



Coupling for Eclipse

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

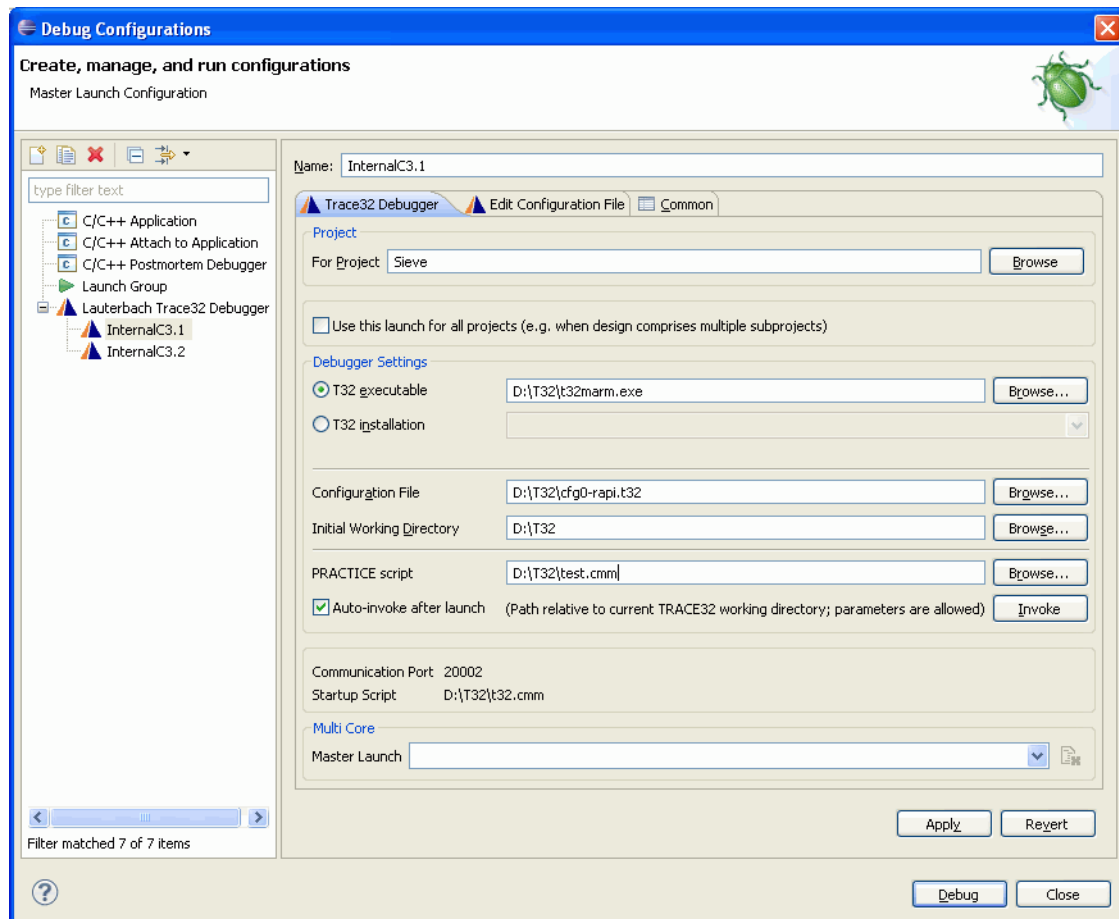
[TRACE32 Index](#)

TRACE32 Documents	
3rd Party Tool Integrations	
Coupling for Eclipse	1
Introduction	3
Supported Eclipse/IDE and CDT version combinations	4
Documentation Updates	5
Brief Overview of Documents for New Users	5
Abbreviations	5
Typical Use Case	6
Installation	7
Requirements	7
Lauterbach TRACE32 Eclipse Plug-In	8
Add the Lauterbach TRACE32 Update Site	8
Install the Plug-In	11
Create Launch Configurations	12
Debugger Startup	16
Parameters for the Startup Script t32.cmm	16
Invoke PRACTICE Scripts	16
Invoke a Script from the Toolbar	17
Add the Lauterbach Logo Button	17
Single-Core Launch with Multiple Eclipse Projects	18
Multi-Core Launch Configurations	18
Breakpoint Synchronization	19
Edit Source Functionality	20
Troubleshooting	21
Eclipse	21
BreakpointSynchronization and EditSource fail	21
Not all Source Code files are defined inside an Eclipse Project	21
Source Code Path names don't match between TRACE32 and Eclipse	21
Plug-in Connect Error Message in TRACE32	22
Update Site not Found (http-ProxySettings OK)	23
Using a LocalUpdateSite for installation:	23

Failed to Connect to TRACE32	23
Unable to Load Class	23
TRACE32	24
Check Your TRACE32 Version	24
Illegal Character (xxxx) for this Context	24
Symbol not Found	24
Symbol not Found in this Context	24
t32.cmm not Executed	24
Help Us Help You	25
Export the Eclipse Error Log	25
Export the Eclipse Configuration	25
Export TRACE32 Information	25
Change Log	26

Introduction

The Lauterbach TRACE32 Eclipse plug-in provides “loose coupling” between Eclipse IDE and the TRACE32 GUI. Both environments are used for the tasks they are best suited for: Eclipse IDE for development (coding, build, version control), TRACE32 for debugging.



The plug-in supports the features most requested by Lauterbach customers:

- Start TRACE32 from an Eclipse launch configuration
- Support for multiple projects (multi-core)
- Synchronization of breakpoints between Eclipse and TRACE32
- 'Open source file' functionality from TRACE32 to Eclipse and vice versa

The plug-in does **not** provide access to the full TRACE32 debug functionality (go, stop, step, variable view, ...) from within Eclipse. If you need this, please consider using TRACE32 as gdbserver (see below).

Supported Eclipse IDE and CDT version combinations

Eclipse 4.5 (Mars)	with CDT 8.7
Eclipse 4.4 (Luna)	with CDT 8.6
Eclipse 4.3 (Kepler)	with CDT 8.2
Eclipse 4.2 (Juno)	with CDT 8.1
Eclipse 3.7 (Indigo)	with CDT 8.0
Eclipse 3.6 (Helios)	with CDT 7.0
Eclipse 3.5 (Galileo)	with CDT 6.0
Eclipse 3.4 (Ganymede)	with CDT 5.0
Eclipse 3.3 (Europa)	with CDT 4.0

Other versions may work, but have not been tested by Lauterbach.

Please understand that **we cannot test, debug and support installations with modified Eclipse or CDT components**. If an IDE is “based on Eclipse” or “derived from Eclipse/CDT”, it usually contains heavily modified components that often break standard plug-in compatibility. For these environments, using TRACE32 as gdbserver (see below) might be a viable alternative.

NOTE: For using the plug-in a **TRACE32 software update may be required**.

For full functionality the TRACE32 software must be from **February 2013 (build rev. 42425 or later)**.

TRACE32 also supports the gdbserver protocol ([api_gdb.pdf](#)). With this, an Eclipse/CDT “external gdb” debug configuration can be used for application debugging with TRACE32.

The latest version of this document is available for download from:
http://www.lauterbach.com/eclipse/doc/int_eclipse.pdf.

Brief Overview of Documents for New Users

Architecture-independent information:

- **”Debugger Basics - Training”** (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**.
- **”RTOS Debugger”** (rtos_<x>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate RTOS manual informs you how to enable the OS-aware debugging.

Abbreviations

GUI	Graphical User Interface
IDE	Integrated Development Environment
JDK	Java Development Kit
JRE	Java Runtime Environment
SMP	Symmetric Multi-Processing

The Eclipse plug-in for Lauterbach TRACE32 provides “loose coupling” between the Eclipse IDE and the TRACE32 GUI.

The plug-in adds an Eclipse Launch Configuration to start an installed TRACE32 debugger.

For TRACE32 instances started via this mechanism (and only for these), the plug-in enables you to:

- Synchronize breakpoints between Eclipse IDE and TRACE32
- Open an Eclipse IDE source editor with the same file as currently displayed in TRACE32 via the TRACE32 context menu entry **Edit Source**.
- Open a TRACE32 **Data.List** window with the same file as displayed in the editor pane from the Eclipse IDE context menu **Open in Trace32**.

A typical workflow is to implement a new feature inside Eclipse, build the executable file, and use TRACE32 for debugging:

1. After the build, in the Eclipse IDE select a **Debug Launch Configuration** that starts TRACE32.
2. TRACE32 invokes a PRACTICE script (provided by you) to set up the target.
3. The PRACTICE script downloads the modified executable to the target.
4. The PRACTICE script starts the executable for debugging with TRACE32.
5. When the position of an error is identified, a right-click on its source line in the TRACE32 **Data.List** window opens a context menu.
6. Selecting **Edit Source** from the context menu jumps back to the same file position in Eclipse.
7. Eclipse opens the requested source file and positions the cursor on the correct line.
8. The error is fixed (e.g., by you).
9. To verify the fix is correct, a breakpoint is set inside Eclipse at the affected source line.
10. The breakpoint is communicated from Eclipse IDE to TRACE32.
11. You re-build the changed executable, load it into the target, and start it (e.g., with a script). The processor will stop at the breakpoint set earlier.

NOTE:

Eclipse needs all source code to be organized within Eclipse projects.

If a source file is not part of an Eclipse project, the plug-in will not be able to communicate breakpoints or provide the **Edit Source** functionality for it.

Requirements

- TRACE32 installation from **February 2013 or later**
- Eclipse IDE and CDT versions as listed in the [Introduction](#) section
- Eclipse configured for Java Runtime Environment (**required minimum is JRE 1.5.0_12**)
- Correct **http-proxy configuration** for download and installation of the plug-in (only required if you use an HTTP proxy at your development site)

If the TRACE32 installation is too old, the plug-in will not work correctly. In the **AREA** window, TRACE32 then prints error messages like

```
illegal character (xxxx) for this context
```

If the Java Runtime Environment used for Eclipse is too old, you will get this error message:

```
Plug-in com.lauterbach.trace32.debug.t32 was unable to load class  
com.lauterbach.trace32.debug.internal.ui.T32LaunchConfigurationTabGroup
```

To use the TRACE32 plug-in for Eclipse, you first need the Eclipse IDE itself, complete with the “Eclipse C/C++ Development Tooling” (CDT). Both are available from www.eclipse.org.

NOTE:

Set the **http-proxy configuration** in the Eclipse dialog **Window > Preferences**, in the section **Install/Update** or in **General > Network Connections**.

(The exact location depends on your Eclipse version.)

Lauterbach TRACE32 Eclipse Plug-In

Install the plug-in via the Eclipse dialog **Help > Software Updates** (up to Eclipse 3.4) or via **Help > Install New Software** (Eclipse 3.5 and later).

NOTE: Your Linux or Windows user account needs to have the necessary rights to install and update plug-ins for Eclipse. Otherwise the Lauterbach TRACE32 Eclipse Plug-In can not be properly installed or updated.

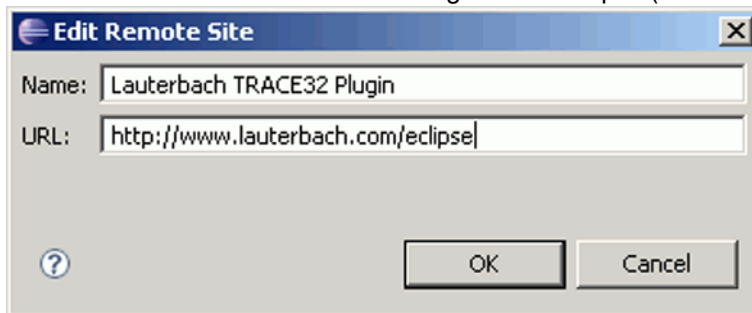
Add the Lauterbach TRACE32 Update Site

First you need to add the **Lauterbach TRACE32 Update Site** to the list of remote sites Eclipse IDE can install updates and new software from. The correct URL is <http://www.lauterbach.com/eclipse>

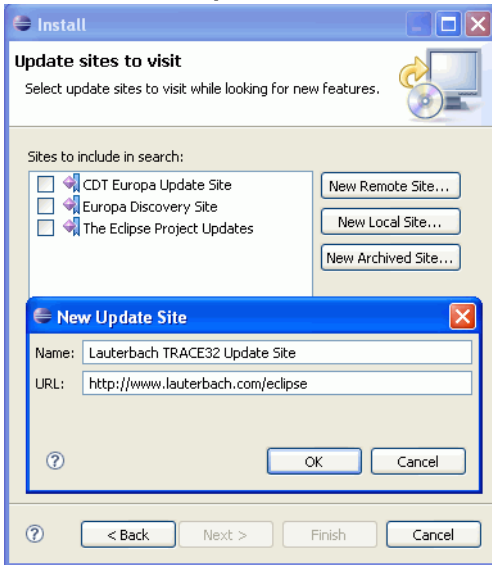
NOTE: If you add a '/' slash character to the end of the URL, older Eclipse IDE versions may have a problem to connect to the Update Site.

The Eclipse IDE option path to install a new plug-in starts in the **Help** menu. The exact dialog sequence and naming has changed with almost every major Eclipse IDE version.

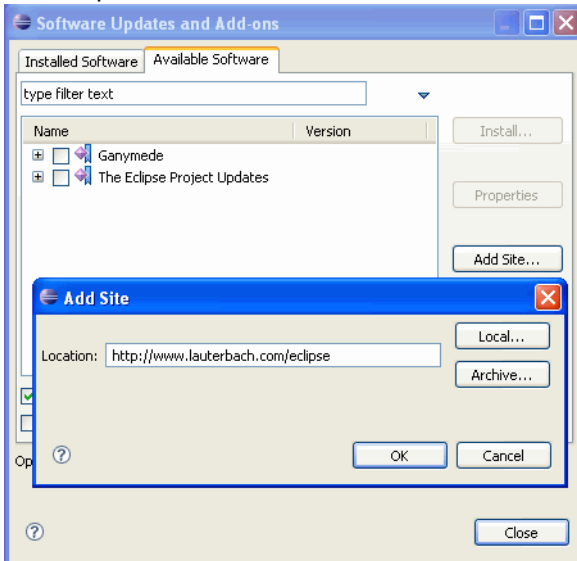
This is how the **Edit Remote Site** dialog looks in Eclipse (before 3.3):



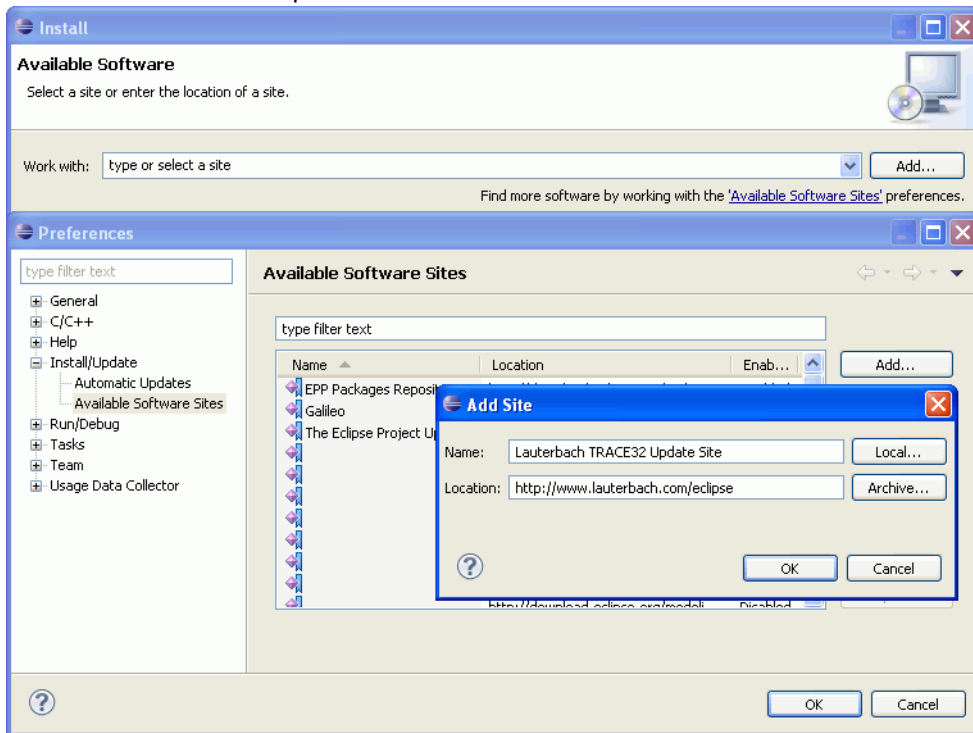
Here is the **New Update Site** variant of Eclipse 3.3:



The Eclipse 3.4 **Add Site** version:



This is how it looks in Eclipse 3.5 and later:

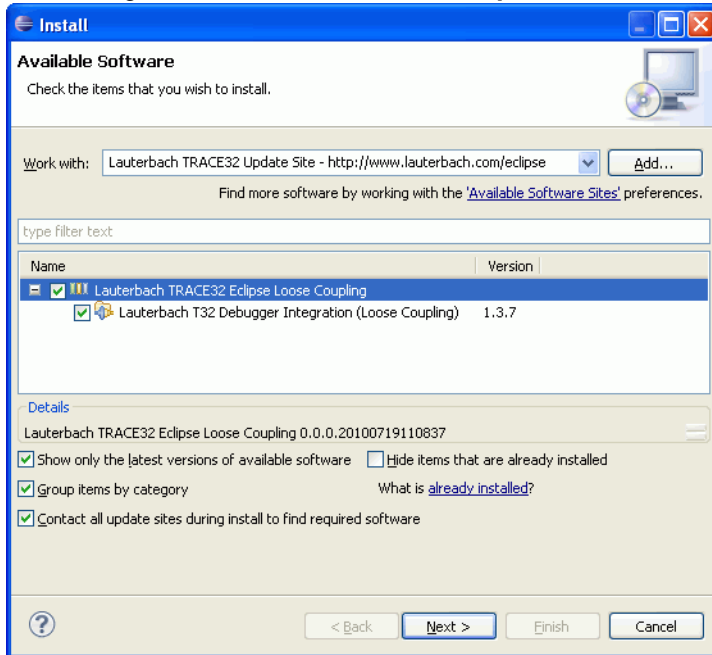


NOTE:

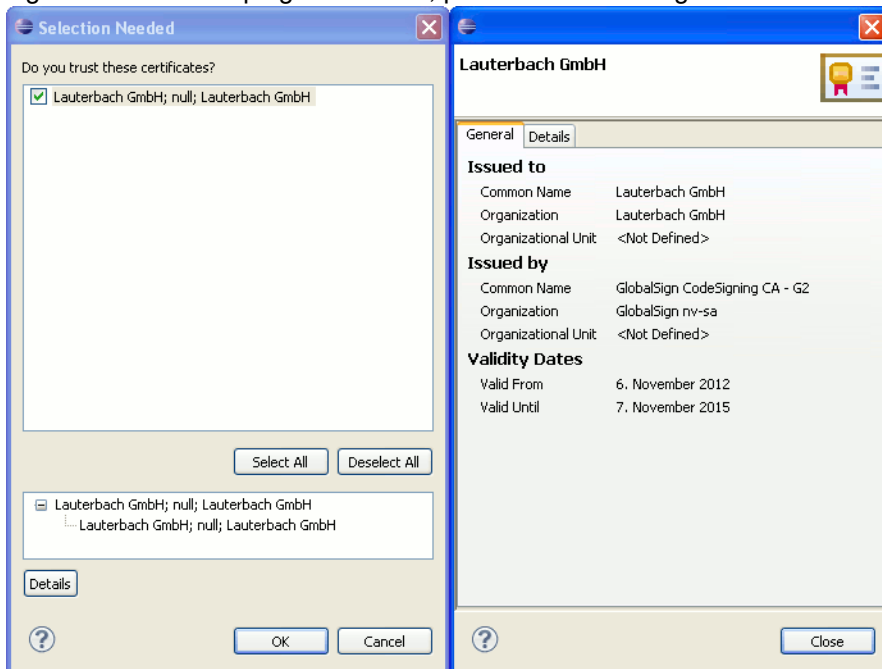
Please make sure you have set up correct **http-proxy settings** for Eclipse before you start the plug-in installation.

If your settings are correct, but installation still fails, please refer to the [Troubleshooting](#) section for a workaround.

1. With configured **Lauterbach TRACE32 Update Site**, select the plug-in in the **Install** dialog:



2. Follow the Eclipse installation wizard.
3. Newer plug-in versions are signed, so you may be prompted to confirm trust to the Lauterbach signature. For older plug-in versions, please confirm unsigned installation:



4. When prompted to do so, please restart Eclipse.

Congratulations, plug-in installation is complete.

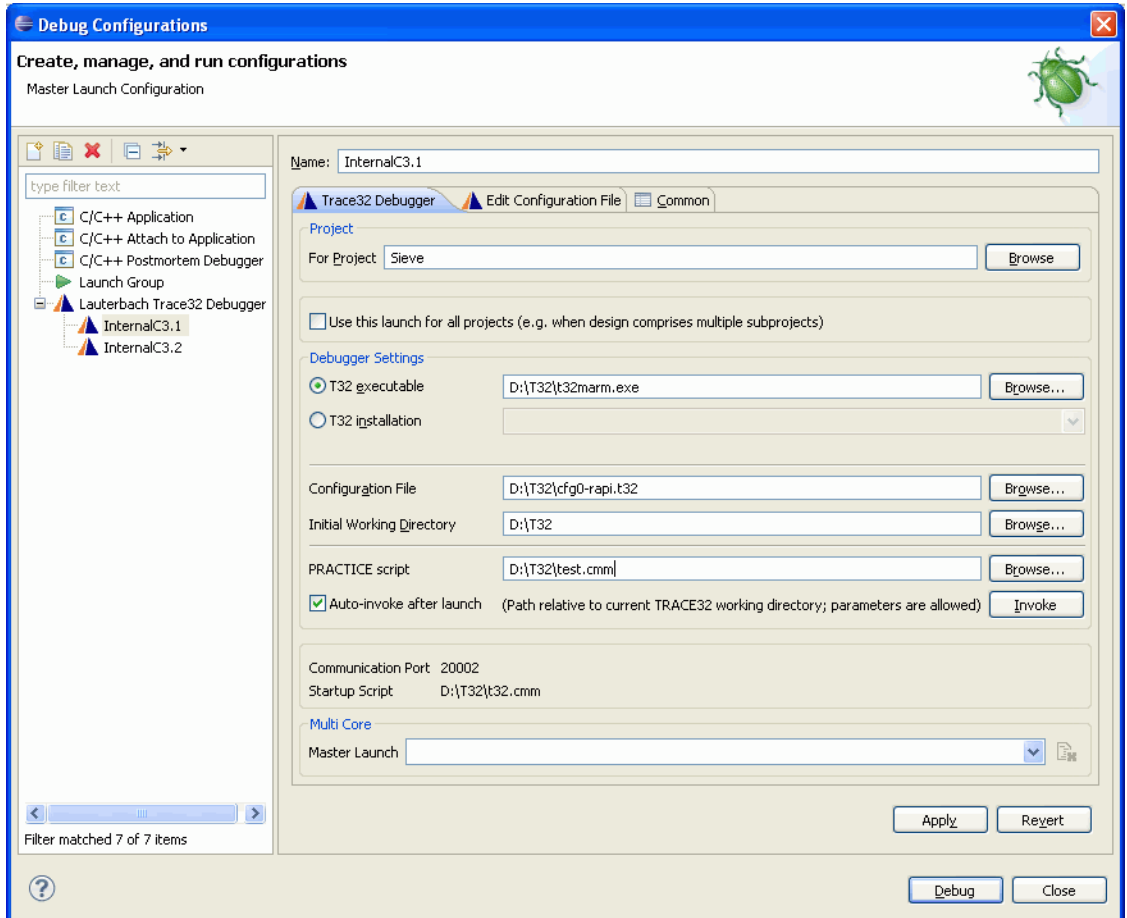
Create Launch Configurations

The Eclipse plug-in starts the TRACE32 GUI via an Eclipse **Launch Configuration**.

This document refers to **Debug Configuration** dialogs only, but a **Run Configuration** dialog can also be used if the Eclipse IDE is not involved in the debugging part (e.g. for just downloading a new build to the target, for performance tests or for tracing).

1. Create or import an Eclipse project. Otherwise the plug-in does not “know” which TRACE32 instance it belongs to. Breakpoint Synchronization, **Edit Source** or **Open in Trace32** will only work with files from an Eclipse project.
2. Open **Run > Debug Configurations** to set up a **Debug Launch Configuration**.

3. In the **Debug Configurations** dialog select **Lauterbach Trace32 Debugger** and add a new configuration with the context menu (right mouse button), opening this dialog:



4. Choose and enter a name for your Debug Configuration.

NOTE:

Please avoid spaces and special characters in the “Name” field if you want to pass parameters to t32.cmm. (“[Parameters for the Startup Script t32.cmm](#)”, page 16.)
E.g. on Windows, a space is the parameter separator, therefore the number and order of parameters on the command line will change if you use spaces within a parameter.

5. Set the **For Project** field to the name of the Eclipse project with your source files.

- In the field **T32 executable**, enter the path to the TRACE32 GUI application that you want to start with this launch configuration. The file name of the TRACE32 executable depends on your target architecture (t32marm.exe for ARM, t32mzsp.exe for ZSP500, etc.).

NOTE: Windows marks executable files with a file name suffix, e.g. t32marm.exe
Linux uses file permissions, the name of the executable is just **t32marm**

- In the field **Configuration File** enter the TRACE32 configuration file to be used with the executable.
- Select the **Edit Configuration File** tab and add this to your TRACE32 configuration file:

```
;T32 API Access
RCL=NETASSIST
PACKLEN=1024
PORT=20006
```

<- mandatory empty line
<- optional comment line

<- mandatory empty line

NOTE: An empty line before and after the text block is required!

From plug-in version 1.3.3, comment lines in the configuration file are ignored, and configuration files generated (on Windows) by **t32start.exe** can be used directly.

The entries above configure TRACE32 to accept commands via the built-in “Remote API” and are a prerequisite for connecting with the Eclipse plug-in. The port number (20006 in the sample) can be chosen rather freely, it just needs to be unique among all concurrently active connections between TRACE32 and Eclipse. It also must not be used by other programs on the host.

- The Eclipse plug-in will parse the chosen configuration file and extract the relevant configuration data.
- Now please check that the values for **Communication Port** and **t32.cmm Startup Script** location are correct:

```
Communication Port: 20000
Startup Script: (none - no t32.cmm found in initial working directory)
```

NOTE: When started, TRACE32 looks for a file with the name **t32.cmm** in its current working directory. If the file is found, the built-in PRACTICE interpreter will automatically read and execute this file as its **Startup Script**.

The **t32.cmm** file which comes with the TRACE32 installation DVD sets the number base (radix), loads extra buttons and the language-specific menu, and restores the command line history.

11. Set the **T32 Initial Working Directory** for TRACE32 when it is launched by the plug-in. This directory will be the initial working directory for scripts started in TRACE32.

NOTE: TRACE32 displays the initial working directory when you enter the **PWD** (=“print working directory”) command directly after TRACE32 starts.

12. With a correct configuration in place, click the **Debug** button to launch TRACE32.

NOTE:

- With plug-in version 1.3.8, CDT Build Variable support was added. For example **\${workspace_loc}** is the current Eclipse workspace directory.
- With plug-in version 1.4.1, CDT Project Variable support was added. For example **\${ProjDirPath}** is the current project directory. (A project must be specified in the “For Project” field)

Browse... resolves any variables to display the selected path/name location and can be used to check if a path/name with variables actually points to what you want. After checking, select **Cancel** to keep your variables, otherwise the resolved 'absolute' path/name will replace the former definition.

Parameters for the Startup Script `t32.cmm`

When Eclipse starts TRACE32, the startup script `t32.cmm` receives several positional parameters:

1. Path to the Eclipse workspace (parameter `&workspace` in the script below)
2. Name of the Eclipse project (`&project`)
3. Name of the launch configuration (`&launchconfig`)
4. Contents of the field **PRACTICE script**, as set in the launch dialog (`&invokeme, ¶m1 ¶m2`)

The sample code below can be included into `t32.cmm`. The `do &invokeme ¶m1 ¶m2` line simulates pressing the **Invoke** button directly after starting up TRACE32:

```
; (+) invoke sample for inclusion in t32.cmm
entry &workspace &project &launchconfig &invokeme &param1 &param2
print "workspace=&workspace project=&project launchconfig=&launchconfig"
do &invokeme &param1 &param2
; (-) invoke sample for inclusion in t32.cmm
```

Invoke PRACTICE Scripts

When you click the **Invoke** button on the **Debug Configuration** dialog, an already running TRACE32 instance (that was started with this launch configuration) executes the given **PRACTICE script** file.

This script can e.g., be used after a rebuild for downloading an object (e.g. ELF) file to the target or to execute other configuration tasks. A relative path name for **PRACTICE script** will use the current working directory of TRACE32 - at the time of invoking the script - as its base directory.

The invoked **PRACTICE script** receives all parameters provided in the text field. The sample script below reads the first 10 parameters (or less, if fewer available) and prints them:

```
entry &a &b &c &d &e &f &g &h &i
print "a=&a b=&b c=&c d=&d e=&e f=&f g=&g h=&h i=&i j=&j"
```

If you want the **PRACTICE script** to be executed directly after TRACE32 launches, you can also call it from `t32.cmm` as shown in the section above.

If this is not possible or desired, the checkbox **Auto-invoke after launch** will make the Eclipse plug-in ask TRACE32 to execute the specified script ca. seven seconds after launching the debugger.

Invoke a Script from the Toolbar

You can have a running TRACE32 instance start a script with the **Lauterbach Logo** button in the Eclipse **C/C++ Perspective** toolbar. This is a shortcut for the **Invoke** button in the **Debug Configuration** dialog.

The **Lauterbach Logo** button executes the TRACE32 **PRACTICE script** set in the **Debug Configuration** for the Eclipse C/C++ project that the currently active editor window belongs to.

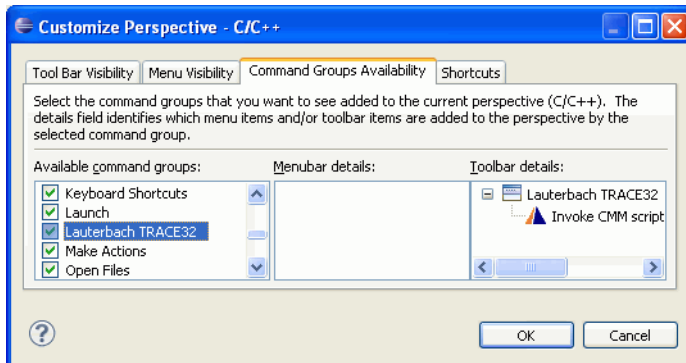
NOTE: **PRACTICE script** name and parameters are used as they were set at the time of the TRACE32 launch (i.e. the settings the debugger was started with).

Changes made later in the **Debug Configuration** dialog (e.g., in parameters passed to the script) are only used when TRACE32 is launched next time.

Add the Lauterbach Logo Button

To add the **Lauterbach Logo** button to the **C/C++ Perspective** toolbar:

1. Open the toolbar's context menu (right-click).
2. Choose **Customize Perspective**.
3. Activate the command group **Lauterbach TRACE32**.



Single-Core Launch with Multiple Eclipse Projects

Larger development projects often organize their source files in multiple separate Eclipse projects.

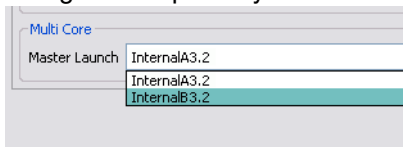
If you have such a project, please specify one of the projects in the launch configuration dialog and check **Use this launch for all projects**. The Lauterbach plug-in will then use this launch setting for all communication (Breakpoint Synchronisation, **Edit Source**, ...) with TRACE32.

Please be aware that this option will currently not work in a multi-core scenario:

- When multiple instances of TRACE32 are started (=one for each core in the target system), the plug-in cannot correlate the current source with the matching debugger instance.
- Therefore a setup with multiple cores and multiple Eclipse projects is currently not supported.

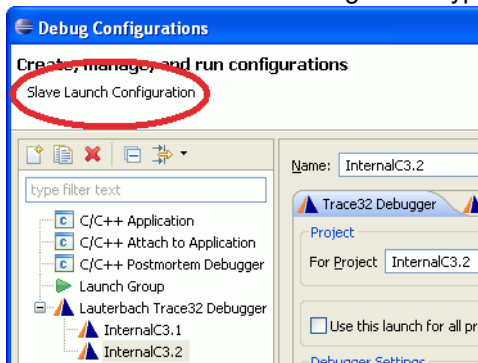
Multi-Core Launch Configurations

For each of your cores in a non-SMP setup, you need to create a separate Eclipse project. This is natural as the cores will each execute their own specific application. For handling such multi-core systems, the launch configuration optionally allows to select a master project:



Whenever the master project is launched, the associated slave projects will also be launched. This can be used to enforce the required start order of the TRACE32 GUI executables.

You can see the Launch Configuration type (**Master** or **Slave**) in the top left corner of the dialog:



NOTE: **Slave** projects cannot be used as **Master** projects for a third project.

If you need a complex launch topology, please either create copies of launch configurations or use **t32start.exe** (Windows only).

Breakpoint Synchronization

The Breakpoint Synchronization feature automatically communicates the breakpoints set in TRACE32 to Eclipse and vice versa. Breakpoints can e.g. be set in TRACE32 while editing source code in Eclipse.

The Eclipse editor works on source lines and symbols and cannot handle memory addresses, assembler instructions or variables. So Breakpoint Synchronization is limited to program breakpoints on HLL lines.

Like in TRACE32, any breakpoint set on a source line that does not have object code associated with it will be promoted to the next following line with object code. Trying to set the breakpoint on line 1 in the sample below will therefore move it to line 3, and then will look like this:

```
1: #define BAR 1
2: #define FOO "hello.c"
o 3: int main(int argc, char** argv) {
```

Sometimes it is required to temporarily disable breakpoint synchronization:

- Breakpoints should not be set during the startup of a target board, until it has been initialized (e.g. by a PRACTICE script that selects the chip, sets register values, enables RAM, etc.)
- Before the debugger has loaded symbol information, any attempt to set an HLL breakpoint will fail with an error message. This should therefore be avoided.

Both cases can occur when breakpoints are set within the Eclipse IDE before starting TRACE32. This can happen either due to user interaction, or because the breakpoints were set in a previous Eclipse session and then reloaded when opening the Eclipse project again.

In these cases, please disable and enable breakpoint synchronization via the TRACE32 command:

```
setup.breaktransfer [on | off]          default: OFF
```

When starting TRACE32, breaktransfer is OFF by default. **If breaktransfer is not 'on' in TRACE32, no breakpoints will be transferred in either direction.** Typically you switch it ON in a startup script:

```
wait 2s                                ; give Eclipse time to connect
setup.breaktransfer OFF                 ; OFF, in case target is restarted
SYStem.Up

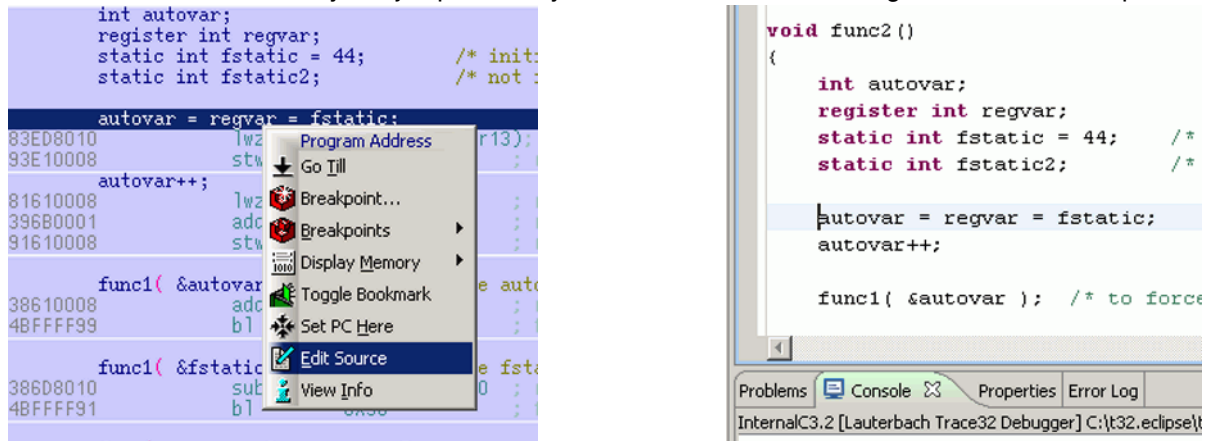
do target_init.cmm                     ; target board initialization
do target_load.cmm                     ; load .elf file with symbol info

setup.breaktransfer ON                  ; now start breakpoint synch.
```

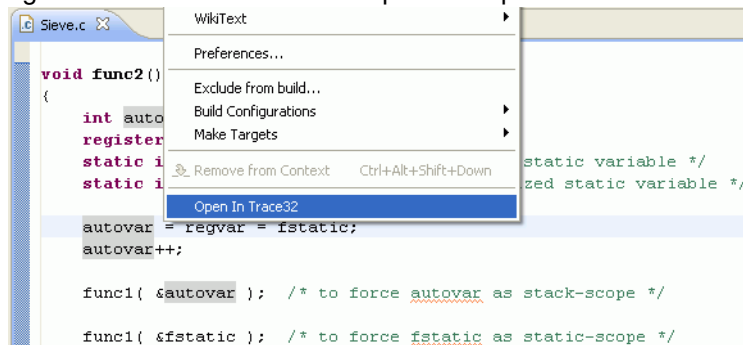
It is possible to set breakpoints in Eclipse while breakpoint transfer is disabled. Eclipse will remember these breakpoints and communicate them to TRACE32 as soon as the transfer is enabled again.

Edit Source Functionality

The **Edit Source** functionality will 'jump' from any TRACE32 window containing source code to Eclipse:



The corresponding Eclipse functionality is **Open In Trace32** from the context menu that opens when you right-click a source line in the Eclipse editor pane:



The TRACE32 **Data.List** command was enhanced (in 2009) to accept line numbers. You can display assembly code and set breakpoints in a **Data.List** window. This is more convenient for debugging work than an **EDIT** window, therefore plug-in versions 1.3.2 and later use **Data.List** for **Open In Trace32**.

NOTE: Without attached Eclipse IDE, the TRACE32 **Edit Source** menu command opens the TRACE32 internal editor.

This default behavior can be restored when an Eclipse IDE is attached with **SETUP.EDITTEXT OFF**, e.g. in your **t32.cmm** startup script.

Eclipse

Breakpoint Synchronization and Edit Source fail

Symptom: Starting TRACE32 from Eclipse works fine but Breakpoint Synchronization and **Edit Source** fail.

Not all Source Code files are defined inside an Eclipse Project

For Eclipse, all source code needs to be organized within Eclipse projects. If a source file is not part of an Eclipse project, the Lauterbach TRACE32 Eclipse plug-in cannot synchronize breakpoints or provide the **Edit Source** functionality.

The problem can also occur when existing projects are imported into Eclipse without selecting the option **Copy projects into workspace**.

Source Code Path names don't match between TRACE32 and Eclipse

The name of the source file path inside Eclipse must match the name of the source file loaded by TRACE32. For projects compiled within Eclipse this should be automatically satisfied.

If source files are compiled outside of Eclipse, in a different location in the file system:

- TRACE32 will find and load the source files based on the location recorded in the debug information from the executable file.
- Eclipse will not recognize the files and will not allow to set any breakpoint there.

The solution is to copy the source files into the Eclipse workspace, and to configure TRACE32 to load the source files from there.

This is done with the **Data.Load** command options **/strippath** and **/stripport**, and with commands like **sYmbol.SourcePATH.SetBaseDir**. (Please see **sYmbol.SourcePATH** for all available options.)

Check the TRACE32 **AREA** window for one of the warnings

```
Eclipse plugin not connected. Disabling breakpoint transfer.
```

```
SETUP.BreakTransfer: no listener registered.
```

```
SETUP.BreakTransfer ON: no listener registered, transfer disabled.
```

The command **SETUP.BreakTransfer ON** causes this warning if it is run before the connection between debugger and Eclipse is established. This may happen if the command is used early during debugger startup (e.g. from **t32.cmm**). To fix this problem, add a short delay in your PRACTICE script (e.g. `wait 1s`) before enabling breakpoint transfer.

Update Site not Found (http-Proxy Settings OK)

Some Eclipse 3.1 installations cannot connect to an update server. The reason seems to be that the Java Runtime Environment executing Eclipse re-uses invalid proxy settings from one of the web browsers that are installed on the host (Firefox, IE, Opera, ...). Configuring the proxy settings for **all web browsers** may solve the problem. One workaround for this issue is to use a **Local Update Site** for installation.

For a discussion about this, see: https://bugs.eclipse.org/bugs/show_bug.cgi?id=101575

Using a Local Update Site for installation:

- A **Local Update Site** consists of **artifacts.jar** and **content.jar** plus two folders **features** and **plugins** with the jar files required for the selected installation.
- A ZIP or JAR archive file that contains such a structure can also be used.
- The complete **Lauterbach Update Site** can be downloaded from <http://www.lauterbach.com/eclipse> and then used as **Local Update Site**.

NOTE:

Eclipse versions 3.4 and earlier have an older plug-in installation system and use **site.xml** file together with **artifacts.xml** and **content.xml**.

The files **artifacts.jar** and **content.jar** are packed (with **zip** or **jar**) versions of **artifacts.xml** and **content.xml**. Just unpack them if you need the XML files.

Failed to Connect to TRACE32

Symptom: Launching TRACE32 from Eclipse works, it starts up fine. However, after some time-out, Eclipse complains 'Failed to connect to TRACE32 on port 20000'.

Please make sure the TRACE32 "Remote API" is enabled (the TRACE32 configuration file contains RCL=NETASSIST and the PORT number is properly set).

Please see [Creation of Launch Configurations](#) for a detailed example.

Unable to Load Class

Please upgrade to Java 1.5 or later, if Eclipse reports this error message

```
Plug-in com.lauterbach.trace32.debug.t32 was unable to load class
com.lauterbach.trace32.debug.internal.ui.T32LaunchConfigurationTabGroup
```

To start Eclipse with a particular JRE installation, you can use this syntax:

```
eclipse.exe -vm \usr\java\jdk1.5.0_10\bin\java
```

Check Your TRACE32 Version

With the **VERSION.view** command, check the build version of your running TRACE32 GUI: to work with the latest Eclipse plug-in it should be from **February, 2013 (build rev. 42425) or later**.

Illegal Character (xxxx) for this Context

TRACE32 prints error messages like “illegal character (xxxx) for this context” in the AREA window.

Please contact **Lauterbach Support** for a TRACE32 update. Please also note: If the TRACE32 maintenance license is expired, the new TRACE32 version will only work in demo mode.

Symbol not Found

Within one debug binary, usually each source file name matches one program module name. Linux and some boot loaders do not follow this rule. If your project contains multiple source files with identical names at different positions in the build file tree, TRACE32 might not be able to determine the correct module from the breakpoint info it gets from Eclipse. To work around this problem, please rename the offending modules.

TRACE32 versions from February, 2013 (build rev. 42425) and later support an extended **Break.Set** command that accepts full source path names. Plug-ins from version 1.3.9 use this mechanism.

Symbol not Found in this Context

Currently the mapping of breakpoints from Eclipse to TRACE32 assumes that the module name is identical to the file name, but some compilers create UPPER-CASE module names while the file name is (as usual) lower-case. One workaround for this problem is using the command **sYmbol.CASE OFF**.

t32.cmm not Executed

If your auto-start PRACTICE file t32.cmm is not executed by TRACE32, please check the TRACE32 Area window for error messages. Any error suspends PRACTICE execution!

Example:

- TRACE32 does not execute your t32.cmm script.
- You see the error message “Online manual (help.t32) fails to load” in the AREA window

In the example the TRACE32 help index and tooltip file was not present in the TRACE32 SYS directory. This error message stops PRACTICE execution. The commands in t32.cmm will not be executed.

To analyze an issue you report to us, we need to create a test environment, as similar to your installation as possible. We also need this to verify that any issue is really fixed e.g. in the new TRACE32 revision.

To set up such a test environment, we need not only to know exactly how your Eclipse installation is configured, but also about TRACE32. Here are the necessary steps to provide this information to us.

Export the Eclipse Error Log

Please include the full Eclipse **Error Log** as a file in your support request:

1. Open the **Error Log** view in Eclipse with **Window > Show View > Error Log**.
2. Click the **Export** icon on the **Error Log** view toolbar.
3. **Save** the log **as** a file.
4. Attach this file to your support request.

Export the Eclipse Configuration

Export the Eclipse configuration settings in text form to the clipboard. With this we can check your Eclipse configuration for any missing or outdated components.

1. Open **Help > About Eclipse**.
2. Click the button **Installation Details**.
3. Select the **Configuration** tab.
4. Click **Copy to Clipboard**.
5. Paste the clipboard content into your support mail to Lauterbach.

Export TRACE32 Information

As an easy way to collect the necessary TRACE32 data (which TRACE32 revision, which Operating System and 32bit or 64bit variant, which Lauterbach hardware, which firmware version you use, which target architecture and CPU you are debugging with etc. etc.), please just follow these simple steps:

1. Download **support.cmm** from <http://www.lauterbach.com/support/static/support.cmm>
2. Start TRACE32 as usual. If possible, connect to the target and stop on a breakpoint.
3. Execute the downloaded support script (in TRACE32) with **DO support.cmm** (on Windows you can drag and drop it from the Explorer window into the TRACE32 command line).
4. The script will first show a form for contact data. If this is your first inquiry, please fill it in (Name, Address and Email are vital, to make sure we can reach you with our response).
5. Then press the **Save to File** button.
6. Please attach the generated output (the saved text-file) in your support request.

Plug-in Version 1.4.2

- Read project active configuration.

Plug-in Version 1.4.1

- Add CDT Project Variables support for Trace32 Debugger Launch

Plug-in Version 1.4.0

- Improve TRACE32 connection handling and launch error messages

Plug-in Version 1.3.9 - requires **TRACE32 from February, 2013** (rev.42425) or later

- Add support for multiple source files with identical file names in the build tree

Plug-in Version 1.3.8

- Add CDT Build Variables support for Trace32 Debugger Launch
- Improve documentation (clarity, usability)

Plug-in Version 1.3.7

- Update Lauterbach Logo to current version

Plug-in Version 1.3.6

- Fix breakpoint synchronization problem (ignore conditional TRACE32 breakpoints)

Plug-in Version 1.3.5

- Improve Breakpoint processing for “OnChip” breakpoints

Plug-in Version 1.3.4

- Fix toolbar launch button problem with Ganymede and later versions
- Improve status information for **Startup Script** and **Port number** setting
- Fix problem with TRACE32 Port number 20000

Plug-in Version 1.3.3 - requires **TRACE32 from January, 2009**

- Improve TRACE32 configuration file parsing (ignore comments)
- Fix lock-up problem caused by non-canonical file names in debug binaries (e.g. ELF)
- Auto-start scripts: add seven second wait time after Eclipse start to allow long Startup Scripts to finish before breakpoint synchronization attempt

[Earlier plug-in version notes removed.]