WGDB6
WINDOWS DEBUGGER
FOR THE ST6 FAMILY

Getting Started

Release 2.00

October 1999

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED.

STMicroelectronics PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF ST Microelectronics.
As used herein:

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

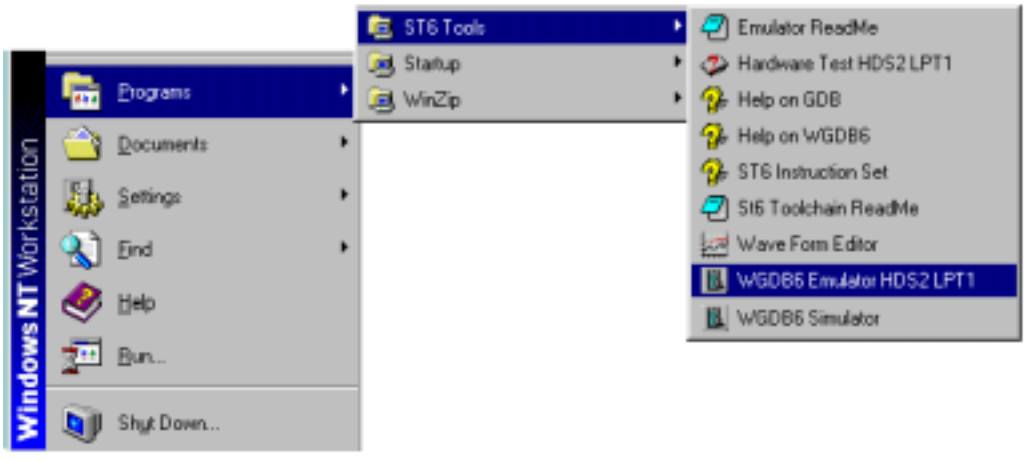
Table of Contents

Table of Contents

1 INTRODUCTION

1.1 Starting WGDB6

If you are using Windows 95, click the **Start** button, point to **Programs**, then **ST6 Tools**, then click **WGDB6 Emulator** if you have an emulator, otherwise click **WGDB6 Simulator**.



1.2 What is the difference between WGDB6 Simulator and WGDB6 Emulator?

WGDB6 Simulator and **WGDB6 Emulator** are two distinct software tools for debugging ST6 programs.

– **WGDB6 Simulator** controls the execution of ST6 programs running in the memory of the development PC.

– **WGDB6 Emulator** controls the execution of ST6 programs running in the memory of an ST6 emulator.

However both tools have much in common since they both:

- Are supplied in the same installation package.
- Have the same user interface.
- Are described in this manual and jointly referred to as **WGDB6**.
- Can be used for debugging programs developed using the STMicroelectronics AST6 Macro-Assembler and LST6 linker.
- Run under Windows 3.x™, Windows 95™, Windows 98™or Windows NT ™.

When you run **WGDB6 Emulator**, if the emulator probe is connected to your application board, the I/O signals will interact with your hardware in the same way as they would if your program was programmed in an ST6 device on your application board.

WGDB6 Simulator does not require the presence of the ST6 emulator. It can work with an ST6 Starter Kit (for performing simple emulating functions). If you install **WGDB6 Simulator** with a starter kit, the name of the program icon installed in the Start menu will be **WGDB6 sim stkit**.

1.3 What Can I Use WGDB6 For?

WGDB6 enables you to execute ST6 programs, and view the contents of the ST6 data and program memory as the program progresses. You can examine source assembler code. Program execution history can be viewed at source or instruction level using the Trace and Stack displays. WGDB6 lets you read and write all ST6 registers and memory locations.

WGDB6 can display data in either ‘normal’, ‘hot’ or ‘real-time’. Normal display data is displayed as it was when you chose to view it, it is not automatically updated. Normal display data is displayed on a white background. Hot display data is updated every time the execution of the program you are debugging is suspended, and is displayed on a yellow background. Real-time display data is updated as the program is running, and is displayed on a red background.

WGDB6 enables you to save and load workspaces. Workspaces are snapshots of windows and option choices that are taken when you close a program. Each program you debug using WGDB6 has its own default workspace definition. When you load a program, the workspace that you were using when you last closed it is restored, thus you can continue working from where you left off. You can also save workspace definitions at any time, so that you can restore them at a later date. See “Working with Workspaces” on page 34 for further details.

WGDB6 includes the following debugging features:

- Lets you set software breakpoints on source or disassembled code, that stop the program running when a chosen instruction is reached.
- Lets you set memory breakpoints, which stop the program running when a predefined area of memory is accessed and optionally when a hardware-related condition is met.
- Enables you to execute source code and machine code line by line. A function call can optionally be considered as a single instruction, depending on the level of detail required.
- Keeps a trace of memory access during program execution.
- Enables you to view the stack contents.
- Enables you to view and modify the simulated/emulated ST6 memory and register contents.

- Enables you to view and modify data symbol values in real time, that is as your program is running.
- Automatically executes GDB6 command batch files, with or without the WGDB6 graphical interface, at start-up.

1.4 What Can I use Wave Form Editor For?

Wave form editor is a standalone program for simulating input signals by software. It is intended for use with **WGDB6 Simulator**. You do not need to use it with **WGDB6 Emulator**, since the emulator works with input signals from the application hardware.

1.5 What are Pretty Format Addresses?

To access an object in areas of paged memory within the program or data space you can use an address format known as ‘pretty address’.

Pretty addresses have the following format:

`P:pp:nnn` to address the program space.

or

`D:pp:nn` to address the data space.

where `pp` is the page number in hexadecimal and `nnn` or `nn` is the offset from the beginning of the page in hexadecimal.

Each page in the program space is 2048 bytes (nnn in the range 0x000 to 0x7FF). An offset greater than or equal to 0x800 always means page number 1 (the static page): in this case pp is insignificant.

Each page in the data space is 64 bytes (nn in the range 0x00 to 0x3F). An offset greater than or equal to 0x40 means a data address outside the dynamic page area: in this case pp is insignificant.

Examples:

`P:00:0F0` addresses byte 240 (F0h) in page 0 of the program space.

`D:08:1A` addresses byte 26 (1Ah) in page 8 of the data space.

`P:01:080` is equivalent to `P:00:880` or `P:01:880` or `P:02:880`.

`D:00:80` is equivalent to `D:01:80`.

2 PREPARING PROGRAMS FOR DEBUGGING

You can debug programs that were generated using the STMicroelectronics ST6 Macro-Assembler.

Some example macro-assembler programs that are fully prepared for debugging are provided in the simex subdirectory of the WGDB6 installation directory.

To be able to load a program, WGDB6 must have access to the *.hex file in the Intel hexadecimal format. With the *.hex file, WGDB6 can recognise the symbol files that were generated by the toolchain. For the STMicroelectronics ST6 Macro-Assembler compiler, the symbol files may include .SYM, .MAP., .DSD and .LIS files.

To generate files with the correct debug information for WGDB6, when you assemble programs using the STMicroelectronics ST6 Macro-Assembler, you must use the PP_ON directive (refer to the AST6/LST6 user guide).

`ast6 -l -o filename.asm` for each source module, and `lst6 -i -m -s -o filename filename.obj` for each object module

If no symbol file is provided with the .hex file, you can still perform some debugging tasks, such as viewing ST6 memory contents, viewing disassembled code, displaying and modifying register values, executing programs and viewing trace buffer contents.

Note: If the program is in a single source file and the generated code is less than 4K bytes, it is possible to generate code without using the linker, by entering the following command line:

```
ast6 -l -m -s filename.asm
```

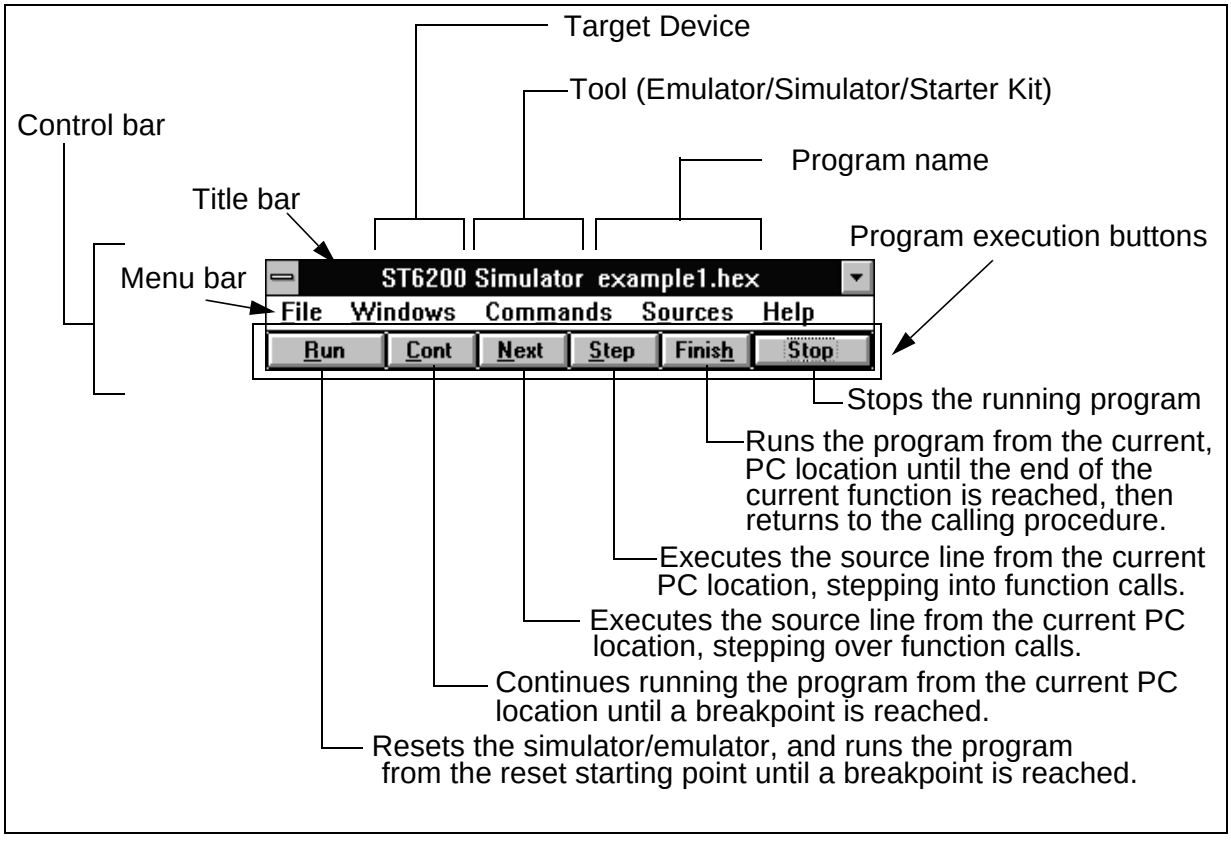
However this option should be used with caution, as the debug information is not suffient for WGDB6 and correct debugging functionality is not guaranteed.

3 USING WGDB6

This section describes how to perform the basic tasks that you will typically carry out when you debug a program using WGDB6. It aims to get you quickly started with WGDB6. For detailed information about windows, menus and dialog boxes, and on how to perform all tasks, use the WGDB6 online help (see “Getting Help” on page 10).

3.1 Using the Control Bar

Once WGDB6 starts, the Control Bar opens:



The Title bar displays the debugging tool (Emulator, Simulator or Starter Kit, the ST6 family name currently being simulated/emulated and the name of the program that is currently loaded (see “Loading a Program” on page 11).

The Menu bar displays the available menus. Clicking a menu name opens it. Since no program is loaded, all program-related menus are greyed out and thus cannot be accessed.

The program execution buttons provide you with quick access to the program execution functions.

Note: All the task procedures described in this book start from the control bar.

3.2 Getting Help

WGDB6 enables you to access two types of help:

- Task-based help, where you choose a task about which you want information.
- Context-sensitive help, that displays information about the window or dialog box that is currently active.

To access task-based help:

1. From the **Help** menu in the Control Bar click **Contents**.
2. Click a topic from the list to display the information.

To access context-sensitive help on the window or dialog box that is currently active:

1. In the window or dialog box, click the **Help** button if there is one, otherwise click the [F1] key.

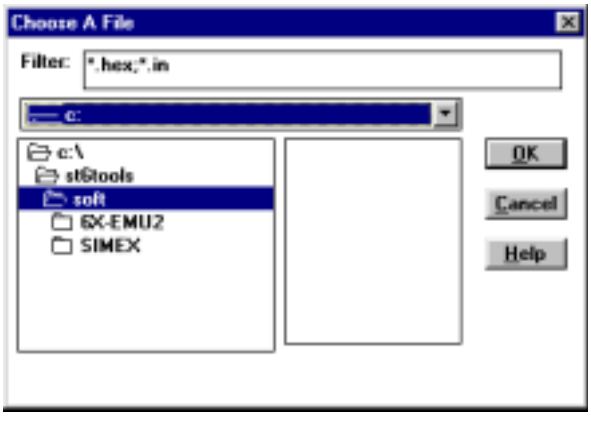
   A help window opens displaying the window or dialog box that is currently active.
2. In the help window, click on the item, such as a menu name or field, about which you want further information.

3.3 Loading a Program

When you load a program, you must specify the .HEX file of the program to load. The symbols are automatically loaded from the appropriate symbol file. For details on the file types that you can debug, see “Preparing Programs for Debugging” on page 8.

1. In the **File** menu, click **Open**.

   The Choose a file dialog box opens:



2. In the drive list, click the drive that contains the program.
3. In the box beneath the drive list, double-click the name of the folder that contains the program. Continue double-clicking subfolders until you open the subfolder that contains the program.
4. In the list of files, click the program name.
5. Click **OK**.

When you load a program, a reset is implicitly performed. To open the Disassembler or module window at the current PC position, in the **Command** menu: click **Display PC** or click the **Reset** button.

To open a program you've used recently, click its name at the bottom of the File menu.

3.4 Viewing Program Information

3.4.1 Viewing a Source Module

To view a source module:

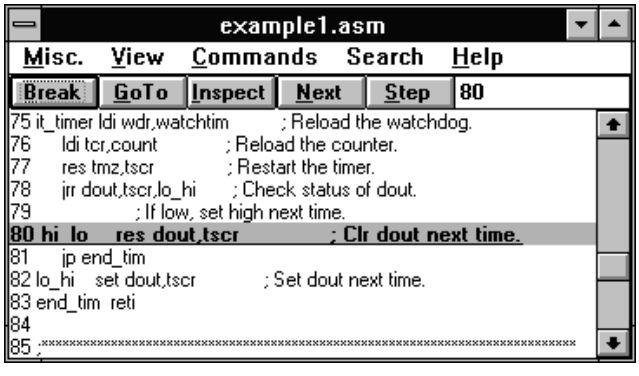
1. Open the **Sources** menu.
2. Either:

   Click **Modules List...** then click the module you want to open.

   or

   Click the module you want to open from the list.

   The module window opens displaying the module you just opened:



From the module window, you can perform the following operations:

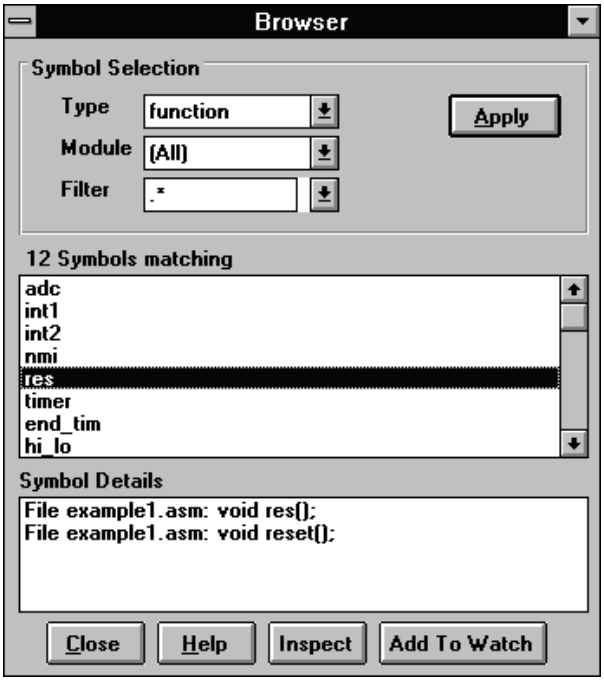
To do this:	Do this:
View further details about any item displayed in the window.	Click the item then click the **Inspect** button.
View a line of code disassembled.	Click the line number, then click the **Inspect** button.
Step or go to another part of the loaded program.	Click the appropriate button: **Next**, **Step** or **Goto**.
Find a character string within the module.	Click **Find** In the **Search** menu.
Insert a software breakpoint.	Click where you want to insert the breakpoint, then click the **Break** button.

3.4.2 Finding and Viewing Symbols

To find symbols and view detailed information about them:

1. In the **Windows** menu, click **Browser**.

The Browser window opens:



2. In the **Symbol Selection** box:

   i Select the symbol type you want to find from the **Type** drop-down list.

   ii Select the module that contains the symbol you want to find from the **Module** drop-down list.

   iii Enter the search operators in the **Filter** field:

To find:	Use this operator:	Examples:
All names	.*	
All the names containing a substring	The substring	“at” finds “data”, “date” and “bat”.
All the names starting with a substring	^ the substring	“^ba” finds “batch” and “back”
All the names ending with a substring	the substring $	“de$” finds “node” and “side”

   iv Click the **Apply** button.

5. The **Symbols matching** box lists the symbols found using the selection criteria entered in the **Symbol Selection** box.

6. You can now view further details about the found symbol by:

   Clicking the symbol and viewing the **Symbol details** box.

Clicking the symbol then clicking the **Inspect** button. If you selected a function, the function body is displayed in either the Disassembler window (see “Viewing/Editing Disassembled Program Code” on page 17) or the Module window (see “Viewing a Source Module” on page 12), depending on the available source file types. If you selected data or a constant, its value and type are displayed in the Inspect window.

Clicking the symbol then clicking the **Add To Watch** to see its type and value in the Watch window (see below).

3.4.3 Watching Variable or Expression Values

WGDB6 enables you to watch the values of variables or expressions, which are updated each time program execution is suspended (for example, after a next or step instruction).

Note: Variables located in data RAM banks cannot be watched.

To view a variable or expression value in the Watch window:

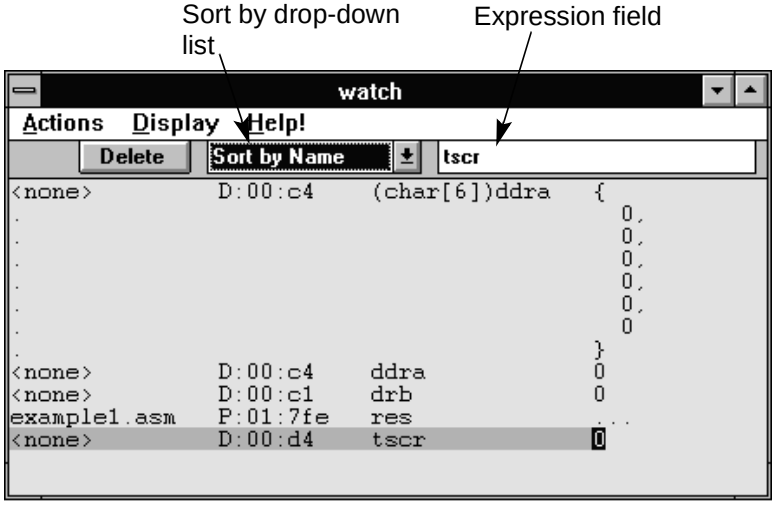
1. Either:

   Select the variable whose value you want to watch in the Browser window, then click the **Add to Watch** button.

   or

   In the **Windows** menu, click **Watch**, then type the name of the expression you want to watch in the expression box then press [Enter].

   The Watch window now opens, displaying the value of the selected variable:



From the Watch window, you can perform the following operations:

To do this:	Do this:
Change the value of a symbol.	Click on the value and overtype it.
Change the format in which newly displayed symbols are displayed.	Choose the appropriate option from the **Preferences** submenu In the **Display** menu.
Change the type of data displayed.	Choose the appropriate option from the **Format** submenu In the **Display** menu.
Change the radix in which the value of the selected symbol is displayed.	Choose the appropriate option from the **Base** submenu In the **Display** menu.
Choose the symbol type that data is sorted by.	Select the symbol type from the Sort by drop-down list.
View the value of an expression.	Enter an expression in the expression box.
View a range of *n* bytes starting from a variable or an address.	Enter the variable name or * followed by the address, then followed by @*n* in the expression box. For example, to view the 5 bytes starting at address D:00:80, enter: *D:00:80@5.
View a subrange of *n* values of an array.	Enter * followed by the array or pointer name followed by @*n* in the expression box.
View a selected function disassembled.	Select the function, then click **Disassembler** In the **Actions** menu.
View and modify the data held in memory for a selected symbol.	Select the symbol, then click **Dump** In the **Actions** menu.
View the value of a selected symbol in the Inspect window.	Select the symbol, then click **Inspect** In the **Actions** menu.
View a selected function within the source code.	Select the symbol, then click **Sources** In the **Actions** menu.

3.4.4 Viewing/Editing Data Symbols in Real Time

1. Either:

   Select a data variable using the Browser window (see “Finding and Viewing Symbols” on page 12), then click the **Inspect** button,

   or

   In the module window, type the data symbol name in the field in the top-right corner of the window, then press the Enter key (“Viewing a Source Module” on page 12).

   The Inspect window opens:

2. In the **Misc.** menu, click **Real Time**.

   The window background color becomes red, indicating that it is running in real-time mode. Executing the program will now result in the displayed values being updated.

3. To update the displayed data value while the program is running:

   In the **Edit** menu, click **Modify**.

   Type a new value, then click the **Set** button.

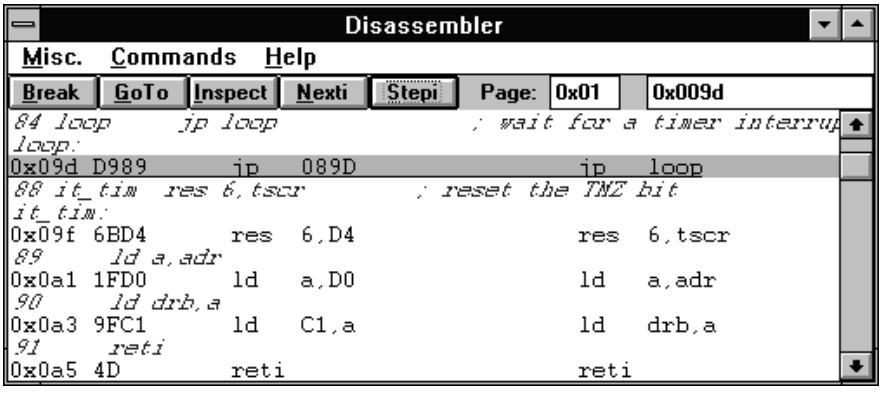
   The data value is now updated accordingly.

   Note: This function cannot be used in programs including Stop or Wait instructions

3.5 Viewing/Editing Disassembled Program Code

To view the disassembled program code that is loaded in memory:

1. In the **Windows** menu, click **Disassembler**.
2. The Disassembler window opens:



The Disassembler window shows each line of source code followed by its disassembled code.

From the Disassembler window, you can perform the following operations:

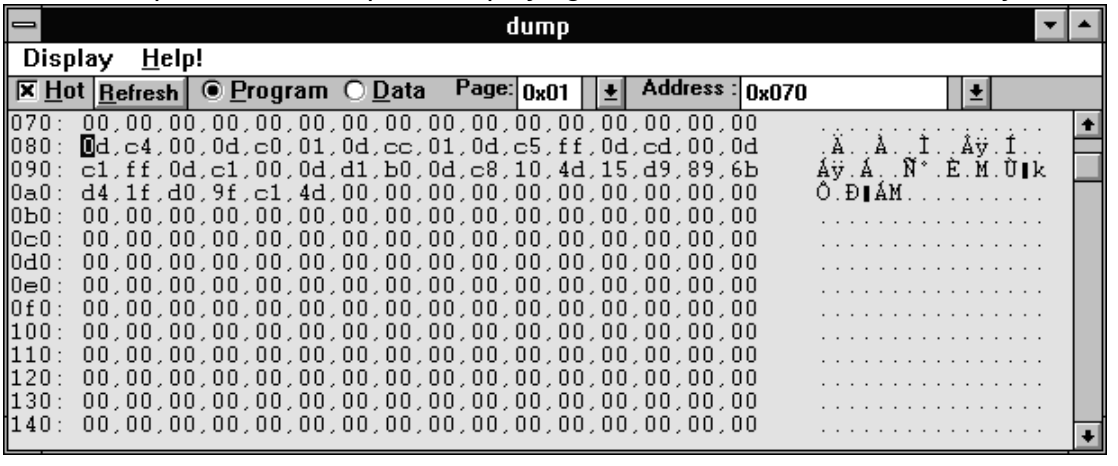
To do this:	Do this:
Display the memory contents in a different format	Choose another format from the **Options** command in the **Misc.** menu.
Display contents starting from a selected page	Click the **Page** field and enter a page number.
View more detailed information about a selected symbol	Select the symbol then click the **Inspect** button.
Display the contents starting from a selected address	Enter a symbol name or any valid C expression in the **Address** field.
Enter and assemble instructions online in the ST6 memory.	Click on the disassembled line you want to modify, then in the **Commands** menu, click **Online Assemble**. For more information about, this refer to the online help

Note: The destination address of conditional jumps is not displayed correctly in the disassembler window. The jumps are executed correctly

3.6 Viewing/Editing ST6 Memory Contents

To view the contents of the ST6 memory:

1. In the **Windows** menu, click **Dump**.
2. The Dump window now opens, displaying the contents of the ST6 memory:



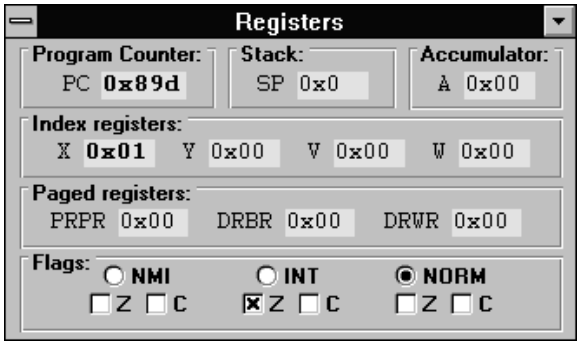
Note that before you have loaded a program, the ST6 program memory will be set to 0. From the Dump window, you can perform the following operations:

To do this:	**Do this:**
Display the memory contents in a different format	Choose another format from the **Displa**y menu.
Make the window Hot (so its contents are updated each time program execution is stopped)	Click the **Hot** checkbox (the window background becomes yellow indicating that the window is hot).
Display contents starting from a selected page (for ST6 devices with dynamic pages)	Click the **Program** or **Data** radio button and select or enter the **Page** value from the DRPR register for data or the PRPR register for programs.
Display contents starting from a selected address	Choose the memory space (Program or Data) whose contents you want to view, by clicking the appropriate button, then enter the page whose contents you want to view in the **Page** field, and the offset to the beginning of the area you want to view in the **Address** field. Press the Enter key.
Display contents starting from a symbol name	Enter the symbol name in the **Address** field.
Edit contents of a selected address	Click the value that you want to modify and enter a new value on the keyboard.

3.7 Viewing/Editing Register Contents

You can view and modify register contents using WGDB6:

1. In the **Windows** menu, click **Registers**.
2. The Registers window now opens:



3. To modify the contents of a register, select it, type in the new value and press the [Enter] key.

   The table below describes the fields in the Registers window:

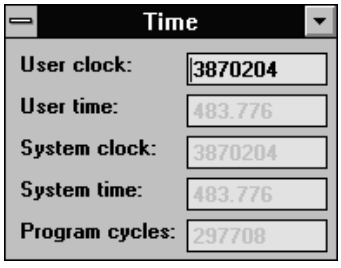
This field:	Displays this:
Program Counter	The Program Counter (PC) register value. This is a 12-bit register that stores the address of the next instruction to be executed.
Stack	The Stack Pointer (SP) value. This value is in the range 0 to 5 indicating the current nested call or interrupt level.
Accumulator	The Accumulator (A). This is an 8-bit general purpose register used to hold operands and the results of arithmetic and logic calculations as well as data manipulations.
Index Registers	The Indirect Registers (X and Y) and the Short Direct registers (V and W). The Indirect registers are used during register-indirect addressing mode as pointers to memory locations in the data space. The Short Direct registers are used to save one byte in short direct addressing mode.

This field:	Displays this:
Paged Registers	PRPR is the Program ROM Page Register. This defines the 2-Kbyte program space page to be addressed. The area 0 to 0x7FF in the program space is paged. DRBR is the Data RAM/EEPROM Bank Register. This selects the 64-byte RAM/EEPROM data space page to be addressed. The area 0 to 0x3F in the data space is paged. DRWR is the Data ROM Window Register. This defines the data ROM window location. This area is addressed as part of the data space between 0x40 to 0x7F.
Flags	NMI is the Non-Maskable Interrupt Flag. This indicates that the ST6 memory is in non-maskable interrupt mode, which is the default mode after reset. INT is the Interrupt Mode Flag. This is used during interrupt mode operation. When the Int bit is set, all interrupts are disabled except for NMI. Clearing this bit enables them. NORM is the Normal Flag. This indicates that the ST6 is in normal mode, and not currently executing an interrupt. The carry (C) flag is set when a carry or borrow occurs during arithmetic operations, otherwise it is cleared. The zero (Z) flag is set when the result of the last arithmetic or logical operation was zero, otherwise it is cleared. See the appropriate Databook for the ST6 family type you are using for further details.

3.8 Viewing/Editing Time Information

You can only perform this task if you are using the ST6 simulator.

1. In the **Windows** menu, click **Time**.
2. The Time window now opens:



The table below describes the fields in the Time window:

This field:	Displays this value:
User clock	Displays the current user clock time. This is the time (expressed as a number of external clock oscillations) that elapsed between the loaded program being executed an that execution being stopped. By default, this value is the same as the system clock value, however this value can be modified. You can modify this value by over typing the displayed value and pressing the [Enter] key.
User time	Displays the user time value (in milliseconds). This is the user clock value divided by simulated clock frequency (to view or update the frequency, click **Frequency** In the **Commands** menu).
System clock	Displays the system clock value. This value cannot be modified.
System time	Displays the system time value. This is the system clock value divided by simulated clock frequency (to view or update the frequency, click **Frequency** In the **Commands** menu).
Program cycles	Displays the number of cycles that the program has executed. In ST6 family microcontrollers, 1 cycle = 13 oscillator pulses (1.625 µs).

3.9 Executing Loaded Programs

WGDB6 includes the following program execution commands, which are available either as buttons in the Control Bar or other windows (where appropriate) or In the **Commands** menu. The following table explains what each command does:

This command:	Does this:
Run	Resets the simulator/emulator, and runs the program from the reset starting point until a breakpoint is reached.
Cont	Continues running the program from the current PC until a breakpoint is reached.
Next	Executes the source line at the current PC location, stepping over any function calls.
Step	Executes the source line at the current PC location, stepping into any function calls.
Finish	Runs the program from the current PC line until the end of the current function is reached, then returns to the calling procedure.
Stop	Stops the running program.
Reset	Resets the ST6 simulator/emulator. The PC is set to the starting point and A, X, Y, PRPR, DRBR, DRWR are set to 0 and mode is set to NMI (see “Viewing/Editing Register Contents” on page 19 for further details about these values).
Goto	Runs the program from the current PC position to the specified marker position.
Stepi	Steps one instruction immediately following the PC line, stepping inside any call instructions.
Nexti	Steps one instruction immediately following the PC line, executing any call instructions.

TIP: To execute a program from a specific address, set the PC and the PRPR values to where you want to start running the program in the Register window (see “Viewing/Editing Register Contents” on page 19), then click the **Cont** button.

3.10 Using Software Breakpoints

3.10.1 Setting Software Breakpoints

Software breakpoints are breakpoints that are triggered when a source line or a disassembled instruction is executed.

You set software breakpoints in either the module window or the Disassembler window. To open the module window, click the **Source** menu, then click the name of the source module in which you want to set a breakpoint. To open the Disassembler window, In the **Windows** menu, click **Disassembler**.

In the module or disassembler window:

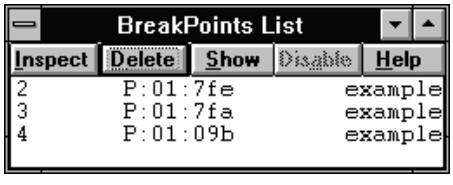
1. Click the line of code on which you want to set a breakpoint.
2. Click the **Break** button.

A breakpoint is now set on the selected line. This line is highlighted in bold (by default), indicating that it includes a breakpoint.

Note: If you try to place a breakpoint on an inappropriate line, such as on a While condition or a comment, WGDB6 will set the breakpoint on the next available line.

3.10.2 Managing Software Breakpoints

You can perform all software breakpoint management tasks in the BreakPoints List window. Note that this is only available when at least one software breakpoint is set. To open the BreakPoints List window, In the **Windows** menu click **Soft Breakpoints**.



The BreakPoints List window is as follows:

From the BreakPoints List window, you can perform the following operations:

To do this:	Do this:
Delete a breakpoint	Select the breakpoint, then click the **Delete** button.
Display the selected breakpoint location within the source code.	Select the breakpoint, then click the **Show** button.
Enable/disable the selected breakpoint.	Select the breakpoint, then click the **Enable/ Disable** button.

3.11 Using Memory Breakpoints

3.11.1 Setting Memory Breakpoints

You can set memory breakpoints on the following events:

- When a variable or constant is accessed.
- When a data memory address or range of addresses are accessed.
- When a program memory address or a range of addresses are accessed or its contents are executed.

- When specified hardware conditions are met.

Note: Memory breakpoints are not saved in workspace so they will not be available when you reload your application.

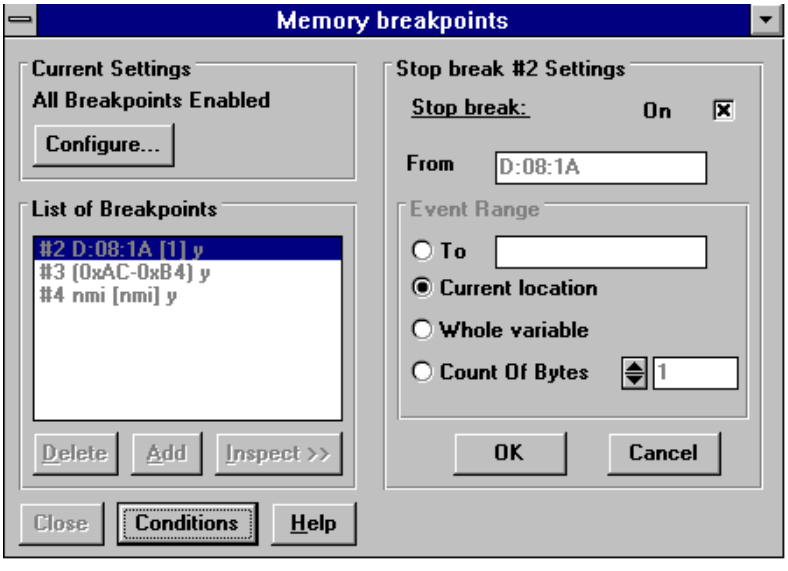
To set a memory breakpoint:

1. In the **Windows** menu, point to **Hardware Events** then click **Memory Breakpoints**.

   The Memory breakpoints window opens.

2. In the Memory breakpoints window, click the **Add** button.

   The New Settings box opens:



3. In the **From** field, enter the start address of the range for which the breakpoint is activated. This can be expressed either as a variable (symbolic name) or as a Pretty address (see “What are Pretty Format Addresses?” on page 7).

4. In the **Event range** box, choose the range on which the breakpoint is triggered:

   Select **To** to enter the end address of the memory range on which the breakpoint is set. This must be expressed as a Pretty address.

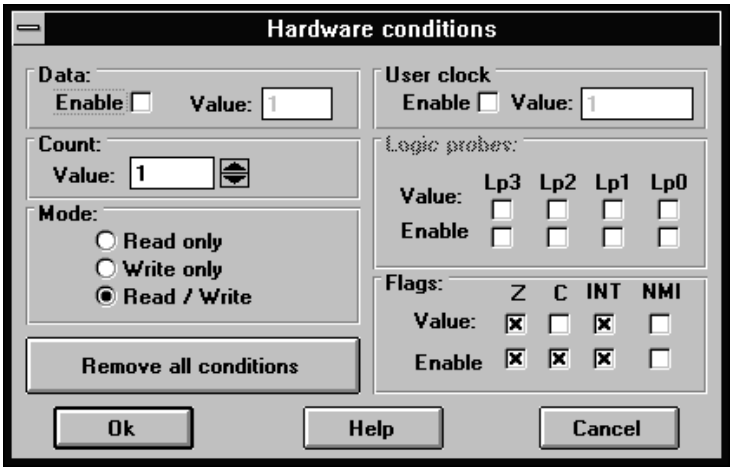
   Select **Current location** to activate the breakpoint only if the address defined in the from field is accessed (range = 1 byte).

   Select **Whole variable** to activate the breakpoint on the whole variable range which is directly linked to the symbol type specified in the from field (for example on whole fields of data symbols for a C struct data type).

   Select **Count Of Bytes** to define the number of bytes from the location specified in the **From** field for which the breakpoint is activated.

5. To attach specific hardware conditions to memory breakpoints, such as activation when a specified value is written to or read from memory, click the **Conditions** button.

   The Hardware conditions dialog box opens:



Follow the instructions in the table below to set conditions on memory breakpoints:

To do this:	**Do this:**
Trigger the breakpoint when a specified data value is read or written. This condition will apply to all breakpoints set on data.	Check the **Enable** check box in the **Data:** box, and enter the value you want to trigger the breakpoint on in the **Value:** field.
Trigger the breakpoint after it has been reached a specified number of times. This condition will apply to all breakpoints.	Enter the number of times you want the breakpoint to be reached before it is triggered in the **Value:** field in the **Count** box. For example, if you enter 2 here, the breakpoint is triggered the second time it is reached.
Choose an access type on which a breakpoint that is set on data is triggered. This condition will apply to all breakpoints set on data.	In the **Mode** box: Select **Read only** to trigger the breakpoint on read access. Select **Write only** to trigger the breakpoint on write access. Select **Read/Write** only to trigger the breakpoint on read or write access.
Trigger a breakpoint on the simulated clock counter value (simulator only).	In the **User clock** box, click **Enable,** then enter the simulated clock value at which you want the breakpoint to be triggered in the **Value** field. This value may range between 0 and 2**32 - 1. Its default value is 0.

To do this:	Do this:
Trigger a breakpoint on a specified logic probe value pattern (emulator only).	In the **Logic probes:** box, check the **Enable** boxes associated with the logic probes whose values you want to match, then click the **Value:** boxes to specify the values you want to match.
Trigger a breakpoint set on data on a specified flags value pattern. This condition will apply to all breakpoints set on data.	In the **Flags:** box, check the **Enable** boxes associated with the flags whose values you want to match, then click the **Value:** boxes to specify the values you want to match.

3.11.2 Managing Memory Breakpoints

You can perform all memory breakpoint management tasks from the Memory breakpoints window (refer to “Setting Memory Breakpoints” on page 23). The table below lists the memory breakpoint management tasks you can perform and explains how to perform them:

To do this:	Do this:
Enable/disable a breakpoint.	Select the breakpoint from the **List of Breakpoints**, click the **Inspect** button, then click the **On** check box in the **New Settings** box. When the **On** box is checked, the breakpoint is enabled.
Delete a breakpoint.	Select the breakpoint from the **List of Breakpoints**, then click the **Delete** button.
Enable/disable all breakpoints (memory and software).	In the Windows menu, point to Hardware Events, then click Memory Breakpoints. The Memory Breakpoints window opens. Click the Configure button. The Hardware Event Advanced Settings opens Click the Enable/Disable All check box in the Breakpoints box to enable/disable all breakpoints. When it is checked, all breakpoints are enabled.

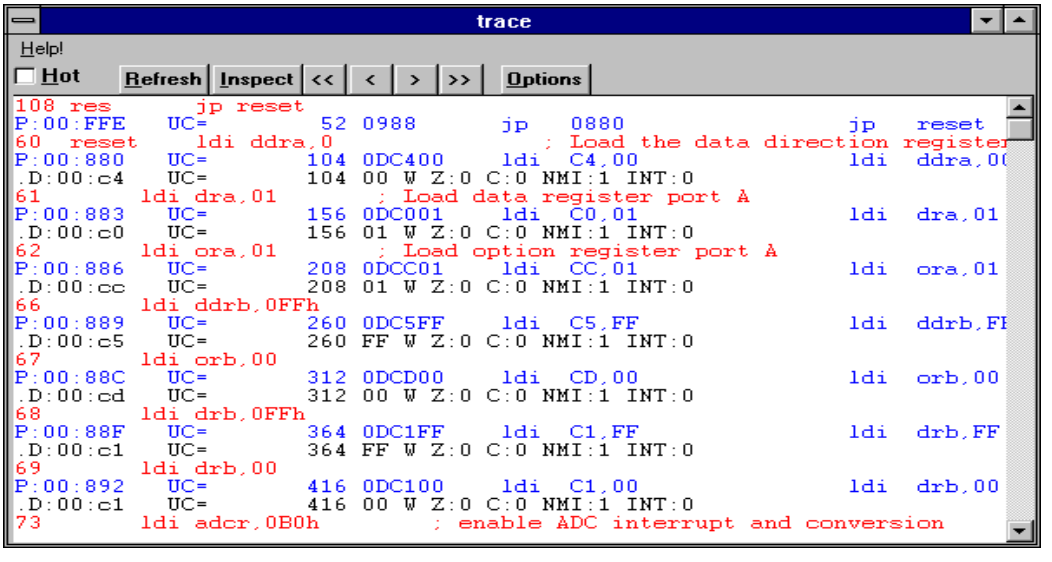
3.12 Working with the Trace Buffer

The emulator trace buffer records some hardware events that occur when a program is executed. Since its size is limited, you can specify what type of events you want to

record. WGDB6 enables you to view either all the trace buffer contents or selected event types. The following paragraphs explain how to view trace buffer contents, how to select the type of events to view and how to specify what type of events are recorded in it.

3.12.1 Viewing Trace Buffer Contents

1. In the **Windows** menu, click **Trace**.
2. The Trace window opens, displaying the events recorded in the emulator hardware trace buffer. Source code is displayed in red, assembler code in blue, and all other lines are displayed in black:



From the Trace window, you can perform the following operations:

To do this:	Do this:
Set the window as Hot, so that its contents are updated every time program execution is suspended.	Click the **Hot** check box. When the **Hot** box is checked, window is Hot.
Update the Trace window contents.	Click the **Refresh** button.
View the value of a constant or data.	Select the constant, variable, or register, then click the **Inspect** button.
Display the disassembled code of a selected address.	Select the address then click the **Inspect** button.
View the source code of a selected function.	Select the function, its address or line number, then click the **Inspect** button.
Browse through the flow of the traced code.	Click the '**<<**', '**>>**' buttons to browse through source code. Click the '**<**', '**>**' buttons to browse through assembler code.

View selected event types.	Click the **Options** button, then select the event types you want to display from the Trace Options window.
View source code, opcodes and operand binary dump, hexadecimal operands of disassembled instructions with variable names each instruction cycle in the Trace window.	Click the **Options** button, then select the appropriate options from the **Disassembly options** in the Trace Options window.

Note: Setting this window as Hot can reduce the execution speed of WGDB6. You can save time by not setting this screen as hot, and updating the display when required using the **Refresh** button.

3.12.2 Selecting the Type of Events to be Recorded in the Trace Buffer

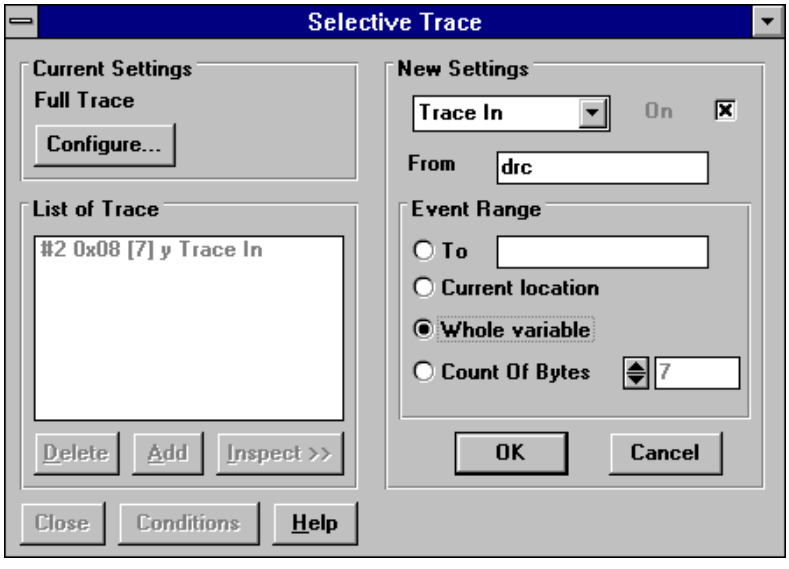
By default, event selection is disabled. This corresponds to the 'full-trace' selection mode. When event selection is disabled, the following information is recorded in the trace buffer:

- The instruction fetch cycle, that provides information on the instruction that was executed.
- The last cycle of the instruction, that provides information on the data that was accessed: its address, value, access mode and other related information, such as the current ST6 mode and flag values.

To record selected event types only:

1. In the **Windows** menu in the Control Bar, click **Hardware Events**, then click **Selective Trace**.

The Selective Trace dialog box opens.



2. Click the **Configure** button.

   i The Hardware Event Advanced Settings dialog box opens.

   ii To only record the cycles defined as TRACEIN events (see below), select **Sampling Filter**.

   iii To only record the cycles that occur between a TRACEON event and a TRACEOFF event (see below), select **Trace On / OFF**.

   iv Click **OK**.

5. In the Selective Trace dialog box, click the **Add** button.

   The New Settings box opens.

6. Select the event type you want to define from the event type drop down list in the New Settings box:

Select this:	To define:
TRACEIN	An area of memory whose related events are recorded in the trace buffer (**Sampling Filter** mode only).
TRACEON	An area of memory, which when accessed, starts event recording in the trace buffer (**Trace On / Off** mode only).
TRACEOFF	An area of memory, which when accessed, ends event recording in the trace buffer (**Trace On / Off** mode only).

7. In the **From** field, enter the start of the address range whose related events you want to define. This can be expressed as a variable (symbolic name) or as a pretty address.
8. In the **Event Range** box, choose the address range whose related events you want to define:

   Select **To** to enter the end address of the memory range whose related events you want to define. This can be expressed either as a variable (symbolic name) or as a Pretty address.

   Select **Current location** to set the related events to only the address defined in the **From** field is accessed (range = 1 byte).

   Select **Whole variable** to set the related event to the whole variable range that is directly linked to the symbol type specified in the **From** field (for example, on whole fields of data symbols for a C-language struct data type).

   Select **Count of bytes** to define the number of bytes, starting from the address pointed to by the **From** filed, whose related events you want to define.
9. Click **OK**.

Repeat steps 3 to 7 if you want to define more event types.

You can also delete event types by selecting the definition and clicking the **Delete** button, or enable/disable definitions by clicking the event definition in the **List of Trace** field then clicking the **Inspect** button, then checking/unchecking the **On** check-box and clicking **OK**.

To reinstate the default (full trace mode) so that all cycles are recorded in the trace buffer:

1. In the Selective Trace dialog box, click the **Configure** button.
2. In the Hardware Event Advanced Settings dialog box, select **Full Trace**.
3. Click **OK** in both the In the Hardware Event Advanced Settings dialog box and the Selective Trace dialog box.

4 CUSTOMISING WGDB6

You can customize the following features in WGDB6:

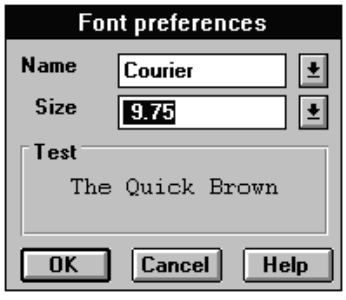
- The screen fonts.
- The breakpoint line, Program Counter line and currently selected line colors.
- The action taken when events occur.

The following paragraphs explain how to do this.

4.1 Changing the Screen Fonts

1. In the **File** menu, point to **Preferences**, then click **Screen Fonts**.

   The Font preferences dialog box opens:



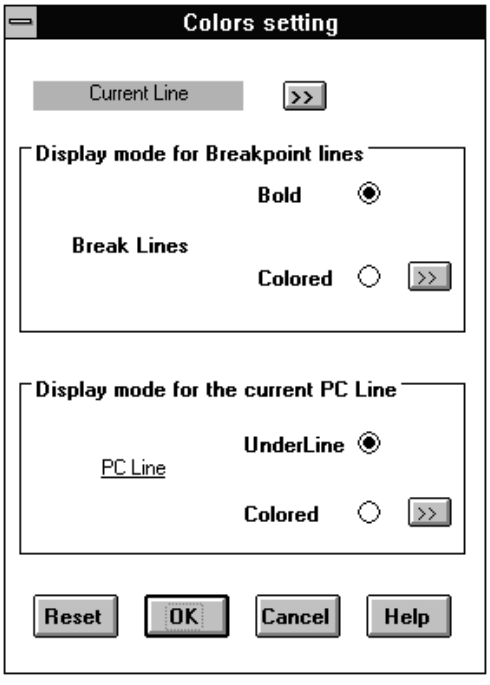
2. Select a new font type from the **Name** drop-down list.
3. Select a new font size from the **Size** drop-down list.
4. Click **OK** to implement the changes you made.

4.2 Changing the Color Settings

To change the breakpoint line, Program Counter line and currently selected line colors and appearance:

1. In the **File** menu, point to **Preferences**, then click **Color Setting**.

The Colors setting dialog box opens:

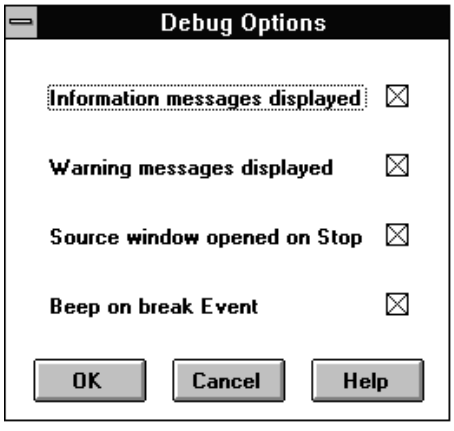


2. To change the color of the current line marker, click >> next to the **Current Line** caption to list the available marker colors and select one.
3. To change the display mode for breakpoint lines, in the **Display mode for breakpoint lines** box:
   - **i** Select the **Bold** button to display the current breakpoint line in bold.
   - **ii** Select the **Colored** button then click >> to list the available marker colors and select one for the current breakpoint line.
3. To change the display mode for the current PC line, in the **Display mode for the current PC line** box:
   - **i** Select the **Bold** button to display the current PC line in bold.
   - **ii** Select the **Colored** button then click >> to list the available marker colors and select one for the current PC line.
3. Click **OK** to implement the changes you made.

4.3 Selecting Which Events are Indicated

1. In the **File** menu point to **Preferences**, then **Debug Options**.

The Debug Options dialog box opens.



2. Click the appropriate check box:

   When the **Information messages displayed** box is checked, information messages are displayed. Information messages inform you, for example, when a memory breakpoint is triggered or a stack overflow occurs.

   When the **Warning messages displayed** box is checked, warning messages are displayed. Warning messages indicate, for example, when required symbol information is not available.

   When the **Source window opened on Stop** box is checked, the module window is displayed when program execution is stopped.

   When the **Beep on break Event** box is checked, a beep sound is made when a breakpoint is reached.

5 WORKING WITH WORKSPACES

Workspaces are made up of the following definitions relating to each program you load to WGDB6:

- Open windows.
- Window positions.
- Software breakpoint definitions.
- Current cursor position in each module window.
- Window states (Snapshot, Hot or Real-time).
- Disassembly options.
- Trace options.

When you load a program, the default workspace configuration that you were using when you last closed it is restored, thus you can continue working from where you left off the previous time.

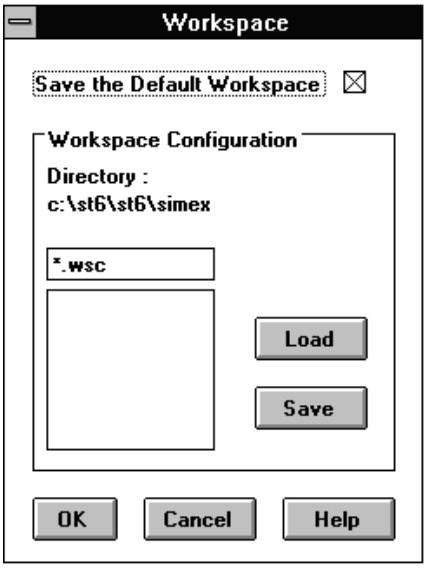
WGDB6 enables you to save a workspace definition, so that you can restore it at a later date. It also enables you to enable/disable automatic default workspace saving, so each time you close a program its workspace is saved.

Note: Since workspaces are directly linked to programs, you can only load or save workspace configurations after the program to which they pertain have been loaded.

5.1 Saving and Loading Workspace Definitions

1. In the **File** menu point to **Preferences**, then click **Work Space**.

The **Workspace** dialog box opens:



2. To load a saved workspace settings file:

   i Select the file you want to load from the list.

   ii Click the **Load** button

   To save the current workspace settings:

   i Type the file name in the file name box.

   ii Click the **Save** button.

3. Click **OK** to implement the changes you made.

5.2 Enabling/Disabling Automatic Default Workspace Saving

To enable/disable automatic workspace saving when you close a program:

1. In the **File** menu point to **Preferences**, then click **Work Space**.

   The Workspace dialog box opens.

2. In the Workspace dialog box, click the **Save the Default Workspace** check box.

   When this box is checked, any workspace modifications you make are saved when you close programs.

6 USING GDB6 COMMANDS

WGDB6 is a Windows interface to GDB6 commands (that is GNU and specific ST6 commands). When you choose a WGDB6 menu option, button, or enter a field, WGDB6 executes an appropriate GDB6 command. Using WGDB6, you can:

- Execute GDB6 commands when WGDB6 is started.
- Execute GDB6 command batch files.
- Enter GDB6 commands using your keyboard.
- View the GDB6 commands executed by WGDB6.
- Record GDB6 commands in a log file.

The following paragraphs explain how to perform these tasks.

6.1 Executing GDB6 Commands on Startup

When you load a program, WGDB6 executes either of the following files if they exist:

hardware.gdb

<program>**.gdb**

where <program> is the name of the loaded program. This enables you to create program-specific commands each time you load a program.

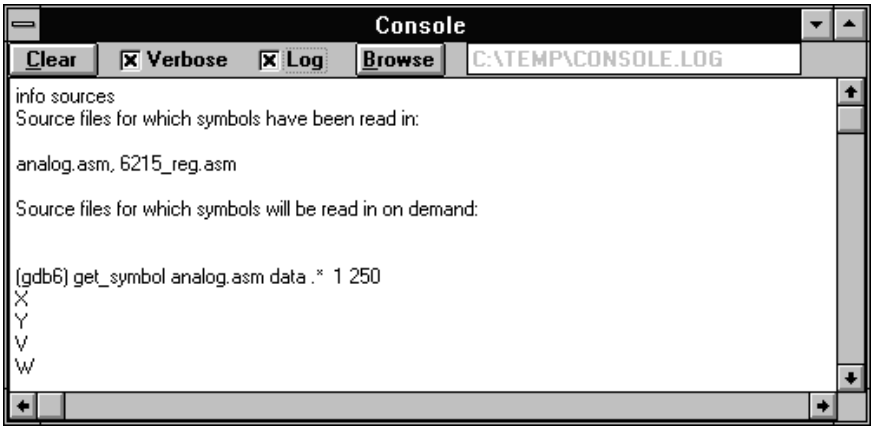
You can create GDB6 files using any ASCII text editor. Each command must be on a separate line. For details about the available commands and their syntax, click GDB Commands on Contents page of the WGDB6 online help.

6.2 Entering GDB6 Commands Using Your Keyboard

WGDB6 lets you enter GDB6 commands using your keyboard. To enter GDB6 commands using your keyboard:

1. In the **Windows** menu, click **Console**.

The Console window now opens, enabling you to enter commands using your keyboard.



2. Click the first available line in the Console window.
3. Type the command, then press the Enter key. GDB6 interprets the line on which the cursor was when you pressed Enter. Therefore, to reissue a command, place the cursor on the command, and press Enter. GDB6 repeats the last command if Enter is pressed on an empty line.

Note that you can access help on GDB commands either by:

- Typing `help` followed by the command name about which you want help in the Console window, then pressing Enter

  or

- In the WGDB6 Control Bar, clicking **Help|Contents** then choosing GDB Commands for STMicroelectronics-specific commands

  or

- In the WGDB6 Control Bar, clicking **Help|GDB Commands** for general GDB commands.

6.3 Viewing GDB6 Commands Executed by WGDB6

To view the GDB6 commands that WGDB6 executes:

1. In the **Windows** menu, click **Console**.

   The Console window now opens.

2. In the Console window, check the **Verbose** check box.

You can now view all the GDB6 commands that are executed by WGDB6 in the Console window.

6.4 Recording GDB6 Commands in a Log FIle

WGDB6 lets you record all the GDB6 commands that it executes and their results in a session in a log file:

1. In the **Windows** menu, click **Console**.

   The Console window now opens.

2. In the Console window, check the **Log** check box.
3. To create a new log file, click the **Browse** button and select the drive and folder you want to create the new file in, then enter the file name in the **File Name** box.
4. To use an existing log file, click the **Browse** button and select the drive and folder containing the file, then select the file name in the **File Name** list.

You can also store input commands (the results of commands generated by WGDB6 and sent to GDB6) and command outputs (those commands generated by GDB6 and sent to WGDB6) separately in files. To record input commands, use the GDB6 command: set remotelogfile <filename>. To record command outputs, use the commands:

`wfopen <filename>`

where `<filename>` is the file to record command outputs in, and

`wfclose`

when you have finished.

7 WGDB6 QUESTIONS AND ANSWERS

7.1 What does the hour glass cursor mean?

When the cursor appears as an hour glass (), this means that WGDB6 is executing a command.

Note that if you move the cursor to outside a WGDB6 window, or a non-active WGDB6 window, although the cursor returns to its normal shape, you cannot use other programs.

7.2 What does the information message “Program stopped at Stack Overflow breakpoint” mean?

This means that the program has exceeded the maximum of six nested functions or has been interrupted in the ST6 stack during execution.

To disable break on stack overflow, In the **Windows** menu, select **Hardware Events** then **Memory breakpoints**, click the **Configure** button, then uncheck the **Stack Overflow Stop** check box.

7.3 Why is the Locals window empty?

This is either because you are debugging a macro-assembler application—macro assembler language does not have local variables, or you are debugging a C-language application, and the current function does not have any variables or parameters.

7.4 How do I specify the location of source files?

To find source files, WGDB6 uses a 'source path' that works like the MS-DOS path feature: the directories specified in the source path are read one after another until the file name is found.

By default, source files must be located in the same directory as the program file. If your source files are in another directory or in several directories, you can use the `directory <pathname>` command to the front of the source path.

You can specify several path names using the "`;`" separator. When used without parameters, the directory command resets the source path to the current directory.

Example:

```
directory c:\tester\sources;c:\tester\library
```

You can put this command into the program startup file named `<program>.GDB` to automatically set the path to the application source files (see “Executing GDB6 Commands on Startup” on page 36).

7.5 How can I modify a memory breakpoint?

You cannot modify the address or address range of a memory breakpoint. You must delete the breakpoint then and create a new one. The conditions that you set on the deleted breakpoint are not affected since they apply to all defined memory breakpoints.

7.6 Why are some software breakpoints never triggered?

If a software breakpoint is not triggered when you think it should be, check that it is not disabled in the BreakPoints List window (see “Managing Software Breakpoints” on page 23). Next, check that the **Enable/Disable All** check box is checked in the Advanced Settings dialog box (**Windows|Hardware events|Memory Breakpoints|Configure** button).

Note that in some older versions of the HDS emulator (monitor version 0.0), the program is stopped at the instruction immediately following the breakpoint.

Symbols

A

B

C

D

E

F

S

T

U

V

W

NOTES: