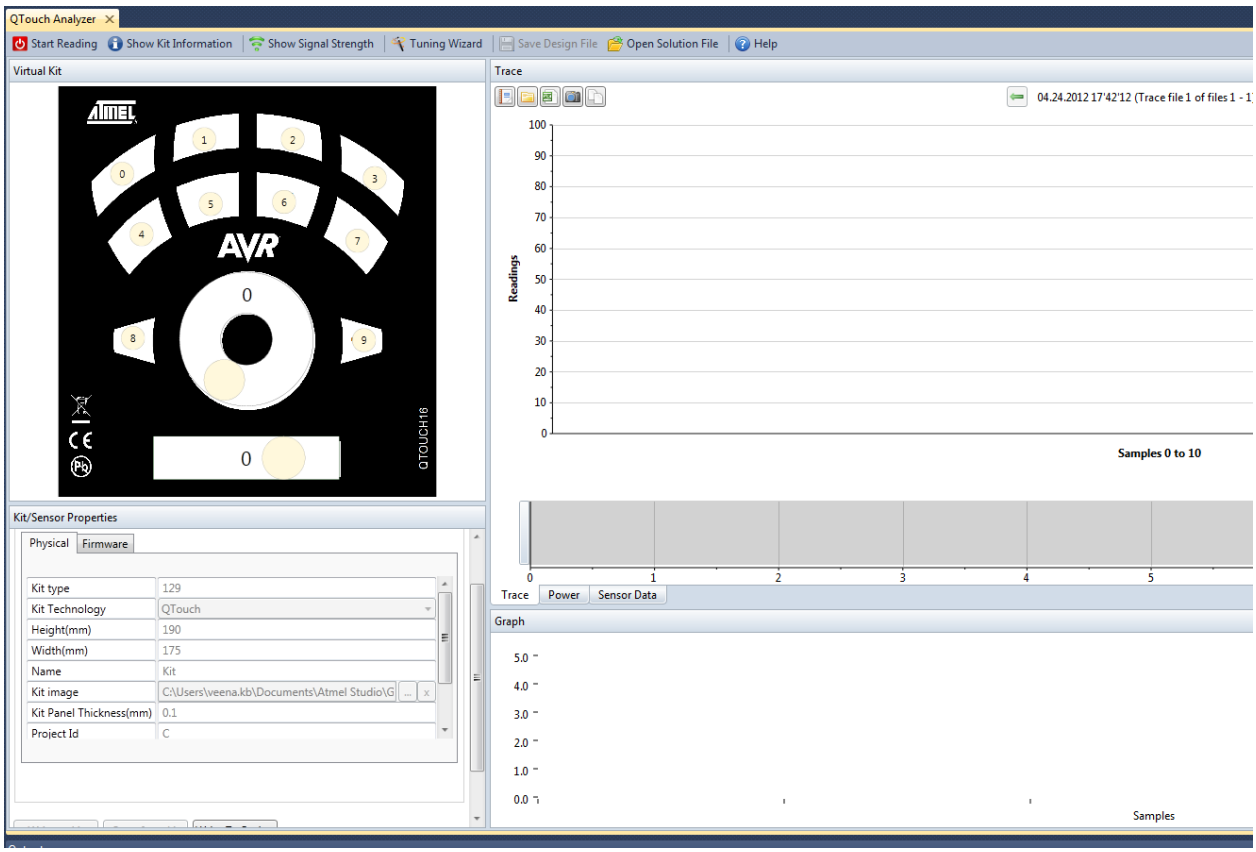
The Virtual Kit view shows touch events such as button press, wheel, and slider use. The image is updated based on the touch data read from the connected Touch Kit.

The Kit/Sensor Properties view allows you to view and modify the kit/sensor configuration options.

The Sensor Data View provides touch data information of the currently connected kit.

The Graph View displays one or more selected touch data's on a graph. Graph shall display most recent touch data. The data set to show can be selected from the data set list at the right side of the view. The datasets are displayed in tabbed pages representing the Signals, Deltas, References, and Wheel/Slider positions. Each data set selection list follows normal selection convention; click on an item in the list to selected that one item. To select a continuous range of items first click on first item then hold down the SHIFT key and select the last item in the range. Multiple items can also be selected one at a time by holding down the CTRL key prior to selecting the next item in the list. In the last case the items need not be in a continuous range. Using CTRL select method also allows deselection of individual items from a selection of multiple items.

The Trace contains one or more data series with touch data in a chart. The trace keeps all historical data of one single reading session (pressing start and stop reading), and the data can be saved in a separate file and opened again later.



8.6. Scripting Extensions

Atmel Studio provides some scripting hooks that can be executed automatically by the IDE. These extensions are mainly written in Python, and will execute for instance when a breakpoint is hit, when an expression is evaluated or when the program is being reset.

8.6.1. Debug Scripting

The debug scripting interface is function based and depends on certain, named functions to be defined in a Python® file. The function will then be called when the corresponding event is occurring inside Atmel Studio.

Attention:
Error checking is kept at a minimum for the functions exported into the Python environment so that the time used on initialization during normal sessions are kept low. This means that there are many ways to crash Atmel Studio through this interface.

To load a Python file, place a file named `debughooks.py` in the `Debug` folder of your project, next to the ELF file, or one folder up where the project file is. It is also possible to place this file inside the Atmel Studio installation directory to make the script load for all projects.

Note: The Python file is loaded and compiled when a project is launched, so changes to the Python file during a debug session will not be active until the next debug session is started. The Python file is running in an IronPython context, with full access to .NET and a Python 2.7 runtime. See http://ironpython.net/documentation/dotnet/ for more information of the runtime.

The functions that Atmel Studio will try to load is shown below with its function signature.

```
def should_process_breakpoint(studio_interface,
                              breakpoint_address,
                              breakpoint_id,
                              obj):
    """
    Called to determine if a breakpoint should cause Atmel Studio to enter debug mode.

    If this function returns False, Atmel Studio will not break at the breakpoint.
    """

    return True


def has_processed_breakpoint(studio_interface,
                             breakpoint_address,
                             breakpoint_id,
                             obj):
    """
    This function is called if Atmel Studio is breaking at a breakpoint.
    The GUI is now in halted mode.
    """

    pass


def on_reset(studio_interface,
             reset_address):
    """
    This function is called when the target is reset. The address
    where the reset went to is 'reset_address'.
    """

    pass


def on_eval_expr(studio_interface,
                 expression):
    """
    This function is called for each expression that is evaluated in Atmel Studio.

    This includes the watch window and other windows that show data from the target.
    Pass the 'expression' string through to evaluate it, or return another expression
    to be evaluated to override the expression. This override is not visible in the
    Atmel Studio GUI.
    """

    return expression
```

Note: Atmel Studio expects all these functions to be available if the script has been found and is loaded correctly. If for instance the `should_process_breakpoint` is undefined, breakpoints might start to misbehave as the return value of a undefined function is in itself undefined.

In the code shown above, the main interface back into the Atmel Studio is the `studio_interface` object. This object contains some functions to show messages and do target interaction.

The `Print` function in the `studio_interface` object is used to show text in the output window inside Atmel Studio. The function takes two arguments, the string to print and the name of the tab in the output window. The example below prints all evaluated expression to the “Expressions” tab.

```
def on_eval_expr(studio_interface, expression):
    studio_interface.Print("Evaluating {}".format(expression), "Expressions")
    return expression
```

Note: The severity level of text sent through `Print` is set to `INFO`, which means that the output may be masked by Atmel Studio. To lower the threshold, go to **Tools** > **Tools**, select **Status Management**, and set the **Display Threshold** to **INFO**.

The `ExecStmt` function in the `studio_interface` object is used to execute statements in the debugger. This can for instance be used to set variables. See MSDN Debugger.ExecuteStatement Method for more information.

The `WriteMemory` and `ReadMemory` are symmetric functions for reading and writing memory on the target. It is important to use a `System.Array[System.Byte]` object to pass the data between the script and Atmel Studio.

```
import System

def should_process_breakpoint(studio_interface,
                              breakpoint_address,
                              breakpoint_id,
                              obj):
    vals = System.Array[System.Byte]([1, 2, 3, 4, 5, 6, 7, 8, 9])

    studio_interface.WriteMemory(data=vals, adr=0, type="eeprom")

    ret = studio_interface.ReadMemory(adr=0, type="eeprom", count=9)

    studio_interface.Print("ret == vals => {!r}".format(ret == vals), "Python")
    return True
```

The `CalcNumericValue` is a shorthand for the `CalcValue` call. It will return the numeric value of the symbol or the provided default value if the function fails to retrieve the value of the symbol.

```
def should_process_breakpoint(studio_interface,
                              breakpoint_address,
                              breakpoint_id,
                              obj):
    a = studio_interface.CalcNumericValue("a", 0)
    if a == 0:
     studio_interface.Print("a was 0 or default", "Value scripts")
    else:
     studio_interface.Print("a = {}".format(a), "Value scripts")
    return True
```

The `CalcValue` function is used to retrieve information about a symbol in the scope where the target code is running. The return value of this call is a list of information, containing the address of the symbol, symbol information and value. The objects sent in this list contains all known information about a symbol, but the most useful field is the last element which contains the value of the evaluated symbol.

```
def should_process_breakpoint(studio_interface,
                              breakpoint_address,
                              breakpoint_id,
                              obj):
    a = studio_interface.CalcValue("a")
    # a now contains all information about the variable a.
    # It is a list with the following members:
    # a = [
    #   <Atmel.VsIde.AvrStudio.Services.TargetService.TCF.Services.ValueInfo>,
    #   <Atmel.VsIde.AvrStudio.Services.TargetService.TCF.Services.SymbolInfo>,
    #   <Atmel.VsIde.AvrStudio.Services.TargetService.TCF.Services.ExpressionInfo>,
    #   '1' ] <-- This is the value of the symbol as a string, here it had the value 1

    studio_interface.Print("Value of a = {}".format(a[3]), "Value Scripts")
    return True
```

Note: The different objects returned by the `CalcValue` call contains objects that are either internal, or documented in the Atmel Studio SDK. Use the python `dir()` command to look at the fields that are exported.

9. Menus and Settings

9.1. Customizing Existing Menus and Toolbars

You can add or remove commands on any menu or toolbar, or change the order and grouping of those commands. You can also add toolbars, and change the layout, position, and content of existing toolbars in the integrated development environment (IDE).

To add a command to a menu or toolbar

1. On the **Tools** menu, click **Customize**.
2. In the **Customize** dialog box, on the **Commands** tab, under **Choose a menu or tool bar to rearrange**, select the menu or tool bar you want to change and then click **Add command**.
3. In the **Add Command** dialog box, select a category name on the **Categories** list and then, on the **Commands** list, select the command you want to add.
4. Click **OK**.
5. Click **Close**.

To remove a command from a menu or toolbar

1. On the **Tools** menu, click **Customize**.
2. In the **Customize** dialog box, on the **Commands** tab, under **Choose a menu or toolbar to rearrange**, select the menu or toolbar you want to change.
3. Select the command you want to remove, and then click **Delete**.
4. Click **Close**.

To separate commands on a menu or toolbar

1. On the **Tools** menu, click **Customize**.
2. In the **Customize** dialog box, on the **Commands** tab, under **Choose a menu or toolbar to rearrange**, select the menu or toolbar you want to change.
3. Select the command you want to separate from the commands above it.
4. In the **Modify Selection** list, select **Begin a Group**.
5. A separator bar appears on the list of commands, above the selected command.
6. Click **OK**.
7. Click **Close**.

The command appears on the menu or toolbar with a separator before it.

To add a new menu

1. On the Tools menu, click **Customize.**
2. In the Customize dialog box, on the Commands tab, click **Add New Menu**. The menu appears, named New Menu.
3. In the Modify Selection list, enter the name for the new menu.
4. Click **OK**.
5. Click **Close**.

The command appears on the menu or toolbar before it.

To change the order of menus

1. On the Tools menu, click **Customize**.

2. In the Customize dialog box, on the Commands tab, under Choose a menu or toolbar to rearrange, select the menu or toolbar you want to move.
3. Select Move Up or Move Down to move the command.
4. Click **OK**.
5. Click **Close**.

The command appears on the menu or toolbar with a separator before it.

To create a toolbar

1. On the Tools menu, click **Customize**.
2. In the Customize dialog box, on the Toolbars tab, click New.
3. In the **New Toolbar** dialog box, type a name for the toolbar.
4. Use the steps described earlier in this topic to add commands to the toolbar.

Changing Toolbar Layout

You can arrange toolbars by dragging them in the main docking area, or by using the Customize dialog box to move them to other docking areas.

To arrange toolbars in the main docking area

1. Drag a toolbar by its left edge to move it where you want it.
2. Surrounding toolbars will be automatically rearranged.
3. To change the docking location of a toolbar.
4. On the Tools menu, click **Customize**.
5. In the Customize dialog box, on the Toolbars tab, on the Modify Selection list, select a dock location.
6. Click **Close**.

For more information about how to improve the usability and accessibility of toolbars, see How to: Set Accessibility Options.

Resetting the Main Menu and Shortcut Menus

If you change the locations of commands or change command icons, you can reset them to their original configurations.

To reset a menu or toolbar

1. On the Tools menu, click **Customize**.
2. In the **Customize** dialog box, on the **Commands** tab, under **Choose a menu or toolbar to rearrange**, select the menu or toolbar you want to reset.
3. Click **Reset all**.

The selected menu bar, toolbar, or context menu returns to its original configuration.

9.2. Reset Your Settings

You can reset the integrated development environment (IDE) to a previous state using the Import and Export Settings wizard. All settings and categories are applied by default; if you want to specify which settings to change, use the option Import selected environment settings.

To reset your settings

1. On the Tools menu, click **Import and Export Settings**.

2. On the **Welcome to the Import and Export Settings Wizard** page, click **Reset all settings**, and then click **Next**.
3. If you want to save your current settings combination, click **Yes, save my current settings**, specify a file name, and then click **Next**.
   —or—

   If you want to delete your current settings combination, choose No, just reset settings, overwriting my current settings, and then click **Next**. This option does not delete default settings, which will still be available the next time you use the wizard.
4. In **Which collection of settings do you want to reset to**, select a settings collection from the list.
5. Click **Finish**.

The **Reset Complete** page alerts you to any problems encountered during the reset.

9.3. Options Dialog Box

The **Options** dialog box enables you to configure the integrated development environment (IDE) to your needs. For example, you can establish a default save location for your projects, alter the default appearance and behavior of windows, and create shortcuts for commonly used commands. There are also options specific to your development language and platform. You can access **Options** from the **Tools** menu.

Note: The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu.

Layout of the Options dialog box

The **Options** dialog box is divided into two parts: a navigation pane on the left and a display area on the right. The tree control in the navigation pane includes folder nodes, such as **Environment**, **Text Editor**, **Projects and Solutions**, and **Source Control**. Expand any folder node to list the pages of options that it contains. When you select the node for a particular page, its options appear in the display area.

Options for an IDE feature do not appear in the navigation pane until the feature is loaded into memory. Therefore, the same options might not be displayed as you begin a new session that were displayed as you ended the last. When you create a project or run a command that uses a particular application, nodes for relevant options are added to the **Options** dialog box. These added options will then remain available as long as the IDE feature remains in memory.
Note: Some settings collections scope the number of pages that appear in the navigation pane of the **Options** dialog box. You can choose to view all possible pages by selecting **Show all settings**.

How options are applied

Clicking **OK** in the **Options** dialog box saves all settings on all pages. Clicking on **Cancel** any page cancels all change requests, including any just made on other **Options** pages. Some changes to option settings, such as those made on **Fonts and Colors**, **Environment**, **Options Dialog Box**, will only take effect after you close and reopen Atmel Studio.

9.3.1. Environment Options

The pages in the Environment folder in the Options dialog box let you set how certain elements of the integrated development environment (IDE) display and behave. You can access the Environment pages by clicking Options on the Tools menu, and then clicking Environment.

9.3.1.1. General Environment Settings

Items shown in Window menu

Customizes the number of windows that appear in the Windows list of the Window menu. Type a number between 1 and 24. By default, the number is 10.

Items shown in recently used lists

Customizes the number of most recently used projects and files that appear on the **File** menu. Type a number between 1 and 24. By default, the number is 10. This is an easy way to retrieve recently used projects and files.

Automatically adjust visual experience based on client performance

Specifies whether Atmel Studio sets the adjustment to the visual experience automatically or you set the adjustment explicitly. This adjustment may change the display of colors from gradients to flat colors, or it may restrict the use of animations in menus or pop-up windows.

Enable rich client experience

Enables the full visual experience of Atmel Studio, including gradients and animations. Clear this option when using Remote Desktop connections or older graphics adapters, because these features may have poor performance in those cases. This option is available only when you clear the Automatically adjust visual experience based on client option.

Use hardware graphics acceleration if available

Uses hardware graphics acceleration if it is available, rather than software acceleration.

Show status bar

Displays the status bar. The status bar is located at the bottom of the IDE window and displays information about the progress of ongoing operations.

Close button affects active tool window only

Specifies that when the Close button is clicked, only the tool window that has focus is closed and not all of the tool windows in the docked set. By default, this option is selected.

Auto Hide button affects active tool window only

Specifies that when the Auto Hide button is clicked, only the tool window that has focus is hidden automatically and not all of the tool windows in the docked set. By default, this option is not selected.

Restore File Associations

Registers file types that are typically associated with Atmel Studio. Registration causes Windows to display the correct icons in Windows Explorer, and to recognize Atmel Studio as the correct application for opening these file types.

This option can be useful if you have two different versions of Atmel Studio installed on the same computer, and you later uninstall one of the versions. After uninstalling, the icons for Atmel Studio files no longer appear in Windows Explorer. In addition, Windows no longer recognizes Atmel Studio as the default application for editing these files. This option restores those associations.

9.3.1.2. Add-in/Macros Security

Add-in Security Settings

To enhance security by preventing malicious add-ins from automatically activating, Atmel Studio provides settings in a Tools Options page named Add-in/Macros Security.

In addition, this options page allows you to specify the folders in which Atmel Studio searches for .Addin registration files. This enhances security by allowing you to limit the locations where .Addin registration files can be read, helping prevent malicious .Addin files from inadvertently being used.

The settings in the Add-in/Macros Security, Environment, and Options Dialog Box that relate to add-in security are:

- Allow add-in components to load. Checked by default. When checked, add-ins are allowed to load in Atmel Studio. When unchecked, add-ins are prohibited from loading in Atmel Studio.
- Allow add-in components to load from a URL. Unchecked by default. When checked, add-ins are allowed to be loaded from external Web sites. When unchecked, remote add-ins are prohibited from loading in Atmel Studio. If an add-in cannot load for some reason, then it cannot be loaded from the web. This setting controls only the loading of the add-in DLL. The .Addin registration files must always be located on the local system.

Default .Add-In File Search Locations

In addition to the security settings, the options page has a list containing folders in which to search for .Addin registration files. By default, the following tokens are included:

- %ALLUSERSDOCUMENTS%
- %ALLUSERSPROFILE%
- %APPDATA%
- %VSAPPDATA%
- %VSCOMMONAPPDATA%
- %VSMYDOCUMENTS%

When Atmel Studio begins searching for .AddIn files, it replaces these tokens with the following path strings:

Table 9-1. AddIn Files Search Path Tokens

Token	Path
%ALLUSERSDOCUMENTS%	%PUBLIC%\Documents
%ALLUSERSPROFILE%	%ALLUSERSPROFILE% (defined by OS)
%APPDATA%	%USERPROFILE%\AppData
%VSAPPDATA%	%USERPROFILE%\AppData\Roaming\Microsoft\AVR Studio 5\<Version> --OR-- %USERPROFILE%\AppData\Local\Microsoft\Atmel Studio 6\<Version>
%VSCOMMONAPPDATA%	%ProgramData%\Microsoft\Atmel Studio 6\<Version>
%VSMYDOCUMENTS%	<My Documents>\Atmel Studio 6

Note: Some of the default paths may resolve to targets that do not exist on your system.

You can remove these predefined tokens from the list by highlighting the token and clicking Remove. To add other folders to the search list, click Add and specify a folder in the Browse for Folder dialog box. For more information, see Add-In Registration.

9.3.1.3. AutoRecover

Use this page of the Options dialog box to specify whether or not files are automatically backed up. This page also allows you to specify whether or not modified files are restored when the integrated development environment (IDE) shuts down unexpectedly. You can access this dialog box by selecting

the Tools menu and choosing Options, and then selecting the Environment folder and choosing the AutoRecover page. If this page does not appear in the list, select Show all settings in the Options dialog box.

Save AutoRecover information every <*n*> minutes

Use this option to customize how often a file is automatically saved in the editor. For previously saved files, a copy of the file is saved in `\...\My Documents\Atmel Studio 6.2\Backup Files \<projectname>`. If the file is new and has not been manually saved, the file is auto-saved using a randomly generated file name.

Keep AutoRecover information for <*n*> days

Use this option to specify how long Atmel Studio keeps files created for auto recovery.

9.3.1.4. Find and Replace

Use this page of the Options dialog box to control message boxes and other aspects of a find and replace operation. You can access this dialog box from the Tools menu by clicking Options, expanding Environment, and then clicking **Find and Replace**. If this page does not appear in the list, select Show all settings in the Options dialog box.

Display informational messages

Select this option to display all **Find and Replace** informational messages that have the Always show this message option. For example, if you chose not to display the message "Find reached the starting point of the search.", selecting this option would cause this informational message to appear again when you use **Find and Replace**.

If you do not want to see any informational messages for **Find and Replace**, clear this option.

When you have cleared the Always show this message option on some, but not all, **Find and Replace** informational messages, the Display informational messages check box appears to be filled but not selected. To restore all optional **Find and Replace** messages, clear this option and then select it again.

Note: This option does not affect any **Find and Replace** informational messages that do not display the Always show this message option.

Display warning messages

Select this option to display all cautionary **Find and Replace** messages that have the Always show this message option. For example, if you chose not to display the Replace All warning message that appears when you attempt to make replacements in files not currently opened for editing, selecting this option would cause this warning message to appear again when you attempt to Replace All.

If you do not want to see any cautionary messages for **Find and Replace**, clear this option.

When you have cleared the Always show this message option on some, but not all, **Find and Replace** warning messages, the Display warning messages check box appears to be filled but not selected. To restore all optional **Find and Replace** messages, clear this option and then select it again.

Note: This option does not affect any **Find and Replace** warning messages that do not display the Always show this message option.

Automatically populate Find What with text from the editor

Select this option to paste the text on either side of the current editor's insertion point into the Find what field when you select any view of the **Find and Replace** Window window from the Edit menu. Clear this option to use the last search pattern from the previous search as the Find what string.

Hide Find and Replace window after a match is located for Quick Find or Quick Replace

Select this option to automatically close the **Find and Replace** window when the first match is found for Quick Find. To go to the next match, use the shortcut key for Edit.FindNext, usually F3, or display the **Find and Replace** window again.

9.3.1.5. Fonts and Colors

The Fonts and Colors page of the Options dialog box lets you establish a custom font and color scheme for various user interface elements in the integrated development environment (IDE). You can access this dialog box by clicking Options on the Tools menu, and then selecting the Fonts and Colors page in the Environment folder. If this page does not appear in the list, select Show all settings in the Options dialog box.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Color scheme changes do not take effect during the session in which you make them. You can evaluate color changes by opening another instance of Atmel Studio and producing the conditions under which you expect your changes to apply.

Show settings for

Lists all of the user interface elements for which you can change font and color schemes. After selecting an item from this list you can customize color settings for the item selected in Display items.

Text Editor

Changes to font style, size, and color display settings for Text Editor affect the appearance of text in your default text editor. Documents opened in a text editor outside the IDE will not be affected by these settings. For information about changing your default text editor, see How to: Change or Add a Default Editor.

Printer

Changes to font style, size, and color display settings for Printer affect the appearance of text in printed documents.

Note: As needed, you can select a different default font for printing than that used for display in the text editor. This can be useful when printing code that contains both single-byte and double-byte characters.

Statement Completion

Changes the font style and size for the text that appears in statement completion pop-up in the editor.

Editor Tool tip

Changes the font style and size for the text that appears in ToolTips displayed in the editor.

Environment Font

Changes the font style and size for all IDE user interface elements that do not already have a separate option in Show settings for. For example, this option applies to the Start Page but would not affect the Output window.

[All Text Tool Windows]

Changes to font style, size, and color display settings for this item affect the appearance of text in tool windows that have output panes in the IDE. For example, Output window, Command window, Immediate window, etc.

Note: Changes to the text of [All Text Tool Windows] items do not take effect during the session in which you make them. You can evaluate such changes by opening another instance of Atmel Studio.

Use Defaults/Use

Resets the font and color values of the list item selected in Show settings for. The Use button appears when other display schemes are available for selection. For example, you can choose from two schemes for the Printer.

Font (bold type indicates fixed-width fonts)

Lists all the fonts installed on your system. When the drop-down menu first appears, the current font for the element selected in the Show settings for field is highlighted. Fixed fonts — which are easier to align in the editor — appear in bold.

Size

Lists available point sizes for the highlighted font. Changing the size of the font affects all Display items for the Show settings for selection.

Display items

Lists the items for which you can modify the foreground and background color.

Note: PlainText is the default display item. As such, properties assigned to PlainText will be overridden by properties assigned to other display items. For example, if you assign the color blue to PlainText and the color green to Identifier, all identifiers will appear in green. In this example, Identifier properties override PlainText properties.

Some of display items include:

Display items

Description.

Plain Text

Text in the editor.

Selected Text

Text that is included in the current selection when the editor has focus.

Inactive Selected Text

Text that is included in the current selection when the editor has lost focus.

Indicator Margin

The margin at the left of the Code Editor where breakpoints and bookmark icons are displayed.

Line Numbers

Optional numbers that appear next to each line of code.

Visible White Space

Spaces, tabs and word wrap indicators.

Bookmark

Lines that have bookmarks. Bookmark is visible only if the indicator margin is disabled.

Brace Matching (Highlight)

Highlighting that is typically bold formatting for matching braces.

Brace Matching (Rectangle)

Highlighting that is typically a grey rectangle in the background.

Breakpoint (Enabled)

Specifies the highlight color for statements or lines containing simple breakpoints. This option is applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Breakpoint (Error)

Specifies the highlight color for statements or lines containing breakpoints that are in an error state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Breakpoint (Warning)

Specifies the highlight color for statements or lines containing breakpoints that are in a warning state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Breakpoint - Advanced (Disabled)

Specifies the highlight color for statements or lines containing disabled conditional or hit-counted breakpoints. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Breakpoint - Advanced (Enabled)

Specifies the highlight color for statements or lines containing conditional or hit-counted breakpoints. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Breakpoint - Advanced (Error)

Specifies the highlight color for statements or lines containing conditional or hit-counted breakpoints that are in an error state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Breakpoint - Advanced (Warning)

Specifies the highlight color for statements or lines containing conditional or hit-counted breakpoints that are in a warning state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Code Snippet Dependent Field

A field that will be updated when the current editable field is modified.

Code Snippet Field Editable

Field when a code snippet is active.

Collapsible Text

A block of text or code that can be toggled in and out of view within the Code Editor.

Comment

Code comments.

Compiler Error

Blue squiggles in the editor indicating a compiler error.

Coverage Not Touched Area

Code that has not been covered by a unit test.

Coverage Partially Touched Area

Code that has been partially covered by a unit test. Coverage Touched Area Code that has been completely covered by a unit test.

Current list location

Current line navigated to in a list tool window, such as the Output window or Find Results windows.

Current Statement

Specifies the highlight color for the source statement or line that indicates the current step position when debugging.

Debugger Data Changed

The color of text used to display changed data inside the Registers and Memory windows. Definition Window Background The background color of the Code Definition window.

Definition Window Current Match

The current definition in the Code Definition window.

Disassembly File Name

The color of text used to display file name breaks inside the Disassembly window.

Disassembly Source

The color of text used to display source lines inside the Disassembly window. Disassembly Symbol The color of text used to display symbol names inside the Disassembly window.

Disassembly Text

The color of text used to display op-code and data inside the Disassembly window. Excluded Code that is not to be compiled, per a conditional preprocessor directive such as #if.

Identifier

Identifiers in code such as the class names, methods names, and variable names.

Keyword

Keywords for the given language that are reserved. For example: class and namespace.

Memory Address

The color of text used to display the address column inside the Memory window.

Memory Changed

The color of text used to display changed data inside the Memory window.

Memory Data

The color of text used to display data inside the Memory window.

Memory Unreadable

The color of text used to display unreadable memory areas within the Memory window.

Number

A number in code that represents an actual numeric value. Operators such as +, -, and !=.

Other Error

Other error types not covered by other error squiggles. Currently, this includes rude edits in <guimenuitem>Edit and Continue</guimenuitem>.

Preprocessor Keyword

Keywords used by the preprocessor such as #include. Read-Only Region Code that cannot be edited. For example code displayed in the Code Definition View window or code that cannot be modified during Edit and Continue.

Register Data

The color of text used to display data inside the Registers window.

Register NAT

The color of text used to display unrecognized data and objects inside the Registers window.

Stale Code

Superseded code awaiting an update. In some cases, Edit and Continue cannot apply code changes immediately, but will apply them later as you continue debugging. This occurs if you edit a function that must call the function currently executing, or if you add more than 64 bytes of new variables to a function waiting on the call stack. When this happens, the debugger displays a "Stale Code Warning" dialog box, and the superseded code continues to execute until the function in question finishes and is called again. Edit and Continue applies the code changes at that time.

String

String literals.

Syntax Error

Parse errors. Task List Shortcut. If a Task List shortcut is added to a line, and the indicator margin is disabled, the line will be highlighted.

Tracepoint (Enabled)

Specifies the highlight color for statements or lines containing simple tracepoints. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Tracepoint (Error)

Specifies the highlight color for statements or lines containing tracepoints that are in an error state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Tracepoint (Warning)

Specifies the highlight color for statements or lines containing tracepoints that are in a warning state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Tracepoint - Advanced (Disabled)

Specifies the highlight color for statements or lines containing disabled conditional or hit-counted tracepoints. This option is applicable only if statement-level tracepoints are active or the Highlight entire

source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Tracepoint - Advanced (Enabled)

Specifies the highlight color for statements or lines containing conditional or hit-counted tracepoints. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Tracepoint - Advanced (Error)

Specifies the highlight color for statements or lines containing conditional or hit-counted tracepoints that are in an error state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Tracepoint - Advanced (Warning)

Specifies the highlight color for statements or lines containing conditional or hit-counted tracepoints that are in a warning state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box.

Track Changes after save

Lines of code that have been modified since the file was opened but are saved to disk.

Track Changes before save

Lines of code that have been modified since the file was opened but are not saved to disk.

User Types

Types defined by users. User Types (Delegates) Type color for delegates.

User Types (Enums)

Type color used for enums. User Types (Interfaces) Type color for interfaces.

User Types (Value types)

Type color for value types such as structs in C.

Warning

Compiler warnings.

Warning Lines

Path Used for Static Analysis warning lines.

XML Attribute

Attribute names.

XML Attribute Quotes

The quote characters for XML attributes.

XML Attribute Value

Contents of XML attributes.

XML Cdata Section

Contents of <![CDATA[…]]>.

XML Comment

The contents of <!-- -->. XML Delimiter XML Syntax delimiters, including <, <?, <!, <!--, -->, ?>, <![,]]>, and [,].

XML Doc Attribute

The value of an XML documentation attribute, such as <param name="I"> where the "I" is colorized.

XML Doc Comment

The comments enclosed in the XML documentation comments.

XML Doc Tag

The tags in XML documentation comments, such as /// <summary>.

XML Keyword

DTD keywords such as CDATA, IDREF, and NDATA.

XML Name Element

Names and Processing Instructions target name.

XML Processing Instruction

Contents of Processing Instructions, not including target name.

XML Text Plain

Text element content.

XSLT Keyword

XSLT element names.

Item foreground

Lists the available colors you can choose for the foreground of the item selected in Display items. Because some items are related, and should therefore maintain a consistent display scheme, changing the foreground color of the text also changes the defaults for elements such as Compiler Error, Keyword, or Operator. Automatic Items can inherit the foreground color from other display items such as Plain Text. Using this option, when you change the color of an inherited display item, the color of the related display items also change automatically. For example, if you selected the Automatic value for Compiler Error and later changed the color of Plain Text to Red, the Compiler Error would also automatically inherit the color Red. Default the color that appears for the item the first time you start AVR Studio 5. Clicking the Use Defaults button resets to this color. Custom Displays the Color dialog box to allow you to set a custom color for the item selected in the Display items list.

Note: Your ability to define custom colors may be limited by the color settings for your computer display. For example, if your computer is set to display 256 colors and you select a custom color from the Color dialog box, the IDE defaults to the closest available Basic color and displays the color black in the Color preview box.

Item background

Provides a color palette from which you can choose a background color for the item selected in Display items.

Because some items are related, and should therefore maintain a consistent display scheme, changing the background color of text also changes the defaults for elements such as Compiler Error, Keyword, or Operator.

Automatic Items can inherit the background color from other display items such as Plain Text.

Using this option, when you change the color of an inherited display item, the color of the related display items also change automatically. For example, if you selected the Automatic value for Compiler Error and later changed the color of Plain Text to Red, Compiler Error would also automatically inherit the color Red.

Clicking the Use Defaults button resets to this color. Custom Displays the Color dialog box to allow you to set a custom color for the item selected in the Display items list. Bold Select this option to display the text of selected Display items in bold text. Bold text is easier to identify in the editor. Sample Displays a sample of the font style, size, and color scheme for the Show settings for and Display items selected. You can use this box to preview the results as you experiment with different formatting options.

9.3.1.6. Language and International Settings

The International Settings page allows you to change the default language when you have more than one language version of the integrated development environment (IDE) installed on your machine.

You can access this dialog box by selecting **Options** from the **Tools** menu and then choosing **International Settings** from the **Environment** folder. If this page does not appear in the list, select **Show all settings** in the **Options** dialog box.

Any changes you make on this page apply only to the default IDE and do not take effect until the environment is restarted.

Language

Lists the available languages for the installed product language versions. This option is unavailable unless you have more than one language version installed on your machine. If multiple languages of products or a mixed language installation of products share the environment, the language selection is changed to Same as Microsoft Windows.

Caution:
In a system with multiple languages installed, the build tools are not affected by this setting. These tools use the version for last language installed and the tools for the previously installed language are overwritten because the build tools do not use the satellite DLL model.

9.3.1.7. Keyboard Settings

The shortcut key combinations in the scheme currently applied, (Default), depend on the settings you have selected as well as any customizations you might have made. For more information about the shortcut keys associated with a settings combination, see Working with Settings. Visual Studio also includes seven other keyboard mapping schemes, each of which differs from the others in the shortcut key combinations assigned by default to various UI elements. For a list of these combinations, organized by mapping scheme, see Pre-defined Keyboard Shortcuts.

Commands with shortcut key combinations that are part of the Global scope can be superseded by commands in other scopes depending on the current context of the integrated development environment (IDE). For example, if you are editing a file, commands that are part of the Text Editor scope have precedence over commands in the Global scope that start with the same key combination. For example, if several Global commands have key combinations that start with CTRL + K and the Text Editor also has several commands with key combinations that start with CTRL + K, when you are editing code the Text Editor key combinations will work and the Global key combinations will be ignored.

Note: The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. This Help page

was written with General Development Settings in mind. To change your settings, from the Tools menu, choose Import and Export Settings. For more information, see Working with Settings.

Determine the Shortcut Key Assigned to a Command

You can manually search for a command to determine whether or not it has an assigned shortcut key combination.

To determine the shortcut key combination for a command

1. On the Tools menu, click Options.
2. Expand the Environment folder and select Keyboard.
   Note: If you do not see the Keyboard page, check Show all settings located in the lower left of the Options dialog box.

   In the Show commands containing box, enter the name of the command without spaces. For example, solutionexplorer.
3. In the list, select the correct command.
   For example, View.SolutionExplorer.
4. If a shortcut key combination exists for the command, the combination appears in the Shortcut(s) for selected command drop-down list.

Create Custom Shortcut Keys

You can create new shortcut key combinations for any command or change the shortcut key combination for commands with existing combinations.

To create a new shortcut key combination

1. On the Tools menu, click Options.
2. Expand the Environment folder, and select Keyboard.
   Note: If you do not see the Keyboard page, check Show all settings located in the lower left corner of the Options dialog box. In the Show commands containing box, enter the name of the command without spaces.

   For example, solutionexplorer.
3. In the list, select the command you want to assign to a shortcut key combination.
4. On the Use new shortcut in drop-down list, select the feature area in which you want to use the shortcut. For example, you can choose Global if you want the shortcut to work in all contexts. Unless the same shortcut is mapped (as Global) in another editor, you can use it. Otherwise, the editor overrides the shortcut.
   Note: The following keys cannot be assigned to a command in Global: PRINT SCRN/SYS RQ, SCROLL LOCK, PAUSE/BREAK, TAB, CAPS LOCK, INSERT, HOME, END, PAGE UP, PAGE DOWN, Windows logo keys, Application key, any of the ARROW keys, or ENTER; NUM LOCK, DEL, or CLEAR on the numeric keypad; or CTRL+ALT+DELETE.
5. Place the cursor in the Press shortcut key(s) box, and then use the keyboard to enter the key combination you intend to use for the command.
   Note: Shortcuts can contain the SHIFT, ALT, and/or CTRL keys in combination with letters. Be sure to check the Shortcut currently used by box to see if the key combination is already assigned to another command in the mapping scheme. Press BACKSPACE to delete the key combination, if the combination is already in use, before trying another combination.
6. Click Assign.
   Note: Changes made by using the Assign button are not canceled if you click the Cancel button.

Exporting and Importing Shortcut Keys

You can share the shortcut key combinations in the current keyboard mapping scheme by exporting the information to a file so others can import the data.

To export shortcut keys only

1. On the Tools menu, choose **Import and Export Settings Wizard**.
2. Select **Export select environment settings** and then click **Next**.
3. Under **What settings do you want to export?**, clear all categories selected by default.
4. Expand **Options** and then expand **Environment**.
5. Select **Keyboard** and then click **Next**.
6. For **What do you want to name your settings file?**, enter a name and then click **Finish**.

To import only shortcut keys

1. On the Tools menu, click **Import and Export Settings Wizard**.
2. Select **Import select environment settings** and then click **Next**.
3. Click No, just import new settings, overwriting my current settings and then click Next.
4. Under **My Settings**, select the settings file that contains the shortcut keys you want to import, or click Browse to locate the correct settings file.
5. Click **Next**.
6. Under **Which settings do you want to import?**, clear all categories.
7. Expand **Options** and then expand **Environment**.
8. Select **Keyboard** and then click **Finish**.

9.3.1.8. Start-up Page — to Change the Default UI Displayed when You Start Atmel Studio

1. On the **Tools** menu, chose **Options**.
2. Expand **Environment** and then chose **Startup**.
3. From the At startup drop-down list, chose one of the options. For more information, see Startup, Environment, Options Dialog Box.
4. Click **OK**.

Your changes take affect the next time you start Atmel Studio.

Use this page to specify what content or user interface (UI), if any, is displayed when you start Atmel Studio. To access this page, on the **Tools** menu, click **Options**, expand **Environment**, and then click **Startup**. If this page does not appear in the list in the **Options** dialog box, select Show all settings.

Note: The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. This Help page was written with General Development settings in mind. To change your settings, on the Tools menu, click **Import and Export Settings**.

At start-up

You can specify what you want to view every time you start AVR Studio 5.

Open Home Page

Displays the default Web page specified by the Home page option in Web Browser, Environment, Options Dialog Box.

Load last loaded solution

Loads the last saved solution in its previous state. Any files that were open in the solution when it was last closed are opened and displayed when you start Atmel Studio. If no solution is loaded when you exit the product, no solution is loaded when you return.

Show Open Project dialog box

Displays the Open Project dialog box when you start Atmel Studio. The dialog box uses the folder set in the Atmel Studio Projects location field of the Projects and Solutions, Environment, Options Dialog Box.

Show New Project dialog box

Displays the New Project dialog box when you open Atmel Studio.

Show empty environment

Displays an empty integrated development environment (IDE) when you start Atmel Studio.

Show Start Page

Displays the Start Page associated with the settings that you have currently applied when you start Atmel Studio.

Start Page news channel

Specifies the RSS feed used to display content in the Atmel Studio News section of the Start Page.

Download content every *n* minutes

Specifies how often the IDE checks for new RSS feed content and product headlines for the Start Page. If this setting is not selected, RSS feed content and product headlines are not downloaded to the Start Page.

Customize Start Page

If you have custom Start Pages installed, you can specify which Start Page to load. The Customize Start Page drop-down list includes an (Default Start Page) entry to load the default Atmel Studio Start Page, and an entry for each custom Start Page on your system.

Any .XAML file in your user start pages directory is considered a custom start page. For more information, see Custom Start Pages.

9.3.1.9. Import and Export Settings

Use this page of the Options dialog box to set preferences for saving settings files as well as specifying whether or not to use team settings files stored on a server. You can access this dialog box by selecting Options from the Tools menu and choosing the Import and Export Settings page from the Environment folder.

Tip:
If this page does not appear in the list, select Show all setting in the Options dialog box.

Note: The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. This Help page was written with General Development Settings in mind. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Automatically load and save settings

Automatically save my settings to this file:

Displays the location and name of the .vssettings file you are currently using. When you close the IDE, any changes you have made, such as moving windows or changing option selections, are saved to the current file. The next time you start the IDE, your settings are loaded.

Team settings

Use team settings file:

When selected, allows you to navigate to a shared .vssettings file by using the Browse button. This settings file is automatically re-applied each time Atmel Studio detects if a newer version is available.

Note: The location of the team settings file must be specified as a UNC path or local path. URLs and other protocols are not supported paths.

9.3.1.10. Task List

This Options page allows you to add, delete, and change the comment tokens that generate Task List reminders. To display these settings, select Options from the Tools menu, expand the Environment folder, and choose Task List.

Confirm deletion of tasks

When selected, a message box is displayed whenever a User Task is deleted from the Task List, allowing you to confirm the deletion. This option is selected by default.

Note: To delete a Task Comment, use the link to find the comment, and then remove it from your code.

Hide full file paths

When selected, the File column of the Task List displays only the names of files to be edited, not their full paths.

Tokens

When you insert a comment into your code whose text begins with a token from the Token List, the Task List displays your comment as new entry whenever the file is opened for editing. You can click this Task List entry to jump directly to the comment line in your code. For more information, see How to: Create Task List Comments.

Token List

Displays a list of tokens, and allows you to add or remove custom tokens. Comment tokens are case sensitive.

Note: If you do not type the desired token exactly as it appears in the Token List, a comment task will not be displayed in the Task List.

Priority

Sets the priority of tasks that use the selected token. Task comments that begin with this token are automatically assigned the designated priority in the Task List.

Name

Enter the token string. This enables the Add button. On Add, this string is included in the Token List, and comments that begin with this name will be displayed in the Task List.

Add

Enabled when you enter a new Name. Click to add a new token string using the values entered in the Name and Priority fields.

Delete

Click to delete the selected token from the Token List. You cannot delete the default comment token.

Change

Click to make changes to an existing token using the values entered in the Name and Priority fields.

Note: You cannot rename or delete the default comment token, but you can change its priority level.

9.3.1.11. Web Browser Options

Sets options for both the internal Web browser and Internet Explorer. To access this dialog box, click Options on the Tools menu, expand the Environment folder, and select Web Browser.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Attention:
Opening certain files or components from the Web can execute code on your computer. For more information, see Code Access Security.

Home page

Sets the page displayed when you open the Integrated Development Environment Web Browser.

Search page

Lets you designate a Search page for the internal Web browser. This location can differ from the search page used by instances of Internet Explorer initiated outside of the integrated development environment (IDE).

View Source in

Sets the editor used to open a Web page when you choose View Source on the page from the internal Web browser.

Source editor

Select to view source in the Code and Text Editor.

HTML editor

Select to view source in the HTML Designer. Use this selection to edit the Web page in one of two views: Design view or the standard text-based Source view.

External editor

Select to view source in another editor. Specify the path of any editor you choose, for example, Notepad.exe.

Internet Explorer Options

Click to change options for Internet Explorer in the Internet Properties dialog box. Changes made in this dialog box affect both the internal Web browser and instances of Internet Explorer initiated outside of the Atmel Studio IDE (for example, from the Start menu).

9.3.1.12. Custom Start Pages

The Atmel Studio Start Page is a Windows Presentation Foundation (WPF) Extensible Application Markup Language (XAML) page that runs in an Atmel Studio tool window. The Start Page tool window can run Atmel Studio internal commands. When Atmel Studio starts, it opens the current default Start Page. If you have installed a third-party Start Page, you can set that page as the default by using the Options dialog box.

Installing and Applying a Custom Start Page

You can install custom Start Pages by using the Online Gallery section of Extension Manager. You can also install directly from a Web site or local intranet page by locating and opening a .vsix file that contains a custom Start Page, or by copying the Start Page files and pasting them into in the Documents\Atmel Studio\StartPages\ folder on your computer.

You can apply a custom Start Page by selecting it in the Options dialog box. Start pages installed by Extension Manager will appear in the **Customize Start Page** list as [InstalledExtension] Extension Name. Start pages dropped into the \StartPages folder will include a partial file path in the list entry, as shown in the following example.

```
Documents\Atmel Studio 6\StartPages\StartPage.xaml
```

To apply a custom Start Page

1. On the Tools menu, click **Options**.
2. On the left side of the **Options** dialog box, expand the Environment node, and then click **Startup**.
3. In the **Customize Start Page** list, select the Start Page you want.
4. This list includes every `.xaml` file in your user Start Pages folder and any installed extensions of type StartPage.
5. Click **OK**.

Troubleshooting

It is possible for an error in a third-party Start Page to cause Atmel Studio to crash. If this happens, you can start Atmel Studio in safe mode by adding the */SafeMode* switch to the application, i.e. `avrstudio5.exe /SafeMode`.

This prevents the bad Start Page from loading. You can then return to the Options dialog box and reset Atmel Studio to use the default Start Page.

9.3.2. Project Options

9.3.2.1. General Settings

Sets the default path of Atmel Studio project folders, and determines the default behavior of the Output window, Task List, and Solution Explorer as projects are developed and built. To access this dialog box, on the Tools menu, click **Options**, expand **Projects and Solutions**, and click **General**.

Note: The options are available in the dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. This Help page was written with the General Development settings in mind. To view or change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Projects location

Sets the default location where new projects and solution folders and directories are created. Several dialog boxes also use the location set in this option for folder starting points. For example, the Open Project dialog box uses this location for the My Projects shortcut.

User project templates location

Sets the default location that is used by the New Project dialog box to create the list of My Templates. For more information, see How to: Locate and Organize Project and Item Templates.

User item templates location

Sets the default location that is used by the Add New Item dialog box to create the list of My Templates. For more information, see How to: Locate and Organize Project and Item Templates.

Always show Error List if build finishes with errors

Opens the Error List window on build completion, only if a project failed to build. Errors that occur during the build process are displayed. When this option is cleared, the errors still occur but the window does not open when the build is complete. This option is enabled by default.

Track Active Item in Solution Explorer

When selected, Solution Explorer automatically opens and the active item is selected. The selected item changes as you work with different files in a project or solution, or different components in a designer. When this option is cleared, the selection in Solution Explorer does not change automatically. This option is enabled by default.

Show advanced build configurations

When selected, the build configuration options appear on the Project Property Pages dialog box and the Solution Property Pages dialog box. When cleared, the build configuration options do not appear on the Project Property Pages dialog box and the Solution Property Pages dialog box for projects that contain one configuration or the two configurations debug and release. If a project has a user-defined configuration, the build configuration options are shown.

When deselected, the commands on the Build menu, such as Build Solution, Rebuild Solution, and Clean Solution, are performed on the Release configuration and the commands on the Debug menu, such as Start Debugging and Start Without Debugging, are performed on the Debug configuration.

Always show solution

When selected, the solution and all commands that act on solutions are always shown in the IDE. When cleared, all projects are created as standalone projects and you do not see the solution in Solution Explorer or commands that act on solutions in the IDE if the solution contains only one project.

Save new projects when created

When selected, you can specify a location for your project in the New Project dialog box. When cleared, all new projects are created as temporary projects. When you are working with temporary projects, you can create and experiment with a project without having to specify a disk location.

Warn user when the project location is not trusted

If you attempt to create a new project or open an existing project in a location that is not fully trusted (for example, on a UNC path or an HTTP path), a message is displayed. Use this option to specify whether the message is displayed each time that you attempt to create or open a project in a location that is not fully trusted.

Show Output window when build starts

Automatically displays the Output Window in the IDE at the outset of solution builds. For more information, see How to: Control the Output Window. This option is enabled by default.

Prompt for symbolic renaming when renaming files

When selected, displays a message box asking whether or not AVR Studio 5 should also rename all references in the project to the code element.

9.3.2.2. Build and Run Options

Determines whether changed files are automatically saved when a project or its solution is built, the maximum number of Visual C++ projects that can build at the same time, and certain default behavior on Run. To access this dialog box, on the Tools menu, click Options, click Projects and Solutions, and then click Build and Run.

Save all changes

Automatically saves changes to the solution file and all project files that were changed since the last build when you press F5, or click Start on the Debug menu or Build on the Build menu. No prompt is given. Items are saved with their current names. By default, this option is enabled.

Save changes to open documents only

Automatically saves changes to all open documents when you press F5, or click Start on the Debug menu or Build on the Build menu. No prompt is given.

Prompt to save all changes

When selected, displays a dialog box that asks whether you want to save changes to the solution and project items when you press F5 or click Start on the Debug menu or Build on the Build menu. The Save As dialog box is displayed so that you can assign a name and location to your project. If this option is not selected, the project runs by using the memory image that contains your changes, but the changes are not saved.

Don't save any changes

When you run your project, the integrated development environment (IDE) runs the code version in the open documents and does not save changes to open documents.

Maximum number of parallel project builds

Specifies the maximum number of projects that can build at the same time. To optimize the build process, the maximum number of parallel project builds is automatically set to the number of CPUs of your computer. The maximum is 32. For more information, see Multiprocessor Builds.

Only build start-up projects and dependencies on Run

When selected, pressing F5 or clicking Start on the Debug menu or Build on the Build menu only builds the start-up project and its dependencies. When this option is cleared, pressing F5 builds all projects, dependencies, and solution files. By default, this option is cleared.

Always build

The message box is not displayed and the out of date project configuration is built. This option is set when you select Do not show this dialog again in the message, and then click Yes.

Never build

The message box is not displayed and the out of date project configuration is not built. This option is set when you select Do not show this dialog again in the message, and then click No.

Prompt to build

Displays the message box every time that a project configuration is out of date.

Prompt to launch

Displays the message box every time that build errors occur.

Do not launch

The message box is not displayed and the application is not started. This option is set when you select Do not show this dialog again in the message box, and then click No.

Launch old version

The message box is not displayed and the newly built version of the application is not started. This option is set when you select Do not show this dialog again in the message box, and then click Yes.

For new solutions use the currently selected project as the start-up project

If selected, new solutions use the currently selected project as the start-up project.

MSBuild project build output verbosity

Sets the verbosity level for the build output. For more information, see the /verbosity switch in MSBuild Command Line Reference.

MSBuild project build log file verbosity

Sets the verbosity level for the build log file. For more information, see the /verbosity switch in MSBuild Command Line Reference.

9.3.3. Source Control

If you have plugins for source control (SVN, ClearCase, Vault, Git, etc.) installed, you should select it from the drop-down list in this section, to activate and use your plugin with the source repository.

9.3.4. Text Editor Options

9.3.4.1. General Settings

This dialog box lets you change global settings for the Visual Studio Code and Text Editor. To display this dialog box, click Options on the Tools menu, expand the Text Editor folder, and then click General.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Settings

Drag and drop text editing

When selected, this enables you to move text by selecting and dragging the text with the mouse to another location within the current document or any other open document.

Automatic delimiter highlighting

When selected, delimiter characters that separate parameters or item-value pairs, as well as matching braces, are highlighted.

Track changes

When selected, the code editor's selection margin displays a vertical yellow line to mark code recently changed, and vertical green lines next to unchanged code.

Auto-detect UTF-8 encoding without signature

By default, the editor detects encoding by searching for byte order marks or charset tags. If neither is found in the current document, the code editor attempts to auto-detect UTF-8 encoding by scanning byte sequences. To disable the auto-detection of encoding, clear this option.

Display

Selection margin

When selected, a vertical margin along the left edge of the editor's text area is displayed. You can click this margin to select an entire line of text, or click and drag to select consecutive lines of text. **Selection Margin on / Selection Margin off**

Indicator margin

When selected, a vertical margin outside the left edge of the editor's text area is displayed. When you click in this margin, an icon and ToolTip that are related to the text appear. For example, breakpoint or task list shortcuts appear in the indicator margin. Indicator Margin information does not print.

Vertical scroll bar

When selected, a vertical scrollbar which allows you to scroll up and down to view elements that fall outside the viewing area of the Editor is displayed. If vertical scrollbars are not available, you can use the Page Up, Page Down, and cursor keys to scroll.

Horizontal scroll bar

When selected, a horizontal scrollbar which allows you to scroll from side-to-side to view elements that fall outside the viewing area of the Editor is displayed. If horizontal scrollbars are unavailable, you can use the cursor keys to scroll.

9.3.4.2. File Extensions and Associations

There you can specify tool association of the source file extensions.

9.3.4.3. General Language Options

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To open this dialog box, select Options from the Tools menu. Within the Text Editor folder, expand the All Languages sub folder and then select General.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the General options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the sub folder for that language and select its option pages.

A grayed checkmark is displayed when an option has been selected on the General options pages for some programming languages, but not for others.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Statement Completion

Auto list members

When selected, pop-up lists of available members, properties, values, or methods are displayed by IntelliSense as you type in the editor. Choose any item from the pop-up list to insert the item into your code. Selecting this option enables the Hide advanced members option. For more information, see List Members.

Hide advanced members

When selected, shortens pop-up statement completion lists by displaying only those items most commonly used. Other items are filtered from the list.

Parameter information

When selected, the complete syntax for the current declaration or procedure is displayed under the insertion point in the editor, with all of its available parameters. The next parameter you can assign is displayed in bold. For more information, see Parameter Info.

Settings

Enable virtual space

When this option is selected and Word wrap is cleared, you can click anywhere beyond the end of a line in the Code Editor, and type. This feature can be used to position comments at a consistent point next to your code.

Word wrap

When selected, any portion of a line that extends horizontally beyond the viewable editor area is automatically displayed on the next line. Selecting this option enables the Show visual glyphs for word wrap option.

Note: The Virtual Space feature is turned off while Word Wrap is on.

Show visual glyphs for word wrap

When selected, a return-arrow indicator is displayed where a long line wraps onto a second line.

Clear this option if you prefer not to display these indicators.

Note: These reminder arrows are not added to your code, and do not print. They are for reference only.

Apply Cut or Copy commands to blank lines when there is no selection

This option sets the behavior of the editor when you place the insertion point on a blank line, select nothing, and then Copy or Cut.

When this option is selected, the blank line is copied or cut. If you then Paste, a new and blank line is inserted.

When this option is cleared, the Cut command removes blank lines. However, the data on the Clipboard is preserved. Therefore, if you then use the Paste command, the content most recently copied onto the Clipboard is pasted. If nothing has been copied previously, nothing is pasted.

This setting has no effect on Copy or Cut when a line is not blank. If nothing is selected, the entire line is copied or cut. If you then Paste, the text of the entire line and its endline character are pasted.

Tip:
To display indicators for spaces, tabs, and line ends, and thus distinguish indented lines from lines that are entirely blank, select Advanced from the Edit menu and choose View White Space.

Display

Line numbers

When selected, a line number appears next to each line of code.

Note: These line numbers are not added to your code, and do not print. They are for reference only.

Enable single-click URL navigation

When selected, the mouse cursor changes to a pointing hand as it passes over a URL in the editor. You can click the URL to display the indicated page in your Web browser.

Navigation bar

When selected, the Navigation bar at the top of the code editor is displayed. Its drop-down Objects and Members lists allow you to choose a particular object in your code, select from its members, and navigates to the declaration of the selected member in the Code Editor.

9.3.4.4. Tabs Dialog

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To display these options, select Options from the Tools menu. Within the Text Editor folder expand the All Languages sub folder, and then choose Tabs.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the Tabs options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the sub folder for that language and select its option pages.

If different settings are selected on the Tabs options pages for particular programming languages, then the message "The indentation settings for individual text formats conflict with each other," is displayed for differing Indenting options; and the message "The tab settings for individual text formats conflict with each other," is displayed for differing Tab options.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Indenting

None

When selected, new lines are not indented. The insertion point is placed in the first column of a new line.

Block

When selected, new lines are automatically indented. The insertion point is placed at the same starting point as the preceding line.

Smart

When selected, new lines are positioned to fit the code context, per other code formatting settings and IntelliSense conventions for your development language. This option is not available for all development languages.

For example, lines enclosed between an opening brace ({) and a closing brace (}) might automatically be indented an extra tab stop from the position of the aligned braces.

Tab and indent size

Sets the distance in spaces between tab stops and for automatic indentation. The default is four spaces. Tab characters, space characters, or both will be inserted to fill the specified size.

Insert spaces

When selected, indent operations insert only space characters, not TAB characters. If the Tab and Indent size is set to 5, for example, then five space characters are inserted whenever you press the TAB key or the Increase Indent button on the Formatting toolbar.

Keep tabs

When selected, each indent operation inserts one TAB character.

9.3.4.5. AVR Assembler Language-specific Settings

General Language Options

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To open this dialog box, select Options from the Tools menu. Within the Text Editor folder, expand the All Languages sub folder and then choose General.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the General options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the sub folder for that language and select its option pages.

A grayed checkmark is displayed when an option has been selected on the General options pages for some programming languages, but not for others.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Statement Completion

Auto list members

When selected, pop-up lists of available members, properties, values, or methods are displayed by IntelliSense as you type in the editor. Choose any item from the pop-up list to insert the item into your code. Selecting this option enables the Hide advanced members option. For more information, see List Members.

Hide advanced members

When selected it shortens the pop-up statement completion lists by displaying only those items most commonly used. Other items are filtered from the list.

Parameter information

When selected, the complete syntax for the current declaration or procedure is displayed under the insertion point in the editor, with all of its available parameters. The next parameter you can assign is displayed in bold. For more information, see Parameter Info.

Settings

Enable virtual space

When this option is selected and Word wrap is cleared, you can click anywhere beyond the end of a line in the Code Editor and type. This feature can be used to position comments at a consistent point next to your code.

Word wrap

When selected, any portion of a line that extends horizontally beyond the viewable editor area is automatically displayed on the next line. Selecting this option enables the Show visual glyphs for word wrap option.

Note: The Virtual Space feature is turned off while Word Wrap is on.

Show visual glyphs for word wrap

When selected, a return-arrow indicator is displayed where a long line wraps onto a second line.

Clear this option if you prefer not to display these indicators.

Note: These reminder arrows are not added to your code, and do not print. They are for reference only.

Apply Cut or Copy commands to blank lines when there is no selection

This option sets the behavior of the editor when you place the insertion point on a blank line, select nothing, and then Copy or Cut.

When this option is selected, the blank line is copied or cut. If you then Paste, a new, blank line is inserted.

When this option is cleared, the Cut command removes blank lines. However, the data on the Clipboard is preserved. Therefore, if you then use the Paste command, the content most recently copied onto the Clipboard is pasted. If nothing has been copied previously, nothing is pasted.

This setting has no effect on Copy or Cut when a line is not blank. If nothing is selected, the entire line is copied or cut. If you then Paste, the text of the entire line and its endline character are pasted.

Tip:
To display indicators for spaces, tabs, and line ends, and thus distinguish indented lines from lines that are entirely blank, select Advanced from the Edit menu and choose View White Space.

Display

Line numbers

When selected, a line number appears next to each line of code.

Note: These line numbers are not added to your code, and do not print. They are for reference only.

Enable single-click URL navigation

When selected, the mouse cursor changes to a pointing hand as it passes over a URL in the editor. You can click the URL to display the indicated page in your Web browser.

Navigation bar

When selected, displays the Navigation bar at the top of the code editor. Its dropdown Objects and Members lists allow you to choose a particular object in your code, select from its members, and navigates to the declaration of the selected member in the Code Editor.

Tabs Dialog
This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To display these options, select Options from the Tools menu. Within the Text Editor folder expand the All Languages sub folder, and then choose Tabs.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the Tabs options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the sub folder for that language and select its option pages.

If different settings are selected on the Tabs options pages for particular programming languages, then the message "The indentation settings for individual text formats conflict with each other," is displayed for differing Indenting options; and the message "The tab settings for individual text formats conflict with each other," is displayed for differing Tab options.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Indenting

None

When selected, new lines are not indented. The insertion point is placed in the first column of a new line.

Block

When selected, new lines are automatically indented. The insertion point is placed at the same starting point as the preceding line.

Smart

When selected, new lines are positioned to fit the code context, per other code formatting settings and IntelliSense conventions for your development language. This option is not available for all development languages.

For example, lines enclosed between an opening brace ({) and a closing brace (}) might automatically be indented an extra tab stop from the position of the aligned braces.

Tab and indent size

Sets the distance in spaces between tab stops and for automatic indentation. The default is four spaces. Tab characters, space characters, or both will be inserted to fill the specified size.

Insert spaces

When selected, indent operations insert only space characters, not TAB characters. If the Tab and Indent size is set to 5, for example, then five space characters are inserted whenever you press the TAB key or the Increase Indent button on the Formatting toolbar.

Keep tabs

When selected, each indent operation inserts one TAB character.

9.3.4.6. AVR GCC Language-specific Settings

General Language Options
This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To open this dialog box, select Options from the Tools menu. Within the Text Editor folder, expand the All Languages sub folder and then choose General.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the General options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the sub folder for that language and select its option pages.

A grayed checkmark is displayed when an option has been selected on the General options pages for some programming languages, but not for others.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Statement Completion

Auto list members

When selected, pop-up lists of available members, properties, values, or methods are displayed by IntelliSense as you type in the editor. Choose any item from the pop-up list to insert the item into your code. Selecting this option enables the Hide advanced members option. For more information, see List Members.

Hide advanced members

When selected it shortens the pop-up statement completion lists by displaying only those items most commonly used. Other items are filtered from the list.

Parameter information

When selected, the complete syntax for the current declaration or procedure is displayed under the insertion point in the editor, with all of its available parameters. The next parameter you can assign is displayed in bold. For more information, see Parameter Info.

Settings

Enable virtual space

When this option is selected and Word wrap is cleared, you can click anywhere beyond the end of a line in the Code Editor and type. This feature can be used to position comments at a consistent point next to your code.

Word wrap

When selected, any portion of a line that extends horizontally beyond the viewable editor area is automatically displayed on the next line. Selecting this option enables the Show visual glyphs for word wrap option.

Note: The Virtual Space feature is turned off while Word Wrap is on.

Show visual glyphs for word wrap

When selected, a return-arrow indicator is displayed where a long line wraps onto a second line.

Clear this option if you prefer not to display these indicators.

Note: These reminder arrows are not added to your code, and do not print. They are for reference only.

Apply Cut or Copy commands to blank lines when there is no selection

This option sets the behavior of the editor when you place the insertion point on a blank line, select nothing, and then Copy or Cut.

When this option is selected, the blank line is copied or cut. If you then Paste, a new, blank line is inserted.

When this option is cleared, the Cut command removes blank lines. However, the data on the Clipboard is preserved. Therefore, if you then use the Paste command, the content most recently copied onto the Clipboard is pasted. If nothing has been copied previously, nothing is pasted.

This setting has no effect on Copy or Cut when a line is not blank. If nothing is selected, the entire line is copied or cut. If you then Paste, the text of the entire line and its endline character are pasted.

Tip:
To display indicators for spaces, tabs, and line ends, and thus distinguish indented lines from lines that are entirely blank, select Advanced from the Edit menu and choose View White Space.

Display

Line numbers

When selected, a line number appears next to each line of code.

Note: These line numbers are not added to your code, and do not print. They are for reference only.

Enable single-click URL navigation

When selected, the mouse cursor changes to a pointing hand as it passes over a URL in the editor. You can click the URL to display the indicated page in your Web browser.

Navigation bar

When selected, displays the Navigation bar at the top of the code editor. Its drop-down Objects and Members lists allow you to choose a particular object in your code, select from its members, and navigates to the declaration of the selected member in the Code Editor.

Tabs Dialog
This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To display these options, select Options from the Tools menu. Within the Text Editor folder expand the All Languages subfolder, and then choose Tabs.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the Tabs options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

If different settings are selected on the Tabs options pages for particular programming languages, then the message "The indentation settings for individual text formats conflict with each other," is displayed for differing Indenting options; and the message "The tab settings for individual text formats conflict with each other," is displayed for differing Tab options.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Indenting

None

When selected, new lines are not indented. The insertion point is placed in the first column of a new line.

Block

When selected, new lines are automatically indented. The insertion point is placed at the same starting point as the preceding line.

Smart

When selected, new lines are positioned to fit the code context, per other code formatting settings and IntelliSense conventions for your development language. This option is not available for all development languages.

For example, lines enclosed between an opening brace ({) and a closing brace (}) might automatically be indented an extra tab stop from the position of the aligned braces.

Tab and indent size

Sets the distance in spaces between tab stops and for automatic indentation. The default is four spaces. Tab characters, space characters, or both will be inserted to fill the specified size.

Insert spaces

When selected, indent operations insert only space characters, not TAB characters. If the Tab and Indent size is set to 5, for example, then five space characters are inserted whenever you press the TAB key or the Increase Indent button on the Formatting toolbar.

Keep tabs

When selected, each indent operation inserts one TAB character.

9.3.4.7. Plain Text Settings

General Language Options

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To open this dialog box, select Options from the Tools menu. Within the Text Editor folder, expand the All Languages subfolder and then choose General.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the General options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

A grayed checkmark is displayed when an option has been selected on the General options pages for some programming languages, but not for others.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Statement Completion

Auto list members

When selected, pop-up lists of available members, properties, values, or methods are displayed by IntelliSense as you type in the editor. Choose any item from the pop-up list to insert the item into your code. Selecting this option enables the Hide advanced members option. For more information, see List Members.

Hide advanced members

When selected, it shortens the pop-up statement completion lists by displaying only those items most commonly used. Other items are filtered from the list.

Parameter information

When selected, the complete syntax for the current declaration or procedure is displayed under the insertion point in the editor, with all of its available parameters. The next parameter you can assign is displayed in bold. For more information, see Parameter Info.

Settings

Enable virtual space

When this option is selected and Word wrap is cleared, you can click anywhere beyond the end of a line in the Code Editor and type. This feature can be used to position comments at a consistent point next to your code.

Word wrap

When selected, any portion of a line that extends horizontally beyond the viewable editor area is automatically displayed on the next line. Selecting this option enables the Show visual glyphs for word wrap option.

Note: The Virtual Space feature is turned OFF while Word Wrap is ON.

Show visual glyphs for word wrap

When selected, a return-arrow indicator is displayed where a long line wraps onto a second line.

Clear this option if you prefer not to display these indicators.

Note: These reminder arrows are not added to your code, and do not print. They are for reference only.

Apply Cut or Copy commands to blank lines when there is no selection

This option sets the behavior of the editor when you place the insertion point on a blank line, select nothing, and then Copy or Cut.

When this option is selected, the blank line is copied or cut. If you then Paste, a new, blank line is inserted.

When this option is cleared, the Cut command removes blank lines. However, the data on the Clipboard is preserved. Therefore, if you then use the Paste command, the content most recently copied onto the Clipboard is pasted. If nothing has been copied previously, nothing is pasted.

This setting has no effect on Copy or Cut when a line is not blank. If nothing is selected, the entire line is copied or cut. If you then Paste, the text of the entire line and its endline character are pasted.

Tip:
To display indicators for spaces, tabs, and line ends, and thus distinguish indented lines from lines that are entirely blank, select Advanced from the Edit menu and choose View White Space.

Display

Line numbers

When selected, a line number appears next to each line of code.

Note: These line numbers are not added to your code, and do not print. They are for reference only.

Enable single-click URL navigation

When selected, the mouse cursor changes to a pointing hand as it passes over a URL in the editor. You can click the URL to display the indicated page in your Web browser.

Navigation bar

When selected, displays the Navigation bar at the top of the code editor. Its drop-down Objects and Members lists allow you to choose a particular object in your code, select from its members, and navigates to the declaration of the selected member in the Code Editor.

Tabs Dialog
This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To display these options, select Options from the Tools menu. Within the Text Editor folder expand the All Languages subfolder, and then choose Tabs.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the Tabs options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

If different settings are selected on the Tabs options pages for particular programming languages, then the message "The indentation settings for individual text formats conflict with each other," is displayed for differing Indenting options; and the message "The tab settings for individual text formats conflict with each other," is displayed for differing Tab options.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Indenting

None

When selected, new lines are not indented. The insertion point is placed in the first column of a new line.

Block

When selected, new lines are automatically indented. The insertion point is placed at the same starting point as the preceding line.

Smart

When selected, new lines are positioned to fit the code context, per other code formatting settings and IntelliSense conventions for your development language. This option is not available for all development languages.

For example, lines enclosed between an opening brace ({) and a closing brace (}) might automatically be indented an extra tab stop from the position of the aligned braces.

Tab and indent size

Sets the distance in spaces between tab stops and for automatic indentation. The default is four spaces. Tab characters, space characters, or both will be inserted to fill the specified size.

Insert spaces

When selected, indent operations insert only space characters, not TAB characters. If the Tab and Indent size is set to 5, for example, then five space characters are inserted whenever you press the TAB key or the Increase Indent button on the Formatting toolbar.

Keep tabs

When selected, each indent operation inserts one TAB character.

9.3.4.8. XML Settings

General Language Options

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To open this dialog box, select Options from the Tools menu. Within the Text Editor folder, expand the All Languages subfolder and then choose General.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the General options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

A grayed checkmark is displayed when an option has been selected on the General options pages for some programming languages, but not for others.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Statement Completion

Auto list members

When selected, pop-up lists of available members, properties, values, or methods are displayed by IntelliSense as you type in the editor. Choose any item from the pop-up list to insert the item into your code. Selecting this option enables the Hide advanced members option. For more information, see List Members.

Hide advanced members

When selected, shortens pop-up statement completion lists by displaying only those items most commonly used. Other items are filtered from the list.

Parameter information

When selected, the complete syntax for the current declaration or procedure is displayed under the insertion point in the editor, with all of its available parameters. The next parameter you can assign is displayed in bold. For more information, see Parameter Info.

Settings

Enable virtual space

When this option is selected and Word wrap is cleared, you can click anywhere beyond the end of a line in the Code Editor and type. This feature can be used to position comments at a consistent point next to your code.

Word wrap

When selected, any portion of a line that extends horizontally beyond the viewable editor area is automatically displayed on the next line. Selecting this option enables the Show visual glyphs for word wrap option.

Note: The Virtual Space feature is turned off while Word Wrap is on.

Show visual glyphs for word wrap

When selected, a return-arrow indicator is displayed where a long line wraps onto a second line.

Clear this option if you prefer not to display these indicators.

Note: These reminder arrows are not added to your code, and do not print. They are for reference only.

Apply Cut or Copy commands to blank lines when there is no selection

This option sets the behavior of the editor when you place the insertion point on a blank line, select nothing, and then Copy or Cut.

When this option is selected, the blank line is copied or cut. If you then Paste, a new, blank line is inserted.

When this option is cleared, the Cut command removes blank lines. However, the data on the Clipboard is preserved. Therefore, if you then use the Paste command, the content most recently copied onto the Clipboard is pasted. If nothing has been copied previously, nothing is pasted.

This setting has no effect on Copy or Cut when a line is not blank. If nothing is selected, the entire line is copied or cut. If you then Paste, the text of the entire line and its endline character are pasted.

Tip:
To display indicators for spaces, tabs, and line ends, and thus distinguish indented lines from lines that are entirely blank, select Advanced from the Edit menu and choose View White Space.

Display

Line numbers

When selected, a line number appears next to each line of code.

Note: These line numbers are not added to your code, and do not print. They are for reference only.

Enable single-click URL navigation

When selected, the mouse cursor changes to a pointing hand as it passes over a URL in the editor. You can click the URL to display the indicated page in your Web browser.

Navigation bar

When selected, displays the Navigation bar at the top of the code editor. Its drop-down Objects and Members lists allow you to choose a particular object in your code, select from its members, and navigates to the declaration of the selected member in the Code Editor.

Tabs Dialog
This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To display these options, select Options from the Tools menu. Within the Text Editor folder expand the All Languages subfolder, and then choose Tabs.

Caution:
This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the Tabs options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

If different settings are selected on the Tabs options pages for particular programming languages, then the message "The indentation settings for individual text formats conflict with each other," is displayed for differing Indenting options; and the message "The tab settings for individual text formats conflict with each other," is displayed for differing Tab options.

Note: The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see Working with Settings.

Indenting

None

When selected, new lines are not indented. The insertion point is placed in the first column of a new line.

Block

When selected, new lines are automatically indented. The insertion point is placed at the same starting point as the preceding line.

Smart

When selected, new lines are positioned to fit the code context, per other code formatting settings and IntelliSense conventions for your development language. This option is not available for all development languages.

For example, lines enclosed between an opening brace ({) and a closing brace (}) might automatically be indented an extra tab stop from the position of the aligned braces.

Tab and indent size

Sets the distance in spaces between tab stops and for automatic indentation. The default is four spaces. Tab characters, space characters, or both will be inserted to fill the specified size.

Insert spaces

When selected, indent operations insert only space characters, not TAB characters. If the Tab and Indent size is set to 5, for example, then five space characters are inserted whenever you press the TAB key or the Increase Indent button on the Formatting toolbar.

Keep tabs

When selected, each indent operation inserts one TAB character.

XML Formatting Options

This dialog box allows you to specify the formatting settings for the XML Editor. You can access the Options dialog box from the Tools menu.

Note: These settings are available when you select the Text Editor folder, the XML folder, and then the Formatting option from the Options dialog box.

Attributes

Preserve manual attribute formatting Attributes are not reformatted. This is the default.

Note: If the attributes are on multiple lines, the editor indents each line of attributes to match the indentation of the parent element.

Align attributes each on their own line

Aligns the second and subsequent attributes vertically to match the indentation of the first attribute. The following XML text is an example of how the attributes would be aligned.

```
<item id = "123-A"
      name = "hammer"
      price = "9.95">
      </item>
```

Auto Reformat

On paste from the Clipboard

Reformats XML text pasted from the Clipboard.

On completion of end tag

Reformats the element when the end tag is completed.

Mixed Content

Preserve mixed content by default

Determines whether the editor reformats mixed content. By default, the editor attempts to reformat mixed content, except when the content is found in an `xml:space="preserve"` scope.

If an element contains a mix of text and markup, the contents are considered to be mixed content. The following is an example of an element with mixed content.

```
<dir>c:\data\AlphaProject\
  <file readOnly="false">test1.txt</file>
  <file readOnly="false">test2.txt</file>
  </dir>
```

XML Miscellaneous Options

This dialog box allows you to change the autocompletion and schema settings for the XML Editor. You can access the Options dialog box from the Tools menu.

Note: These settings are available when you select the Text Editor folder, the XML folder, and then the Miscellaneous option from the Options dialog box.

Auto Insert

Close tags

If the autocompletion setting is checked, the editor automatically adds an end tag when you type a right angle bracket (>) to close a start tag, if the tag is not already closed. This is the default behavior.

The completion of an empty element does not depend on the autocompletion setting. You can always autocomplete an empty element by typing a backslash (/).

Attribute quotes

When authoring XML attributes, the editor inserts the =" " characters and positions the caret (^) inside the double quotes.

Selected by default.

Namespace declarations

The editor automatically inserts namespace declarations wherever they are needed.

Selected by default.

Other markup (Comments, CDATA)

Comments, CDATA, DOCTYPE, processing instructions, and other markup are auto-completed.

Selected by default.

Network

Automatically download DTDs and schemas

Schemas and document type definitions (DTDs) are automatically downloaded from HTTP locations. This feature uses System.Net with auto-proxy server detection enabled.

Selected by default.

Outlining

Enter outlining mode when files open

Turns on the outlining feature when a file is opened.

Selected by default.

Caching

Schemas

Specifies the location of the schema cache. The browse button (...) opens the Directory Browse dialog box at the current schema cache location. You can select a different directory, or you can select a folder in the dialog, right-click, and choose Open to see what is in the directory.

9.3.5. Debugger

9.3.5.1. Usage

In Atmel Studio, you can specify various settings for debugger behavior, including how variables are displayed, whether certain warnings are presented, how breakpoints are set, and how breaking affects running programs. You specify debugger settings in the Options dialog box.

To set debugger options

On the **Tools** menu, click **Options**.

In the **Options** dialog box, open the **Debugging** folder.

In the **Debugging** folder, choose the category of options you want.

9.3.5.2. AVR Debugger Settings

AVR Communication Timeout

Shows the timeout delay used for communication with the back-end. If the watchdog detects that timeout is exceeded the back-end is restarted. 20000ms by default.

AVR Debugger Path

Shows the path to the AVR Debugger.

AVR Debugger Port

Indicates the Windows Comm API Port number, used by the AVR debugger. 0 by default.

RPC transaction times

File name to put statistic logging in. This is log data from the communication with the back-end. Empty means no logging. Note that the file must be written to a directory where the user has write permission. E.g. C:/tmp/transactionlog.csv

User Tool polling

Use internal port polling method for hardware tool discovery, instead of relying on Windows Comm Framework. Must restart Atmel Studio if activated, it may slow down your PC considerably, so use it only if you have errors related to Windows Comm Framework. Disabled by default.

9.3.6. Atmel Software Framework Settings

Path of the application used to compare files

An application is normally used to compare files in the Atmel Software Framework, as such you must specify a path here.

Command line arguments used for file comparison

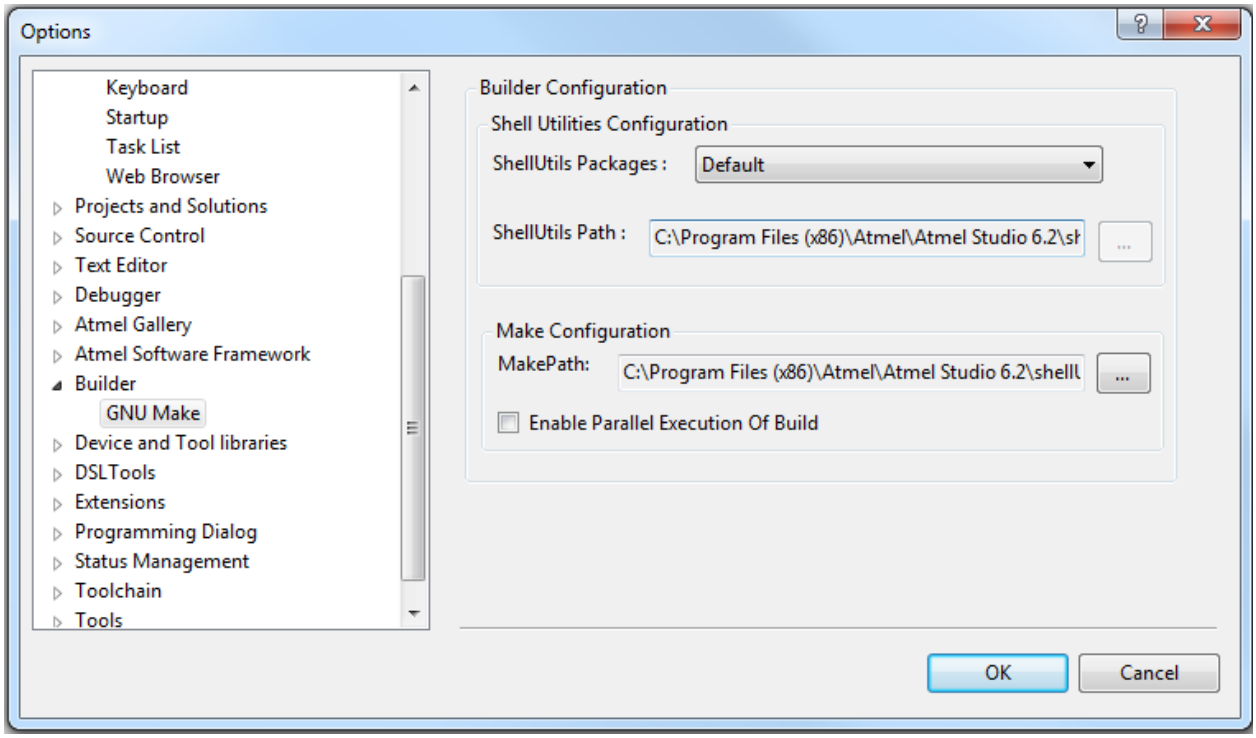
Command line argument macros:

- %original - Path of the original Software Framework file.
- %mine - Path of the modified file in the local project

If the command line for the configured file compare application is `FileCompare.exe filepath1 filepath2`, specify `%original` for `filepath1` and `%mine` for `filepath2`. For example, if configuring WinMerge as the compare application, specify the following command line arguments: `%original %mine /s /u`.

9.3.7. Builder

Figure 9-1. Builder



ShellUtils Packages

It will list Default, Custom, and installed Shell Utility extensions.

ShellUtils Path

Based on the package selected the ShellUtils Path will point to the corresponding utilities folder. If you select a custom ShellUtil package then you can configure a custom Shell utilities folder by clicking on ... button. If you select default or installed shell extension package then the path will be read only and point to the package path.

Make Configuration

You can configure the path to the Make executable by clicking on ... button by default it points to INSTALLDIR\shellUtils\make.exe and you can enable parallel build of projects by checking the box.

9.3.8. Device and Tool Libraries

In the **Devices** sub-menu you can specify the path to custom libraries for your device. In the **Tools** sub-menu, you can specify the path to custom tools for your device.

9.3.9. Status Management

Contains path to the log files and logging settings.

Location

Path to the log file. You can change it by clicking and browsing to the desired location.

Severity threshold

How severe the incident must be in order to generate a log entry. You can choose whether you want to have an output when all operations are successful - **OK** level, when some unorthodox code is present - **Info** level, when some operations have been canceled - **Cancel** setting. If you want to generate output

only in the case when the code is potentially unstable or erroneous, choose either **Warning** or **Error** setting.

Component filter

Filter messages coming from the source code for standard or custom components in your design.

Severity threshold

Meaning identical to the Severity threshold for your source code log generation.

Use filter

Whether the logging process should use a filter to separate components output from your code output.

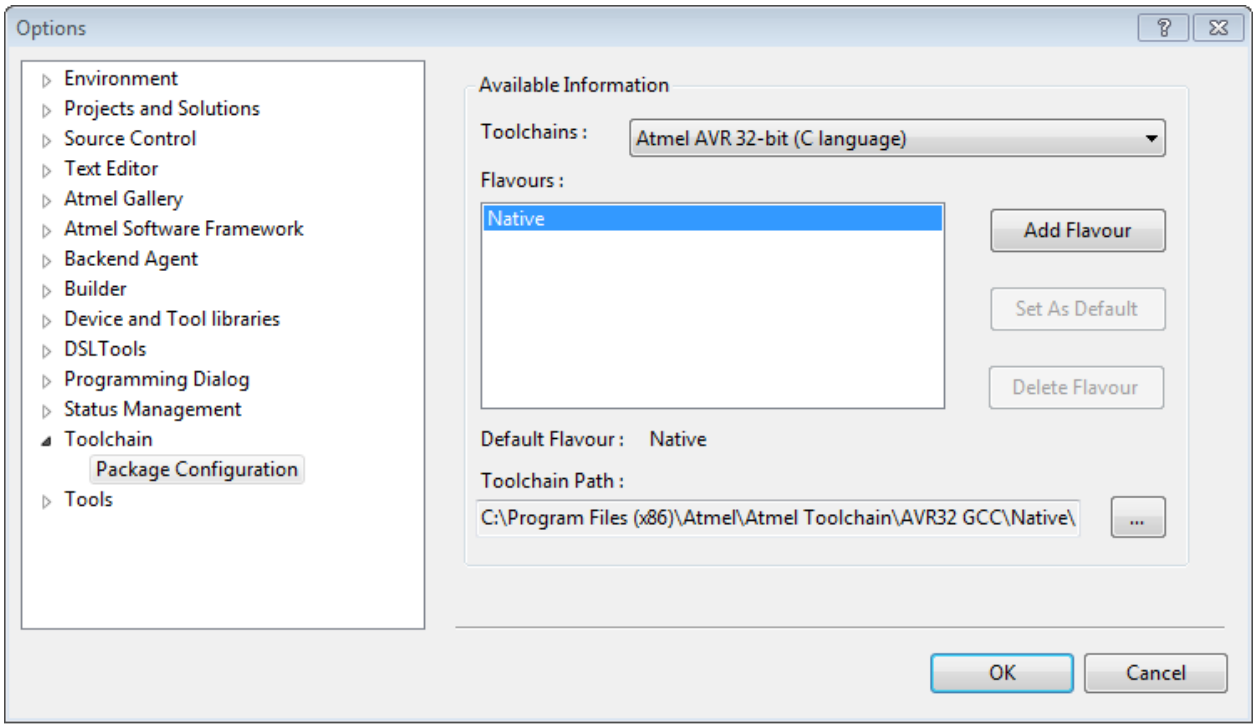
9.3.10. Text Templating

Show security message

Display a dialog prompting the user to ensure that the text templates are from a trusted source when a text transformation operation is initiated.

9.3.11. Toolchain

Figure 9-2. Toolchain Flavor Configuration



Toolchain

Toolchain is used to compile, link, and transform the source code to an executable form targeting the AVR devices. By default, AVR Studio has the following Toolchain Type extensions.

Table 9-2. Toolchain Options

Toolchain type	Language	Description
AVR Assembler	Assembly	Used for building 8-Bit Assembler projects
Atmel AVR 8-bit	C	Used for building 8-Bit C/C++ projects
	C++	
Atmel AVR 32-bit	C	Used for building 32-Bit C/C++ projects
	C++	
Atmel ARM 32-bit	C	Used for building ARM C/C++ projects
	C++	

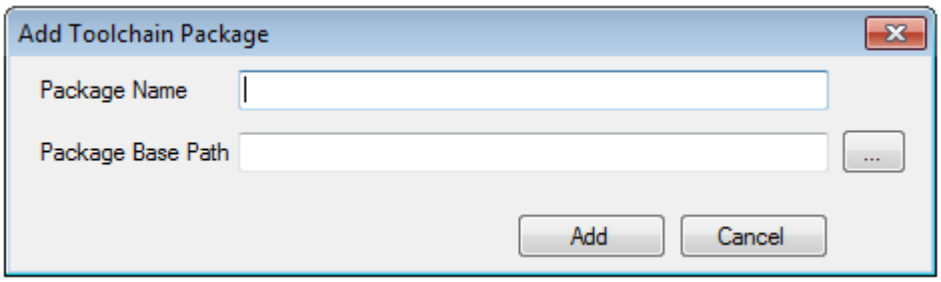
9.3.11.1. Flavor

Flavor identifies a particular version of Toolchain extension of a desired Toolchain type. You could have different flavors of same Toolchain type extensions installed for Atmel Studio.

Add Flavor

1. Select a Toolchain type for which the new Flavor is to be added.

Figure 9-3. Add Toolchain Flavor



2. Enter a new Flavor Name.
3. Configure the Toolchain path for the Flavor. The path should contain desired Toolchain executable, e.g. `avr-gcc.exe` for Atmel AVR 8-bit.
4. Click the **Add** button.

Set Default Flavor

1. Select a Flavor to set as default. The flavor would be the default for the selected toolchain type. Hence, a new project using the toolchain type, would use the configured Flavor settings.
2. You can view and switch between various Flavors after creating the project through the project properties page shown in Advanced Options.

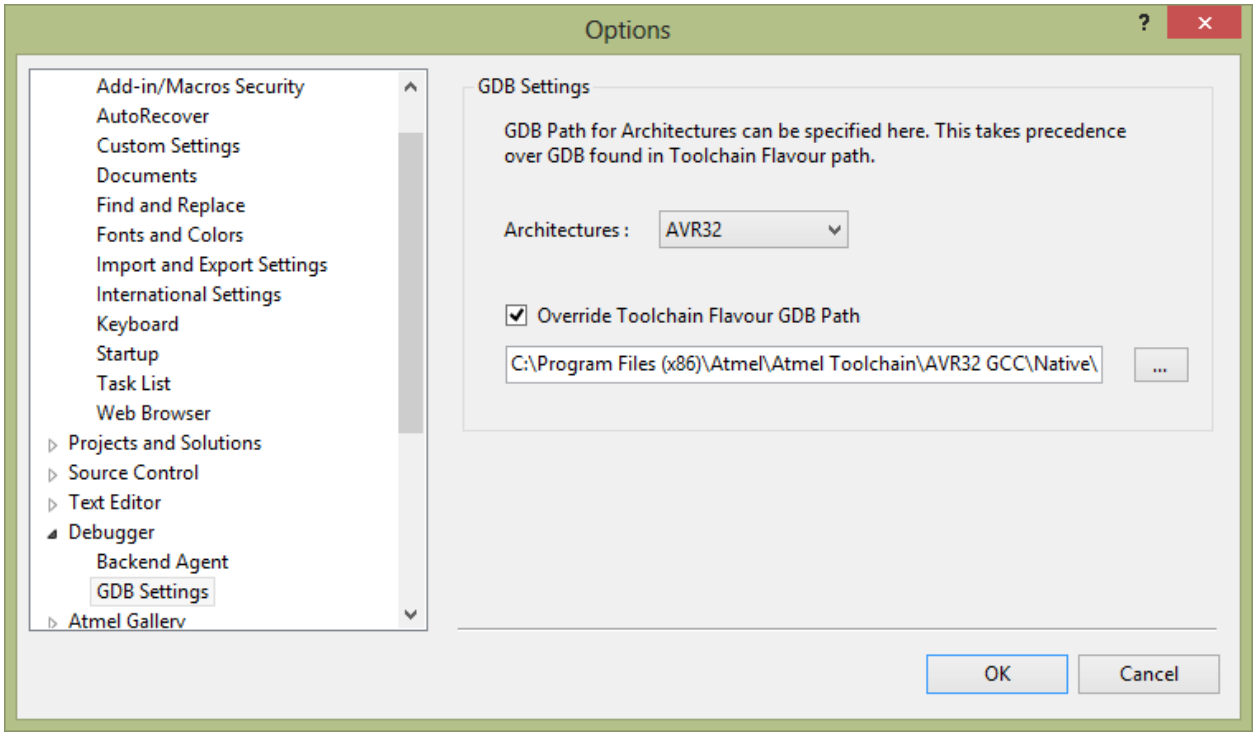
Delete Flavor

Pressing the **Delete Flavor** button deletes the Flavor configuration.

Note: If the customized default flavor is deleted, then the **Native** flavor will be set as default. Also the projects that were configured with the deleted flavor will be changed to the default flavor of the respective toolchain type when the project is opened the next time.

9.3.12. GDB Settings

We can configure architecture specific GDB path in this page. This will override the default toolchain flavor GDB path.



9.4. Code Snippet Manager

Code snippets are particularly useful when writing AVR GCC applications. You can use the Code Snippets Manager to add folders to the folder list that the Code Snippet Picker scans for XML .snippet files. Having these building blocks of code at your disposal can facilitate project development.

The Code Snippets Manager can be accessed from the Tools menu.

9.4.1. Managing Code Snippets

To access the Code Snippets Manager

On the Tools menu, click Code Snippets Manager.

To add a directory to the Code Snippet Manager

1. In the Language: drop-down list, select the language that you want to add a directory to.
2. Click Add. This opens the Code Snippets Directory window.
3. Select the directory that you want to add to the Code Snippets Manager and click OK. The directory will now be used to search for available code snippets.

To remove a directory from the Code Snippet Manager

1. Select the directory that you want to remove.
2. Click Remove.

To import a code snippet into the Code Snippet Manager

1. In the Language: drop-down list, select the language that you want to add the code snippet to.
2. Select the existing folder that you want to place the imported code snippet into.

3. Click Import. This opens the Code Snippets Directory window.
4. Select the code snippet file that you want to add to the Code Snippets Manager and click OK. The code snippet is now available for insertion into the code editor.

9.4.2. Code Snippet Manager Layout

Language

Selects the development language whose code snippet folders are displayed in the folder list.

Location

Displays the path to the folders in the folder list, or to the code snippet file selected there.

Folder list

Shows the set of sub-folders, if any, and the code snippet files available for the Language selected. Click any folder to expand it and list its files.

Description

Displays information on the folder or code snippet file selected in the folder list. When a code snippet file is selected, displays the text from its Author, Description, Shortcut, and Type fields.

Add

Opens the Code Snippet Directory dialog box. Allows you to navigate to the desired snippets folder on your local drive or server, and include it in the folder list.

Remove

Removes a selected top-level folder and its contents from the folder list. Does not physically delete the folder.

Import

Opens the Code Snippet Directory dialog box. Allows you to navigate to the desired snippet on your local drive or server, and add it to an existing code snippet folder.

Security

Whenever you store a new snippet in a folder accessed by the Code Snippets Manager, you are responsible for ensuring that its code is constructed as securely as the rest of your application. Because using code snippets saves development time, snippets can be reused frequently as you construct applications. You should therefore make sure that model code saved in snippets is designed to address security issues. Development teams should establish procedures to review code snippets for compliance with general security standards.

9.4.3. Modifying Existing Code Snippets

IntelliSense Code Snippets are XML files with a .snippet file name extension that can be easily modified using any XML editor, including Atmel Studio.

To modify an existing IntelliSense Code Snippet

1. Use the Code Snippets Manager to locate the snippet that you want to modify.
2. Copy the path of the code snippet to the clipboard and click OK.
3. On the File menu, click Open, and click File.
4. Paste the snippet path into the File location box and click OK.
5. Modify the snippet.
6. On the File menu, click Save. You must have write access to the file to save it.

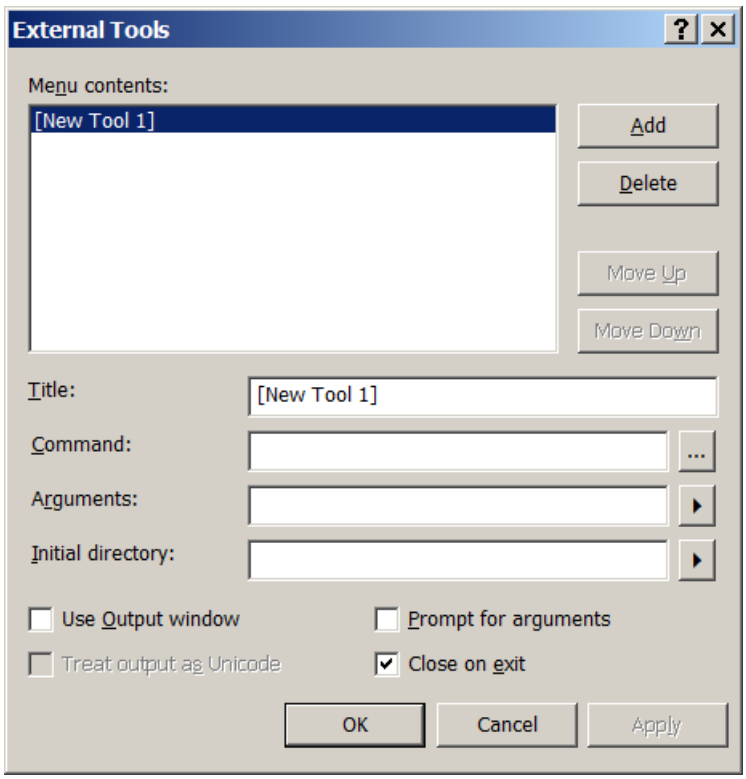
9.5. External Tools

You can add items to the Tools menu that allow you to launch external tools from within Visual Studio. For example, you can add an item to the Tools menu to launch utilities such as **avrdude** or a diffing tool.

9.5.1. Add an External Tool to the Tools Menu

You can add a command to the Tools menu to start another application, such as Notepad, from within the integrated development environment (IDE).

Figure 9-4. External Tool Dialog



The dialog contains a list box where all previously defined external tools are listed. If you have not defined any tool, the list box will be empty.

- On the Tools menu, choose External Tools
- In the External Tools dialog box, choose Add, and enter a name for the menu option in the Title box

Tip:
Type an ampersand before one of the letters in the tool name to create an accelerator key for the command when it appears on the **Tools** menu. For example, if you use M&y External Tool, the letter 'y' will be the accelerator key. See Assign a Keyboard Shortcut for more information.

- In the **Command** box, enter the path to the file you intend to launch, or choose **Browse (...)** to navigate to the file. Files types that you can launch include .exe, .bat, .com, .cmd, and .pif.

  Note: If the file resides on the system path, you can enter just the file name. If not, enter the full path to the file.
- Select Use output window and Close on exit, as appropriate, and then choose OK

9.5.2. Pass Variables to External Tools

You can specify that certain information be passed to a command when it is launched, such as command line switches for console applications.

Fill in the **Arguments** box with the necessary launch arguments, either manually or using the auto-fill button.

The auto-fill argument button can provide you with the macros described in the table below.

Table 9-3. External Tools Macros

Name	Argument	Description
Item Path	`$(ItemPath)`	The complete file name of the current source (defined as drive + path + file name); blank if a non-source window is active.
Item Directory	`$(ItemDir)`	The directory of the current source (defined as drive + path); blank if a non-source window is active.
Item File Name	`$(ItemFilename)`	The file name of the current source (defined as file name); blank if a non-source window is active.
Item Extension	`$(ItemExt)`	The file name extension of the current source.
Current Line	`$(CurLine)`	The current line position of the cursor in the editor.
Current Column	`$(CurCol)`	The current column position of the cursor in the editor.
Current Text	`$(CurText)`	The selected text.
Target Path	`$(TargetPath)`	The complete file name of the item to be built, (defined as drive + path + file name).
Target Directory	`$(TargetDir)`	The directory of the item to be built.
Target Name	`$(TargetName)`	The file name of the item to be built.
Target Extension	`$(TargetExt)`	The file name extension of the item to be built.
Binary Directory	`$(BinDir)`	The final location of the binary that is being built (defined as drive + path).
Project Directory	`$(ProjectDir)`	The directory of the current project (defined as drive + path).
Project file name	`$(ProjectFileName)`	The file name of the current project (defined as drive + path + file name).
Solution Directory	`$(SolutionDir)`	The directory of the current solution (defined as drive + path).
Solution file name	`$(SolutionFileName)`	The file name of the current solution (defined as drive + path + file name).

9.5.3. Initial Directory

You can also specify the working directory for the tool or command. For example, if the tool reads file system data from the current directory, the tool requires that certain program components are present in the current directory at start-up.

9.5.4. Run Behavior

Underneath the argument boxes you can modify the tool behavior.

Use output window - if this box is checked, the tool will output processing information to the Atmel Studio output window, otherwise the output will be suppressed.

Close on exit - if the box is checked the tool window, if any will be automatically closed after completing all operations.

Prompt for arguments - used for toolchain automation. If the box is checked, external tool will require user intervention to input additional processing parameters, otherwise the tool will be silent.

Treat output as Unicode - internationalization option. Some tools have a capacity to output Unicode results for better interpretation. This option allows for correct output rendering if you are using such a tool.

9.5.5. Assign a Keyboard Shortcut

To assign a shortcut (accelerator) to a command, add an ampersand (&) in the title of the tool, just before the letter that you want to use as the access key.

After the ampersand has been added the accelerator needs to be included as a keyboard shortcut.

- On the **Tools** menu, click **Options**
- Select **Keyboard** on the **Environment** page
- In the **Show commands containing** list, type **Tools**
- In the **Command names** list, locate the appropriate **External Command n** entry

  Note: You can define keyboard shortcuts for up to twenty external tools. External tools are listed as External Command 1-20 in the **Command names** list. The numbers correspond to the number to the left of the custom external command name on the **Tools** menu. If the menu command already has a shortcut assigned to it, that information appears in the **Shortcuts for selected command** list.
- Put the cursor in the **Press shortcut keys** box, and then press the keys you want to assign to the external tool

  Note: If the keyboard shortcut is already assigned to another command, the **Shortcut currently assigned to** list will display that information.
- Click **Assign**

9.6. Predefined Keyboard Shortcuts

The Atmel Studio uses the Visual Studio Shell framework from Microsoft Visual Studio 2010 and therefore the integrated development environment (IDE) includes several predefined keyboard shortcut schemes, identical to those in the Visual Studio. When you start Atmel Studio for the first time and select your settings, the associated schemes are automatically set. Thereafter, by using the keyboard options page in the Options dialog box, you can choose from additional schemes and you can also create your own keyboard shortcuts.

Designers and Editors, Shared Shortcuts

These shortcuts work in both designers and editors.

Command	Description	General development, web
Edit.Copy	Copies the selected item to the Clipboard.	CTRL+C or CTRL +INSERT
Edit.Cut	Deletes the selected item from the file and copies it to the Clipboard.	CTRL+X or SHIFT +DELETE
Edit.CycleClipboardRing	Pastes an item from the Clipboard ring to the cursor location in the file. To paste the next item in the Clipboard ring instead, press the shortcut again.	CTRL+SHIFT+V
Edit.Delete	Deletes one character to the right of the cursor.	DELETE
Edit.Find	Displays the Quick tab of the Find and Replace dialog box.	CTRL+F
Edit.FindAllReferences	Displays the list of references for the selected symbol.	SHIFT+ALT+F
Edit.FindinFiles	Displays the In Files tab of the Find and Replace dialog box.	CTRL+SHIFT+F
Edit.FindNext	Finds the next occurrence of the search text.	F3
Edit.FindNextSelected	Finds the next occurrence of the currently selected text, or the word at the cursor.	CTRL+F3
Edit.FindPrevious	Finds the previous occurrence of the search text.	SHIFT+F3
Edit.FindPreviousSelected	Finds the previous occurrence of the currently selected text, or the word at the cursor.	CTRL+SHIFT+F3
Edit.FindSymbol	Displays the Find Symbol pane of the Find and Replace dialog box.	ALT+F12
Edit.GoToFindCombo	Puts the cursor in the Find/Command box on the Standard toolbar.	CTRL+D
Edit.IncrementalSearch	Activates incremental search. If incremental search is on, but no input is passed, the previous search query is used. If search input has been found, the next invocation searches for the next occurrence of the input text.	CTRL+I
Edit.Paste	Inserts the Clipboard contents at the cursor.	CTRL+V or SHIFT +INSERT
Edit.QuickFindSymbol	Searches for the selected object or member and displays the matches in the Find Symbol Results window.	SHIFT+ALT+F12

Command	Description	General development, web
Edit.NavigateTo	Displays the Navigate To dialog box.	CTRL+,
Edit.Redo	Repeats the most recent action.	CTRL+Y or SHIFT+ALT +BACKSPACE or CTRL +SHIFT+Z
Edit.Replace	Displays the replace options on the Quick tab of the Find and Replace dialog box.	CTRL+H
Edit.ReplaceinFiles	Displays the replace options on the In Files tab of the Find and Replace dialog box.	CTRL+SHIFT+H
Edit.SelectAll	Selects everything in the current document.	CTRL+A
Edit.StopSearch	Stops the current Find in Files operation.	ALT+F3, S
Edit.Undo	Reverses the last editing action.	CTRL+Z or ALT +BACKSPACE
View.ViewCode	For the selected item, opens the corresponding file and puts the cursor in the correct location.	CTRL+ALT+0

Text Navigation

These shortcuts are for moving around in an open document.

Command	Description	Shortcut
Edit.CharLeft	Moves the cursor one character to the left.	LEFT ARROW
Edit.CharRight	Moves the cursor one character to the right.	RIGHT ARROW
Edit.DocumentEnd	Moves the cursor to the last line of the document.	CTRL+END
Edit.DocumentStart	Moves the cursor to the first line of the document.	CTRL+HOME
Edit.GoTo	Displays the Go To Line dialog box.	CTRL+G
Edit.GoToDefinition	Navigates to the declaration for the selected symbol in code.	ALT+G
Edit.GoToNextLocation	Moves the cursor to the next item, such as a task in the Task List window or a search match in the Find Results window. Subsequent invocations move to the next item in the list.	F8
Edit.GoToPrevLocation	Moves the cursor back to the previous item.	SHIFT+F8
Edit.IncrementalSearch	Starts incremental search. If incremental search is started but you have not typed any characters, recalls the previous pattern. If text has been found, searches for the next occurrence.	CTRL+I

Command	Description	Shortcut
Edit.LineDown	Moves the cursor down one line.	DOWN ARROW
Edit.LineEnd	Moves the cursor to the end of the current line.	END
Edit.LineStart	Moves the cursor to the start of the line.	HOME
Edit.LineUp	Moves the cursor up one line.	UP ARROW
Edit.NextBookmark	Moves to the next bookmark in the document.	CTRL+K, CTRL +N
Edit.NextBookmarkInFolder	If the current bookmark is in a folder, moves to the next bookmark in that folder. Bookmarks outside the folder are skipped. If the current bookmark is not in a folder, moves to the next bookmark at the same level. If the Bookmark window contains folders, bookmarks in folders are skipped.	CTRL+SHIFT+K, CTRL+SHIFT+N
Edit.PageDown	Scrolls down one screen in the editor window.	PAGE DOWN
Edit.PageUp	Scrolls up one screen in the editor window.	PAGE UP
Edit.PreviousBookmark	Moves the cursor to the location of the previous bookmark.	CTRL+K, CTRL +P
Edit.PreviousBookmarkInFolder	If the current bookmark is in a folder, moves to the previous bookmark in that folder. Bookmarks outside the folder are skipped. If the current bookmark is not in a folder, moves to the previous bookmark at the same level. If the Bookmark window contains folders, bookmarks in folders are skipped.	CTRL+SHIFT+K, CTRL+SHIFT+P
Edit.ReverseIncrementalSearch	Changes the direction of incremental search to start at the bottom of the file and progress toward the top.	CTRL+SHIFT+I
Edit.ScrollLineDown	Scrolls text down one line. Available in text editors only.	CTRL+DOWN ARROW
Edit.ScrollLineUp	Scrolls text up one line. Available in text editors only.	CTRL+UP ARROW
Edit.ViewBottom	Moves to the last visible line of the active window.	CTRL+PAGE DOWN
Edit.ViewTop	Moves to the first visible line of the active window.	CTRL+PAGE UP

Command	Description	Shortcut
Edit.WordNext	Moves the cursor to the right one word.	CTRL+RIGHT ARROW
Edit.WordPrevious	Moves the cursor to the left one word.	CTRL+LEFT ARROW
View.NavigateBackward	Moves to the previously browsed line of code.	CTRL+-
View.NavigateForward	Moves to the next browsed line of code.	CTRL+SHIFT+-
View.NextError	Moves to the next error entry in the Error List window, which automatically scrolls to the affected section of text in the editor.	CTRL+SHIFT +F12
View.NextTask	Moves to the next task or comment in the Task List window.	

Visual Assist shortcuts

These shortcuts are for Visual Assist.

Command	Description	Shortcut
VAssistX.FindReference	Find all references to the marked text.	SHIFT+ALT+F
VAssistX.FindSymbolDialog	Opens the symbols dialog listing all symbols in the project.	SHIFT+ALT+S
VAssistX.GotoImplementation	Go to implementation.	ALT+G
VAssistX.ListMethodsInCurrentFile	Opens the list of all methods in the current file.	ALT+M
VAssistX.OpenCorrespondingFile	Opens the corresponding file (i.e. .h/.c).	ALT+O
VAssistX.OpenFileInSolutionDialog	Displays a list of all files in the solution.	SHIFT+ALT+O
VAssistX.Paste	Shows the paste history menu.	CTRL+SHIFT+V
VAssistX.RefactorContextMenu	Shows the refactor context menu.	SHIFT+ALT+Q
VAssistX.RefactorRename	Shows the rename dialog.	SHIFT+ALT+R
VAssistX.ScopeNext	Jump to next scope.	ALT+Down Arrow
VAssitX.ScopePrevious	Jump to previous scope.	ALT+Up Arrow

Text Selection

These shortcuts are for selecting text in an open document.

Command	Description	Shortcut
Edit.CharLeftExtend	Moves the cursor one character to the left and extends the current selection.	SHIFT+LEFT ARROW
Edit.CharLeftExtendColumn	Moves the cursor to the left one character, extending the column selection.	SHIFT+ALT+LEFT ARROW
Edit.CharRightExtend	Moves the cursor one character to the right and extends the current selection.	SHIFT+RIGHT ARROW
Edit.CharRightExtendColumn	Moves the cursor to the right one character, extending the column selection.	SHIFT+ALT+RIGHT ARROW
Edit.DocumentEndExtend	Selects the text from the cursor to the last line of the document.	CTRL+SHIFT+END
Edit.DocumentStartExtend	Selects the text from the cursor to the first line of the document.	CTRL+SHIFT +HOME
Edit.LineDownExtend	Extends text selection down one line, starting at the location of the cursor.	SHIFT+DOWN ARROW
Edit.LineDownExtendColumn	Moves the pointer down one line, extending the column selection.	SHIFT+ALT+DOWN ARROW
Edit.LineEndExtend	Selects text from the cursor to the end of the current line.	SHIFT+END
Edit.LineEndExtendColumn	Moves the cursor to the end of the line, extending the column selection.	SHIFT+ALT+END
Edit.LineStartExtend	Selects text from the cursor to the start of the line.	SHIFT+HOME
Edit.LineStartExtendColumn	Moves the cursor to the start of the line, extending the column selection.	SHIFT+ALT+HOME
Edit.LineUpExtend	Selects text up, line by line, starting from the location of the cursor.	SHIFT+UP ARROW
Edit.LineUpExtendColumn	Moves the cursor up one line, extending the column selection.	SHIFT+ALT+UP ARROW
Edit.PageDownExtend	Extends selection down one page.	SHIFT+PAGE DOWN
Edit.PageUpExtend	Extends selection up one page.	SHIFT+PAGE UP
Edit.SelectCurrentWord	Selects the word that contains the cursor or the word to the right of the cursor.	CTRL+W
Edit.SelectionCancel	Cancels the current selection.	ESC

Command	Description	Shortcut
Edit.ViewBottomExtend	Moves the cursor and extends the selection to the last line in view.	CTRL+SHIFT +PAGE DOWN
Edit.ViewTopExtend	Extends the selection to the top of the active window.	CTRL+SHIFT +PAGE UP
Edit.WordNextExtend	Extends the selection one word to the right.	CTRL+SHIFT +RIGHT ARROW
Edit.WordNextExtendColumn	Moves the cursor to the right one word, extending the column selection.	CTRL+SHIFT+ALT +RIGHT ARROW
Edit.WordPreviousExtend	Extends the selection one word to the left.	CTRL+SHIFT+LEFT ARROW
Edit.WordPreviousExtendColumn	Moves the cursor to the left one word, extending the column selection.	CTRL+SHIFT+ALT +LEFT ARROW

Text Viewing

These shortcuts are for changing how text is displayed without changing the text itself, for example, by hiding a selected area or by outlining methods.

Command	Description	Shortcut
Edit.ClearBookmarks	Removes all bookmarks in all open documents.	CTRL+K, CTRL+L
Edit.CollapseAllOutlining	Collapses all regions on the page to show just the outermost groups in the hierarchy; typically the using/imports section and the namespace definition.	CTRL+M, CTRL+A
Edit.CollapseCurrentRegion	Collapses the region that contains the cursor to show just the top line of the region, followed by an ellipsis. Regions are indicated by triangles on the left edge of the document window.	CTRL+M, CTRL+S
Edit.CollapseTag	Hides the selected HTML tag and displays an ellipsis (. . .) instead. You can view the complete tag as a tooltip by putting the mouse pointer over the ellipsis.	CTRL+M, CTRL+T
Edit.CollapsetoDefinitions	Collapses existing regions to provide a high-level view of the types and members in the source file.	CTRL+M, CTRL+O
Edit.EnableBookmark	Enables bookmark usage in current document.	
Edit.ExpandAllOutlining	Expands all collapsed regions on the page.	CTRL+M, CTRL+X
Edit.ExpandCurrentRegion	Expands the current region. Put the cursor on a collapsed region to use this command.	CTRL+M, CTRL+E

Command	Description	Shortcut
Edit.HideSelection	Hides the selected text. A signal icon marks the location of the hidden text in the file.	CTRL+M, CTRL+H
Edit.StopHidingCurrent	Removes the outlining information for the currently selected region.	CTRL+M, CTRL+U
Edit.StopOutlining	Removes all outlining information from the whole document.	CTRL+M, CTRL+P
Edit.ToggleAllOutlining	Toggles all previously collapsed outlining regions between collapsed and expanded states.	CTRL+M, CTRL+L
Edit.ToggleBookmark	Sets or removes a bookmark at the current line.	CTRL+K, CTRL+K
Edit.ToggleOutliningExpansion	Toggles the currently selected collapsed region between the collapsed and expanded state.	CTRL+M, CTRL+M
Edit.ToggleTaskListShortcut	Sets or removes a shortcut at the current line.	CTRL+K, CTRL+H
Edit.ToggleWordWrap	Enables or disables word-wrap in an editor.	CTRL+E, CTRL+W
Edit.ViewWhiteSpace	Shows or hides spaces and tab marks.	CTRL+R, CTRL+W

Text Manipulation

These shortcuts are for deleting, moving, or formatting text in an open document.

Command	Description	Shortcut
Edit.BreakLine	Inserts a new line.	ENTER
Edit.CharTranspose	Swaps the characters on either side of the cursor. For example, AC|BD becomes AB|CD.	CTRL+T
Edit.CommentSelection	Applies comment characters for the current language to the current selection.	CTRL+K, CTRL+C
Edit.CompleteWord	Completes the current word in the completion list.	ALT+RIGHT ARROW or CTRL+SPACEBAR
Edit.DeleteBackwards	Deletes one character to the left of the cursor.	BACKSPACE
Edit.FormatDocument	Formats the current document according to the indentation and code formatting settings specified on the Formatting pane in the Options dialog box, for the current language.	CTRL+K, CTRL+D
Edit.FormatSelection	Formats the current selection according to the indentation and code formatting settings specified on the Formatting pane in the Options dialog box, for the current language.	CTRL+K, CTRL+F

Command	Description	Shortcut
Edit.InsertSnippet	Displays the Code Snippet Picker. The selected code snippet will be inserted at the cursor position.	CTRL+K, CTRL+X
Edit.InsertTab	Indents the line of text a specified number of spaces.	TAB
Edit.LineCut	Cuts all selected lines, or the current line if nothing has been selected, to the Clipboard.	CTRL+L
Edit.LineDelete	Deletes all selected lines, or the current line if no selection has been made.	CTRL+SHIFT+L
Edit.LineOpenAbove	Inserts a blank line above the cursor.	CTRL+SHIFT+ENTER
Edit.LineOpenBelow	Inserts a blank line below the cursor.	CTRL+ENTER
Edit.LineTranspose	Moves the line that contains the cursor below the next line.	SHIFT+ALT+T
Edit.ListMembers	Invokes the IntelliSense completion list.	CTRL+J
Edit.MakeLowercase	Changes the selected text to lowercase characters.	CTRL+U
Edit.MakeUppercase	Changes the selected text to uppercase characters.	CTRL+SHIFT+U
Edit.OvertypeMode	Toggles between insert and over-type insertion modes.	INSERT
Edit.ParameterInfo	Displays the name, number, and type of parameters required for the specified method.	CTRL+SHIFT +SPACEBAR
Edit.SurroundWith	Displays the Code Snippet Picker. The selected code snippet will be wrapped around the selected text.	CTRL+K, CTRL+S
Edit.TabifySelectedLines	Replaces spaces with tabs in the selected text.	
Edit.TabLeft	Moves selected lines to the left one tab stop.	SHIFT+TAB
Edit.UncommentSelection	Removes the comment syntax from the current line of code.	CTRL+K, CTRL+U
Edit.UntabifySelectedLines	Replaces tabs with spaces in selected text.	
Edit.WordDeleteToEnd	Deletes the word to the right of the cursor.	CTRL+DELETE

Command	Description	Shortcut
Edit.WordDeleteToStart	Deletes the word to the left of the cursor.	CTRL+BACKSPACE
Edit.WordTranspose	Transposes the words on either side of the cursor. For example, |End Sub would be changed to read Sub End|.	CTRL+SHIFT+T

File and Project Operations

These shortcuts are for file and project operations, and can be used anywhere in the IDE.

Command	Description	Shortcut
Build.BuildSelection	Builds the selected project and its dependencies.	
Build.BuildSolution	Builds all the projects in the solution.	F7
Build.Cancel	Stops the current build.	CTRL+BREAK
Build.Compile	Creates an object file that contains machine code, linker directives, sections, external references, and function/ data names for the selected file.	CTRL+F7
Build.RebuildSolution	Rebuilds the solution.	CTRL+ALT+F7
File.NewFile	Displays the New File dialog box so that you can add a new file to the current project.	CTRL+N
File.NewProject	Displays the New Project dialog box.	CTRL+SHIFT+N
File.OpenFile	Displays the Open File dialog box.	CTRL+O
File.OpenProject	Displays the Open Project dialog box so that you can add existing projects to your solution.	CTRL+SHIFT+O
File.Print	Displays the Print dialog box so that you can select printer settings.	CTRL+P
File.Rename	Lets you modify the name of the item selected in Solution Explorer.	F2
File.SaveAll	Saves all documents in the current solution and all files in the external files project.	CTRL+SHIFT+S
File.SaveSelectedItems	Saves the selected items in the current project.	CTRL+S
File.SaveSelectedItemsAs	Displays the Save File As dialog box when items are selected in the editor.	
Project.AddExistingItem	Displays the Add Existing Item dialog box, which lets you add an existing file to the current project.	

Command	Description	Shortcut
Project.AddNewItem	Displays the Add New Item dialog box, which lets you add a new file to the current project.	
Project.Properties	Displays the Project Properties dialog box for the current project in the editing frame.	

Window Management

These shortcuts are for moving, closing, or navigating in tool windows and document windows.

Command	Description	Shortcut
View.FullScreen	Toggles Full Screen mode ON and OFF.	SHIFT+ALT +ENTER
Window.ActivateDocumentWindow	Closes a menu or dialog box, cancels an operation in progress, or puts focus in the current document window.	ESC
Window.CloseDocumentWindow	Closes the current tab.	CTRL+F4
Window.CloseToolWindow	Closes the current tool window.	SHIFT+ESC
Window.Dock	Returns a floating tool or document window to its most recent docked location in the IDE.	
Window.NextDocumentWindow	Cycles through the open documents.	CTRL+F6
Window.NextDocumentWindowNav	Displays the IDE Navigator, with the first document window selected.	CTRL+TAB
Window.NextPane	Moves to the next pane of the current tool or document window.	ALT+F6
Window.NextToolWindow	Moves to the next tool window.	
Window.NextToolWindowNav	Displays the IDE Navigator, with the first tool window selected.	ALT+F7
Window.PreviousDocumentWindow	Moves to the previous document in the editor.	CTRL+SHIFT+F6
Window.PreviousDocumentWindowNav	Displays the IDE Navigator, with the previous document window selected.	CTRL+SHIFT +TAB
Window.PreviousPane	Moves to the previously selected window.	SHIFT+ALT+F6
Window.ShowEzMDIFileList	Displays a pop-up listing all open documents only.	CTRL+ALT +DOWN ARROW

Tool Windows

These shortcuts are for opening tool windows anywhere in the IDE.

Command	Description	Shortcut
Tools.CodeSnippetsManager	Displays the Code Snippets Manager, which lets you search for and insert code snippets in files.	CTRL+K, CTRL+B
Tools.GoToCommandLine	Puts the pointer in the Find/Command box on the Standard toolbar.	CTRL+/
View.BookmarkWindow	Displays the Bookmark window.	CTRL+K, CTRL+W
View.CallHierarchy	Displays the Call Hierarchy window.	CTRL+ALT+K
View.CommandWindow	Displays the Command window, where commands can be invoked to make changes to the IDE.	CTRL+ALT+A
View.EditLabel	Lets you change the name of the selected item in Solution Explorer.	F2
View.ErrorList	Displays the Error List window.	CTRL+\, E
View.FindSymbolResults	Displays the Find Symbol Results window.	CTRL+ALT+F12
View.Output	Displays the Output window to view status messages at run time.	CTRL+ALT+O
View.SolutionExplorer	Displays Solution Explorer, which lists the projects and files in the current solution.	CTRL+ALT+L
View.TaskList	Displays the Task List window, which displays custom tasks, comments, shortcuts, warnings, and error messages.	CTRL+\, T
View.WebBrowser	Displays the Web Browser window, which lets you view pages on the Internet.	CTRL+ALT+R
Window.PreviousToolWindow	Brings focus to the previous tool-window.	
Window.PreviousToolWindowNav	Displays the IDE Navigator, with the previous tool window selected.	SHIFT+ALT+F7

Bookmark Window

These shortcuts are for working with bookmarks, either in the Bookmarks window or in the editor. For more information, see .

Command	Description	Shortcut
Edit.ClearBookmarks	Removes all bookmarks in all open documents.	CTRL+K, CTRL+L
Edit.EnableBookmark	Enables bookmark usage in current document.	
Edit.NextBookmark	Moves to the next bookmark in the document.	CTRL+K, CTRL+N

Command	Description	Shortcut
Edit.NextBookmarkInFolder	If the current bookmark is in a folder, moves to the next bookmark in that folder. Bookmarks outside the folder are skipped. If the current bookmark is not in a folder, moves to the next bookmark at the same level. If the Bookmark window contains folders, bookmarks in folders are skipped.	CTRL+SHIFT+K, CTRL+SHIFT+N
Edit.ToggleBoomark	Toggles a bookmark on the current line in the document.	CTRL+K, CTRL+K
View.BookmarkWindow	Displays the Bookmark window.	CTRL+K, CTRL +W
Edit.PreviousBookmark	Moves the cursor to the location of the previous bookmark.	CTRL+K, CTRL+P
Edit.PreviousBookmarkInFolder	If the current bookmark is in a folder, moves to the previous bookmark in that folder. Bookmarks outside the folder are skipped. If the current bookmark is not in a folder, moves to the previous bookmark at the same level. If the Bookmark window contains folders, bookmarks in folders are skipped.	CTRL+SHIFT+K, CTRL+SHIFT+P

Debugging

These shortcuts are for debugging code.

Command	Description	Shortcut
Debug.Autos	Displays the Auto window, which displays variables used in the current line of code and the previous line of code.	CTRL+ALT+V, A
Debug.BreakAll	Temporarily stops execution of all processes in a debugging session. Available only in Run mode.	CTRL+F5
Debug.BreakatFunction	Displays the New Breakpoint dialog box.	CTRL+B
Debug.Breakpoints	Displays the Breakpoints dialog box, where you can add, remove, and modify breakpoints.	ALT+F9 or CTRL +ALT+B
Debug.CallStack	Displays the Call Stack window, which displays a list of all active methods or stack frames for the current thread of execution.	ALT+7 or CTRL +ALT+C
Debug.DeleteAllBreakpoints	Clears all the breakpoints in the project.	CTRL+SHIFT+F9

Command	Description	Shortcut
Debug.Disassembly	Displays the Disassembly window.	CTRL+ALT+D or ALT+8
Debug.EnableBreakpoint	Toggles the breakpoint between disabled and enabled.	CTRL+F9
Debug.Exceptions	Displays the Exceptions dialog box.	CTRL+ALT+E
Debug.Immediate	Displays the Immediate window, where expressions can be evaluated.	CTRL+ALT+I
Debug.Locals	Displays the Locals window, which displays the local variables and their values for each method in the current stack frame.	ALT+4 or CTRL +ALT+V, L
Debug.Memory1	Displays the Memory 1 window to view large buffers, strings, and other data that do not display clearly in the Watch or Variables windows.	CTRL+ALT+M, 1
Debug.Memory2	Displays the Memory 2 window to view large buffers, strings, and other data that do not display clearly in the Watch or Variables windows.	CTRL+ALT+M, 2
Debug.Memory3	Displays the Memory 3 window to view large buffers, strings, and other data that do not display clearly in the Watch or Variables windows.	CTRL+ALT+M, 3
Debug.Memory4	Displays the Memory 4 window to view large buffers, strings, and other data that do not display clearly in the Watch or Variables windows.	CTRL+ALT+M, 4
Debug.Modules	Displays the Modules window, which lets you view the .dll or .exe files that are used by the program. In multiprocess debugging, you can right-click and then click Show Modules for all Programs.	CTRL+ALT+U
Debug.ParallelStacks	Opens the Parallel Stacks window.	CTRL+SHIFT+D, S
Debug.ParallelTasks	Opens the Parallel Tasks window.	CTRL+SHIFT+D, K
Debug.Processes	Displays the Processes window. Available in Run mode.	CTRL+ALT+Z
Debug.QuickWatch	Displays the QuickWatch dialog box that has the current value of the selected expression. Available only in Break mode. Use this command to examine the current value of a variable, property, or other expression for which you have not defined a watch expression.	CTRL+ALT+Q or SHIFT+F9

Command	Description	Shortcut
Debug.Registers	Displays the Registers window, which displays registers content for debugging native code applications.	ALT+5 or CTRL +ALT+G
Debug.RunToCursor	In Break mode, resumes execution of your code from the current statement to the selected statement. The Current Line of Execution margin indicator appears in the Margin Indicator bar. In Design mode, starts the debugger and executes your code to the pointer location.	CTRL+F10
Debug.Start	Launches the application under the debugger based off of the settings from the start-up project. When in Break mode, invoking this command will run the application until the next breakpoint.	F5
Debug.StepInto	Executes code one statement at a time, following execution into method calls.	F11
Debug.StepIntoCurrentProcess	Available from the Processes window.	CTRL+ALT+F11
Debug.StepOver	Sets the execution point to the line of code you select.	F10
Debug.StopDebugging	Stops running the current application under the debugger.	CTRL+SHIFT+F5
Debug.Threads	Displays the Threads window to view the running threads.	CTRL+ALT+H
Debug.ToggleBreakpoint	Sets or removes a breakpoint at the current line.	F9
Debug.Watch1	Displays the Watch window, which displays the values of selected variables or watch expressions.	CTRL+ALT+W, 1
Debug.Watch2	Displays the Watch2 window to view the values of selected variables or watch expressions.	CTRL+ALT+W, 2
Debug.Watch3	Displays the Watch3 window to view the values of selected variables or watch expressions.	CTRL+ALT+W, 3
Debug.Watch4	Displays the Watch4 window to view the values of selected variables or watch expressions.	CTRL+ALT+W, 4

Help

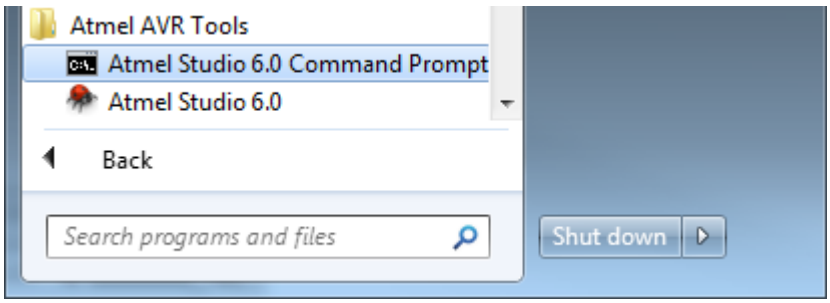
These shortcuts are for viewing topics in Help and moving among them.

Command	Description	Shortcut
Help.F1Help	Displays a topic from Help that corresponds to the user interface that has focus.	F1
Help.ManageHelpSettings	Displays the Help Library Manager.	CTRL+ALT+F1
Help.WindowHelp	Displays a topic from Help that corresponds to the user interface that has focus.	SHIFT+F1

10. Command Line Utility (CLI)

Atmel Studio comes with a command line software utility called `atprogram.exe`.

Tip:
You can start a command shell with the PATH set up to run atprogram by clicking on **Start** > **All Programs** > **Atmel AVR Tools** > **Atmel Studio 6.2 Command Prompt** as shown in the figure below.



`atprogram.exe` can be used to:

- Program a .bin .hex or .elf file to a device
- Verify that the programming was correct
- Read, write, and erase the device memories
- Program fuses, lock bits, security bits, user page, and user signature
- Program a production file to a device [6]
- List out all connected tools
- Set interface and interface clock speeds

To get help on how to use the utility, execute: `atprogram.exe`.

This will print out the atprogram CLI help text on stdout.

[6] The ELF production file format can hold the contents of both Flash, EEPROM and User Signatures (XMEGA devices only) as well as the Fuse- LockBit configuration in one single file. The format is based on the Executable and Linkable Format (ELF).

The production file format is currently supported for tinyAVR, megaAVR, and XMEGA. See Creating ELF Files with Other Memory Types for description on how to configure the project in order to generate such files.

11. Frequently Asked Questions

Frequently asked questions about Atmel Studio.

What is the Atmel USB Driver?

The Atmel USB Driver is a cumulative installer that bundles the Jungo USB driver for the AVR tools and the Segger USB Driver for SAM tools.

I get an error during installation of the Atmel USB Driver Package.

During installation of the Atmel USB Driver Package, you might get the error *0x800b010a - A certificate chain could not be built to a trusted root authority.* This means that the certificate that signs the installer could not be validated using the certificate authority built in to Windows.

The reason for not being able to validate the certificate is because the certificate chain needs to be updated through Windows Update. Make sure that you have received all updates, so that Windows is able to validate the certificate.

If you are not able to update your computer due to the computer being offline or restricted in some way, then the root certificate update can be downloaded from http://support2.microsoft.com/kb/931125.

Will Atmel Studio work in parallel with older versions of Atmel Studio, AVR Studio, and AVR32 Studio?

Yes, it will work side by side between major and minor versions. Side by side installation with different build numbers are not possible. If you are uninstalling AVR Studio 4.0 or AVR32 Studio be careful when you manually delete folders or registry entries after uninstall, as there might be other keys and folders deployed by Atmel Studio inside the Atmel folder and registry paths. Note that drivers may be incompatible between versions.

I have AVR Studio 4 in my PC. When installing Atmel Studio it updated the Jungo USB driver. Will AVR Studio 4 still work?

Yes, it will work. If Jungo driver is already present and its version is anything less than the new one, then the installer will update the Jungo driver you already have. The newest Jungo USB driver (version 12) breaks compatibility with older versions. See KB: Downgrading tools for how to switch between Jungo versions.

Atmel Studio cannot find any debuggers or programmers when Norton AntiVirus is running.

Atmel Studio might not show any connected tools if Norton AntiVirus is running. To make it work make sure Norton AntiVirus allows `atprogram.exe` to communicate with Atmel Studio by adding `atbackend.exe` as an exception in the Norton AntiVirus allowed programs. This is the same with any anti-virus program that by default blocks ports.

Windows shows a message box with the following message when attempting to run Atmel Studio installer: "Windows cannot access the specified device, path or file. You may not have the appropriate permissions to access the item. "

This might be caused by an anti-virus program blocking the installation of the Atmel Studio. We have seen this with the Sophos antivirus package. Temporarily disable the Sophos service running on the machine (or any corresponding anti-virus service), and attempt installation.

Atmel Studio takes a very long time to start, but runs well in a VM environment.	The Visual Studio shell (and thus Atmel Studio) does a considerable amount of processing during start-up. Parts of the operations are WPF operations which benefits greatly by updated graphics libraries and drivers. Installing the latest graphics driver may give a performance boost both during normal operation and during start-up.
Verification and programming often fails with a serial port buffer overrun error message when using STK500.	This is a known issue. Due to DPC latency, serial communication can have buffer overruns on the UART chipset. A workaround which works for most systems is to use an USB to serial adapter.
When launching from a guest account, the following error is displayed when starting Atmel Studio: "Exception has been thrown by the target of an invocation".	Atmel Studio neither installs under guest account and nor runs under it.
Can install and run Atmel Studio from within a Virtual Machine.	Yes, with simulator there should be no issues. However with physical devices like debuggers and programmers, the VM must offer support for physical USB and Serial port connections.
How can I reduce the start-up time of Atmel Studio?	• Make sure you have uninstalled unwanted extensions • Disable *Allow Add-in components to load*: 2.1. Go to *Tools, Options, Add-in/Macro Security*. 2.2. Then uncheck the Allow Add-in components to load option. • Disable the start-up page: 3.1. Go to *Tools, Options, Environment, Startup, At Startup*. 3.2. Select the *Show empty environment* option.
How to improve studio performance for any supported version of Windows?	• Make sure your system has the latest version of the Windows Automation API • Exclude the following directories and files from your antivirus scanner: – The Atmel Studio installation directory, and all files and folders inside it – *%AppData%\Roaming\Atmel* directory, and all files and folders inside it – *%AppData%\Local\Atmel* directory, and all files and folders inside it – Your project directories • Visual Studio Shell requires a lot of swap space. Increase the paging file. Also put the system to maximize performance. Both options are found in the *System, Properties, Performance, Settings* menu.

Should I install the latest Windows Automation API 3.0?	Yes, if your OS is any of the following: • Windows Server 2008
How can I make sure my system has the latest Windows Automation API 3.0?	Your system has the latest Windows Automation API if you have Windows 7 or Windows 8. Only Windows XP, Windows Vista, Windows Server 2003, and Windows Server 2008 have the old version of the API. Find the *UIAutomationCore.dll* file in your system (normally found in the windows folder) and compare the version number of that file. The version should be 7.X.X.X. for the new API. The latest API can be found at http://support.microsoft.com/kb/971513.
My Project is large and it takes a long time to open. Is there any option to avoid this delay?	Visual Assist X parses all the files when we opening the existing project. You could disable this option: 1. Go to *VAssistX, Visual Assist X Options, Performance*. 2. Uncheck the *Parse all files when opening the project*.
I have the limited RAM size in my system and I work long hours in the same instance of Atmel Studio. After some time, Atmel Studio becomes slow on my system.	Press *Ctrl+Shift+Alt+F12* twice to force Atmel Studio to garbage collect.
Does Atmel Studio perform better on multi-core processors than on single-core systems?	Yes, Atmel Studio performs better on a multi-core system.
How can I make my projects build faster?	You can enable parallel build Option from *Tools, Options, Builder, GNU Make, Make Parallel Execution Of Build*. This option will enable the parallel execution feature in the GNU make utility. This option may cause the build log to be displayed unordered.

11.1. Compatibility with Legacy AVR Software and Third-party Products

11.1.1. How do I Import External ELF Files for Debugging?

Use the **File → Open object file for debugging**.

11.1.2. How do I Reuse My AVR Studio 4 Projects with the New Atmel Studio?

1. Click the menu **File→Import AVR Studio 4 project**.
2. An "**Import AVR Studio 4 Project**" dialog will appear.
3. Type in the name of your project or browse to the project location by clicking the **Browse** button of the **APFS File location** Tab.
4. Name the new solution resulting from the conversion of your project in the **Solution Folder** Tab.
5. Click **Next**.
6. Atmel Studio will proceed with conversion. Depending on the complexity and specificity of your project there might be some warnings and errors. They will be shown in the **Summary** window.
7. Click **Finish** to access your newly converted project.

11.2. Atmel Studio Interface

11.2.1. How can I Start Debugging My Code? What is the Keyboard Shortcut for Debugging?

Unlike the AVR Studio 4 to build your project, without starting debugging, you should press F7.

If you need to rebuild your project after a change to the source files, press Ctrl Alt F7 .

To Start debugging - press F5.

To open the Debugging Interface without running directly, press the **Debug→Start Debugging and Break** menu button, or press F11.

To start a line-by-line debugging press F10, to start an instruction by instruction debugging session - press F11.

To run your project without debugging, press the **Debug→Start Without Debugging** menu button.

11.2.2. What is a Solution?

A solution is a structure for organizing projects in Atmel Studio. The solution maintains the state information for projects in .sln (text-based, shared) and .suo (binary, user-specific solution options) files.

11.2.3. What is a Project

A project is a logic folder that contains references to all the source files contained in your project, all the included libraries and all the built executables. Projects allow seamless reuse of code and easy automation of the build process for complex applications.

11.2.4. How can I use an External Makefile for my Project?

The usage of external makefiles and other project options can be configured in the project properties.

Remember that an external makefile has to contain the rules needed by Atmel Studio to work.

11.2.5. When Watching a Variable, the Debugger says `Optimized away`

Most compilers today are what is known as an optimizing compiler. This means that the compiler will employ a number of tricks to reduce the size of your program, or speed it up.
Note: This behavior is usually controlled by the `-On` switches.

The cause of this error is usually trying to debug parts of code that does nothing. Trying to watch the variable `a` in the following example may cause this behavior.

```
int main() {
   int a = 0;
   while (a < 42) {
       a += 2;
   }
}
```

The reason for `a` to be optimized away is obvious as the incrementation of `a` does not affect any other part of our code. This example of a busy wait loop is a prime example of unexpected behavior if you are unaware of this fact.

To fix this, either lower the optimization level used during compilation, or preferably declare `a` as `volatile`. Other situations where a variable should be declared volatile is if some variable is shared between the code and a ISR[7].

For a thorough walk through of this issue, have a look at Cliff Lawsons excellent tutorial on this issue.

[7] Interrupt Service Routine

11.2.6. When Starting a Debug Session, I get an Error Stating that Debug Tool is not Set

The reason for this message is that there are no tool capable to debug that are selected for your project. Go to the Tool project pane and change the to a supported tool.

If the tool you have selected does support debug, then check that the correct interface is chosen and that the frequency is according to the specification. If the issue persist, try to lower the frequency to a frequency where programming is stable, and then slowly increase the frequency as long as it keeps stable.

11.3. Performance Issues

11.3.1. Atmel Studio Takes a Very Long Time to Start on My PC, but Runs Well in a VM Environment. Is there Something I Can do With This?

Visual Studio shell (and thus Atmel Studio) uses WPF as a graphics library and does a lot of processing in the GUI thread. WPF has support for hardware acceleration. Some graphics card drivers does not utilize this well and spend time in kernel space even when no graphics update is required. Installing the latest graphics driver may give a performance boost.

11.3.2. Verification and Programming often Fails with a Serial Port Buffer Overrun Error Message when using STK500

This is a known issue. Interrupt DPC latency for serial communication may be disrupted by other drivers, thus causing buffer overruns on the UART chipset. A workaround which works for most systems is to use a USB to serial adapter.

11.3.3. I've connected my Tool through a USB Hub, and now I get Error Messages and Inconsistent Results while Programming and Debugging

Tools and devices should be connected directly to an USB port on your debugging PC. If this is not an option, you may reduce/eliminate problems by:

- Disconnect any other USB devices connected to the hub
- Switch ports on the USB hub
- Set the tool clock frequency low. *E.g. Set JTAG Clock < 600kHz.*
- If *Use external reset* is an option for your tool/device combination, enable this

Note: The AVR Dragon should be connected through a powered USB hub. This because the power supply on the Dragon can be too weak if the motherboard does not provided enough power. If the Dragon times out or freezes, then the hub might be of to low quality.

11.4. Driver and USB Issues

11.4.1. How do I get my Tool to be Recognized by Atmel Studio?

This should happen automatically, but sometimes the Windows driver does not recognize the tool correctly. To correct this, you have to check that the tool is listed under the **Jungo** item in the device manager in Windows. If your tool is not listed, try to find it under **Unknown devices**. If it is located there, try to reinstall the driver by double clicking the tool, click the **Driver** tab and choose **Update Driver**. Let Windows search for the driver. The driver should be reinstalled and the tool should be displayed under **Jungo**. Now, the tool should be usable from Atmel Studio.

11.4.2. The Firmware upgrade Process fails or is Unstable on a Virtualized Machine

Most tools will perform a reset when asked to switch from normal operation mode to firmware upgrade mode. This forces the tool to re-enumerate on the USB bus, and the virtualization software may not reattach to it making your virtualized system with a disconnected tool.

Normal virtualization software supports the idea of USB filters where you set a collection of USB devices you want to automatically attach to a given guest operating system. Check the manual for your virtualization solution to see how this is done, or see the Firmware Upgrade Fails on VirtualBox.

11.4.3. Debugging never Breaks under a Virtualized Machine

Some virtualization solutions have a limit on how many USB endpoints it supports. This may become an issue if the number of endpoints is lower than the required number for the tool. Usually this causes programming to work as expected but debug not to work as debug events are transmitted on a higher endpoint number.

Check with your virtualization software how many endpoints are available, and on other endpoint specific issues with your virtualization software regarding this.

11.4.4. No Tool is recognized by Atmel Studio, but the Driver seems to be Working

On some systems the **Jungo** driver is known not to activate properly. This can be seen as the **WinDriver** unit under **Jungo** in the device manager in Windows is missing. To remedy this, try the following:

1. In your Device Manager, right click on your computer name (the very top item) and choose **Add Legacy Hardware**.
2. Click next, and choose to install the hardware manually.
3. Choose the **Show All Devices** item on the top of the list, and click next.
4. Click the **Have Disk** button.
5. Navigate to the folder **Atmel USB** which is located under the install directory for Atmel Studio (typical location is **C:\Program Files (x86)\Atmel\Atmel USB**.
6. Choose the **usb32** or **usb64** folder depending on the architecture you are running.
7. Inside there should be only one file named **windrvr#.inf**, where the hash is the revision number for the driver. Double click this, click OK, and the WinDriver should appear in the list. If you get an error message, you probably have chosen the wrong architecture.
8. Click Next until finished.
9. Verify that the **WinDriver** has appeared under **Jungo**.

The tools should be working straight away, but you may have to restart your machine if you are still having problems.

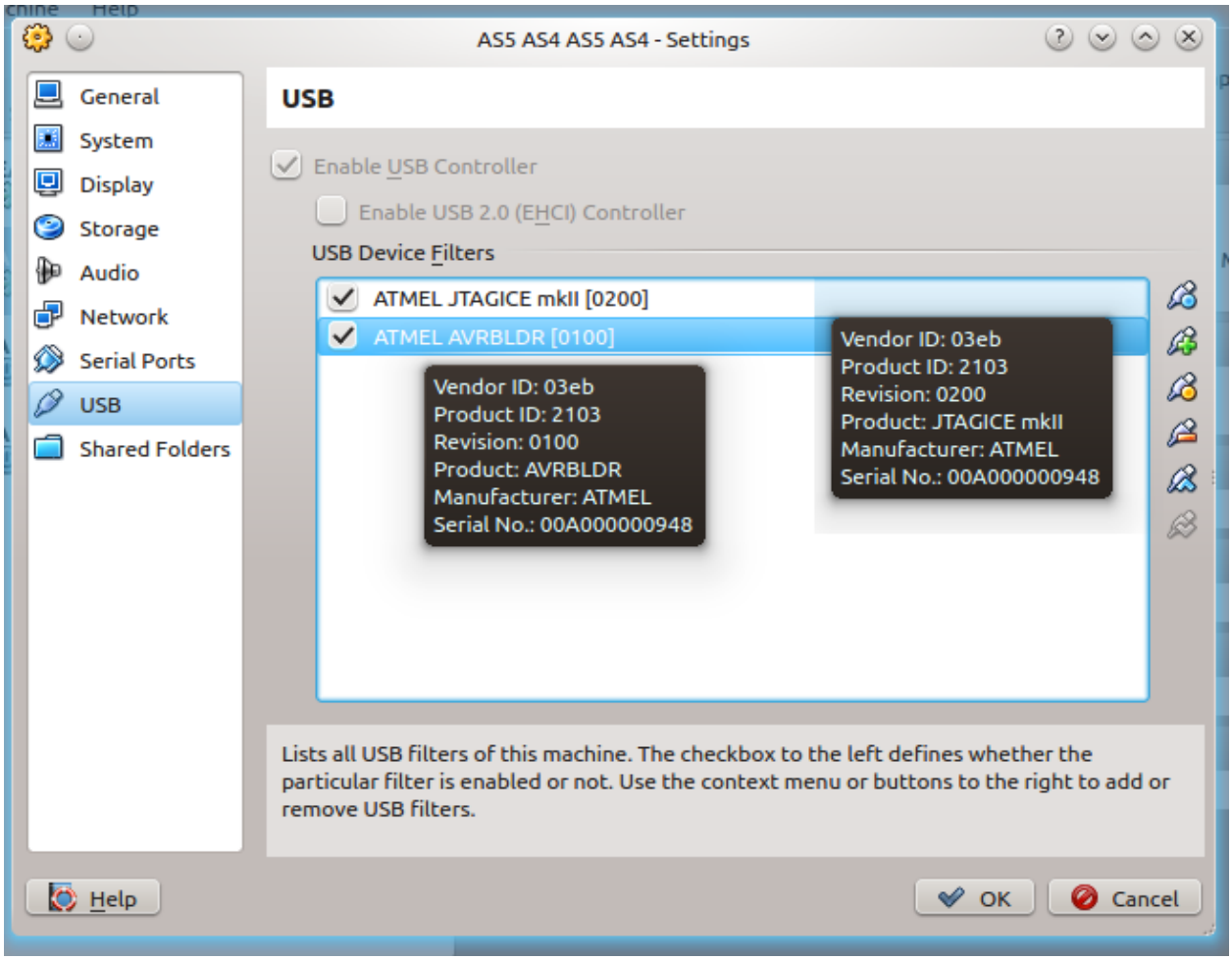
11.4.5. Firmware Upgrade Fails on VirtualBox

When doing a firmware upgrade on any tool, the tool needs to be reconnected in another mode than the one used during regular operation. This causes the tool to be re-enumerated, and can cause the tool to be disconnected from the VirtualBox instance and returned to the host operating system.

To make the tool connect automatically to the VirtualBox instance, you need to set up a couple of USB filters. More information on USB filters can be found in the VirtualBox documentation.

Make two filters that are similar to the two shown in the figure below.

Figure 11-1. VirtualBox USB Filter



Note that the example in the figure above is specific for the JTAGICE mkII. There are one entry for the tool, here the JTAGICE mkII, and one for *AVRBLDR*, which is the firmware upgrade mode for the tool. The name, serial, Vendor ID, and Product ID may be different for your tool, so change those values accordingly.

Note: This section contains specifics to VirtualBox. The same logic applies to other virtualization software, but the steps may differ.

11.4.6. Common Jungo USB Errors

Jungo is the driver stack that is used for older programmers and debuggers, up to the JTAGICE3.

Common Jungo USB Error Codes

Table 11-1. Common Jungo USB Errors

Error	Cause	Resolution
Internal system error	USB subsystem malfunctions	Reinstall driver and check Driver and USB Issues page
Conflict between read and write operations	Directional error in data	Disconnect and reconnect the tool

Error	Cause	Resolution
Data mismatch	Expected and received/sent data error	Make sure that you use the latest driver for your USB controller and the latest firmware for your tool
Packet size is zero	Sent or received a zero packet	
Insufficient resources	Unable to set up send/receive buffers due to memory limitation	Free more memory or try to restart your machine
USB descriptor error	Error in control data on USB bus	Try connection tool to another USB port
Wrong unique ID		
Device not found		
Wrong unique ID		
Timeout expired		
Error	Cause	Resolution

11.4.7. Issues with ARM Compatible Tools

In some rare instances all ARM compatible tools disappears from Atmel Studio. This has been tracked down to different dll load strategies used in different versions of Windows.

To check that it is a dll load error, try to read out the chip information using `atprogram`. This can be done by opening the Atmel Studio command prompt from the **Tools** menu inside Atmel Studio or from the start menu. In the command prompt, enter the following command and check that it does not fail.

```
atprogram -t <tool> -i <interface> -d <device> info
```

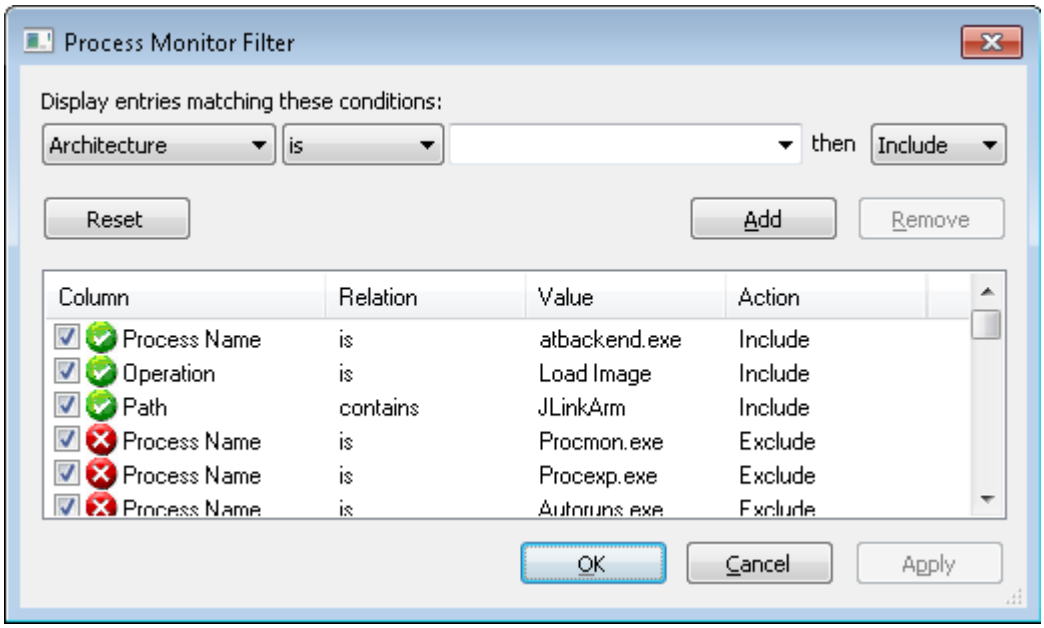
In the snippet above, replace `<tool>` with the tool name, e.g. `atmelice`, `samice`, or `edbg`. Likewise, replace `interface` with the used interface and the `device` with the full device name, e.g. `atsam3s4c`.

Invoking the above command should output information about the memory layout, the supply voltage for the chip, and the fuse settings. If it fails it is likely a driver issue, which is covered by Driver and USB Issues.

If `atprogram` is able to communicate with the device it means that the issue is most likely a wrong version of `JLinkArm.dll` being loaded due to loader precedence. To check this, use the Procmon tool to check what dll is being loaded.

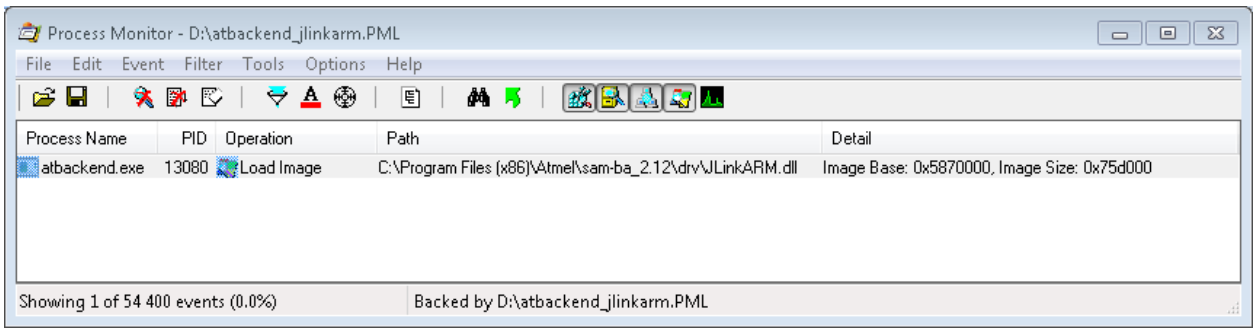
Download the Procmon tool, open it, and configure the filter shown in the figure below. Then start Atmel Studio. A couple of seconds after Atmel Studio has started, one line should become visible showing the path to where the dll is being loaded from. It should be loaded from the `atbackend` folder inside the Atmel Studio installation directory.

Figure 11-2. Procmon Filter Configuration



If the path of the dll is different it means that Atmel Studio has picked up the wrong dll, and this dll is incompatible with the dll shipped with Atmel Studio. An example of this is shown in the figure below.

Figure 11-3. Procmon Filter Configuration



To solve the above issue, we recommend backing up the dll that is being loaded and then replacing it with the `JLinkARM.dll` found in the `atbackend` directory inside the Atmel Studio installation directory. This can be done given the assumption that the dll bundled with Atmel Studio is newer than the one that is being loaded, and the dll is backwards compatible.

Note: Remember to back up the offending `JLinkARM.dll` before replacing it, as it is not given that it will be compatible with the program that deployed it.

12. Document Revision History

Doc Rev.	Date	Comments
42167B	09/2016	Section "Power Debugger" is added
42167A	07/2016	Initial document release.

Index

A

AsmToolchainOptions 3, 60
Atmel Studio 1
AVR Studio 1

C

Choose file 201

D

Device selection 26, 33

T

ToolchainOptions 5, 163, 173

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2016 Atmel Corporation. / Rev.: Atmel-42167B-Atmel-Studio_User Guide-09/2016

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, AVR®, megaAVR®, STK®, tinyAVR®, XMEGA®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, Cortex®, and others are the registered trademarks or trademarks of ARM Ltd. Windows® is a registered trademark of Microsoft Corporation in U.S. and or other countries. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.