LAUTERBACH
DEVELOPMENT TOOLS

Integration for EasyCase

MANUAL

Integration for EasyCase

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 🗁

3rd-Party Tool Integrations ..... 🗁

Integration for EasyCase **1**

Overview **4**

Brief Overview of Documents for New Users **4**

Operation Theory **5**

Installation **6**

Startup Sequence **7**

Menu Commands **8**

EasyCODE V6.x Menu 8

Go 8

Break 8

Step Into 8

Step Over 8

Set Breakpoint 9

Delete Breakpoint 9

Add Watch 9

Remove Watch 9

Restart Target 9

Start TRACE32 10

Connect 10

Disconnect 10

Synchronize 10

Get Selection 10

About AddIn 10

EasyCODE V7.x Menu 11

Start 11

Restart 11

Break 11

End 11

Step Over 11

Step Into 12

Step Out 12

Breakpoint 12
Show Value 12
Add Watch 12
Remove Watch 13
Go to Selection 13

Working with the EasyCODE Integration 14

Known Bugs 14

Integration for EasyCase

Version 06-Jun-2024

Overview

This interface integrates the EasyCODE tool (EasyCODE GmbH, Nuremberg, Germany) and TRACE32. Current supported EasyCASE versions are:

Versions: EasyCASE Version 6.0, 6.32, 6.5, 7.0

Hosts: All Windows versions

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Training Basic Debugging"** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"T32Start"** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **"General Commands"** (general_ref_*<x>*.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **"OS Awareness Manuals"** (rtos_*<os>*.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Operation Theory

In EasyCODE 7.x, the interface connects the “Debugging” menu of EasyCODE to TRACE32.

In EasyCODE 6.x, the interface installs an additional menu to EasyCODE, which allows to control TRACE32 main features.

The interface consists of an helper application ("EasyT32.exe") that manages the communication between EasyCODE and TRACE32. EasyCODE passes all requests via Windows messaging to the helper application. This application converts the requests and sends them to the TRACE32 application via sockets (important: WINSOCK has to be installed!). The TRACE32 application transports the requests to the TRACE32. The answer goes exactly the same way backwards.

EasyCODE --> EasyT32 --> t32m*xxx*.exe --> TRACE32

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.

Installation

Follow these steps to install the EasyCODE extension for TRACE32:

1. Copy the file `"~~/demo/env/easycode/*/easyt32.exe"` to your EasyCODE directory, where `*` stands for the *<version>* number.

2. Edit your "config.t32" file: Add between two empty lines the line "RCL=NETASSIST", if not already there.

3. Add a line “sYmbol.CASE OFF” in the system-settings.cmm or your personal start-up script file of TRACE32 (find it in the TRACE32 system directory, e.g. C:\t32). This is necessary because TRACE32 and EasyCODE sometimes uses different cases for module names.

Versions 7.x:

4. If your "`EasyCODE.exe`" file is not in the search path, or if you want to start EasyCODE with parameters, then create a symbol or shortcut to "`easyt32.exe`", specifying the commandline for EasyCODE as parameters. E.g. “`easyt32.exe`” without parameters searches for “`EasyCODE.exe`" in the search path. “`easyt32.exe c:\easy\EasyCODE test.c`” will start "`EasyCODE`" in the specified path with the specified file.

Versions 6.x:

5. Edit your "easy-cpp.ini" file: Insert (or add) an "[AddIn]" section as defined in the file `"~~/demo/env/easycode/*/easy_cpp.ini"`, where `*` stands for the *<version>* number. You can open this file and your .ini file and use "cut&paste" to copy the entries into your file. Do not modify the "[AddIn]" entries - with one exception: Behind the command id, you can specify a shortcut key to the command. See the entry “AddInCmd2”, which is defined with the shortcut key “Shift+F2”.

   NOTE: you won’t see the shortcuts inside the menu.

6. If your "`easy-cpp.exe`" file is not in the search path, or if you want to start EasyCODE with parameters, then create a symbol or shortcut to "`easyt32.exe`", specifying the commandline for EasyCODE as parameters. E.g. “`easyt32.exe`” without parameters searches for “`easy-cpp.exe`" in the search path. “`easyt32.exe c:\easy\easy-cpp test.c`” will start "`easy-cpp`" in the specified path with the specified file.

Startup Sequence

For the first try on the EasyCODE integration, follow the steps below. If you once got experienced, you can customize your own start-up sequence, e.g. starting the TRACE32 driver within EasyCODE.

1. Switch on power on TRACE32 and your target.
2. Start the TRACE32 debugger.
3. Change the directory inside TRACE32 to the demo example.
4. Run the demo example onto “main”.
5. Start EasyT32. It will automatically start EasyCODE. EasyCASE should contain a pull down menu called “Debugging” (V7.x) or "TRACE32" (V6.x).
6. EasyCODE V7.x: EasyT32 automatically connects EasyCODE to TRACE32.
   EasyCODE V6.x: Connect to TRACE32 by invoking the menu "TRACE32 -> Connect".
7. After the application came up, try "Run", "Step" etc.

Menu Commands

EasyCODE V6.x Menu

EasyT32 adds a menu to the EasyCODE menu bar. This chapter describes the added menu items.

Go

Continue application

Equivalent debugger command: "Go"

This command continues the application execution in the debugger.

Break

Stop application

Equivalent debugger command: "Break"

This command stops the application execution in the debugger immediately.

Step Into

Step into function call

Equivalent debugger command: "Step.HII"

Performs an HLL single step. If a function is called, the execution will stop at the beginning of this function.

Step Over

Step over function call

Equivalent debugger command: "Step.Over"

Performs an HLL single step. If a function is called, the execution will stop when the function returned to the caller.

Set Breakpoint

Set breakpoint on current line

Equivalent debugger command: "Break.Set"

Sets a program breakpoint on the line currently marked by the cursor.

Delete Breakpoint

Delete breakpoint on current line

Equivalent debugger command: "Break.Delete"

Deletes the program breakpoint on the line currently marked by the cursor.

Add Watch

Add variable to watch window

Equivalent debugger command: "Var.AddWatch"

Adds the variable marked or selected by the cursor to the watch window inside the TRACE32 display. If no watch window is currently active, one is created.

Remove Watch

Remove variable from watch window

Equivalent debugger command: "Var.DelWatch"

Removes the variable marked or selected by the cursor from the active watch window inside the TRACE32 display.

Restart Target

Do a target restart

Equivalent debugger command: "DO restart"

Performs any practice command. Normally used for restarting the target by invoking a practice file. For this reason the default is "do restart", which implies that there is a practice file called "restart.cmm" containing all restart activities, such as reset and reload.

Start TRACE32

Start debugger application

Start an instance of the TRACE32 application. You are prompted for the start command.

Connect

Connect to TRACE32

Connects EasyCODE to the TRACE32 application. After successful connection a "Synchronize" is performed automatically.

Disconnect

Disconnect TRACE32

Disconnects EasyCODE from the TRACE32 application.

Synchronize

Synchronize editor and program counter

This command synchronizes the EasyCODE editor to the current program counter. The source code is loaded into EasyCODE (if not already loaded), and the cursor is placed at the current source line.

Get Selection

Load source of TRACE32 selection

If a source line in a TRACE32 window was marked (e.g. in a "d.l" or "a.l" window), this command gets the last selection and displays it in the EasyCODE editor. The cursor is placed at the line selected.

About AddIn

Information about “EasyT32.exe”

This menu item gives information about the "EasyT32.exe" file, such as build date and version.

Versions 1.x are 16-bit versions, for use with EasyCODE V6.0.
Versions 2.x are 32.bit versions, for use with EasyCODE V6.32, V6.5.

EasyCODE V7.x Menu

EasyCODE V7.x contains a “Debugging” menu that talks to the debugger. This chapter describes those menu items.

Start

Start debugging

If not yet connected, this command connects EasyCODE to the TRACE32 application. After successful connection, TRACE32 executes a PRACTICE script file with the start-up commands. The default name of this script file is “start.cmm”. If you call EasyT32.exe with the parameter -s<*script_file*>, you can specify any PRACTICE script file to be executed on this menu selection.

Restart

Continue application

Equivalent debugger command: "Go"

This command continues the application execution in the debugger.

Break

Stop application

Equivalent debugger command: "Break"

This command stops the application execution in the debugger immediately.

End

End debugging

Disconnects EasyCODE from the TRACE32 application.

Step Over

Step over function call

Equivalent debugger command: "Step.Over"

Performs an HLL single step. If a function is called, the execution will stop when the function returned to the caller.

Step Into

Step into function call

Equivalent debugger command: "Step.HIl"

Performs an HLL single step. If a function is called, the execution will stop at the beginning of this function.

Step Out

Step out of function

Equivalent debugger command: "Step.Up"

Steps out of this function to the upper level function. The execution will stop when the function returns to the caller.

Breakpoint

Breakpoint arrangement

Sets, deletes, activates and deactivates breakpoints on the line currently marked by the cursor.

Equivalent debugger command: "Break.Set/Delete/ENable/DISable"

Show Value

Show variable value

Show the value of the variable, currently marked by the cursor

Add Watch

Add variable to watch window

Equivalent debugger command: "Var.AddWatch"

Adds the variable marked or selected by the cursor to the watch window inside the TRACE32 display. If no watch window is currently active, one is created.

Remove Watch

Remove variable from watch window

Equivalent debugger command: "Var.DelWatch"

Removes the variable marked or selected by the cursor from the active watch window inside the TRACE32 display.

Go to Selection

Load source of TRACE32 selection

If a source line in a TRACE32 window was marked (e.g. in a "d.l" or "a.l" window), this command gets the last selection and displays it in the EasyCODE editor. The cursor is placed at the line selected.

Working with the EasyCODE Integration

You have to keep some things in mind, when working with the integration. It is not dangerous to use both - EasyCODE integration and TRACE32 debugger - to control the emulator. However in some cases the editor gets confused.

EasyCODE tracks the program counter when single stepping or breaking through its menu. It does not recognize any action done in the TRACE32 debugger. That means, if you perform stepping in TRACE32, EasyCODE will show you the wrong program line.

The EasyCODE integration has its own breakpoint management. Breakpoints that should be displayed in EasyCODE, MUST be set in EasyCODE and MUST be deleted in EasyCODE. EasyCODE will not show breakpoints set in the TRACE32 application. We strongly recommend to use EITHER the EasyCODE integration for breakpoints OR the breakpoint commands in the display driver BUT NOT both mixed together.

Known Bugs

- Multiple HLL breakpoints on single ASM lines

The breakpoint management inside EasyCODE works on HLL lines, while the breakpoint system inside TRACE32 works on hardware addresses. If you set two breakpoints on different HLL lines, which represent the same hardware address (e.g. comments), EasyCODE will have two breakpoints, while TRACE32 will have one. Deleting one of the HLL breakpoints will delete the hardware breakpoint. That means, in EasyCODE there is on breakpoint remaining, while TRACE32 has no breakpoint left.

Workaround: Set breakpoints only on code lines. When detecting multiple breakpoints on a single address, delete all that breakpoints.

- Error message during executing command “Set Breakpoint“:

"Message from TRACE32, Symbol not found in this context".

Solution: add the command sYmbol.CASE OFF inside the startup script of the TRACE32.