

**ICAT Debugger
IBM 4758 Cryptographic Coprocessor
Windows NT and Windows 2000**

January 16, 2001

Second Edition (January, 2001)

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult your IBM representative to be sure you have the latest edition and any Technical Newsletter.

IBM does not stock publications at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office that serves your location.

Reader's comments can be communicated by e-mail to George Dolan, gmdolan@us.ibm.com, or the comments can be addressed to IBM Corporation, Department VM9A, MG81/204, 8501 IBM Drive, Charlotte, NC 28262-8563, U.S.A. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1997, 2001. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Table of Contents

About This Book	1
Introducing the ICAT Debugger	3
Before You Begin	3
Minimum Hardware Requirements	3
Installation	3
Environment Variables	3
Finding Source Files	6
Limitations	6
Getting Started	7
Setting up the Coprocessor	7
Setting up the Host Computer	7
Demonstration Session	7
Starting a Debug Session	8
Using the Tool Buttons	9
Helpful Tips and Hints	10
Troubleshooting	10
Ending the Debugging Session	11
Main Debugging Windows	13
Debug Session Control Window	13
Opening a New Source File	14
Opening a Source File to a Function	14
Locating the Execution Point	14
Saving the Contents of the Threads Pane View	14
Saving the Contents of the Components Pane View	15
Opening the Launch or attach Window	15
Setting Breakpoints	15
Setting a Line Breakpoint	15
Setting a Function Breakpoint	17
Setting an Address Breakpoint	17
Setting a Watchpoint	18
Viewing a List of Breakpoints	18
Setting Debugger Properties	19
Remote Page	20
Source Page	21
Setting Monitor Properties	22
Viewing Your Source	23
Source Window	23
Disassembly Window	24
Mixed Window	25
Executing a Program	26
Monitors Windows	27
Viewing Active Functions for a Particular Thread	27
Menus	28
Viewing Registers for a Particular Thread	28
Menus	29
Viewing Storage Contents and Addresses	29
Menus	30

Monitoring Local Variables	31
Modifying Variables	31
Menus	31
Monitoring Other Variables and Expressions	32
Modifying Variables	32
Expressions Supported	33
Supported Expression Operands	33
Supported Expression Operators	33
Supported Data Types	34
Notices	35
Copying and Distributing Softcopy Files	35
Trademarks	36
Index	37

About This Book

This document contains information to help you install, get started, and perform tasks with the Interactive Code Analysis Tool (ICAT) debugger.

If you need assistance from any window while using the debugger, press F1 from any window or choose the Help menu.

Introducing the ICAT Debugger

The IBM Interactive Code Analysis Tool (ICAT) is a debugger that enables developers to debug applications running on an IBM 4758 PCI Cryptographic Coprocessor. The debugger provides a graphical user interface that enables a developer to:

- Locate the current point of execution within an application and view the source that corresponds to that location.
- Examine and modify an application's state, including variables and registers.
- Set breakpoints and execute machine instructions or source statements one-by-one.
- Dump the contents of the call stack.
- Intercept and diagnose exceptions generated by an application.

The ICAT debugger (hereafter referred to in this document as the debugger) is a source-level debugger that runs on a Windows NT or a Windows 2000 system and interacts with a special version of the CP/Q++ operating system running on the coprocessor. The debugger can communicate with CP/Q++ through the PCI bus (if the coprocessor is installed in the host where the debugger is running) or through a null-modem serial cable.

Before You Begin

This section lists the hardware and software requirements, options that can be used when compiling and linking your program, environment variables, and the search order of source files and modules.

Minimum Hardware Requirements

- Intel® x86® 150MHz processor
- 11 MB hard disk space
- If ICAT is to communicate with CP/Q++ through a null-modem serial cable, the COM port over which ICAT communicates must work reliably at 57600 baud (that is, it should be equipped with buffered UARTs).

Installation

ICAT is part of the IBM 4758 Application Program Development Toolkit, which includes three versions of the CP/Q++ operating system. The *debug* versions incorporate a “probe” that acts as ICAT's proxy and manipulates the application under debug. The *production* version does not. The debug versions must be loaded into segment 2 of the PCI cryptographic coprocessor in order to debug applications. For details, refer to the *IBM 4758 Cryptographic Coprocessor Custom Software Developer's Toolkit Guide* and the *IBM 4758 PCI Cryptographic Coprocessor Custom Software Installation Manual* located on the Library page of the IBM 4758 Web site at www.ibm.com/security/cryptocards/.

Environment Variables

The debugger uses environment variables to manage debugging sessions and remote communication. To set the environmental variables, edit the SETICAT.BAT file. Set the environment variables carefully in the session or command prompt window from which the debugger is invoked. Following is a list of the variables and a description of each:

CAT_COMMUNICATION_TYPE

Specifies whether ICAT is to communicate with CP/Q++ through the PCI bus or through a serial port.

For example, type the following at the command prompt to communicate with CP/Q++ through a serial port:

```
SET CAT_COMMUNICATION_TYPE=ASYNC_SIGBRK
```

For example, type the following at the command prompt to communicate with CP/Q++ through the PCI bus:

```
SET CAT_COMMUNICATION_TYPE=PCI
```

CAT_MACHINE

Specifies which host com port the debugger uses to communicate with the cryptographic adapter. A single host may contain more than one coprocessor card. The host device driver assigns each coprocessor a unique number, starting with zero. (The number assigned to a particular coprocessor depends on the order in which information about devices in the system is presented to the device driver by the host operating system and currently there is no way to tell *a priori* which coprocessor will be assigned number 0, which will be assigned number 1, and so on. If CAT_COMMUNICATION_TYPE=PCI, CAT_MACHINE must be set to the number assigned to the cryptographic coprocessor on which the application to be debugged will run. For example,

```
SET CAT_MACHINE=0
```

If CAT_COMMUNICATION_TYPE=ASYNC_SIGBRK, CAT_MACHINE identifies the host COM port over which ICAT will communicate with CP/Q++. For example,

```
SET CAT_MACHINE=COM1
```

CAT_HOST_BIN_PATH

Tells the debugger where to find your debug binary (the .XLD file with debug information) on your host system. See “Finding Source Files” on page 6 for a description of how this environment variable is used.

For example, type the following at the command prompt:

```
SET CAT_HOST_BIN_PATH=I:\4758TEST
```

CAT_HOST_SOURCE_PATH

Tells the debugger where to find your source (for example, rte.c in the demonstration session). See “Finding Source Files” on page 6 for more details.

For example, type the following at the command prompt:

```
SET CAT_HOST_SOURCE_PATH=I:\4758TEST
```

CAT_PATH_RECURSE

Causes a recursive search of the subdirectories below the subdirectories listed in CAT_HOST_BIN_PATH and CAT_HOST_SOURCE_PATH. For example, with the CAT_HOST_SOURCE_PATH=I:\4758TEST variable, the debugger searches the 4758TEST subdirectory and all subdirectories below 4758TEST as well as their subdirectories. The default is NULL, which means the debugger will not perform a recursive search. When the variable is set to any non-null value, the recursive search is performed.

For example, type the following at the command prompt:

```
SET CAT_PATH_RECURSE=ON
```

IPF_PATH32

Locates the IPF32.DLL for the debugger to run correctly. For example, type the following at the command prompt:

```
SET IPF_PATH32=%SCCTK_FS_ROOT%
```

The debugger finds IPF_PTH32 in %SCCTK_FS_ROOT%\bin.

CAT_OVERRIDE

Specifies a path that the debugger searches first to find the source files used to build your debug binaries. See “Finding Source Files” on page 6 for a complete description of the process.

For example, type the following at the command prompt:

```
SET CAT_OVERRIDE=E:\TEMP\UPDATES
```

CAT_TAB

Specifies the number of spaces between tab stops when source code containing tabs is displayed in a debugger window.

For example, type the following at the command prompt:

```
SET CAT_TAB=5
```

The debugger converts each tab in the source to five spaces when the source is displayed.

CAT_TAB_GRID

Specifies the column positions for the tab stops when source code containing tabs is displayed in a debugger window.

For example, typing the following command at the command prompt sets tab stops at the 6th position:

```
SET CAT_TAB_GRID=6
```

CAT_DEBUG_NUMBER_OF_ELEMENTS

Represents the default number of elements displayed for a variable or structure that has a substantial number of elements. The last element displayed for such a structure is labeled “more elements.” Clicking on this entry displays the next *n* elements of the variable or structure.

Note: CAT_DEBUG_NUMBER_OF_ELEMENTS is an environment variable that is set to an integer, *n*.

For example, type the following at the command prompt:

```
SET CAT_DEBUG_NUMBER_OF_ELEMENTS=100
```

The next 100 elements are displayed.

Finding Source Files

The debugger searches for the source files in the following order:

1. CAT_OVERRIDE environment variable.
2. The subdirectory in which the object file generated from the source was compiled (as indicated by debug information in the executable file.)
3. CAT_HOST_BIN_PATH environment variable.
4. The subdirectory in which the translated (.XLD) file was found.
5. CAT_HOST_SOURCE_PATH environment variable, descending subdirectories if the CAT_PATH_RECURSE environment variable is set.
6. The current directory.
7. The path defined in the INCLUDE environment variable.
8. The last specified subdirectory from the "change source file" function.
9. If ICAT cannot find the source in any of the previously mentioned locations, it prompts the user to enter the location of the required source file.

The debugger searches for the executable (.XLD) file in the following order:

1. Current directory.
2. CAT_HOST_BIN_PATH environment variable.

Note: The CAT_PATH_RECURSE environment variable, if specified, causes the debugger to search recursively all subdirectories of the CAT_HOST_BIN_PATH and CAT_HOST_SOURCE_PATH environment variables.

Limitations

The debugger has the following restrictions:

- The debugger can not currently launch an application to be debugged (that is, cannot cause an application to be loaded into the cryptographic coprocessor and assume control of the application before any instructions in the application have been executed). The earliest point at which the debugger can attach to an application (that is, assume control of the application and place it under debug) is after the application's main entry point has been invoked. If you want to make certain that your application does not make progress before the debugger has a chance to attach, you must code an infinite loop at the beginning of the application (and use the debugger to change the point of execution to the statement following the loop after attaching).
- If the debugger is communicating with CP/Q++ through a serial port, no other application on the host can access that port.
- Applications to be debugged must be compiled and linked in such a way that the application executable incorporates debug information. Otherwise the debugger cannot be used to examine and manipulate the application at the source level. However, only the copy of the application executable that the debugger reads must have this information; the copy downloaded to the coprocessor can be reduced in size by stripping debug information from it before it is downloaded.
- ICAT does not handle certain coding styles well. For example, it can be difficult to debug a program that has more than one source statement on a line—the debug information available to the debugger forces the debugger to treat the entire line as a single statement. Thus, you cannot set a breakpoint on, say the second statement on the line, nor can you step through each statement.

Getting Started

This section describes how to start a debugging session, set up the coprocessor and the host, and how to end a debugging session.

Setting up the Coprocessor

To set up the coprocessor:

1. Follow the instructions for installation in the *IBM 4758 Cryptographic Coprocessor Custom Software Developer's Toolkit Guide* to install the coprocessor and use the CLU utility to load the TPRrrrss.clu (the debug kernels) onto the coprocessor.
2. Use the DRUID utility to load your application and start it running for debugging. The application should have an infinite loop near the beginning of the code, as recommended in *IBM 4758 Cryptographic Coprocessor Custom Software Developer's Toolkit Guide*.

Setting up the Host Computer

To set up the host Windows NT or Windows 2000 computer:

1. Install the IBM 4758 toolkit. The debugger is packaged as part of the toolkit in the ...\\scctk\\bin\\nt directory.
2. Ensure that the ...\\scctk\\bin\\nt directory is in the path.
3. Set the environment variables. See "Environment Variables" on page 3 for more information.

Demonstration Session

The following session demonstrates the debugger manipulating the code on the coprocessor.

1. Inspect the scctk\\src\\samples\\rte subdirectory on your host computer. This subdirectory contains the .C source files and the makefiles to make the binaries that the debugger must see on your host computer.
2. Change the rten.mak makefile to ensure that the variable DEBUG is defined (/DDEBUG). This ensures that the attachment loop is included in the executable.
3. Make the rte executable. The executable will be saved at ...\\scctk\\obj\\samples\\rte\\nt\\(bld_env) where (bld_env) is msvcmasm for Microsoft Visual C++ or vacppmsm for IBM Visual Age C++.
4. Translate the executable file to an XLD file, using the CPQXLT utility.
5. Translate the XLD file into a ROD (read-only disk) file using the sccrodsk utility.
6. On the coprocessor, ensure that you have loaded the debug kernel (TPRrrrss.CLU).
7. Use DRUID to load the ROD file onto the coprocessor.
8. On the host computer, modify SETICAT.BAT as described in "Environment Variables" on page 3 so that the CAT_MACHINE environment variable includes the number of the coprocessor you wish to debug. Now, run SETICAT.BAT to set up the environment variables, run ICATCPW.EXE, and then wait for the Launch or Attach dialog box to display.
9. From the Launch or Attach dialog box, enter rte in the **Program** field. Click the **Attach** button and click **OK** to attach the program. (Program launching is not supported, the loader launches the program when it is loaded into the coprocessor.) It will take some time, but the **Debug Session Control** window will be displayed, with the rte application in the component list.
10. Click the plus sign located beside the path name for the RTE.XLD file in the Debug Session Control window. The path expands to display a list of functions within the file. Double-click rte to open the Source window for rte. The function should pause within the infinite loop.
11. Select a Mixed view and notice that we mix the disassembly with the C source. Switch back to a Source view.
12. Set a breakpoint within the infinite loop, then set the debugger to Run.

- Next do a jump to line 34 (after the infinite loop). At this point, you can debug the program normally. However, if you hit run, the system will stop on `sccGetRequest()`, since no host function has asked for service yet.
- From the host, run the `hre.exe` application.
- When the debugger hits a breakpoint, display a mixed view. Single step the MASM code a couple of times. Switch back to a source view. Next, set a breakpoint at line 58, and run again. When you hit that breakpoint, you can double-click variables, do a call stack unwind, show a Register window or a Storage window, and so on.

This concludes the demonstration session.

Starting a Debug Session

Load the program you wish to debug on the coprocessor.

To start the debugger:

- From the Windows NT or Windows 2000 command prompt, enter `ICATCPW` followed by one of the following parameters:

/P+ Use program profile information (the default).

/P- Do not use any program profile information.

The Launch or attach window is displayed.

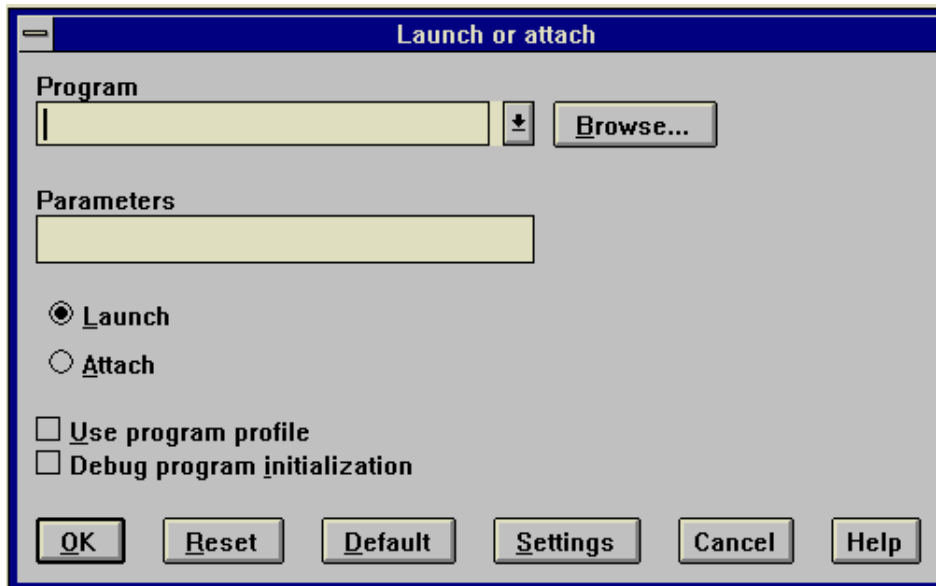


Figure 1. Launch or attach Window

- Note:** It's usually a good idea to have set up your environment variables by way of a `SETICAT.BAT` file before invoking the debugger. See "Environment Variables" on page 3 for more information.
- If you're going to debug a program more than once, select the **Use program profile** check box to reactivate the windows and breakpoints.
 - Enter the name of the program to be launched in the **Program** field. You can click the **Browse** button and select a program from the list of files.
 - Click **Attach**.

Note: The **Launch** option is currently unsupported.

5. Click **OK**. The Debug Session Control window opens displaying the threads and components of your source.

Reset returns the window settings to the values you defined upon initialization of the window.

The **Default** button restores the window's default settings.

The **Settings** button displays the Debugger Properties window, which enables you to select how threads and source files are initially displayed and enables you to set environment variables. See "Setting Debugger Properties" on page 19 for more information.

Using the Tool Buttons

A tool bar has been provided on the debugger windows for easier access to frequently used features. To display buttons in a window, enable the **Tool buttons** choice that is listed under the **Options** menu. The following is a list of features that are provided:



Step over executes the current line in the program. If the current line is a call, execution is halted when the call is completed.



Step into executes the current line in the program. If the current line is a call, execution is halted at the first statement in the called function.




Step debug executes the current line in the program. The debugger steps over any function for which debugging information is not available (for example, library and system routines) and steps into any function for which debugging information is available.



Step return automatically executes the lines of code up to and including the return statement of the current function.



Run enables you to start and stop the program. When the debugger is running, the **Run** button is disabled and the **Halt** button  is enabled. You can click the **Halt** button to halt the program execution. You can also interrupt the program you are debugging by selecting the **Halt** choice from the **Run** menu.



View changes the current source window to one of the other source windows. For example, you can switch from the Disassembly window to the Mixed window.



Monitor Expression enables you to type in the name of the expression you want to monitor.



Call Stack enables you to view all of the active functions for a particular thread including the system calls. The functions are displayed in the order that they were called.



Registers enables you to view all the processor and coprocessor registers for a particular thread. This is useful only for native-binary debugging.



Storage displays the storage contents and the address of the storage. This is useful only for native-binary debugging.



Breakpoints enables you to view all the breakpoints that have been set.



Debug Session Control displays the Debug Session Control window. This is the main window for the debugger and runs during the complete session.



Growth direction enables you to change the direction that items are displayed on the stack.



Delete enables you to delete the selected item.



Delete all enables you to delete all the items in the window.



32-bit float displays the storage contents as a 32-bit floating point/number.



64-bit float displays the storage contents as a 64-bit floating point/number.



32-bit unsigned displays the storage contents as a 32-bit unsigned integer.



32-bit signed displays the storage contents as a 32-bit signed integer.



ASCII displays the storage contents in ASCII.



Hex and ASCII displays the storage contents in Hex and ASCII.



Change representation enables you to change the data representation.

Helpful Tips and Hints

The following tips and hints may be helpful:

- You must have Windows NT or Windows 2000 on your host computer.
- Put any environment variables that you want set in a command file. For example, you could put them in the SETICAT.BAT file or create your own command file.
- The code translator (CPQXLT.EXE) can be used with the /NODEBUG option to strip debug information from the EXE file before building a .ROD file to be loaded on the coprocessor. However, this .XLD file should not be in the CAT_HOST_BIN_PATH of the debugger before the .XLD file containing debug information.
- Using C, you can write your program code with stylistic features that are not supported by the debugger. For example, multiple statements on the same line are difficult to debug. None of the individual statements can be accessed separately when you set breakpoints or when you use step commands.

Troubleshooting

Following are some things to check when the debugger is not doing what you think it should:

- If the debugger can't attach to the application under debug:
 - If you have just loaded your code onto the coprocessor, wait a couple of minutes. The loader may take some time to run the program to the point at which it can be attached.
 - If you are using the serial port to debug, ensure that your serial cable is a "null-modem" cable (a cable that connects the transmit data pin of one machine to the receive data pin of the other).

- If you are using the serial port to debug, ensure that your serial cable is securely attached to both the host computer and the target coprocessor.
- If you are using the serial port to debug, ensure that the serial cable is connected to the COM port on the host computer which is specified with the CAT_MACHINE environment variable and that the CAT_COMMUNICATION_TYPE environment variable is set to ASYNC_SIGBRK.
- If you are using the serial port to debug, ensure that you have buffered UARTs on the host computer and that you are using 57600 baud.
- If you run the debugger on a notebook computer, ensure that your communication port is enabled and powered on.
- Ensure that the debug kernel is installed on the target coprocessor and that your application has been loaded onto the coprocessor.
- Ensure that CAT_HOST_BIN_PATH is set to the proper path.
- If the debugger only displays the disassembly listing of your program and not the source listing:
 - Ensure that the program was compiled with the proper flags to enable source-level debugging.
 - Ensure that the .XLD file in the CAT_HOST_BIN_PATH was not processed by a debug stripper utility. Debug stripper utilities make the DLLs smaller by removing the debug information.
 - Ensure that your CAT_HOST_SOURCE_PATH is set to reference your source files.
 - Ensure that ICATCPW was not run from a directory containing a non-debug version of your XLD files.

Ending the Debugging Session

To end the debugging session, click **Close debugger** (located within the **File** menu) from any of the debugger windows. The Close Debugger window is displayed. Select one of the following choices:

- Select **Yes** to end your debugging session.
- Select **No** to return to the previous screen without exiting the debugger.

You can also end the debugging session by pressing F3 in any of the debugger windows.

Main Debugging Windows

This section introduces the Debug Session Control window and how to perform functions from this window. It also introduces the three source windows that offer different views of your source code.

Debug Session Control Window

The Debug Session Control window is the control window of the debugger and is displayed during the entire debugging session. This window is divided into two panes: *Threads* and *Components*.

- The *Threads* pane contains the threads, their names, and the state of the threads started by your program. To display the state of a thread, click the plus icon to the left of the thread.

Right-click on a selected item to display the Thread menu and press F1 to view help for this item.

- The *Components* pane shows the path names of the modules that you are debugging.

Right-click on a selected item to display the Component menu and press F1 to view help for this item.

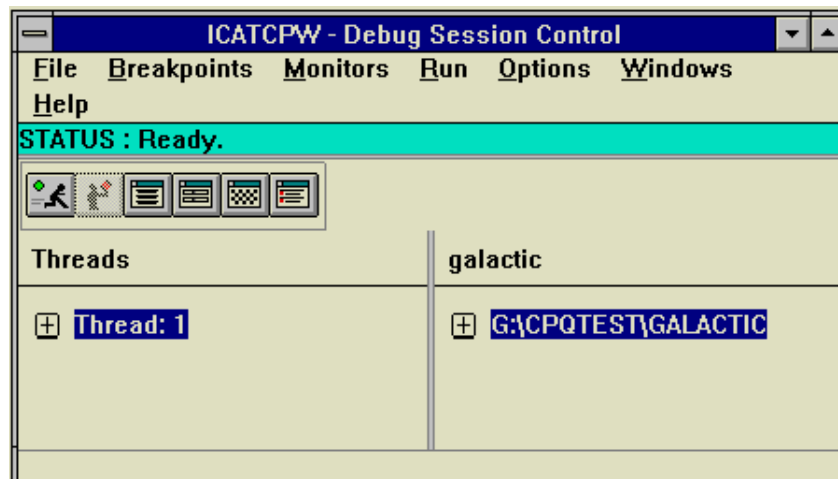


Figure 2. Debug Session Control Window

From the Debug Session Control window you can select menus that enable you to:

- Open a new source file.
- Open a source window to a particular function.
- Open a source window containing the next line to be executed.
- Save the contents of the Threads pane or the Components pane into a file.
- Set line, function, address, watchpoint, or load occurrence breakpoints.
- Display a list of breakpoints that have been set.
- Monitor the call stack for a particular thread.
- Monitor registers and flags for a particular component or thread.
- Display the local variables for your application's current function.
- Execute your application or stop execution.
- Modify how the debugger window is displayed.

Opening a New Source File

You can open additional source files from the Debug Session Control window.

To open a new source file:

1. Click **Open New Source** (located within the **File** menu).
2. Type the name of the object file you want to open the source for in the **Source** field.

For example, to look for the source used to compile A123.OBJ, type the following in the **Source** field:

A123

If you are uncertain of the file name, click the **File list** button to view a list of the files that you can select.

3. Type the name of the executable file in the **Executable** field. The source files for the executable file are displayed in the **Source** field.
4. Select the **All executables** check box if you want to search all the executable files. Clear the **All executables** check box to search for a particular executable file.
5. Select the **Debugging information only** check box if you want to search only the source files that contain debugging information.
6. Click the **OK** button.

Opening a Source File to a Function

You can use the Find Function window to open a source window to a particular function.

1. Click **Find Function** (located within the **File** menu).
2. Type the name of the function you want to search for in the **Function** field.

If the function that you specify is not found, the following message is displayed:

No matching function found. Desired function could be static.

This means it might be a static function or the function you specified does not exist.

The debugger searches each object file for global functions that match the function name specified. If an object file contains the global function that was specified, then it also searches that file for any static function with the same name.

3. Select the **Debugging information only** check box if you want to search only the object files that contain debugging information.
4. Select the **Case sensitive** check box if you want to search for the string exactly as typed. Clear this check box if you want to search for both uppercase and lowercase characters.
5. Click the **OK** button.

Locating the Execution Point

To locate the execution point in your source, click **Where is execution point** (located within the **File** menu). A source window is displayed containing the next line to be executed.

Saving the Contents of the Threads Pane View

If you want to save the contents of the Threads pane view in a file, click **Save thread list in file** (located within the **File** menu). This saves the view in a file named THREADS.OUT. To change the default file name, click **Options, Windows settings**, and then **Display style** (located within the Debug Session Control window) and type the file name in the **Threads output file** field.

Saving the Contents of the Components Pane View

If you would like to save the contents of the Components pane view in a file, click **Save component list in file** (located within the **File** menu). This saves the view in a file named COMPS.OUT. To change the default file name, click **Options, Window Settings**, and then **Display Style** (located within the Debug Session Control window) and type the file name in the **Components output file** field.

Opening the Launch or attach Window

If you want to start a debugging session, click **File** and then **Launch or attach** (located within the Debug Session Control window). The Launch or attach window is displayed. See “Starting a Debug Session” on page 8 for more information.

Setting Breakpoints

You can control program execution by setting breakpoints. A breakpoint stops the execution of your program at a specific location or when a specific event occurs.

To set breakpoints, click the **Breakpoints** menu (located on the Debug Session Control window or located on any source window), and then click the appropriate choice for the type of breakpoint you want to set. When you set a breakpoint in one source window, it is reflected in the other source windows. In addition, you can set a simple line breakpoint in a source window using either the mouse or the keyboard:

- To set a breakpoint with the mouse, double-click in the prefix area of an executable statement (the prefix area is the area to the left of the source code where line numbers or addresses are displayed); the prefix area is displayed in red to indicate that the breakpoint has been set. Double-click in the same prefix area to delete the breakpoint.
- To set a breakpoint with the keyboard, move the cursor to the prefix area and then press the Spacebar to set or delete a breakpoint.

Note: You can set as many breakpoints as you want.

You can set either line, function, address, or watchpoint breakpoints.

Setting a Line Breakpoint

A line breakpoint enables you to stop the execution of your program at a specific line number.

You set a line breakpoint from the Line Breakpoint window. To display the window, from the **Breakpoints** menu, click **Set line**.

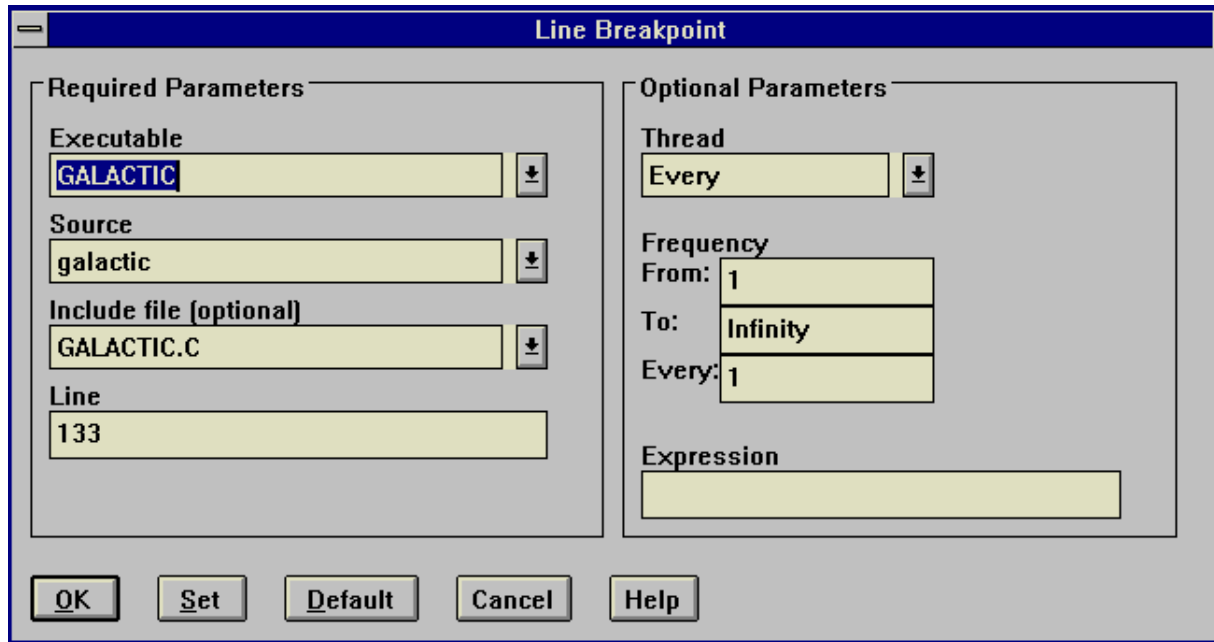


Figure 3. Line Breakpoint Window

The Line Breakpoint window is divided into two group headings: *Required Parameters* and *Optional Parameters*.

Required Parameters:

1. Type the executable name or click an executable located within the drop-down list in the **Executable** field.
2. Select the executable file in which you want to set the breakpoint.
3. Type the source name or click a source name located within the drop-down list in the **Source** field.
4. Select the source name where you want to set the breakpoint.
5. If the source you selected has include files with executable statements, type the name of the file in the **File (optional)** field or click the file name located within the drop-down list. All the file names that contain executable lines are displayed in the drop-down list.
6. Select the file where you want to set the breakpoint.
7. Type the line number where you want to set the breakpoint in the **Line** field.

Optional Parameters:

1. Click the drop-down list in the **Thread** field.
2. Select the thread where you want to set the breakpoint.

Click **Every**, the default, to set a breakpoint in all of the active threads in your program. The Every choice is thread independent. Select one of the individual threads to set a breakpoint in only one thread. Threads are added to the Thread list as new threads are activated. Type a number in the **From** field to activate the breakpoint the nth time the location is encountered.

3. Type a number in the **To** field to stop activating the breakpoint after the nth time the location is encountered.
4. Type a number in the **Every** field to indicate how often the breakpoint should be activated within the From and To range.

Note: The **Frequency** fields (From, To, and Every) are used for location, address, and load occurrence breakpoints.

5. If you are setting an address, function, or line breakpoint, you can also type an expression in the **Expression** field. The execution of the program stops only if this condition tests true. For example, you could type the following:

`(i==1) | | (j==k) && (k!=5)`

Note: Variables in a conditional expression associated with a function breakpoint are limited to any static or global variables that are known to the called function when the function is called. The debugger does not always evaluate local variables and automatic variables correctly. The maximum length of the condition is 256 characters.

6. Click **Set** to set the breakpoint.

Setting a Function Breakpoint

A function breakpoint stops the execution of your application when the first instruction of the function is encountered where the breakpoint has been set.

You set a function breakpoint from the Function Breakpoint window. To display the window, click **Set function** (located within the **Breakpoints** menu).

To set a function breakpoint:

1. Type an executable name or click an executable located within the drop-down list in the **Executable** field.
2. Select the executable file where you want to set the breakpoint.
3. Type the source name or click a source name located within the drop-down list in the **Source** field.
4. Select the source where you want to set the breakpoint.
5. Type the name of the function in the **Function** field where you want to set the breakpoint or click a function located within the Function list.
6. Select the **Debugging information only** check box if you want to search only the object files that contain debugging information.
7. Select the **Case sensitive** check box if you want to search for the string exactly as typed. Clear this check box if you want to search for both uppercase and lowercase characters.
8. Click or type optional parameters (if any).

For a description of the fields under the Optional Parameters group heading, see Optional Parameters on page 16.

9. Click **Set** to set the breakpoint.

Setting an Address Breakpoint

An address breakpoint enables you to stop the execution of your application at a specific address.

You set an address breakpoint from the Address Breakpoint window. To display the window, click **Set address** (located on the **Breakpoints** menu).

To set an address breakpoint:

1. Type the name of the address or expression where you want to set the breakpoint in the Address field.

For example, to set an address breakpoint for the address 0x000A1FCC, type one of the following:

`0x000A1FCC` or `A1FCC`

The 0x is optional.

2. Click or type optional parameters (if any).

For a description of the fields under the Optional Parameters group heading, see Optional Parameters on page 16.

3. Click **Set** to set the breakpoint.

Setting a Watchpoint

A watchpoint stops the execution of your application when the contents of memory at a given address are referenced or when an instruction is fetched from a particular address.

You set a watchpoint from the **Watchpoint** window. To display the window, click **Set watchpoint** (located within the **Breakpoints** menu).

To set a watchpoint:

1. Type a hexadecimal address or an expression that can be evaluated to a hexadecimal address in the **Address (or expression)** field.

Note: If you type ABC in the **Address (or expression)** field, and there is a variable named ABC, the value of the variable is used instead of the hex value ABC. Also, you can type &a in the field to set the watchpoint on the address of variable a.

For example, type the following in the field to set a watchpoint for the address *&variable*.

&variable

2. Click the Watchpoint Type.

The debugger supports four types of watchpoints. They are as follows:

Read	Causes a break when the address is read.
Write	Causes a break when the address is written to.
Read or write	Causes a break when the address is read from or written to.
Instruction fetch	Causes a break when the instruction at that address is fetched.

Attention: If you set a watchpoint that is on the call stack, you should remove the watchpoint before leaving the routine associated with the watchpoint. Otherwise, when you return from the routine, the routine's stack frame is removed from the stack leaving the watchpoint intact. Any other routine that gets loaded on the stack then contains the watchpoint. You can set up to four watchpoints.

3. Click or type optional parameters (if any).

For a description of the Optional Parameters group heading, see Optional Parameters on page 16.

4. Click **Set** to set the watchpoint.

Note: The debugger monitors 1, 2, or 4 bytes for the type of watchpoint operation that you select. This choice is made for you on the **Instruction fetch** type.

Viewing a List of Breakpoints

The Breakpoints List window lists all the breakpoints that have been set in your application. It also displays the state of each breakpoint.

To display the Breakpoints List window, click **List** (located within the **Breakpoints** menu of the Debug Session Control window).

Type	Executable	Source	File	Function	Line	Address
Line	GALACTIC	galactic	GALACTIC.C	main	138	0x040
Line	GALACTIC	galactic	GALACTIC.C	main	141	0x040
Line	GALACTIC	galactic	GALACTIC.C	main	144	0x040
Line	GALACTIC	galactic	GALACTIC.C	main	147	0x040
Line	GALACTIC	galactic	GALACTIC.C	main	151	0x040
Line	GALACTIC	galactic	GALACTIC.C	main	153	0x040

Figure 4. Breakpoints List Window

Note: Multiple breakpoints can be set on the same line. All occurrences are displayed in the breakpoint list.

The following information is provided for each breakpoint:

- The type of breakpoint
- The position of the breakpoint
- The enablement state
- The conditions under which the breakpoint is activated

From the menu on this window you can:

- Close your current debugging session.
- Delete, disable, and modify breakpoints.
- Set line, function, address, watchpoint breakpoints.
- Modify how the information in the window is displayed.
- View a list of open windows and select any open window to display that window.
- Display help.

Setting Debugger Properties

From the Debug Session Control window click **Options, Debugger settings**, and then **Debugger properties** to select how the threads and source files are initially displayed. The Debugger Properties window contains two tabs:

- Remote
- Source

Remote Page

When you click the **Remote** tab (located on the Debugger Properties window), the following page is displayed:

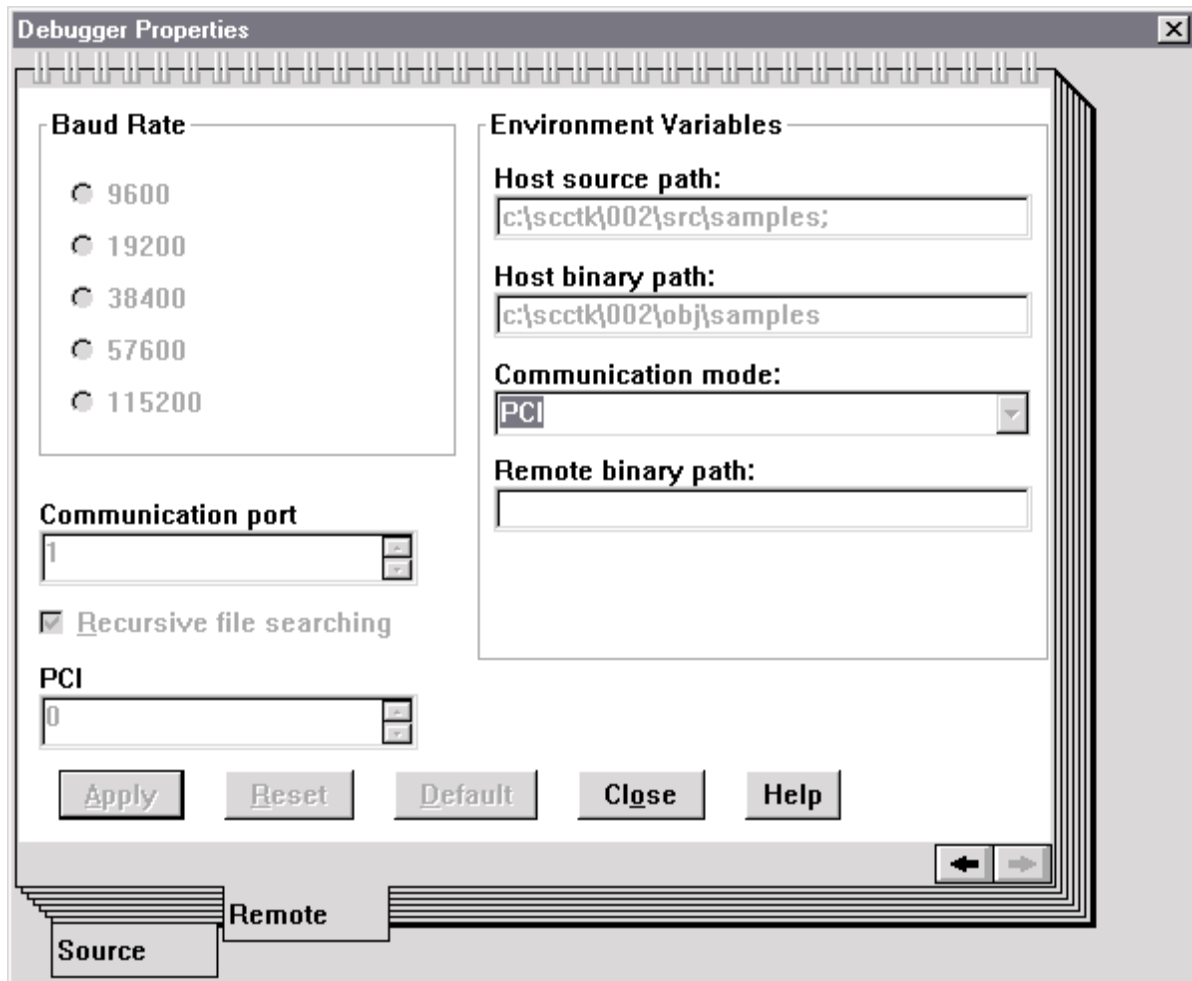


Figure 5. Debugger Properties Window - Remote Page

From the Remote page you can:

- Set the communication baud rate.
- Set the communication port.
- Set the path where the debugger finds the source.
- Set the path where the debugger finds the debug binary modules.
- View the communication mode (ASYNC_SIGBRK for serial or PCI for direct).
- Set the option for recursive subdirectory searching of the source and binary paths.
- Set the remote path for the probe to use to find the debug binary modules.

Note: The values for this window are dithered and cannot be changed after communication has been established with the target coprocessor.

To change your communication setting paths dynamically before communication is established with the target computer, adjust any of the **Environment Variables** fields. See “Environment Variables” on page 3 for detailed information on environment variables.

These fields correspond, respectively, to the following environment variables:

- CAT_MACHINE (first two fields)
- CAT_PATH_RECURSE (check box)
- CAT_HOST_SOURCE_PATH
- CAT_HOST_BIN_PATH
- CAT_COMMUNICATION_TYPE

If you select the **Recursive file searching** check box, the debugger searches all source and binary path subdirectories recursively.

Source Page

When you click the **Source** tab (located on the Debugger Properties window), the following page is displayed:



Figure 6. Debugger Properties Window - Source Page

Use this page to determine:

- When a source window is displayed during a debugging session.
- How to process a source window from which execution has just left. The window can remain displayed, be turned into an icon, or be discarded.

To display the source view of all threads or a particular thread when execution stops, choose any selection located under the **Display at stop** group heading.

In the course of debugging, the **Old Source Disposition** selections enable you to control the behavior of source windows following command execution. These radio buttons control the behavior of source windows within a thread.

The dispositions that the views can take are:

- Keep** Leaves open the source windows that contain the components and threads that you select with **Display at stop**.
- Minimize** Changes into icons the views that contain the components and threads that you select with **Display at stop**.
- Discard** Disposes of the views that contain the components and threads that you select with **Display at stop**.

You can choose to display more than one source window for a particular source file. Enable the **Multiple views** check box located under the **Settings** group heading if you want to have multiple source windows open at the same time.

To select functions you want to perform with the right mouse button, choose the radio button that represents the action located under **Mouse Button 2 Behavior**.

Setting Monitor Properties

To select the settings for monitoring variables or expressions:

1. From the Debug Session Control window click **Options, Debugger settings**, and then **Monitor properties**. The Monitor Properties window is displayed.

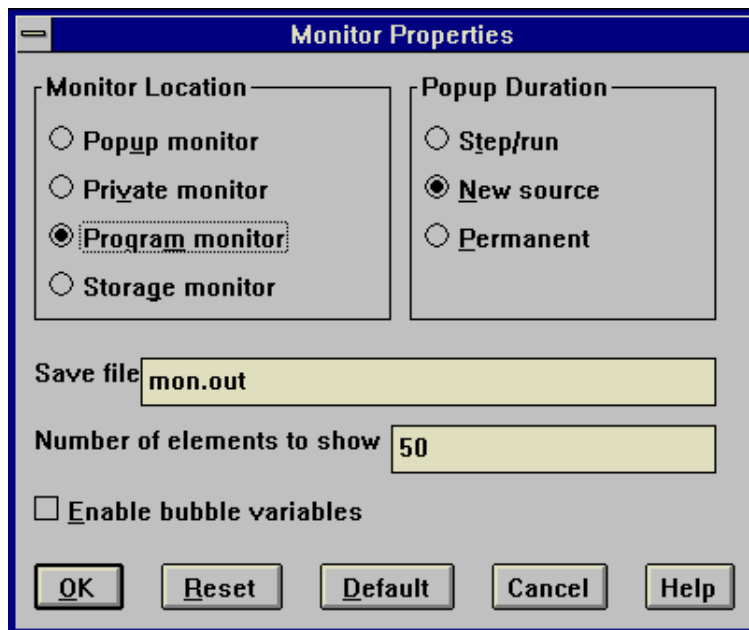


Figure 7. Monitor Properties Window

From this window you can set the window into which the variable or expression being monitored is placed or for expression windows, how long the monitor windows are displayed.

2. Define the monitor window that opens when you select a variable or expression to monitor. Following are the selections you can make and the corresponding windows:

Popup monitor	Display the variable or expression in an expression window.
Private monitor	Display the variable or expression in the Private Monitor window.
Program monitor	Display the variable or expression in the Program Monitor window.
Storage monitor	Display the variable or expression in the Storage window.
3. If you click **Popup monitor**, click one of the following radio buttons to specify how long the expression window is displayed:

Step/run	The monitor window closes when the next step command or Run is executed.
New source	The monitor window closes when execution stops in a new source file.
Permanent	This monitor window is associated with a specific source window and closes when the associated source window closes.
4. Type a file name and extension in the **Save file** field to identify where all monitor windows will save their contents.
5. The **Number of elements to show** field identifies the maximum number of structure or class elements that are displayed at one time for a given variable in the monitors.
6. Select the **Enable bubble variables** check box if you want a bubble value for the contents of a variable to appear as you place the mouse pointer over the variable in the Source, Disassembly, and Mixed view windows.

Viewing Your Source

A source window enables you to view the program you are debugging. You can view your source in one of the following windows:

- Source
- Disassembly
- Mixed

Source Window

A source window is thread specific. Executable lines are initially displayed in blue, and nonexecutable lines are initially displayed in black. Lines with breakpoints have a red prefix, and lines with disabled breakpoints have a green prefix.

The Source window displays the source code for the current function of the program being debugged. If source is available, the Source window is displayed with the Debug Session Control window when the debugging session starts; otherwise, the Disassembly window is displayed.

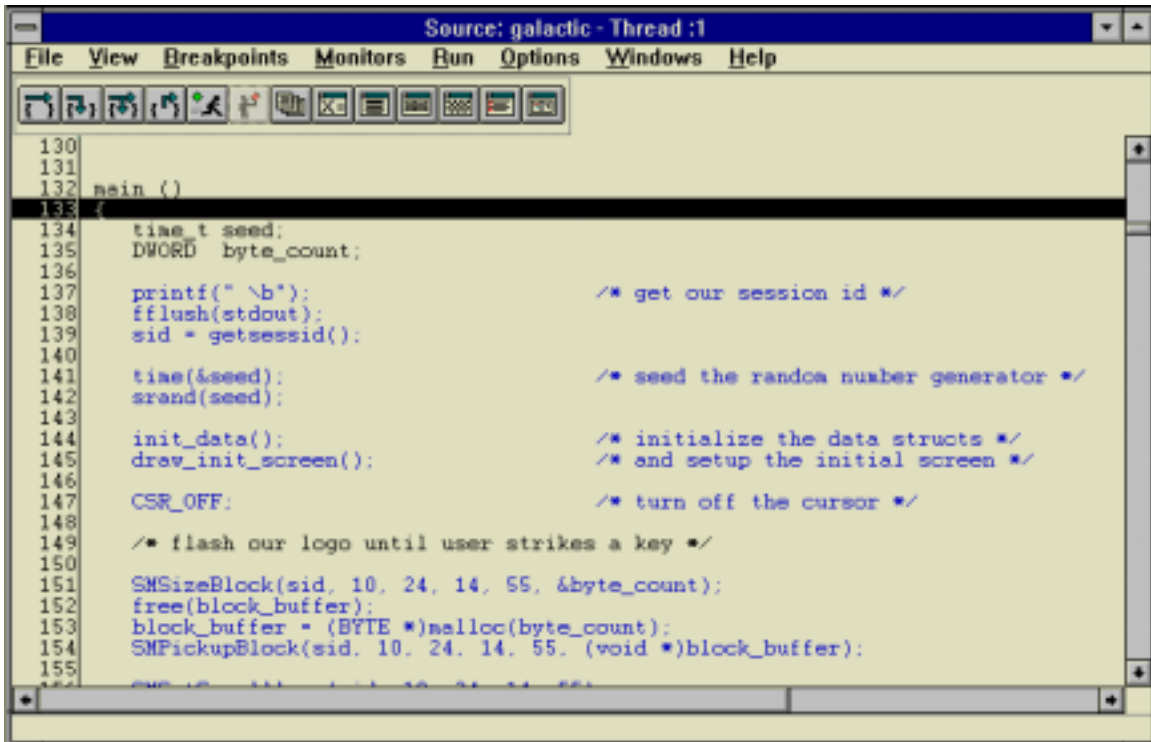


Figure 8. Source Window

Disassembly Window

The Disassembly window displays the assembler or bytecode instructions for your program without symbolic information.

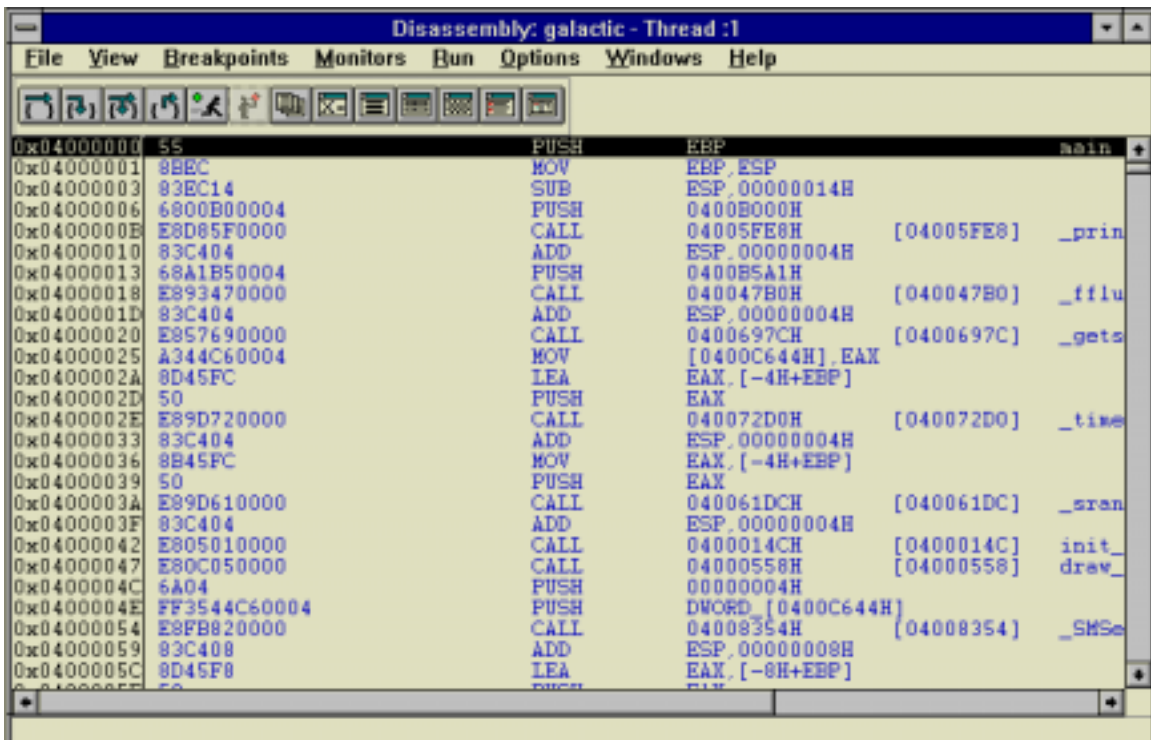


Figure 9. Disassembly Window

Mixed Window

The Mixed window displays your program as follows:

- Each line of source code is prefixed by its line number as in the Source window.
- Each disassembled line is prefixed by an address as in the Disassembly window.
- Source comment lines are also displayed.
- The lines of source code are treated as comments within the lines of disassembly code. You can only set breakpoints or run your program on lines of disassembly code.

Note: The Mixed window cannot be opened if the source code is not available.

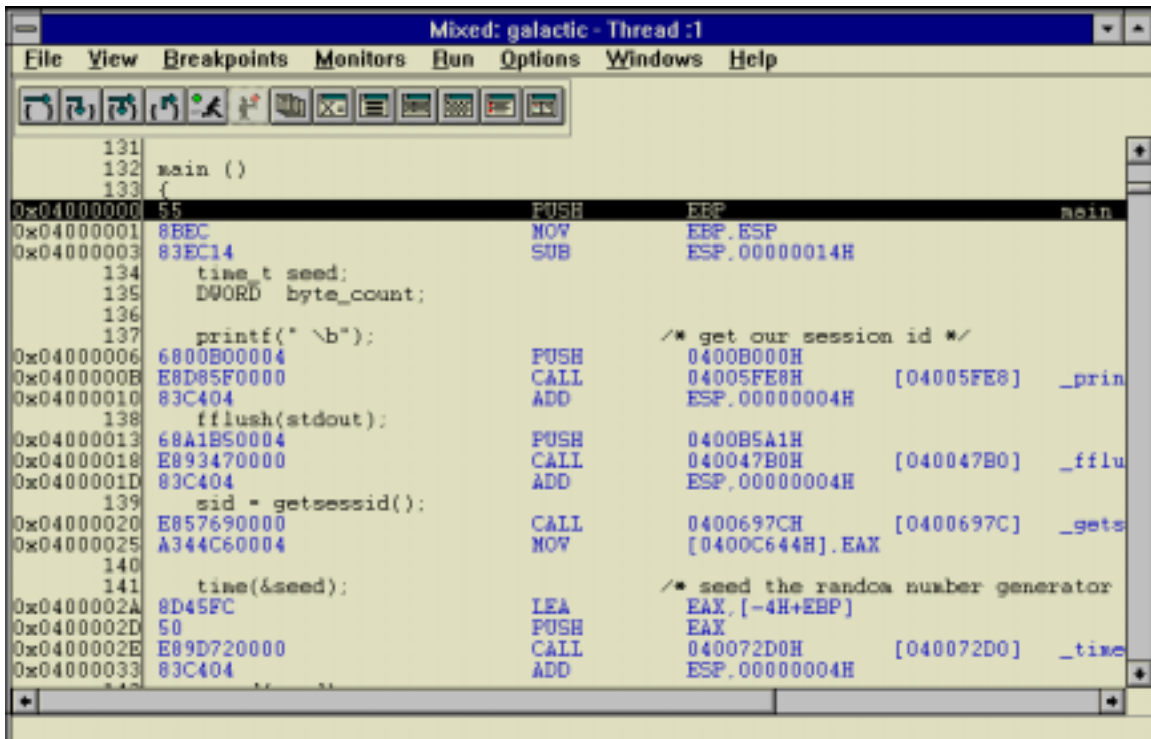


Figure 10. Mixed Window

Each of the source windows has menus. The menus are the same as the Debug Session Control window menus with the following exceptions:

- **File** menu—**Save window in file**

This choice enables you to save the current source view to a named file.

- From the **View** menu, you can:
 - Locate strings of text:
 - Alphabetic and numeric
 - A maximum of 256 characters
 - Uppercase and lowercase characters
 - Scroll to a particular line.

You can also use the Scroll to Line Number window to set a breakpoint. In the Line field, enter the line number and then click the Breakpoint button.

- View include files.
- Change the text file name (specify a file name to be used as the source in the current view).
- Select a different view of your application.

- **Breakpoints** menu—**Toggle at current line** choice.

Toggle at current line sets a breakpoint on the current line or deletes an existing breakpoint from the current line.

- **Monitors** menu—Monitor expression

Enables you to monitor expressions or variables and add them to various monitor windows.

Note: If you need help with any of the menus, press F1 while the menu is selected.

Executing a Program

You can execute a program from any of the source windows (Source, Mixed, or Disassembly) using step commands or the Run command.

Step commands Step commands control the execution of the program. The step commands are located on the tool bar of the source windows and under the **Run** menu of the source windows.

Run command The Run command runs the program until a breakpoint is encountered, the program is halted, or the program ends. You can start the Run command from the **Run** button (located on the tool bar) or the **Run** menu of the source windows.

When you execute a program, a clock icon is displayed to indicate that the program is running and that the program might require input to continue to the next breakpoint or termination of the program.

Monitors Windows


To open Monitors windows, from the **Monitors** menu of the Debug Session Control window, click any of the following choices:

- Call stack
- Registers
- Storage
- Local variables

These windows are also accessible from the tool bar of the source windows. See “Using the Tool Buttons” on page 9 for information about the tool bar.

Viewing Active Functions for a Particular Thread

You can view all of the active functions for a particular thread including system calls from the Call Stack window.

To display the Call Stack window, click Call stack (located within the **Monitors** menu) or click the **Call Stack** button  (located on the tool bar).

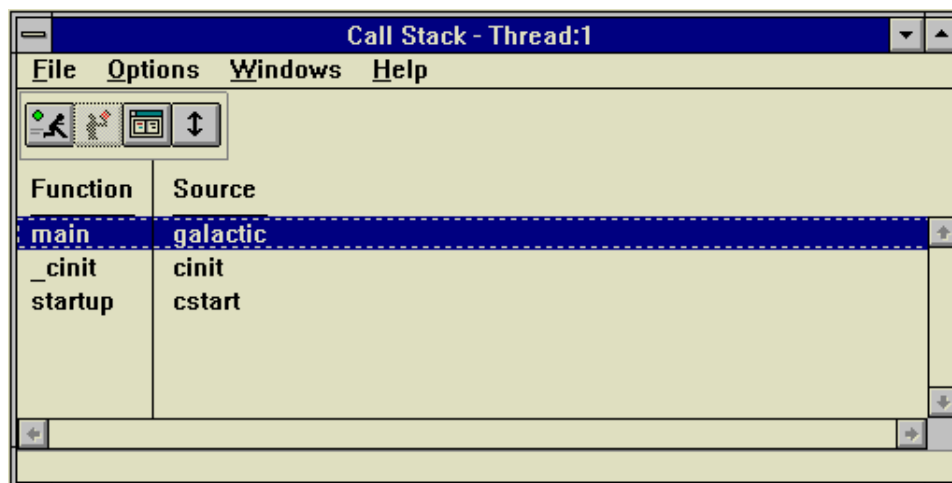


Figure 11. Call Stack Window

Each Call Stack window displays call stack information for only one thread. When the state of the program changes, such as when you execute the program or you update displayed data, the Call Stack window changes to reflect the current state. You can double-click any call stack entry to display the source code for that entry. The line that calls the next stack entry is selected. The remaining stack size shows the bytes left in the stack for the thread.

Note: The stack might not be displayed correctly if the code does not follow standard calling conventions, or if you step into optimized code.

Menus


From the menus of the Call Stack window you can:

- Save the contents of the Call Stack window in a file. Choose the file name by clicking **Options** and then **Display style**. Enter the file name in the **Save file** field.
- End the debugging session.
- Select the type of information displayed in the window and choose how items are displayed.
- Reset all your window settings to their original settings.
- Enable or disable the tool bar.
- Select whether you want hover help to be shown.
- Select to display the information area in the window.
- View a list of open windows and select a window from the list to display it.
- Display help.

Note: If you need help with any of the menus, press F1 while the menu is selected.

Viewing Registers for a Particular Thread

You can view all the processor registers for a particular thread from the Registers window.

To display the processor registers and flags, click **Registers** (located within the **Monitors** menu) or click the **Registers** button  (located on the tool bar).

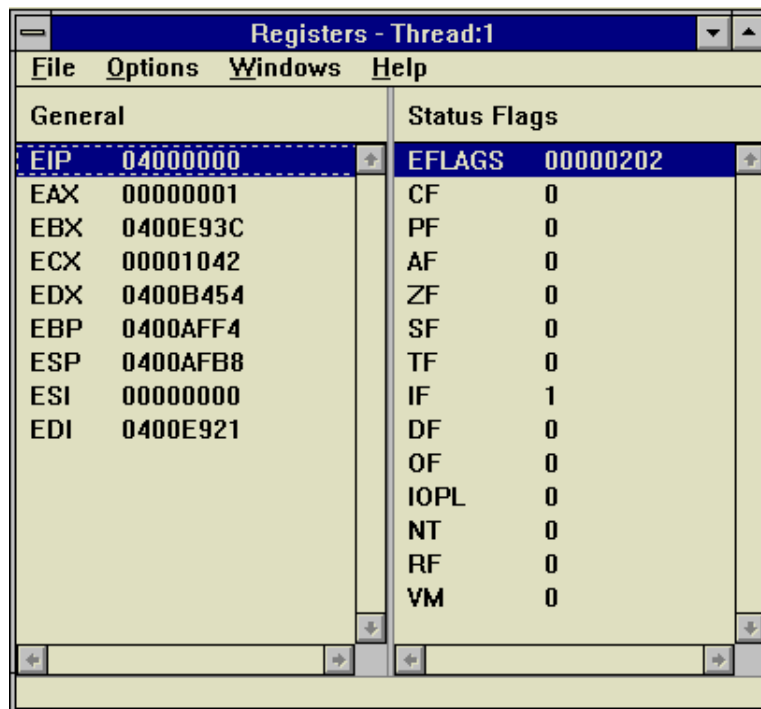


Figure 12. Registers Window

The contents of all of the registers except floating-point registers are displayed in hexadecimal. Registers that change after a run or a step are highlighted. To update a register, double-click the register and a multiple-line field is displayed. Type over the contents and press **Enter**. If you decide not to change the value, press **Esc**.

In the Registers window, floating-point registers are displayed as floating-point decimal numbers. They can be updated with a floating-point decimal number or with a hexadecimal string that represents a floating-point number.

Menus


From the menus of the Registers window you can:

- End the debugging session.
- Select the font to be displayed in the window, select the items you want displayed in the window, restore the defaults, and enable or disable the tool bar.
- Reset all your window settings to their original settings.
- Select whether you want hover help to be shown.
- Select to display the information area in the window.
- View a list of open windows and select any open window to display that window.
- Display help.

Note: If you need help with any of the menus, press F1 while the menu is selected.

Viewing Storage Contents and Addresses

The Storage window shows the storage contents and the address of the storage.

To display the Storage window, click **Storage** (located within the **Monitors** menu) or click the **Storage** button  (located on the tool bar).

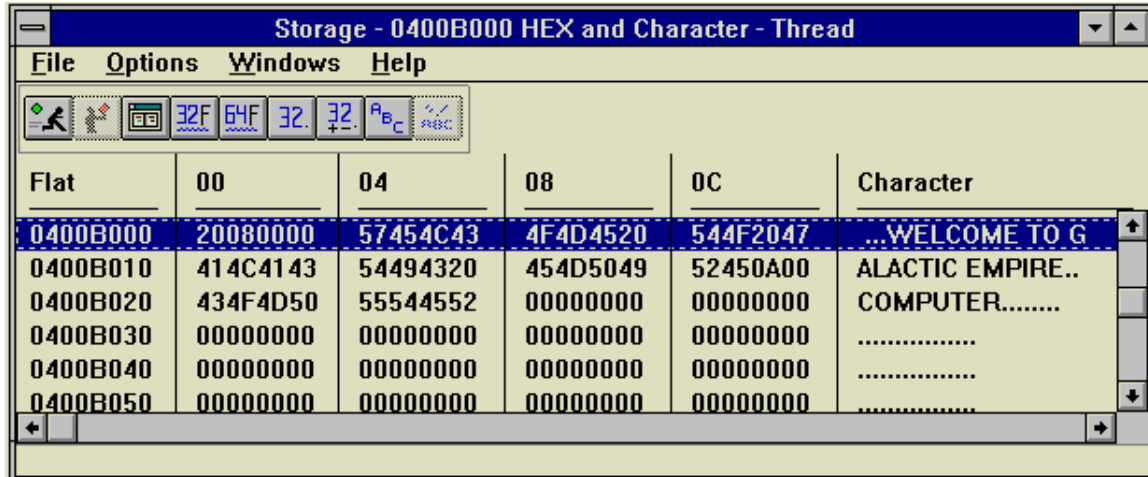


Figure 13. Storage Window

Multiple storage windows can display the same storage. When you run a program or update displayed data, the Storage window is updated to reflect the change.

To update the storage contents and all affected windows, double-click in the multiple-line field that is displayed. Type over the contents of the field. If you decide not to make the change, press **Esc**.

To specify a new address location, type over the address field in the Storage window. The window scrolls to the appropriate storage location.

Menus

From the menus of the Storage window you can:

- Save the contents of the Storage window in a file. Choose the file name by clicking **Options** and then **Display style**. Type the file name in the **Save file** field.
- End the debugging session.
- From the **Options** menu you can:
 1. Select the font to be displayed in the window.
 2. Select the items you want displayed in the window.
 3. Restore the defaults.
 4. Enable or disable the tool bar.
 5. Fill memory with a specific character or hexadecimal pattern.
 6. Identify the expression you want to monitor.

The expression evaluator used is based on the context. For example, if you display the Storage window by clicking **Monitor expression** (located within the **Monitors** menu), the evaluator used is based on the context in the Monitor Expression window. However, if you display the Storage window first and then click **Monitor expression** (located within the **Options** menu of the Storage window), the evaluator used is based on the context of the stopping thread.

Note: You cannot look at variables that have been defined using the #DEFINE preprocessor directive. If the variable is not in scope when the monitor is opened, the default address is displayed. If the variable goes out of scope, the address is changed to a hex constant.


If you select the **Enabled monitor** check box, the monitor updates the stop value of the program to the actual value in storage. However, a disabled monitor suspends this updating and reflects the stop value or the value held when the monitor was disabled.

- Reset all your window settings to their original settings.
- Enable or disable the tool bar.
- Select whether you want hover help to be shown.
- Select to display the information area in the window.
- View a list of open windows and select any open window to display that window.
- Display help.

Note: If you need help with any of the menus, press F1 while the menu is selected.

Monitoring Local Variables

You can monitor and change the local variables (static, automatic, and parameter) for the current execution point in the program from the Local Variables window. The contents of the Local Variables window change each time your program enters or leaves a function.

To display the Local Variables window, click **Local variables** (located within the **Monitors** menu) or select the Monitor Expression button  (located on the tool bar).

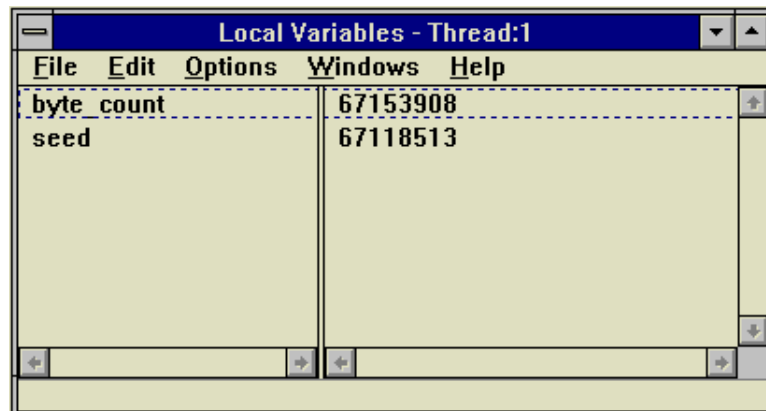


Figure 14. Local Variables Window

Modifying Variables

To modify the value of a variable, double-click the value. When the information area is displayed, type the new value and press **Enter**.

Menus

From the menus of the Local Variables window you can:

- End the debugging session.
- Delete, select, deselect, show other elements, or change representation of the variables. You can copy the selected local variable data to the clipboard. You can also save the Local Variables window contents in a file. Select **Options, Debugger settings**, and then **Monitor properties** from the Debug Session Control window or any of the source windows and enter the file name in the **Save file** field.
- Control how the contents of variables display and set debugger options.
- View a list of open windows and select any open window to display that window.
- Display help.

Note: If you need help with any of the menus, press F1 while the menu is selected.

Monitoring Other Variables and Expressions

The debugger has four other windows that enable you to monitor and change variables and expressions. These windows are as follows:

- Popup Monitor
- Program Monitor
- Private Monitor
- Storage Monitor

A Popup Monitor window monitors single variables or expressions. This window is associated with a specific source window and closes when the associated window closes.

The Program Monitor, Private Monitor, and Storage Monitor windows are used as collectors for individual variables or expressions that might be of interest to you.

The difference between the Private Monitor window and the Program Monitor window is the length of time that each remains open. The Program Monitor window remains open for the entire debugging session. The Private Monitor window is associated with the source window from which it was opened and closes when its associated view is closed.

Modifying Variables

To modify the value of a variable, double-click the value. When the information area is displayed, type the new value and press **Enter**.

Expressions Supported

This section describes the expression language supported by the debugger, which is a subset of C. This includes the operands, operators, and data types.

Note: You can display and update bit fields for C code only. You cannot look at variables that have been defined using the #DEFINE preprocessor directive.

Supported Expression Operands

You can monitor an expression that uses the following types of operands only:

Operand	Definition
Variable	A variable used in your program.
Constant	The constant can be one of the following types: <ul style="list-style-type: none">Fixed or floating-point constant. Note: The largest floating-point constant is 1.8E308. The smallest floating-point is 2.23E-308.A string constant, enclosed in quotation marks (" ")A character constant, enclosed in single quote marks (' ')
Registers	In the case of conflicting names, the program variable names take precedence over the register names. For conversions that are done automatically when the registers display in mixed-mode expressions, general purpose registers are treated as unsigned arithmetic items with a length appropriate to the register.

If you monitor an enumerated variable, a comment displays to the right of the value. If the value of the variable matches one of the enumerated types, the comment contains the name of the first enumerated type that matches the value of the variable. If the length of the enumerated name does not fit in the monitor, the contents display as an empty entry field.

The comment (empty or not) lets you distinguish between a valid enumerated value and an invalid value. An invalid value does not have a comment to the right of the value.

You *cannot* update an enumerated variable by entering an enumerated type. You must enter a value or expression. If the value is a valid enumerated value, the comment to the right of the value updates.

Bit fields are supported for C compiled code only. You can display and update bit fields, but you cannot use them in expressions. You cannot look at variables that have been defined using the #DEFINE preprocessor directive.

Supported Expression Operators

You can monitor an expression that uses the following operators only:

Operator	Coded as
Subscripting	a[b]
Member selection	a.b or a->b

<i>Table 1 (Page 2 of 2). Supported Expression Operators</i>	
Operator	Coded as
Size	<i>sizeof a</i> or <i>sizeof (type)</i>
Logical not	<i>!a</i>
One's complement	<i>~a</i>
Unary minus	<i>-a</i>
Unary plus	<i>+a</i>
Dereference	
Type cast	<i>(type) a</i>
Multiply	<i>a * b</i>
Divide	<i>a / b</i>
Modulo	<i>a % b</i>
Add	<i>a + b</i>
Subtract	<i>a - b</i>
Left shift	<i>a << b</i>
Right shift	<i>a >> b</i>
Less than	<i>a < b</i>
Greater than	<i>a > b</i>
Less than or equal to	<i>a <= b</i>
Greater than or equal to	<i>a >= b</i>
Equal	<i>a == b</i>
Not equal	<i>a != b</i>
Bitwise AND	<i>a & b</i>
Bitwise OR	<i>a b</i>
Bitwise exclusive OR	<i>a ^ b</i>
Logical AND	<i>a && b</i>
Logical OR	<i>a b</i>

Supported Data Types

You can monitor an expression that uses the following typecasting operations:

- 8-bit signed byte
- 8-bit unsigned byte
- 16-bit signed integer
- 16-bit unsigned integer
- 32-bit signed integer
- 32-bit unsigned integer
- 32-bit floating-point
- 64-bit floating-point
- 128-bit floating-point
- Pointers
- User-defined types

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, 10594, USA.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Copying and Distributing Softcopy Files

For online versions of this book, we authorize you to:

- Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.
- Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM

Intel is a registered trademark of the Intel Corporation.

Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

B

- breakpoints, setting 15
- breakpoints, viewing a list 18

C

- components pane view, saving the contents 15

D

- data types, valid 34
- Debug Session Control window 13
- debug session, ending 11
- debug session, starting 8
- debugger properties, setting 19
- demonstration session 7

E

- ending the debugging session 11
- enumerated variable 33
- environment variables 3
- executing a program 26
- execution point, locating 14
- expressions supported 33
 - operands 33
 - operators 33
 - supported data types 34

F

- finding source files 6

G

- getting started 7
 - demonstration session 7
 - setting up the coprocessor 7
 - setting up the host computer 7

H

- hardware requirements 3
- helpful tips and hints 10
- host computer, setting up 7

I

- ICAT 3
- installation 3
- interactive code analysis tool (ICAT) 3
- introducing the debugger 3

L

- Launch or attach window 8, 15
- limitations 6
- local variables, monitoring 31
- locating the execution point 14

M

- monitor properties, setting 22
- monitoring local variables 31
- monitoring other variables and expressions 32
- monitors windows 27
 - Call stack window 27
 - Local Variables window 31
 - Registers window 28
 - Storage window 29

N

- new source file, opening 14

O

- opening a new source file 14
- opening a source file to a function 14
- operands, valid expression 33
- operators, valid expression 33

P

- program, executing 26

R

- restrictions 6

S

- saving the contents of the components pane view 15

- saving the contents of the threads pane view 14
- setting breakpoints 15
- setting debugger properties 19
- setting monitor properties 22
- setting up the coprocessor 7
- setting up the host computer 7
- source file, opening to a function 14
- source, viewing 23
- starting a debug session 8

T

- threads pane view, saving the contents 14
- tool buttons 9
- troubleshooting 10
- typecasting, data types in monitors 34

V

- variables and expressions, monitoring other 32
- variables, environment 3
- viewing a list of breakpoints 18
- viewing active functions for a particular thread 27
- viewing registers for a particular thread 28
- viewing storage contents and addresses 29
- viewing your source 23