DEBUGWAVE

Fast STM32 Debug Server and Flash Programmer

User Guide

DebugWave 1.8, 28 August 2019

What Is DebugWave

DebugWave is GNU GDB-compatible debug server and command-line flash programmer for STM32 microcontrollers. Being a debug server DebugWave provides a mean to use GDB and its graphical front ends for source-level in-circuit debugging. It supports running, stepping and halting targets, setting hardware and software breakpoints, reading and writing registers and in-memory variables.

DebugWave also works as a fast and easy-to-use command-line tool for reading and writing the internal flash memory of STM32 microcontrollers. It supports multiple connected targets and detects their parameters and configuration automatically.

DebugWave is delivered for Windows and Linux and supports ST-Link/V2 and Keil ULINK2 probes.

This guide will walk you through the installation and configuration process and show how to use DebugWave to debug and program your hardware designs.

Contents

1 Setting Up DebugWave

This clause describes how to install and configure DebugWave on Windows and Linux.

1.1 Setting Up DebugWave on Windows

On Windows, DebugWave uses the drivers from STMicroelectronics to communicate with ST-Link/V2 probes. For best performance, we recommend using the latest available drivers, which can be downloaded directly from the ST website:

http://www.st.com/en/development-tools/stsw-link009.html

Once the installation is completed, please reboot your system to make sure the new drivers are in use.

Note: Keil ULINK2 and other CMSIS-DAP probes are USB HID devices and thus do not require installing drivers.

For Windows systems DebugWave is delivered in the form of Windows Installer packages (.msi files) that are available on our download page:

https://www.debugwave.com/download

Running an .msi file will launch a wizard to guide you through the installation. The Using the Windows Installer section provides a detailed description of the process and shows how to use .msi files in some custom scenarios, such as installation in non-interactive mode and installation without administrator privileges.

Now that DebugWave is installed, the next step is to make sure it's prepared to work. That's the topic

of the Making Sure DebugWave Is Prepared to Work section below.

1.2 Configuring Access To Debug Probes on Linux

For security reasons, in the common default Linux configurations only root users have access to USB devices like debug probes. Since working under the root account is potentially dangerous, we recommend to create a group of users, say, `debugprobe`, and install udev rules granting its members access to the supported types of probes. Once this group is configured, it's easy to grant or revoke access for specific users by including them in or excluding them from that group.

To simplify the process, we have prepared a shell script that creates the `debugprobe` group, adds the current, non-root user to it, and installs the udev rules. The script can be downloaded directly from our support repository on GitHub:

```
$ wget https://raw.githubusercontent.com/debugwave/support/master/configure_access.sh
```

Before applying, you may want to customize the script or just examine its content:

```
$ cat configure_access.sh
```

Here's how to make it executable and run:

```
$ chmod +x configure_access.sh
$ sudo ./configure_access.sh
```

Please note that since adding users to a group doesn't affect users who are currently logged in, the changes will be applied as soon as you log out, and log back in.

1.3 Setting Up DebugWave on Ubuntu

On Ubuntu systems, DebugWave can be installed from our APT repository or directly from .deb files as described in Installing DebugWave from DEB Packages. We recommend installing from the repository, as it enables automatic upgrades and guarantees that all packages DebugWave depends on are installed and configured properly.

To set up the repository, we first need to import the key used to sign the DebugWave packages:

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com \
    --recv-keys 710D3239B2BEA573
```

Now we can add the DebugWave repository to the system:

```
$ sudo add-apt-repository 'deb https://www.debugwave.com/apt bionic main'
```

Note: On Ubuntu versions other than 18.04 Bionic Beaver, it's necessary to replace `bionic` with the corresponding codename. Here's the list of currently supported versions:

`bionic` for Ubuntu 18.04 Bionic Beaver (LTS)

`disco` for Ubuntu 19.04 Disco Dingo
`xenial` for Ubuntu 16.04 Xenial Xerus (LTS)
`trusty` for Ubuntu 14.04 Trusty Tahr

The last step is updating the list of available packages and installing DebugWave:

```
$ sudo apt-get update
$ sudo apt-get install debugwave
```

Once DebugWave is installed, we are ready to make sure it's prepared to work. The Making Sure DebugWave Is Prepared to Work section below explains how to do that.

1.4 Making Sure DebugWave Is Prepared to Work

Once DebugWave is installed (along with the latest ST-Link drivers, on Windows systems) and access to debug probes had been configured (on Linux systems), we are ready to check whether DebugWave is set up correctly and can access debug probes.

Note: On Windows systems, the installer adds the location of the DebugWave executable to the `%PATH%` environment variable, which means `debugwave.exe` can be run from any current directory without specifying the complete path. However, if you have followed the advanced installation procedure (described in the Using the Windows Installer section) and decided not to add DebugWave to `%PATH%`, it is possible to use the `%DEBUGWAVE%` environment variable to run DebugWave. The installer always defines `%DEBUGWAVE%` so that it expands to the full path of the executable.

Please note that changing environment variables does not affect current terminal sessions, so it's necessary to restart the terminal session for the changes to take effect.

The simplest way to check whether DebugWave can run (and thus is installed correctly) is to open a terminal and type:

```
> debugwave
```

If DebugWave is set up and the operating system knows it location, the response will be:

```
debugwave: Nothing to do. See 'debugwave help' for usage info.
```

Now, if we attach a debug probe to the host machine, the `id` command should return its name along with the name of the target device connected to that probe, like this:

```
> debugwave id
STM32F030x8/051x8/058  Cortex-M0  64 KB  stlink-53ff7
```

If both the commands above worked as expected, that means DebugWave is ready to work and is able to communicate with your probes and target devices. Otherwise, our developers team at support@debugwave.com is ready to help.

2 The Command-Line Interface

This clause describes DebugWave commands, their arguments and options.

2.1 Identifying Connected Debug Probes and Target Chips

The `id` command brings up a list of targets along with the debug probes they're connected to.

```
> debugwave id
```

DebugWave automatically detects connected target devices and knows how to deal with them. No manual configuration is needed. Each detected target is displayed on a separate line, in the following format:

```
target-name  cpu-core  flash-size  debug-pod
```

where *target-name* is the name of the target device, *cpu-core* is the type of the ARM core the microcontroller is based on, *flash-size* is the size of the main internal program flash memory of the device and *debug-pod* is the type and a few first letters of the serial number of the debug probe the target is connected to. For example:

```
> debugwave id
STM32F030x8/051x8/058    Cortex-M0   64 KB  stlink-53ff7
STM32F101xE/103xE        Cortex-M3  512 KB  ulink-V0930
STM32L151xE/152xE/162xE  Cortex-M3  512 KB  stlink-30363
```

If DebugWave is unable to recognize a target device, one of the following errors will be displayed against the probe identifier:

- `Access denied.` — DebugWave is unable to communicate the probe due to insufficient access rights; Configuring Access To Debug Probes on Linux provides detailed information on how to grant permission to debug probes.
- `Target is powered off or not connected.` — The probe is behaving as if no target device is connected to it.
- `Unknown target.` — A target that can be accessed, but does not identify itself as a device known to DebugWave.

2.2 Debugging Software Running on the Target Device

The `debug` command launches a GDB-compatible debug server.

```
debugwave debug [--port n] [target-device]
```

Once launched, the server listens on a port (specified or default), and executes commands sent to it over a socket from a GDB debugger. DebugWave supports the following GDB features:

- Running, stepping and halting the target.
- Setting and removing breakpoints — both hardware and software ones.
- Reading and writing registers.
- Reading and writing flash memory and RAM.

When the debug server is no longer needed, Ctrl+C closes it.

The `--port` (or `-p`, for short) option specifies the port to listen on. The default port is 2159.

The optional *target-device* argument specifies the device to debug, which is necessary if you have several targets connected at the same time. For example, this command runs a debug server on an

STM32L162xx-A chip:

```
> debugwave debug l162-a
```

The Target Specifiers section provides further details on syntax and meaning of *target-device* arguments.

To demonstrate how to use GDB with DebugWave, let's say we have a small C program compiled into an executable ELF image with debug symbols not stripped. First, we have to attach a probe with the target we want to run the program on connected to it and run DebugWave in the debug server mode. In this example we will use the default port. DebugWave will respond with a message indicating that it is awaiting commands from GDB.

```
> debugwave debug
Listening on port 2159.
```

In another terminal session we now can run GDB for the example program:

```
> arm-none-eabi-gdb test.elf
GNU gdb (7.10-1ubuntu3+9) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from test.elf...done.
```

The last line of the GDB's output above indicates that the debug symbols have been successfully loaded from the specified executable image.

It's time to tell GDB how it can access the target device. For this we use the `target` command with the port number that the previously run DebugWave instance listens on:

```
(gdb) target extended-remote :2159
Remote debugging using :2159 0x00000000 in ?? ()
```

Then we can ask GDB to upload the program to target's memory — flash or RAM, depending on the linker configuration used to build the program:

```
(gdb) load
Loading section .isr\_vector, size 0xc4 lma 0x8000000
Loading section .text, size 0xe0 lma 0x80000c4
Start address 0x8000124, load size 420
Transfer rate: 2 KB/sec, 210 bytes/write.
```

At this point we are fully prepared to actually debug the program. We can start with setting a breakpoint and letting the program hit it:

```
(gdb) br main.c:5
Breakpoint 1 at 0x80000e2: file main.c, line 5.
(gdb) cont
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at main.c:5
5           i++;
```

Now let's see the relevant snippet of the source code, print the current value of the counter variable and

examine the call stack:

```
(gdb) list
2    int main(void) {
3    volatile int i = 0;
4    for(;;) {
5         i++;
6    }
7
(gdb) print i
$1 = 0
(gdb) bt
#0 main () at main.c:5@@
```

Once we are done with debugging, the quit command will terminate the GDB session:

```
(gdb) quit
A debugging session is active.

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y
```

To stop the debug server, we need to get back to the terminal session running DebugWave and press Ctrl+C.

2.3 Reading the Target's Flash Memory

The `read` command reads the flash memory of a connected device.

```
debugwave read [target-device] output-file
```

If there's only one connected target, all that's necessary is the command and the name of the output file, like this:

```
> debugwave read image.bin
```

Adding a *target-device* specifier ensures that DebugWave reads from a specific connected target. For example, if one of the connected chips is an STM32L162xx-A microcontroller, the following command will read from it:

```
> debugwave read l162-a image.bin
```

The Target Specifiers section provides more details on how to specify devices DebugWave should work with.

2.4 Writing and Erasing the Target's Flash Memory

The `write` and `erase` commands respectively write and erase the flash memory of a target device.

```
debugwave write [target-device] input-file
debugwave erase [target-device]
```

The *input-file* argument specifies the name of the file to write to flash memory, e.g.:

```
> debugwave write blinky.bin
```

The size of the file shall be the same or less than the size of the flash memory.

Note: It is important to understand that DebugWave makes no guarantees on the content of flash memory past the end of the input file; it may be erased or left as is depending on what seems best for programming performance. The DebugWave's Programming Algorithm section explains this further.

Also, please note that the `write` and `erase` commands only affect the main program flash memory — not other kinds of flash memory, such as option bytes for example.

To make sure DebugWave will write a specific target, possibly among several connected ones, a *`target-device`* specifier can be used. For example, this command writes to a target connected to the only attached ST-Link probe:

```
> debugwave write stlink blinky.bin
```

Similarly, a target specifier can be used with the `erase` command to prevent accidental erasure of a wrong device. For the following command DebugWave will either erase the only connected STM32F103 chip or generate an error if there are several or none such devices.

```
> debugwave erase f103
```

The Target Specifiers section provides more details on how to specify devices DebugWave should work with.

2.5 Target Specifiers

DebugWave supports multiple connected debug probes. When more than one target is identified, target specifiers are used to determine which of the target devices to work with.

If none (or more than one) of the connected probes and targets match the provided specifier, then DebugWave will generate a corresponding error message. This can help prevent commands being accidentally sent to the wrong device — an especially useful safeguard for commands like `write` and `erase`, which can't be undone.

A target specifier can be any of the following:

- A full target name, e.g., `stm32f401vc`, `l152rb-a`.
- A target family, e.g., `l152`, `L162xC`, `stm32f`, `STM32`.
- A CPU core name, e.g., `Cortex-M3`, `m7`.
- A debug probe name optionally followed with leading serial number characters, e.g., `ulink`, `ulink-V0930`, `ST-Link/V2-53fc4`.
- A combination of the above separated with the colon character (`:`), e.g., `stlink:f407`, `ULINK2:m0`.

The specifier can be all-uppercase or all-lowercase. Lowercase `x` wildcard characters denote any characters and shall remain lowercase in uppercase forms.

Since use of lowercase `x` characters is reserved, it is allowed to use the uppercase `X` in place of a target name suffix in all-lowercase specifiers. For example, `l162vx-x` matches to both -A and -X suffixed targets, while `l162vx-X` only matches to -X suffixed ones.

Target name specifiers that end with periods (`.`) will only match to targets without suffixes. For example, `l162vx.` matches to STM32L162Vx targets, but does not match to STM32L162Vx-A or STM32L162Vx-X targets.

2.6 Batch Use, Diagnostics and Exit Status Codes

DebugWave's command-line interface is specifically designed for easy integration with development and production processes, including scenarios relying on batch processing.

DebugWave is aware of all kinds of hardware errors a target device can possibly signal during flash programming, and never ignores them.

By default, DebugWave operates quietly, generating messages only when necessary — for example, in the case of an incorrect command being issued. DebugWave prefixes diagnostic messages with its name, making it easy to distinguish these messages from similar ones generated by other utilities in the same script or a pipe. For example:

```
> debugwave hello
debugwave: Unknown command 'hello'.
```

On Windows, the installer defines the `%DEBUGWAVE%` environment variable, which expands to the full path of the DebugWave executable. This variable can be used in scripts, to make them independent of a specific DebugWave location.

On both Windows and Linux, DebugWave error messages return a numeric value to the shell, indicating

the type of problem:

- 0 – Success.
- 1 – A common failure that doesn't fit into any of the following categories.
- 2 – An internal check failed. This means DebugWave itself has a problem that needs fixing. If you encounter this error code, please let us know at support@debugwave.com so we can fix the issue as soon as possible.
- 3 – Not enough memory.
- 4 – Usage error. Such errors usually mean DebugWave failed to parse the commands and/or options specified in the command line.
- 5 – Input/output error. Indicates a problem with reading or writing files.
- 6 – Permission denied. DebugWave returns this code when it fails to gain access to a file (or some other system resource) due to insufficient access rights.

Other status codes are reserved for future use.

DebugWave versions with the same major version numbers are backward-compatible in terms of the command-line interface. Getting DebugWave Version Number explains how to programmatically determine the version of the local DebugWave installation.

2.7 Getting DebugWave Version Number

The `--version` option outputs the version number along with other basic information about current DebugWave installation. For example:

```
> debugwave --version
debugwave version v1.7.0-68-g58c5efd
DebugWave - A tool to program and debug STM32 microcontrollers.
Copyright (C) 2018 Ivan Labs. All rights reserved.
https://www.debugwave.com
```

The format of the first line is fixed, and will not be changed in future versions.

The version number (e.g., `v1.7.0`) is comprised of the leading `v` followed by three numbers: major and minor version numbers and a patch number.

Versions of DebugWave with the same major number are guaranteed to be backward-compatible in terms of the command-line interface.

Versions that only differ by the patch number are known to have the same set of supported features, targets and probes. Such versions are usually issued to fix critical bugs.

3 The GDB Monitor Commands

Aside of the standard GDB commands, there is a way to issue custom commands to be passed to and executed directly by the GDB server. These are usually used to initialize and configure the connection with the debug probe and target device, and control their state.

This clause describes the GDB `monitor` commands supported by DebugWave.

3.1 monitor help

```
monitor help
```

The `help` command displays a short summary of supported GDB monitor commands and their parameters.

```
(gdb) monitor help
help          Display this commands summary.
init          Initializes or re-initializes the GDB stub.
reset [halt]  Reset target. When specified, 'halt' is ignored.
(gdb)
```

3.2 monitor init

```
monitor init
```

The `init` command initializes DebugWave's GDB stub or re-initializes it if it has been initialized before. Various integrated development tools and environments issue this command via GDB scripts in order to communicate it to GDB servers that they should prepare themselves for another new debug session.

```
(gdb) monitor init
(gdb)
```

3.3 monitor reset

```
monitor reset [halt]
```

The command forces immediate reset and halts the target device.

DebugWave accepts `halt` as an optional argument for the sake of better compatibility with other tools. It currently has no effect.

```
(gdb) monitor reset
(gdb)
```

4 DebugWave's Programming Algorithm

DebugWave's `write` and `erase` commands use an intelligent algorithm to minimize programming time and flash wear by making sure no portion of flash memory is erased or reprogrammed unless it's necessary.

This means DebugWave only guarantees that the given image file will be in flash memory after programming is completed. It doesn't guarantee the status of flash memory outside the binary image address range, which may be erased or left untouched, depending on the situation.

Thus, the most reliable (and efficient) way to ensure all data gets flashed correctly is to prepare a complete image, then `write` the whole image at once. For example, the way to flash data to sectors 1 and 2 is to prepare and flash an image that contains data for both these sectors, instead of programming them separately.

As of this version, the `write` command does not support areas guaranteed to be unchanged during programming (holes). We're currently considering implementing a feature that allows users to control programming granularity and specify holes, in cases where multi-session programming is preferable over preparing and writing a composite image. If you'd like to see this feature soon, please let us know at support@debugwave.com.

5 Getting Updates

On Windows, newer versions can simply be installed on top of the older ones. All that's needed is to download the new version and run the installer, which will take care of updating installed files.

On Linux, the easiest way to stay up-to-date is by using the DebugWave APT repository. Setting Up DebugWave on Ubuntu describes the details.

We typically release a new version every few months. DebugWave's official Twitter account (https://twitter.com/debugwave) provides regular updates on new features and improvements. Also, our GitHub account (https://github.com/debugwave) always contains the latest support materials related to DebugWave.

6 Feedback and Support

We welcome suggestions and comments, including ideas for improvement and requests for new features. Please feel free to contact us at support@debugwave.com.

The first step in addressing a problem is to check the version of DebugWave currently running, as the issue may have been fixed in later versions. The Getting Updates section provides recommendations on keeping DebugWave up to date.

We also provide paid support for companies and individual professionals who use DebugWave as part of their production infrastructure. The Support page on the DebugWave site provides more details.

If you are an OEM company interested in getting API-level access to the DebugWave's underlying technology, please let us know at info@debugwave.com.

Contacts:

- General requests: info@debugwave.com
- Technical questions: support@debugwave.com

Appendices

This clause contains various supplementary information about using and installing DebugWave.

A Supported Debug Probes and Targets

DebugWave supports all series of STM32 devices (F, L and W), including chips with multiple flash layout configurations and variable parallelism size.

The complete list of the supported target chips is available in our support repository on GitHub:

https://github.com/debugwave/support/supported_targets.txt

DebugWave currently supports ST-Link/V2 (both integrated and standalone) and Keil ULINK2 debug probes.

If you would like DebugWave to support a particular chip or a debug probe, please feel free to let us know at support@debugwave.com.

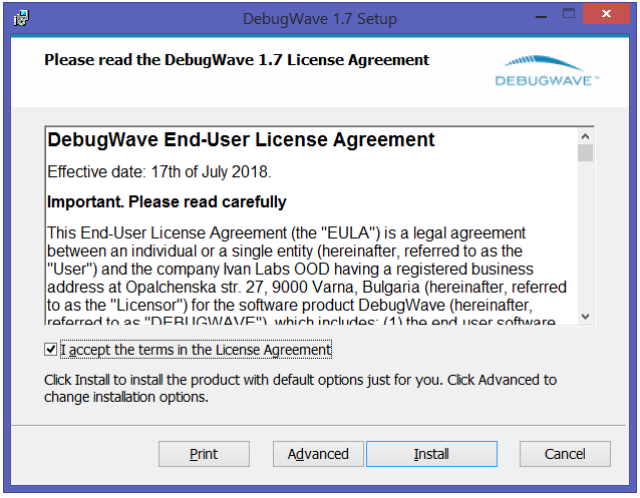
B Using the Windows Installer

On Windows, DebugWave is delivered in the form of .msi files, that is, Windows Installer packages that hold all the information necessary for installing DebugWave, including description of the installation steps and the files to install. It is the current recommended way of installing Windows applications.

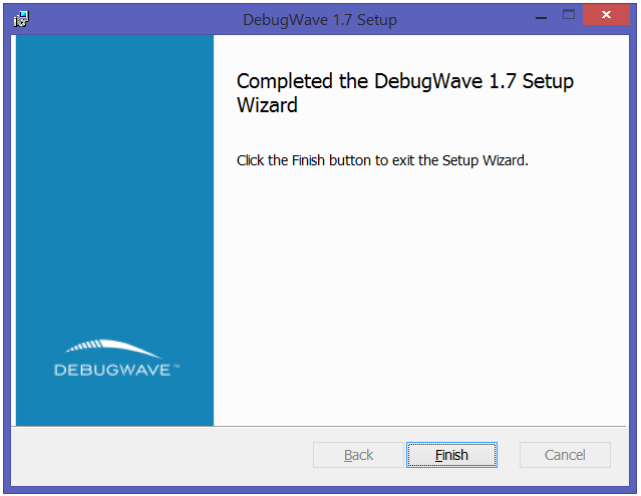
Running an .msi file launches the Windows Installer that unpacks and parses the installation script and then runs the Setup Wizard to let the user configure the installation parameters. Alternatively, it is possible to use the command-line interface of the Windows Installer (the `msiexec.exe` utility) to perform unattended and pre-configured installations, which may be useful for mass deployments and incorporating DebugWave installer packages into third-party installers. The Unattended Installations on Windows section provides some examples.

Once you've made your selections on these dialogues, you can click **Install** to complete the installation.

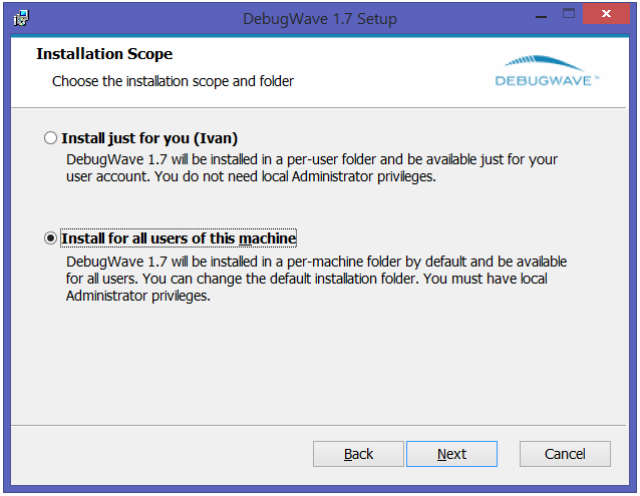
When launched as usual, DebugWave installation package will bring up the Setup Wizard. The first screen of the Setup Wizard asks you to accept the license agreement. You can continue by checking the **I accept** box and clicking **Install**.



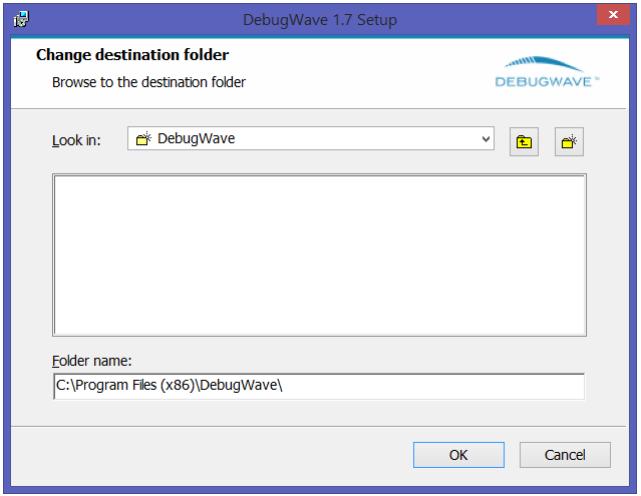
Windows may ask you for permission to continue the installation. Once the Setup Wizard has finished installing DebugWave, it will present a confirmation screen. Clicking **Finish** will exit the Setup Wizard.



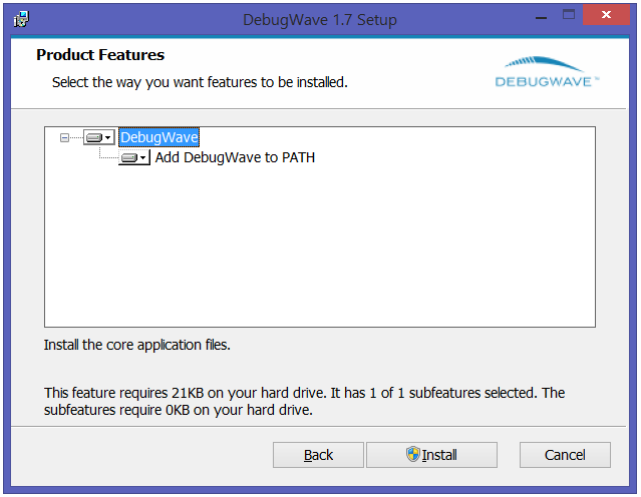
Alternatively, clicking the Setup Wizard's **Advanced** button will bring up a series of additional dialogues, beginning with **Installation Scope**. Here you can choose whether to install DebugWave for all users, or only for yourself:



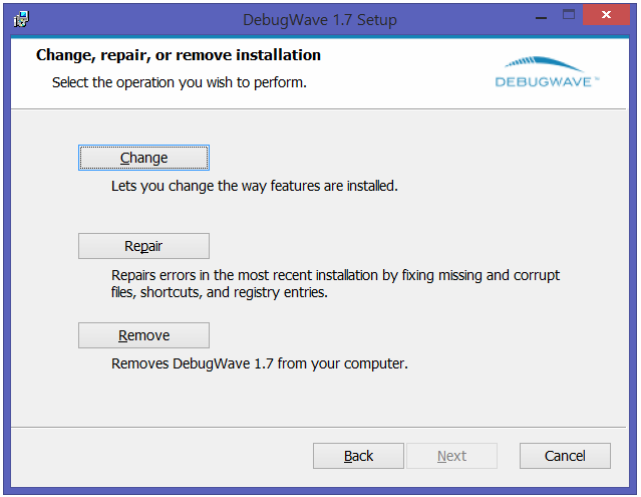
After clicking **Next**, you'll be taken to the **Destination Folder** dialogue, where you can specify an installation path other than the default:



Finally, you'll arrive at the **Product Features** dialogue, where you can choose to disable adding the `%DEBUGWAVE%` environment variable to `%PATH%`:



Running the installer when DebugWave is already installed will bring up an alternate dialogue, offering the options to change, repair or remove DebugWave:



Clicking **Change** will enable you to change features of your DebugWave installation. Choosing **Repair** will cause the installer to attempt an automatic repair of DebugWave. Selecting **Remove** will uninstall DebugWave from your system.

C Unattended Installations on Windows

DebugWave supports installation and removal without user interaction, which may be useful in some custom scenarios, such as mass and remote deployments. This can be done by running the Windows Installer utility, `msiexec.exe`, explicitly at a command line. The utility takes various command-line options that provide some or all of the information that the user would normally specify during interactive installation.

For example, this command will perform silent unattended administrative install or update:

```
> msiexec /a debugwave-1.8.msi /qn
```

Then silent uninstall is as simple as:

```
> msiexec /x debugwave-1.8.msi /qn
```

Or without the installation package, by specifying the DebugWave's product GUID:

```
> msiexec /x {753b0b8b-4a65-40fd-9b99-79fae4c04a9f} /qn
```

Please see the following page for more information about `msiexec.exe` and its options:

https://msdn.microsoft.com/en-us/library/aa367988.aspx

D Adding DebugWave to %PATH%

The installer defines the `%DEBUGWAVE%` environment variable automatically. This means that typing `%DEBUGWAVE%` in a terminal window will run DebugWave regardless of whether the path to the executable is added to `%PATH%`.

Adding `%DEBUGWAVE%` to `%PATH%` enables DebugWave to be run from any directory, without using the `%DEBUGWAVE%` environment variable. The Windows installer includes an option to do this automatically. It's also possible to add DebugWave to `%PATH%` manually:

On Windows 10 and Windows 8

1. In **Search**, search for and then select **System (Control Panel)**.
2. Click the **Advanced system settings** link.
3. Click **Environment Variables**. In the section **System Variables**, find the `%PATH%` environment variable and select it. Click **Edit**.
4. In the **Edit System Variable** window, append `%DEBUGWAVE%` to the value of the variable. Click **OK**.

On Windows 7

1. From the desktop, right click the **Computer** icon.

2. Choose **Properties** from the context menu.
3. Click the **Advanced system settings** link.
4. Click **Environment Variables**. In the section **System Variables**, find the `%PATH%` environment variable and select it. Click **Edit**.
5. In the **Edit System Variable** window, append `%DEBUGWAVE%` to the value of the variable. Click **OK**.

On Windows Vista

1. From the desktop, right click the **My Computer** icon.
2. Choose **Properties** from the context menu.
3. Click the **Advanced system settings** system settings link.
4. Click **Environment Variables**. In the section **System Variables**, find the `%PATH%` environment variable and select it. Click **Edit**.
5. In the **Edit System Variable** window, append `%DEBUGWAVE%` to the value of the variable. Click **OK**.

On Windows XP

1. Select **Start**, select **Control Panel**. Double click **System**, and select the **Advanced** tab.

2. Click **Environment Variables**. In the section **System Variables**, find the `%PATH%` environment variable and select it. Click **Edit**.
3. In the **Edit System Variable** window, append `%DEBUGWAVE%` to the value of the variable. Click **OK**.

E Installing DebugWave from DEB Packages

The recommended way of installing and updating DebugWave on Ubuntu is from the DebugWave APT repository as described in the Setting Up DebugWave on Ubuntu section. However, when necessary, DebugWave can also be installed directly from DEB packages available on our download page:

https://www.debugwave.com/download

The first step is to download and install the DebugWave package itself.

```
$ sudo dpkg -i debugwave_<version>.deb
```

If there are any unsatisfied dependencies, `dpkg` will leave the package in unconfigured state.

In this case `apt-get` can be used to complete the process and resolve the dependencies:

```
$ sudo apt-get install -f
```

F Uninstalling DebugWave on Windows

DebugWave can be uninstalled from a Windows system by running the installer and clicking **Remove**, or from the **Control Panel**:

1. Click the **Start button** to open your **Start Menu**, then click the **Control Panel** menu option.
2. In the **Control Panel** window, click **Uninstall a program** under the **Programs** category.
3. Choose **DebugWave** from the list of programs, and follow the instructions in the **Uninstall Wizard** to remove it from your system.

G Uninstalling DebugWave on Ubuntu

On Ubuntu systems, DebugWave can be uninstalled with `apt-get`, as usual:

```
$ sudo apt-get remove debugwave
```

The following command will also remove the DebugWave repository, if it has been added previously:

```
$ sudo add-apt-repository --remove ppa:debugwave/debugwave
```

H Hardware and Software Requirements

Debugwave requires a Windows or Ubuntu operating system. Microsoft® Windows 95, 98, ME and NT4 are not supported. The supported Ubuntu versions are 19.04 Disco Dingo, 18.04 Bionic Beaver, Ubuntu 16.04 Xenial Xerus and Ubuntu 14.04 Trusty Tahr.

DebugWave also requires the following hardware:

- An Intel® x86 compatible CPU (233 MHz required, 1 GHz or faster recommended).
- 64 MB of RAM or more (1 GB recommended).
- 50 MB of available hard disk space.
- An STMicroelectronics ST-Link/V2 or Keil ULINK2 debug probe.