



# Integration for CodeBlocks

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .....	
3rd-Party Tool Integrations .....	
Integration for CodeBlocks .....	1
Overview .....	3
Supported Code::Blocks versions .....	4
Plug-in Installation .....	5
Plug-in and TRACE32 Configuration .....	6
Plug-in Configuration	6
TRACE32 Configuration	7
Plug-in Menu and Windows .....	8
Plug-in Menu	8
Memory window	11
Watches window	12
Registers window	12
Breakpoints List	13
Debugging Example Application .....	14

This document describes installation and usage of the TRACE32 Code::Blocks integration.

## Overview

---

The TRACE32 Code::Blocks integration is a plug-in allowing the Code::Blocks user to debug his applications on a real target using the TRACE32 debugger or a simulated target using the TRACE32 Instruction Set Simulator.

**NOTE:**

This integration uses internally the [TRACE32 Remote API](#).  
The Remote API has [restrictions](#) if TRACE32 runs in demo mode.  
Please see there for further details.

# Supported Code::Blocks versions

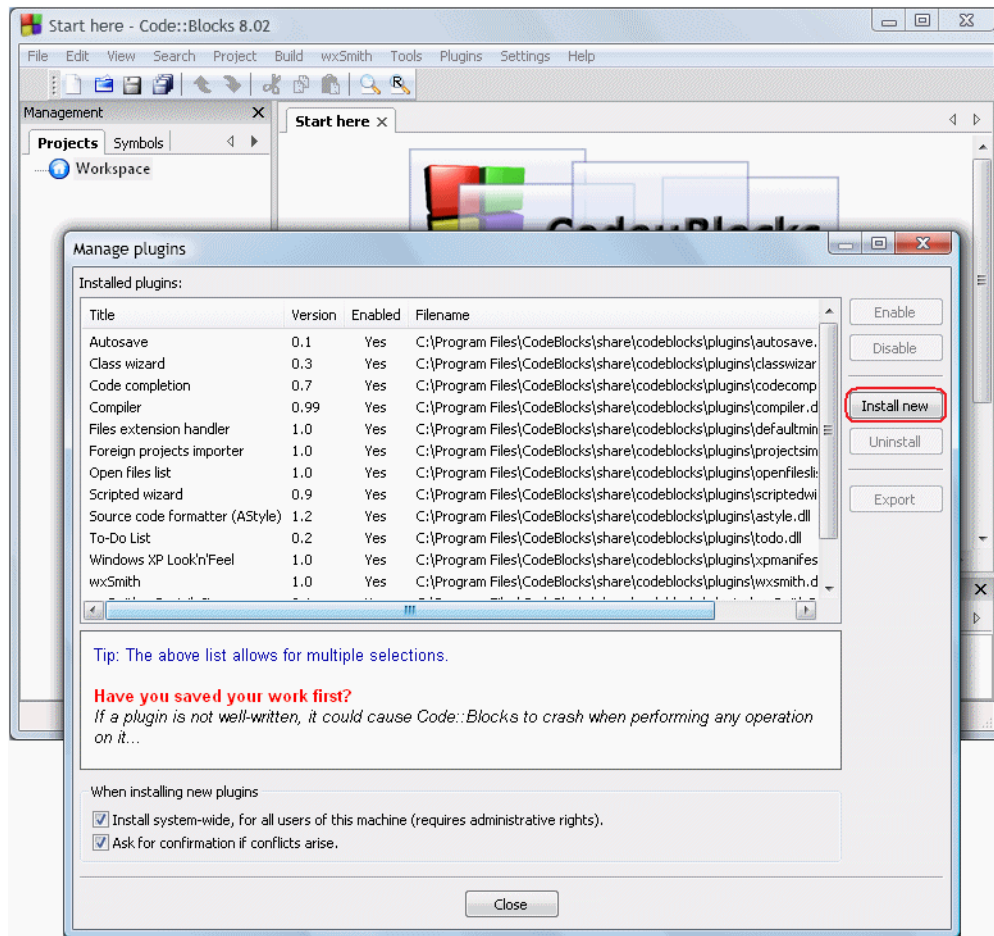
---

The integration plug-in supports following Code::Blocks versions:

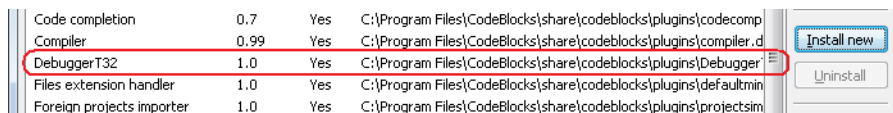
Version	Date
8.02	28 Feb 2008
10.05	30 May 2010
12.11	04 Jul 2013
13.12	06 May 2014

# Plug-in Installation

To install the plug-in, select “Plugins->Manage Plugins” from Code::Blocks menu. Make sure that there is no other debugger plug-in installed in Code::Blocks. To uninstall a debugger plug-in select it from the list and press “Uninstall”. To install TRACE32 integration plug-in select “Install new” and navigate to the `debuggert32.cbplugin` file.



After successful installation DebuggerT32 plug-in appears in the list:



# Plug-in and TRACE32 Configuration

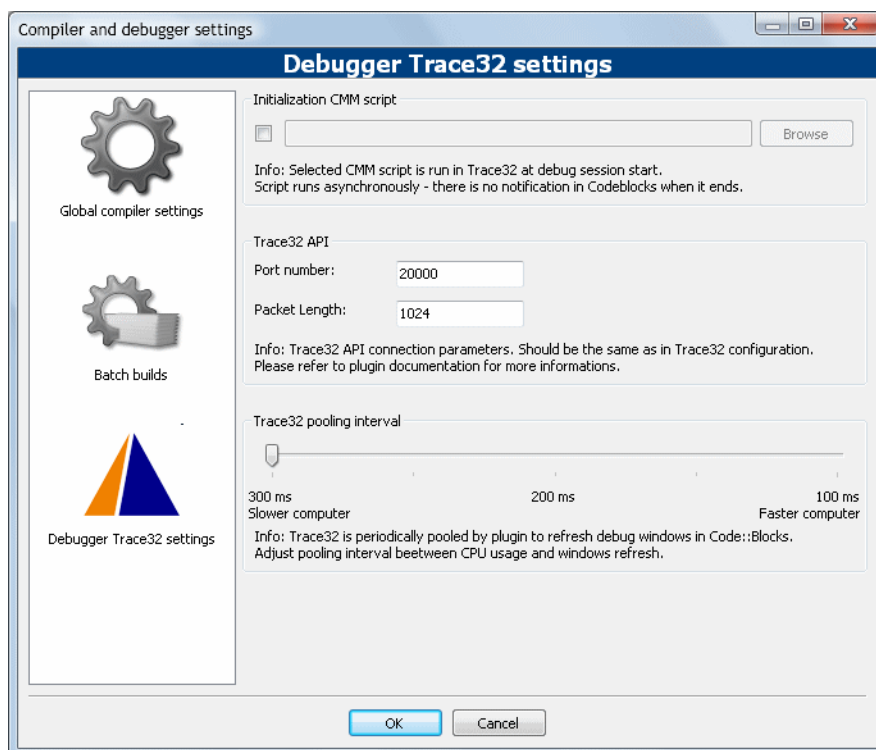
The plug-in and TRACE32 communication is based on TRACE32 API. To enable this communication API UDP port number and packet length need to be configured in both the plug-in and TRACE32.

## Plug-in Configuration

Select “Settings -> Compiler and Debugger -> Debugger TRACE32 Settings” from Code::Blocks menu.

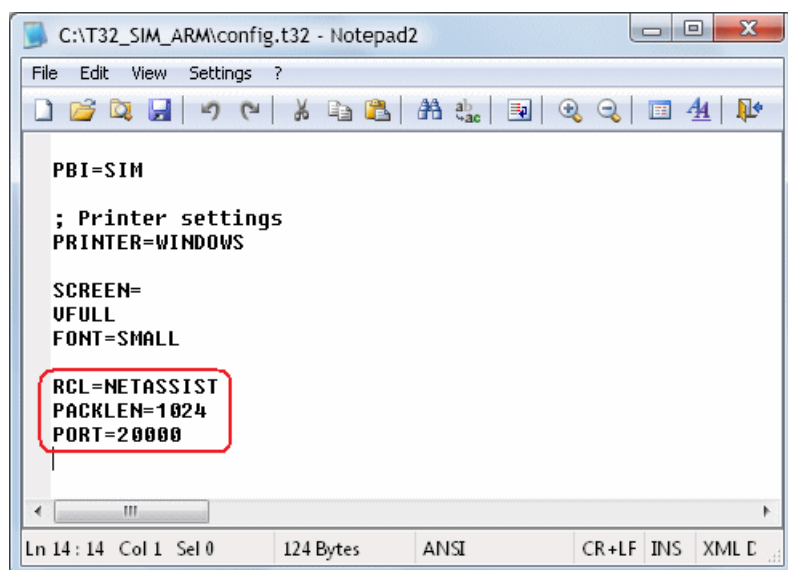
Specify port number and packet length or leave default values.

Additionally a start-up CMM script can be specified. The startup script is executed in TRACE32 each time debug session is started.

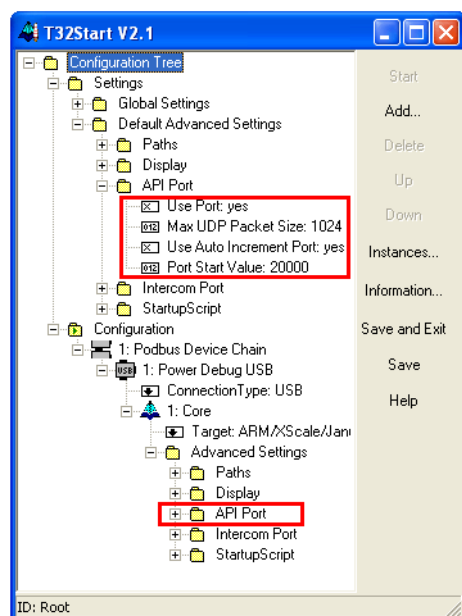


# TRACE32 Configuration

To enable the TRACE32 API communication with the plug-in, some changes need to be made in `config.t32`. This file can be found in your TRACE32 installation directory. Take care to keep one empty line before section with RCL, PORT and PACKLEN.










Changes described above can be also made by using T32Start utility provided with TRACE32:



## Plug-in Menu

After the plug-in is installed, a new Code::Blocks menu with name “TRACE32” appears.

 Debug	F8
 Stop Debugging	Alt-F8
 Download application	Ctrl-F8
 Set next statement	
 Show current statement	
 Show In Trace32	
Memory	▶
Watches	▶
Registers	
 New Breakpoint	F5
Breakpoints list	
 Step	Shift-F7
 Step over	F7
 Go Next	
 Go Return	
 Go Up	Shift-Ctrl-F7
 Go Till...	F4
 Go	Ctrl-F7
 Break	Alt-F7

<b>Debug</b>	Start debug session by connecting to TRACE32 and executing startup script (if specified in plug-in settings. See <a href="#">“Plug-in Configuration”</a> (int_codeblock.pdf) for details)
<b>Stop Debugging</b>	Stop debug session.
<b>Download application</b>	Download application to target. See <a href="#">“Debugging Example Application”</a> (int_codeblock.pdf) for details.
<b>Set next statement</b>	Set program counter to current position in editor.
<b>Show current statement</b>	Open source code for current program counter position.
<b>Show in TRACE32</b>	Open <code>Data.LIST</code> window in TRACE32 with source code at editor’s current position.
<b>Memory</b>	Open memory window. See <a href="#">“Memory window”</a> (int_codeblock.pdf) for details.
<b>Watches</b>	Open watch window. See <a href="#">“Watches window”</a> (int_codeblock.pdf) for details.

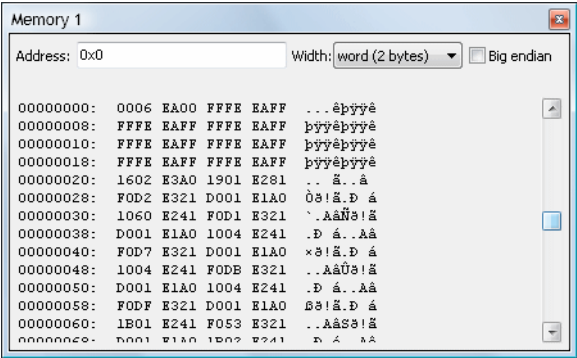


<b>Registers</b>	Open register window. See <a href="#">“Registers window”</a> (int_codeblock.pdf) for details.
<b>New Breakpoint</b>	Set new breakpoint at carret position.
<b>Breakpoints list</b>	Open breakpoints list. See <a href="#">“Breakpoints List”</a> (int_codeblock.pdf) for details.
<b>Step</b>	Step into function call.
<b>Step over</b>	Step over function call.
<b>Go Next</b>	Step to next line. This command can be used to leave loops.
<b>Go Return</b>	Run application to the last instruction in the function.
<b>Go Up</b>	Return to caller function.
<b>Go Till</b>	Run application untill editor’s current position is reached.
<b>Go</b>	Run application.
<b>Break</b>	Break application.

The table below lists features supported in plug-in and their TRACE32 equivalents.

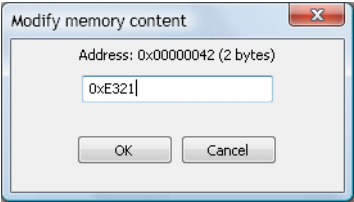
<b>Set next statement</b>	Register.Set PC <address>
<b>Show current statement</b>	Data.List
<b>Show in TRACE32</b>	Data.List <address>
<b>Memory</b>	Data.dump
<b>Watches</b>	Var.Watch
<b>Registers</b>	Register.view
<b>New breakpoint</b>	Break.Set
<b>Breakpoints list</b>	Break.List
<b>Step</b>	Step
<b>Step over</b>	Step.Over
<b>Go Next</b>	Go.Next
<b>Go Return</b>	Go.Return
<b>Go Up</b>	Go.Up
<b>Go Till</b>	Go <address>
<b>Go</b>	Go
<b>Break</b>	Break

# Memory window



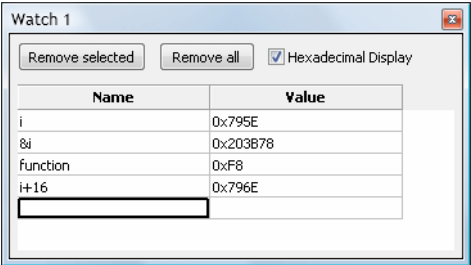
Address	Address expression.
Width	Width of displayed values. Can be either byte, word, long or quad.
Big endian	If activated, values are displayed in big-endian mode.

To modify memory content double click on appropriate value to open “Modify memory content” dialog:



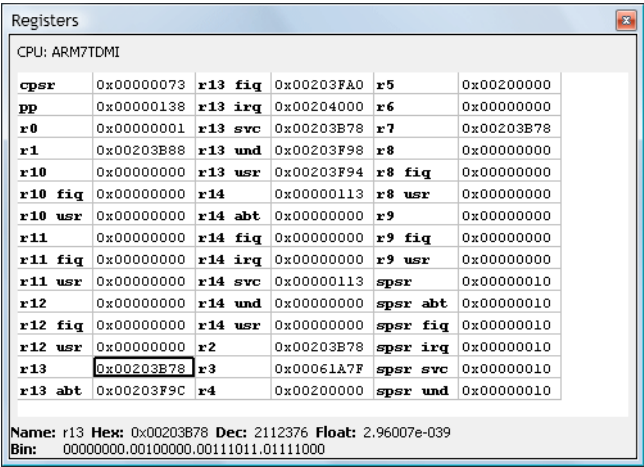
Provided value can be either in hexa-decimal, decimal or floating point format. If floating point value is provided, it is converted to either IEEE754 single or double precision format, depending on width of modified memory content. If width is less than 4 bytes, floating point value cannot be specified.

# Watches window



# Registers window

To change register content, double-click on appropriate value and provide it in hexa-decimal, decimal or floating point value.



# Breakpoints List

This window contains all breakpoints set in current project.

Breakpoints

Address	State	Implement...	Line	File
0x00000102	Enabled	Auto	18	main.c
0x0000010E	Disabled	Auto	20	main.c
0x00000124	Enabled	Auto	22	main.c

✓ Enable

Go to source...

Auto

Software

OnChip

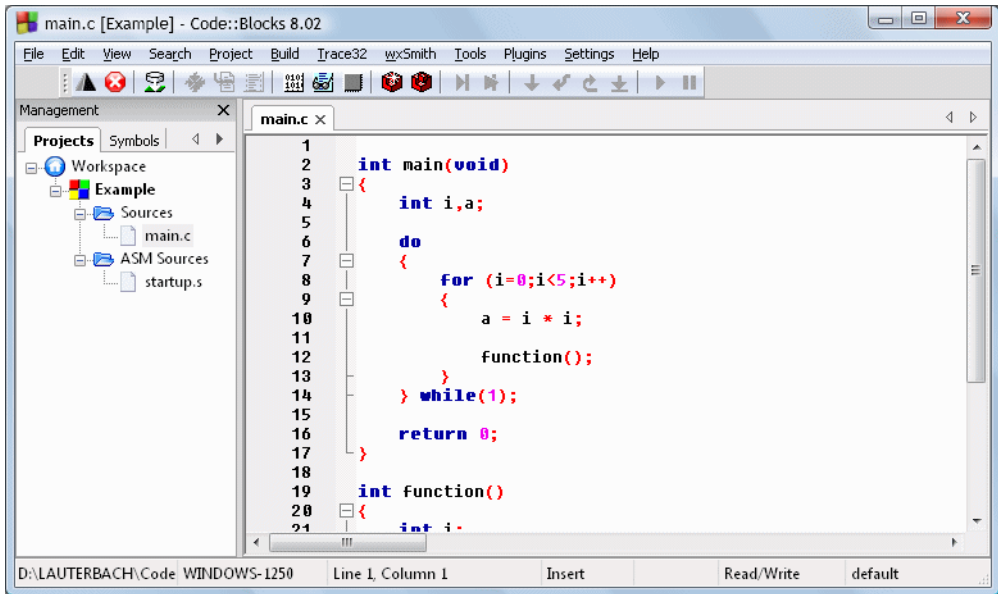
Hardware

Delete

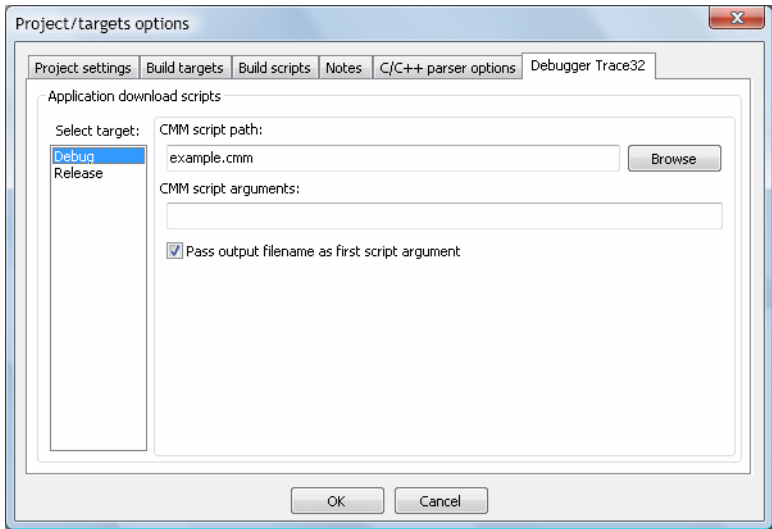
# Debugging Example Application

Start Code::Blocks and the TRACE32 Instruction Set Simulator for ARM.

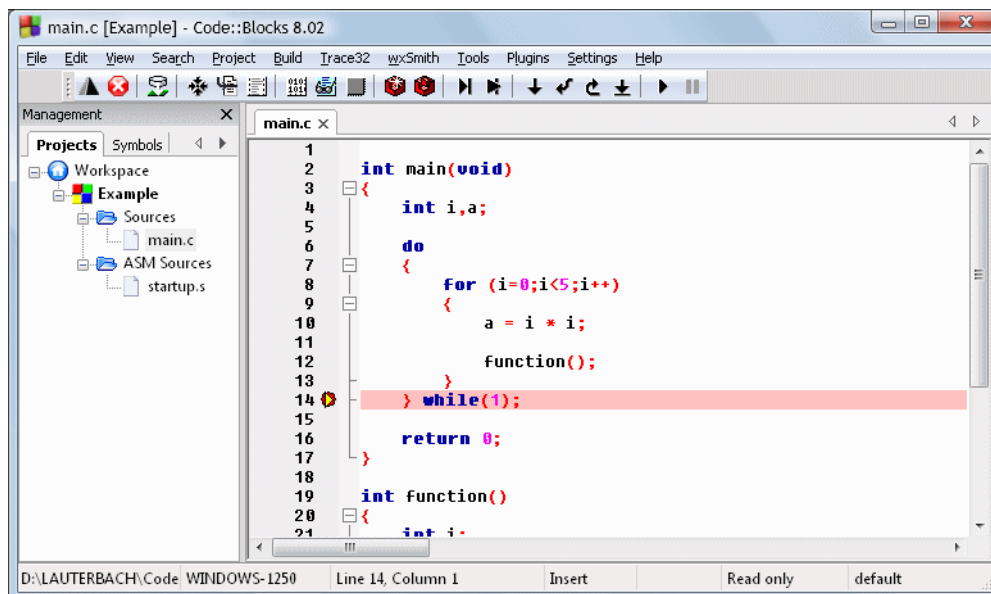
Open project “Example” that is provided with this plug-in and select Debug from plug-in menu to start debug session:



Select “Project -> Properties” from Code::Blocks menu. In tab “Debugger TRACE32” select target “Debug”. These settings specify application download script (in this case example.cmm, that can be found in project directory):



Select “Download application” in the plug-in menu. Download script is executed in TRACE32 and Code::Blocks displays current application state. From this point application can be debugged:



Application source code is also visible in TRACE32. If not, make sure that argument of y.spath command in example.cmm download script points to correct project directory. “Edit source” from context menu can be used to open source code location in Code::Blocks (SETUP.EDITTEXT command with parameter ON need to be specified in download CMM script).

