LAUTERBACH
DEVELOPMENT TOOLS

Integration for X-Tools and X32

MANUAL

Integration for X-Tools and X32

Integration for X-Tools and X32

Version 06-Jun-2024

Overview

This interface integrates the X-TOOLS / X32 tool series (blue river software GmbH) and TRACE32. Current supported versions are:

X-TOOLS Version 4.03
X32 Version 2.0

Hosts: Windows95, Windows NT

In the further description, both X-TOOLS Classic and X32 are referred to as X-TOOLS.

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

X-TOOLS provides an additional menu, which allows to control TRACE32 main features via X-TOOLS. To access these menu items, you have to install an helper application ("WxT32.exe") that manages the communication between X-TOOLS and TRACE32. X-TOOLS passes all requests via DDE messages to the helper application. This application converts the requests and sends them to the TRACE32 display driver via sockets (important: WINSOCK has to be installed!). The display driver transports the requests to the TRACE32. The answer goes exactly the same way backwards.

X-TOOLS --> WxT32 --> t32 --> 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 X-TOOLS extension for TRACE32:

1. Copy the file "~~/demo/env/xtools/wxt32.exe" to your X-TOOLS directory.
2. Edit your "config.t32" file: Add **between two empty lines** the following statements, if not already there:

   ```
   RCL=NETASSIST
   PACKLEN=1024
   ```

   No "`PORT=`" line should follow. The port defaults to 20000, and this is the only one currently usable.
3. Start X-TOOLS, select menu Extras->Tools…
   Press the “Add” button and choose “WxT32.exe”.
   Press “OK”.
4. If you like, you can add the startup call for your TRACE32 debugger to the “Tools” menu.

Startup Sequence

For the first try on the X-TOOLS integration, follow the steps below. If you once got experienced, you can customize your own startup sequence, e.g. starting the WxT32 only, when it is needed.

1. Switch on power on TRACE32 and your target.
2. Start the TRACE32 debugger.
3. Change the directory inside TRACE32 to your application.
4. Setup TRACE32 and load your application.
5. Start X-TOOLS. If you followed the installation steps above, the “Tools” menu should contain an item called “WxT32”.
6. Start WxT32 by choosing this menu item. WxT32 will automatically establish a DDE connection to X-TOOLS and connect to the TRACE32 debugger. X-TOOLS then loads the current source file and places the cursor to the line that represents the current program counter address.
7. After the application came up, try "Step", “Go” etc.

Menu Commands

X-TOOLS provides a menu (“Debugger”) to control TRACE32. The functionality of these menu items are described in the online help of X-TOOLS. Please see there for details. This chapter contains additional information for each menu item.

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.

List of all Breakpoints — Lists the breakpoints

No actions on TRACE32 performed.

X-TOOLS lists all breakpoints known by itself.

Go — Continue application

Equivalent debugger command: **Go.direct**

This command starts the application. The helper application watches the state of TRACE32 and informs X-TOOLS of an eventual break (e.g. due to a breakpoint).

Break — Stop application

Equivalent debugger command: **Break.direct**

This command stops the application.

Go until Cursor — Continue application until this line

Equivalent debugger command: **Go.Till**

A temporary program breakpoint is set on the marked HLL line and the application is continued. As soon as a breakpoint is hit (either the temporary breakpoint or another set breakpoint), the application is stopped, and the temporary breakpoint is deleted.

Step Over — Step over function call

Equivalent debugger command: **Step.Over**

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

Step Into — Step into function call

Equivalent debugger command: **Step.Hll**

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

Watch Variable — Add variable to watch window

Equivalent debugger command: **Var.View**

Adds the variable marked or selected by the cursor to the watch window inside the TRACE32 display. A view window is created for each variable to watch.

Working with the X-TOOLS extensions

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

X-TOOLS 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 steps in TRACE32, X-TOOLS will show you the wrong program line. But that is harmless, because performing the next “Step Into” inside X-TOOLS will get you back to the right line.

The WxT32 integration tool has its own breakpoint management. Breakpoints that should be displayed in X-TOOLS, MUST be set in X-TOOLS and MUST be deleted in X-TOOLS. X-TOOLS will not show breakpoints set in the TRACE32 debugger. We strongly recommend to use EITHER the X-TOOLS extensions for breakpoints OR the breakpoint commands in the display driver BUT NOT both mixed together.

- Multiple HLL breakpoints on single ASM lines

The breakpoint management inside WxT32 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), X-TOOLS will have two breakpoints, while TRACE32 will have one. Deleting one of the HLL breakpoints will delete the hardware breakpoint. That means, in X-TOOLS there is one 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.