

ICAT Debugger
JavaOS for Business - OS/2 Warp 4

Contents

| | |
|--|----|
| About This Book | 1 |
| Introducing the ICAT Debugger | 3 |
| Before You Begin | 3 |
| Minimum Hardware Requirements | 3 |
| Minimum Software Requirements | 3 |
| Environment Variables | 4 |
| Search Order of Source Files and Modules | 6 |
| Limitations | 6 |
| Getting Started | 7 |
| Setting Up the Target Network Computer | 7 |
| Setting Up the Host OS/2 Warp Version 4 Computer | 7 |
| Demonstration Session | 7 |
| Starting a Debug Session | 10 |
| Helpful Tips and Hints | 11 |
| Troubleshooting | 11 |
| Tool Buttons | 11 |
| Using the Drag-and-Drop Function | 13 |
| Ending the Debugging Session | 13 |
| Main Debugging Windows | 15 |
| Debug Session Control Window | 15 |
| Opening a New Source File | 16 |
| Opening a Source File to a Function | 16 |
| Locating the Execution Point | 16 |
| Saving the Contents of the Thread Pane View | 16 |
| Saving the Contents of the Components Pane View | 17 |
| Setting Breakpoints | 17 |
| Setting a Line Breakpoint | 17 |
| Setting a Function Breakpoint | 18 |
| Setting an Address Breakpoint | 19 |
| Setting a Watchpoint | 19 |
| Setting a Load Occurrence Breakpoint | 20 |
| Viewing a List of Breakpoints | 20 |
| Setting Debugger Properties | 21 |
| Remote Page | 21 |
| Source Page | 23 |
| Modules Page | 24 |
| Setting Monitor Properties | 25 |
| Viewing Your Source | 27 |
| Source Window | 27 |
| Drag-and-Drop Function | 27 |
| Disassembly Window | 28 |
| Mixed Window | 28 |
| Executing a Program | 30 |
| Monitors Windows | 31 |
| Viewing Active Functions for a Particular Thread | 31 |
| Menus | 31 |

| | |
|--|-----------|
| Viewing Registers for a Particular Thread | 32 |
| Menus | 32 |
| Drag-and-Drop Function | 33 |
| Viewing Storage Contents and Addresses | 33 |
| Menus | 33 |
| Drag-and-Drop Function | 34 |
| Monitoring Local Variables | 34 |
| Menus | 35 |
| Viewing Messages | 35 |
| Menus | 36 |
| Monitoring Other Variables and Expressions | 36 |
| Drag-and-Drop Function | 37 |
| Hardware Monitor | 37 |
| Menus | 38 |
| Trace Dump Window | 38 |
| Menus | 38 |
| Expressions Supported | 39 |
| Supported Expression Operands | 39 |
| Supported Expression Operators | 39 |
| Supported Data Types | 40 |
| C/C++ | 40 |
| Java | 41 |
| Notices | 43 |
| First Edition (May, 1998) | 43 |
| Copyright Notices | 43 |
| Disclaimers | 43 |

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 OS/2 Warp 4-hosted Interactive Code Analysis Tool (ICAT) for the JavaOS™ for Business™ provides debugging of the JavaOS for Business microkernel, its Java Virtual Machine (JVM), the JVM Java applications, and their (potential) native-method calls (C or C++ code collected in load libraries). It is designed to work with the Sun/IBM JavaOS for Business for network computers and can be run on Intel® and PowerPC 603 target platforms.

The ICAT Debugger (hereafter referred to in this document as the debugger) is a source-level debugger that runs on an OS/2 Warp 4 system. It is an OS/2 Warp 4 application; that is, it will use Presentation Manager (PM) for its presentation space, and its debug engine will run outside of a JVM.

The debugger uses a JavaOS for Business debug probe to manipulate the target environment, and the debugger uses a null-modem cable to communicate with this debug probe. Note that this design allows for remote debugging; that is, the target environment and the JavaOS for Business debug probe will run on a separate network computer from the host computer of the debugger itself. The result is that a user can attach to the target JavaOS for Business with the debugger.

The debugger presents various windows (Source, Storage, Call Stack, variable monitors, and so on) with PM to reflect the current state of the debuggee. The debugger enables you to step, run, catch breakpoints, and so on.

The supported debug file formats for the native-method load libraries (that you may choose to debug) and the microkernel include DWARF1 and STABS (Cygnus Corporation).

Before You Begin

This section lists the hardware and software requirements, environment variables, and the search order of source files and modules.

Minimum Hardware Requirements

- Intel® 486SX processor
- 11 MB hard disk space
- A null-modem cable connected to serial ports on both the host OS/2 Warp 4 and target machine is required.

Minimum Software Requirements

There are software requirements for both the target network computer where the applications are debugged and run and the host computer where the debugger runs.

Target Computer

- JavaOS for Business image
- Debugger probe

OS/2 Warp 4 Host Computer

- ICATJOS.ZIP

Environment Variables

The debugger uses environment variables to manage debugging sessions and remote communication. Therefore, you should set your environment variables before you attempt to debug your application. To set the environment variables, edit the SETICAT.CMD file. Use caution when setting your environment variables; they affect communication with the debug probe, location of your Java application, and other files needed to run the debugger. Following is a list of the variables and a description of each:

CAT_NATIVE_ONLY

This variable tells the daemon that you do *not* want to debug Java applications. This allows the debug probe to ignore Java classes and enhances performance considerably. If you leave this variable as NULL, you can debug your Java applications.

```
SET CAT_NATIVE_ONLY=
```

CAT_MACHINE

Specifies which host COM port the debugger uses to communicate with the debug probe and the baud rate for communication. This variable has the following form:

```
COMx:nnnn
```

where *x* identifies the port (for example, 0 for COM0) and *nnnn* specifies the baud rate. The following rates are supported:

- 9600 (the default)
- 19200
- 38400
- 57600
- 115200

For speeds above 19200, you need buffered UARTs on both the host and target computers.

For example, type the following at the command prompt:

```
SET CAT_MACHINE=COM2:57600
```

CAT_HOST_BIN_PATH

This variable tells the debugger where to find your debug binaries (the JavaOS for Business load image, .class files, and load libraries with debug information) on your host system. If you want to see local Java variables, you *must* compile with the -g flag (for example, javac -g mycode.java).

```
SET CAT_HOST_BIN_PATH=i:\javatest
```

CAT_COMMUNICATION_TYPE

Allows asynchronous communication. This must be set to ASYNC_SIGBRK.

For example, type the following at the command prompt:

```
SET CAT_COMMUNICATION_TYPE=ASYNC_SIGBRK
```

CAT_HOST_SOURCE_PATH

This variable tells the debugger where to find your source (for example, Main.java and find.c).

```
SET CAT_HOST_SOURCE_PATH=i:\javatest
```

CAT_MODULE_LIST

This variable enables the jprobe *not* to report any modules whose names do not pattern match any of the names/substring of names in the set. Since the debugger takes time processing each module, it is often a very good idea for performance to supply this variable.

You can separate the various modules with either a ; or a blank space. JAVAOS.DEBUG is *always* reported independent of this variable.

```
SET CAT_MODULE_LIST=String Main Object Thread
```

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:\javatest variable, the debugger searches the javatest subdirectory and all subdirectories below javatest 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
```

CAT_PACKAGE_PATH

The CAT_PACKAGE_PATH environment variable is used to search for both Java source and Java class files. The debugger uses CAT_PACKAGE_PATH the same way that a JVM uses the CLASSPATH environment variable to find Java class files. Both source files and class files are searched according to the package qualification of the associated class. In addition to containing subdirectories, this environment variable may also include names of zip files in which to search for Java class files.

This environment variable is most useful when you are debugging classes that are contained within a package. If you are using this environment variable to find all of your Java class and source files, then set CAT_HOST_BIN_PATH and CAT_HOST_SOURCE_PATH to find your native-method binaries and source, respectively, or set them to null when you are debugging only Java.

CAT_OVERRIDE

Specifies a path that the debugger searches first to find the source files that were used to build your debug binaries. See “Search Order of Source Files and Modules” 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 5 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_NUMBEROFELEMENTS

CAT_DEBUG_NUMBEROFELEMENTS is an environment variable that is set to an integer, *n*. This integer 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.

For example, type the following at the command prompt:

```
SET CAT_DEBUG_NUMBEROFELEMENTS=100
```

The next 100 elements are displayed.

Search Order of Source Files and Modules

The debugger searches for source files in the following order:

1. CAT_OVERRIDE environment variable
2. CAT_PACKAGE_PATH environment variable using package qualification (only if source file has a .java extension)
3. CAT_HOST_SOURCE_PATH environment variable, descending subdirectories if the CAT_RECURSE_PATH environment variable is set
4. Current subdirectory
5. INCLUDE environment variable (only if the source file does *not* have a .java extension)
6. Last specified subdirectory from a change source file

The debugger searches for Java modules in the following order:

1. CAT_PACKAGE_PATH environment variable using package qualification
2. CAT_HOST_BIN_PATH environment variable, descending subdirectories if the CAT_RECURSE_PATH environment variable is set. (Note: Directories are searched with the unqualified name of the class; zip files are searched with the package-qualified name of the class).
3. Current subdirectory for the unqualified name of the class

The debugger searches for native modules in the following order:

1. CAT_HOST_BIN_PATH environment variable, descending subdirectories if the CAT_RECURSE_PATH environment variable is set.
2. Current subdirectory

Limitations

Refer to README.TXT file provided with the debugger for the current JavaOS for Business and debugger restrictions.

Getting Started

This section describes how to set up the target and host computers, start a debugging session from the OS/2 Warp 4 command prompt, and end a debugging session.

Setting Up the Target Network Computer

To set up the target network computer:

1. Make certain the target computer has a null-modem cable connected for serial port communication.
2. Boot the JavaOS for Business image on your target network computer.

The JavaOS for Business image contains a debugger probe that will be activated when you attach your target computer to the computer on which the debugger is running.

Setting Up the Host OS/2 Warp Version 4 Computer

To set up the host OS/2 Warp 4 computer:

1. Unzip the ICATJOS.ZIP file into the directory where you want to run the debugger. This file contains the debugger and a command file.
2. Set the environment variables. See "Environment Variables" on page 4 for information about setting your environment variables.

Demonstration Session

1. Set up the environment variables needed to run the debugger. For this demonstration session, the following settings were used:

```
CAT_MACHINE=COM1:38400
CAT_SETUP_RATE=9600
CAT_COMMUNICATION_TYPE=ASYNC
CAT_HOST_BIN_PATH=z:\javaos\native\bin
CAT_HOST_SOURCE_PATH=z:\javaos\native\source;z:\javaos\java\source
CAT_MODULE_LIST=JavaExprTest
CAT_PACKAGE_PATH=z:\javaos\java\bin\ROMClasses.zip
CAT_STACK_REFRESH=ON
CAT_RESUME=
CAT_NATIVE_ONLY=
CAT_PATH_RECURSE=
```

Note: See "Environment Variables" on page 4 for a description of each environment variable and how it affects your debug session.

2. Run *icatjos*, and wait for the **Initialization** dialog box to be displayed. Click the **Attach** button. After a few moments, your native code is displayed.

Steps 3 through 5 demonstrate native code debugging, and steps 6 through 12 demonstrate Java code debugging.

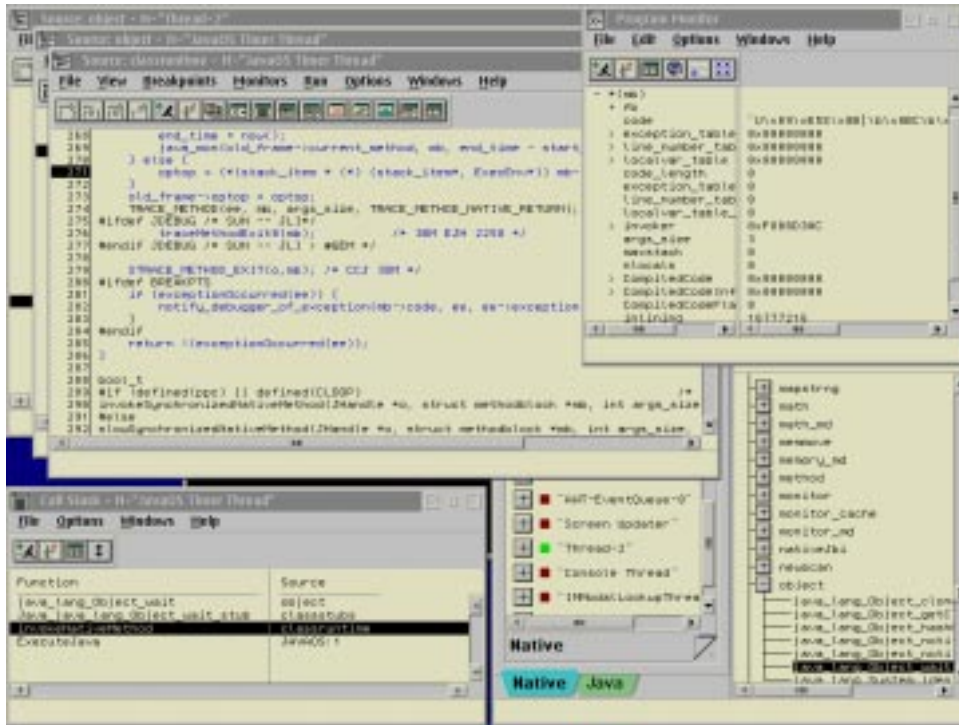
3. Select JAVAOS.DEBUG from the Debug Session Control component pane. You can show all of the compiled units of native code in JavaOS for Business by clicking the "+" located next to the JAVAOS.DEBUG component. Scroll through the compiled units until you see "object.". Now click the "+" located next to the "object" compiled unit. Then double-click **java_lang_Object_wait**. Your source

(Object.c) is displayed at the first line of code in the subroutine java_lang_Object_wait(). Set a breakpoint on line 74 by double-clicking the prefix area in the source window. The prefix area changes to red indicating that a breakpoint has been set. You can click the green light on the title bar, use the "R" key, or select **Run** from the **Run** pull-down menu to run the JavaOS for Business system. Choose one of these options to resume the system. You will hit this breakpoint fairly quickly.

4. Select the **Call stack** choice from the **Monitors** pull-down menu. You will see the calling sequence that led to the java_lang_Object_wait() subroutine. Double-click the classruntime.c entry, and its source is displayed within the invokeNativeMethod() subroutine.

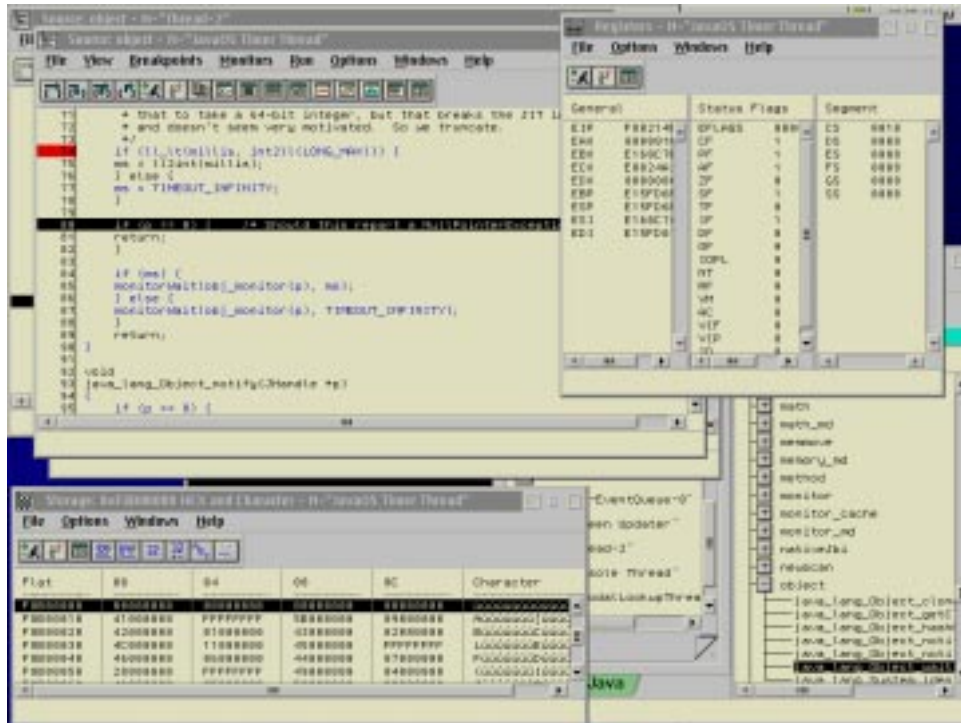
You can view variables that are on the stack from the calling subroutines. In this case, you can view the mb variable from the invokeNativeMethod() subroutine. Double-click the mb variable on line 276. A Program Monitor is displayed containing the mb variable. Double-click the ">" located next to mb in the Program Monitor window to dereference it. You will see the members of the structure that mb points to.

The following illustration shows all of our activity so far.



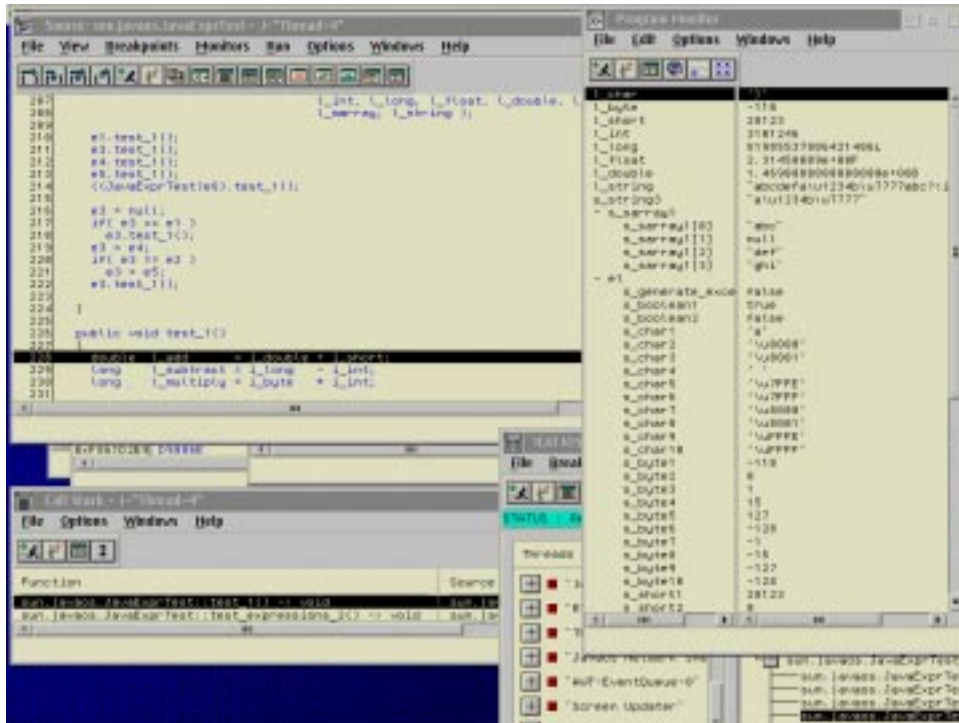
5. Press **Ctrl-p** from the source view. This will always take you back to the current line of execution. Click the toolbar **Step Over** icon and notice that you have stepped over a source line in java_lang_Object_wait(). Now select the **Storage** choice from the **Monitors** pull-down menu. Additionally, select the **Registers** choice from the **Monitors** pull-down menu. Close the **Program Monitor** and the **Call Stack** windows.

The following illustration shows the results of this activity.



Now let's move on to Java code debugging.

6. Clear the breakpoint at line 74 in Object.c.
7. Close the **Storage** and **Registers** windows and the "object" compiled unit in the **Debug Session Control** window. Now click the "+" located next to the module "JavaExprTest.class" in the Debug Session Control component pane. Double-click the main() method, and you are in the source for JavaExprTest.java.
8. Scroll up and set a breakpoint in the Java source at line 197. Run until you hit that breakpoint. You will need to issue the Shell command "etest" from a Telnet session to the target JavaOS for Business to cause the class to be executed.
9. Double-click the local variables listed on lines 178 through 192 (l_boolean, l_char, l_byte, and so on). Notice the support for Java primitive variables in the **Program Monitor** window.
10. Double-click the class variables listed on line 195 (s_string1, s_string3, and s_sarray1). Notice the support for Java strings and arrays in the **Program Monitor** window.
11. Set a breakpoint at line 210. Run until you hit that breakpoint. Double-click on the class variable e1 listed on line 210. Notice the support for Java class variables in the **Program Monitor** window.
12. Step into the call to e1.test_1(). Once in the test_1 method, bring up the **Call Stack** window again. The following illustration shows the results of this activity.



This concludes the demonstration session.

Starting a Debug Session

To start the debugger:

1. From the OS/2 Warp 4 command prompt, type the command `icatjws` followed by one of the following parameters:

/p+ Use program profile information.

/p- Do not use any program profile information.

The Initialization window is displayed.

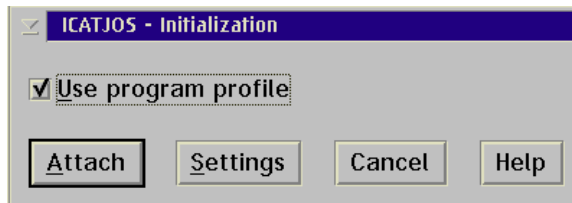


Figure 1. Initialization Window

Note: It's usually a good idea to have set up your environment variables by way of a SETICAT.CMD file before invoking the debugger. See "Environment Variables" on page 4 for more information.

2. If you're going to debug a program more than once, select the **Use program profile** check box to reactivate the windows and breakpoints.
3. Click the **Attach** button to attach the debugger to JavaOS for Business.

4. Click the **Settings** button to display 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 21 for more information.
5. Click **OK**. The Debug Session Control window opens displaying the threads and components of your source.

Helpful Tips and Hints

The following tips and hints might be helpful:

- You must have OS/2 Warp 4 on your host and your target network computer must be an Intel® or PowerPC platform.
- Type, in a command file, any environment variables that you want set. For example, you could type them in the SETICAT.CMD file or create your own command file.
- Any debug stripper utility can be used to strip debug information from native binaries placed on the target computer but should not be used on native binaries placed in the CAT_HOST_BIN_PATH.
- 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

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

- If the debugger can't attach to the target application:
 - Ensure that the JavaOS for Business probe has been booted with the JavaOS for Business load image on the target network computer.
 - Ensure that your CAT_COMMUNICATION_TYPE environment variable is set to ASYNC_SIGBRK.
 - Ensure that you have a null-modem cable connected to the correct serial ports on both the host and target computers.
- If the debugger displays only 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 native binaries in the CAT_HOST_BIN_PATH were **not** processed by a debug stripper utility. Debug stripper utilities make the native binaries smaller by removing the debug information.
 - Ensure that your CAT_HOST_SOURCE_PATH or CAT_PACKAGE_PATH is set to reference your source files.

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 change from the Disassembly window to the Mixed window.



Monitor Expression enables you to specify 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.



Hardware Monitor displays the physical memory, JVM heap statistics, processor type, version, and build information.



Trace Dump displays the buffered trace information collected so far for a trace-enabled JavaOS for Business kernel.



Messages displays printf's emitted by the JavaOS for Business kernel (diagnostic) code.



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



64-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.

Using the Drag-and-Drop Function

Drag-and-drop is supported in the Storage, Registers, Private Monitor, and Program Monitor windows. Additionally, drag is supported from the Source window. When you press and hold the right mouse button then move the mouse, the drag is in operation. Move the mouse to the target location and release the mouse button to perform a drop.

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:

- Click **Yes** to end your debugging session.
- Click **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 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, 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, press F1 to view help for this item.

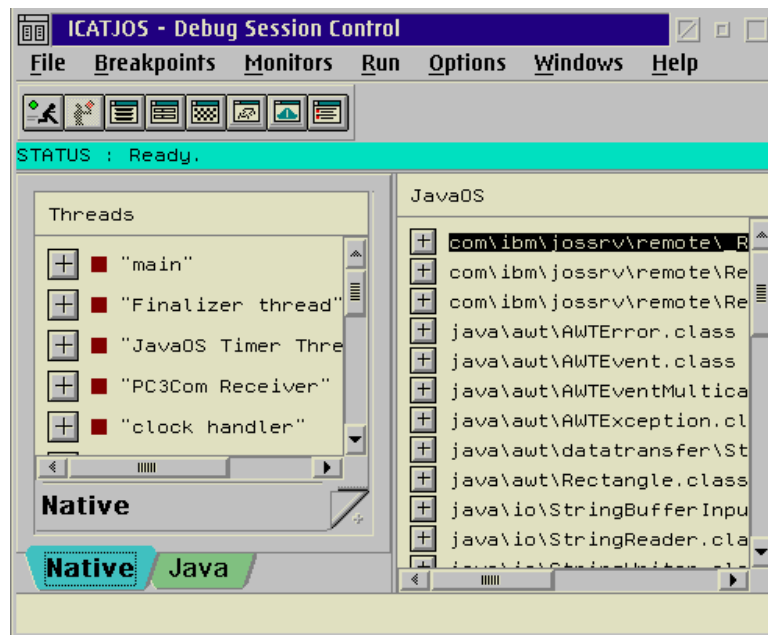


Figure 2. Debug Session Control Window

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 Component 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.
- Monitor the storage in your application.
- Display the local variables for your application's current function.
- Display captured printf's from applications running on the cryptographic adapter.
- 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 Thread 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, Window settings**, and then **Display style** located within the Debug Session Control window and type the file name in the Save file field.

Saving the Contents of the Components Pane View

If you would like to save the contents of the Component 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 Save file field.

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 turns 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, watchpoint, or set load occurrence 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**.

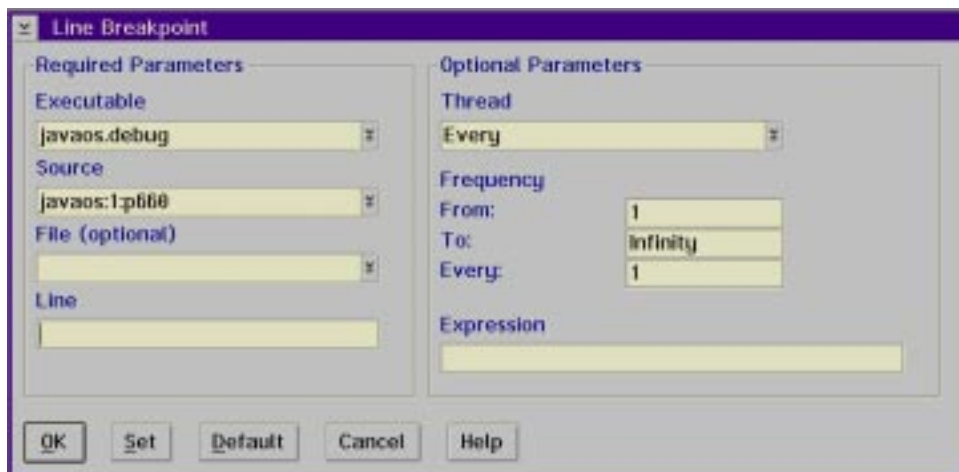


Figure 3. Line Breakpoint Window

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

Required Parameters:

1. Type a component name or click a component 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 located within the drop-down list in the Source field.
4. Select the source 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.

3. Type a number in the From field to activate the breakpoint the *n*th time the location is encountered.
4. Type a number in the To field to stop activating the breakpoint after the *n*th time the location is encountered.
5. 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.

6. If you are setting an address, function, or line breakpoint, you can also type in an expression. 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.

7. 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 a component name or click a component located within the drop-down list in the Executable (optional) field.
2. Select the executable file where you want to set the breakpoint.
3. Type the source name or click a source 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.

If a function is overloaded, a window is displayed with a list of all the overloaded function names. Click the appropriate function from the 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 18.

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 this field.

For example, to set an address breakpoint for the address `0x000A1FCC`, you would type one of the following in the Address field:

`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 18.

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

Setting a Watchpoint

A watchpoint stops the execution of your application when 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 port 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 `A1FCC`.

`A1FCC`

Type the following in the field to set a watchpoint for the expression `&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 18.

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

Note: The debugger will monitor 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.

Setting a Load Occurrence Breakpoint

A load occurrence breakpoint stops the execution of your application when a specific module is loaded.

You set a load occurrence breakpoint from the **Load Occurrence** window. To display this window, click **Set load occurrence** located within the **Breakpoints** menu.

To set a load occurrence breakpoint:

1. Type the name of the module in the Module Name field.

Execution stops when the module is loaded.

To set a load occurrence breakpoint when MY.CLASS is loaded, you would type one of the following in the **Module Name** field:

MY or MY.CLASS

Note: If the CAT_MODULE_LIST environment variable has been defined and the module's name is not contained in the CAT_MODULE_LIST, the module is not reported. If the module cannot be found in the module search path, the module name is not accepted. See "Setting Debugger Properties" on page 21 for information on identifying modules.

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 18.

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

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 **Breakpoint** menu of the Debug Session Control window.

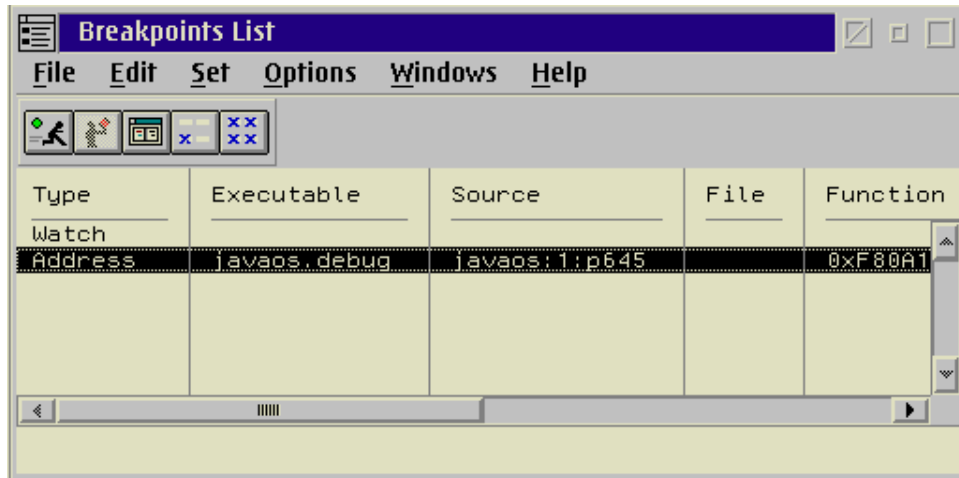


Figure 4. Breakpoints List Window

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, and load occurrence 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

Clicking **Debugger properties** located within the **Options** menu of the Debug Session Control window enables you to select how the threads and source files initially display. The Debugger Properties window contains three tabs:

- Remote
- Source
- Modules

Remote Page

The Remote page is displayed by default if you open the window using the Initialization window. If not, the Source page is displayed.

When you click the **Remote** tab, the following page is displayed:

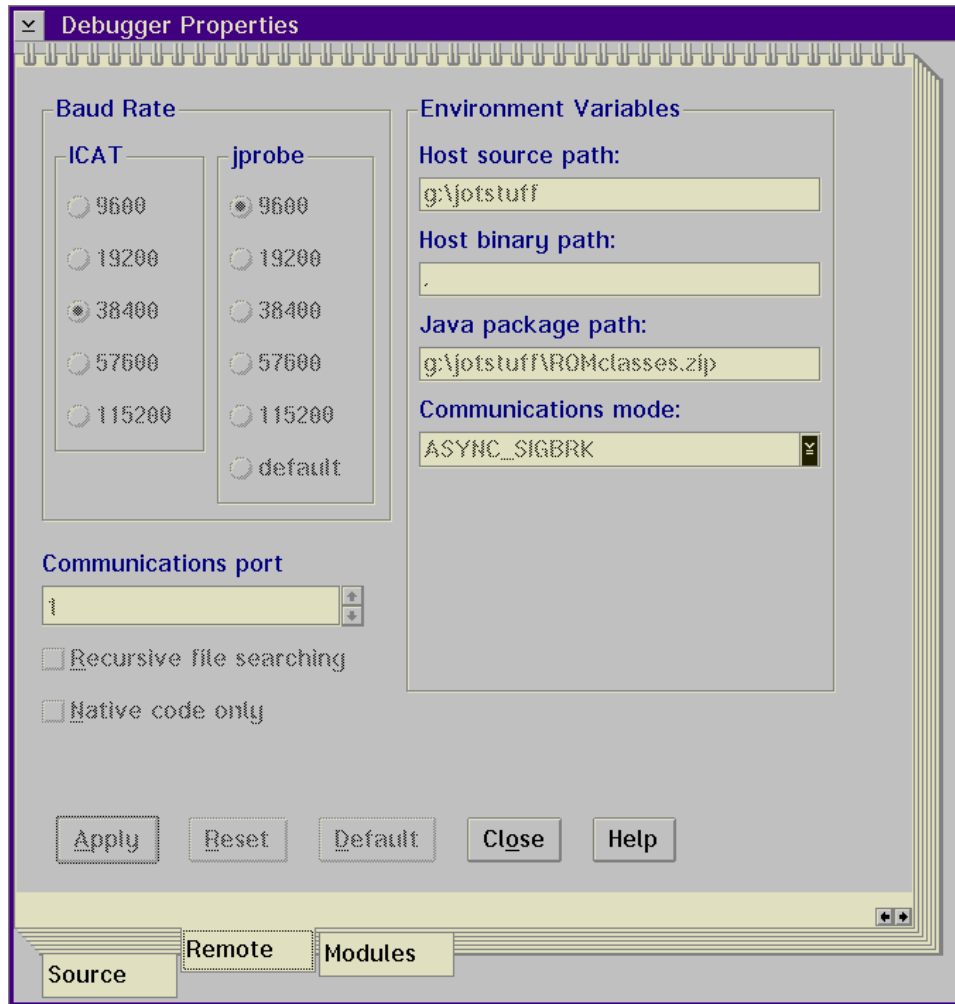


Figure 5. Debugger Properties Window - Remote Page

From the Remote page you can:

- Set the communication baud rate for the debugger.
- Set the initial baud rate for the JavaOS for Business probe.
- Set the communication port for the host.
- Set the path where the debugger finds the source.
- Set the path where the debugger finds the debug binary modules.
- Set the package path where the debugger finds class package and source files.
- Set the path where the debugger finds class files (especially zipped class files) and Java files.
- View the communication mode (ASYNC_SIGBRK is all that is currently available).
- Set the option for recursive subdirectory searching of the source and binary paths.
- Set the option for debugging only native code (no Java debugging).

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

To change your communication setting paths dynamically before communication is established with the target computer, adjust any of the **Environment Variables Group Heading** fields. See “Environment Variables” on page 4 for detailed information on environment variables.

These fields correspond respectively to the following environment variables:

- CAT_MACHINE

- CAT_HOST_SOURCE_PATH
- CAT_HOST_BIN_PATH
- CAT_PACKAGE_PATH
- CAT_COMMUNICATION_TYPE
- CAT_PATH_RECURSE
- CAT_NATIVE_ONLY

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

Selecting the Native code only check box tells the debugger to debug native files and to ignore class files (no Java debugging).

Source Page

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

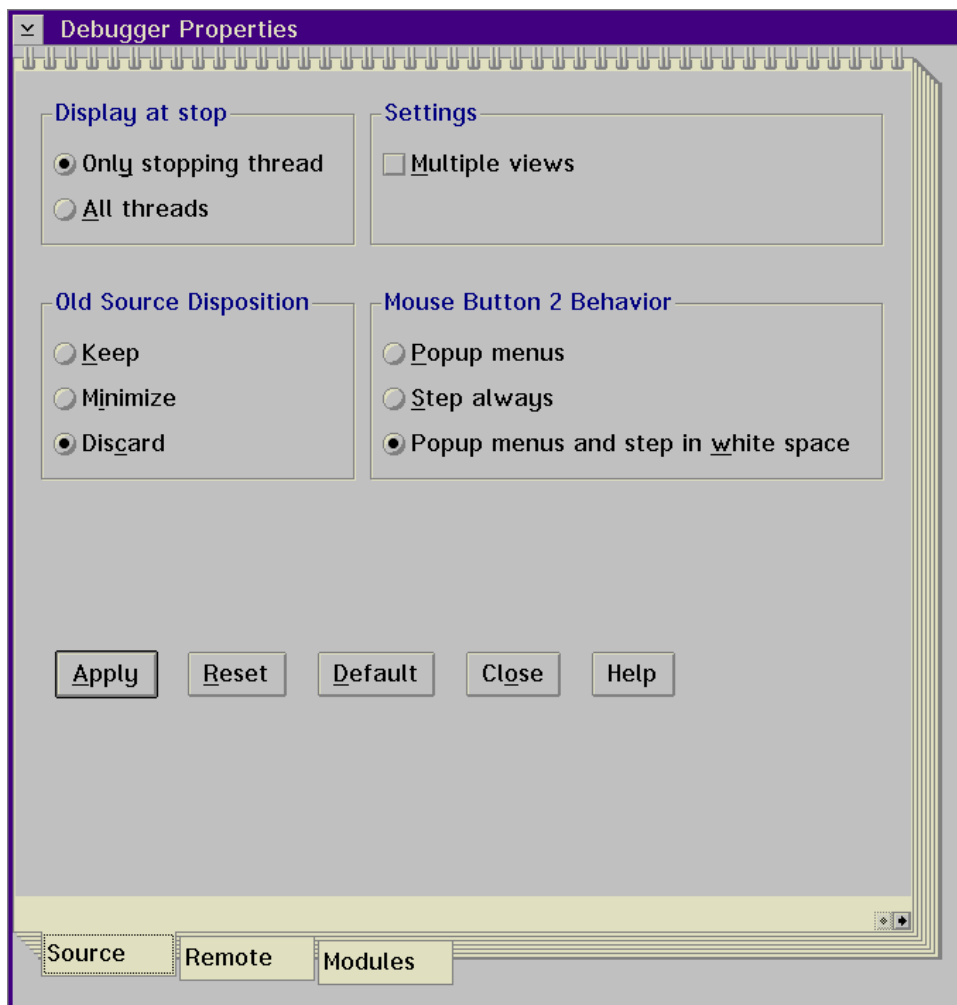


Figure 6. Debugger Properties Window - Source Page

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 the **Mouse Button 2 Behavior** group heading.

Modules Page

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



Figure 7. Debugger Properties Window - Modules Page

From this page you can add a module name to or delete a module name from the `CAT_MODULE_LIST` environment variable. See “Environment Variables” on page 4 for detailed information on this environment variable.

The Modules list box displays a list of the modules that the debugger obtains information about if or when they are loaded. If a module is loaded and it is not in the list, the debugger ignores the module.

To add a new module, type the name of the new module in the **New module** field and click the **Add** button.

You can delete a module or a group of modules, select the modules in the Modules list box that you would like to delete and click either the **Delete** button or the **Delete all** button.

Setting Monitor Properties

To select the settings for monitoring variables or expressions:

1. Click **Monitor properties** located on the Debug Session Control window. The Monitor Properties window is displayed.

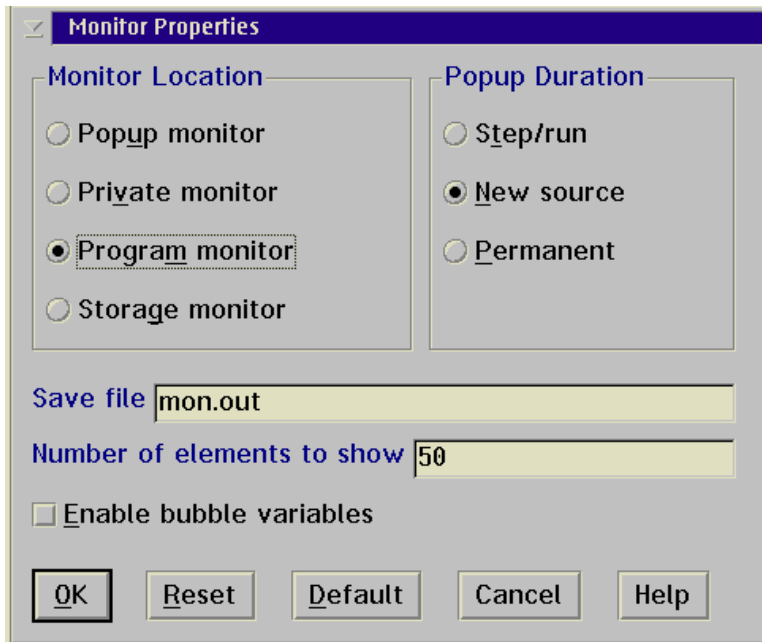


Figure 8. Monitor Properties Window

From this window you can set the following:

- The window into which the variable or expression being monitored is placed.
 - 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. The selections you can make, and the corresponding windows, are:
 - 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 or 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 non-executable 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.

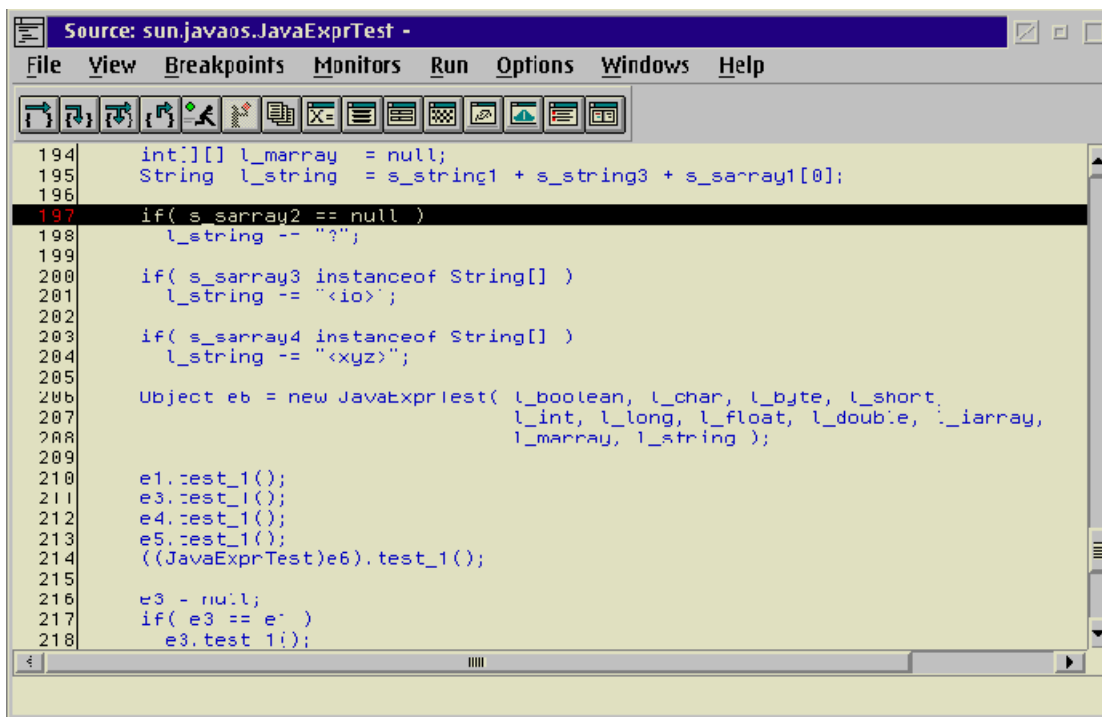


Figure 9. Source Window

Drag-and-Drop Function

The Source window can be used only to drag values. Drop is not supported in the Source window.

To drag an item in the Source window, press and hold the right mouse button on the string that you want to drag while moving the mouse. If you want to drag an extended line, select the line using the left mouse button and then press and hold the right mouse button to move the selected text.

When an item is dragged from the Source window, its context is dragged with it. However, only the monitors windows (Private and Program) use this context on the drop operation. So only values that have global context can be dropped from the Source window onto the Storage window or the Registers window.

See “Using the Drag-and-Drop Function” on page 13 for more information about the drag-and-drop function.

Disassembly Window

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

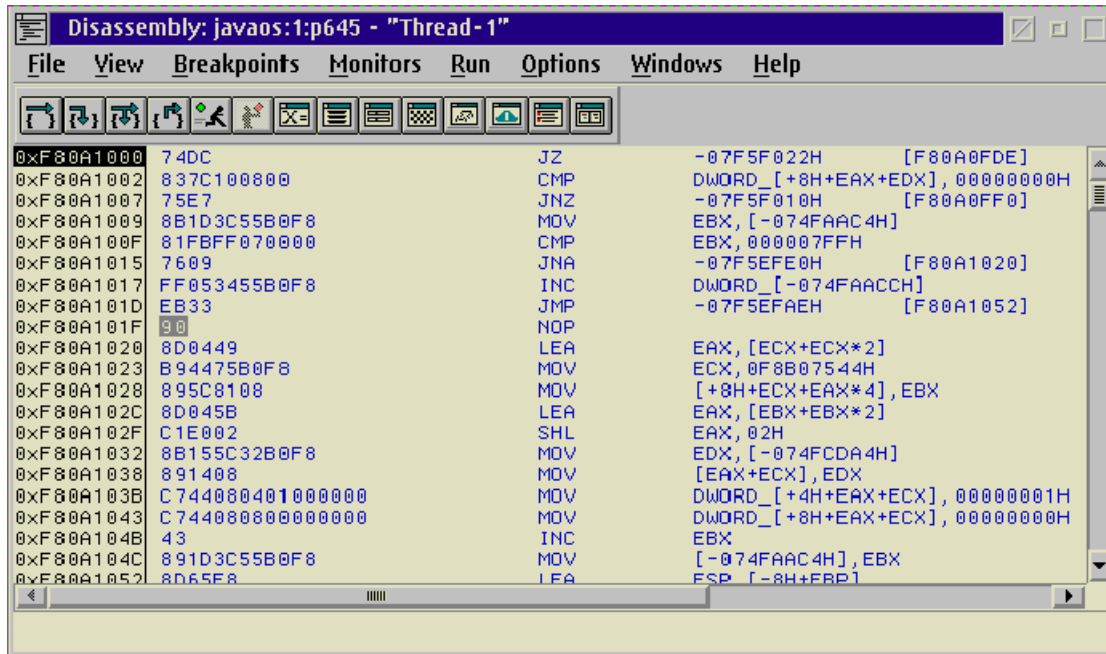


Figure 10. Disassembly Window

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.

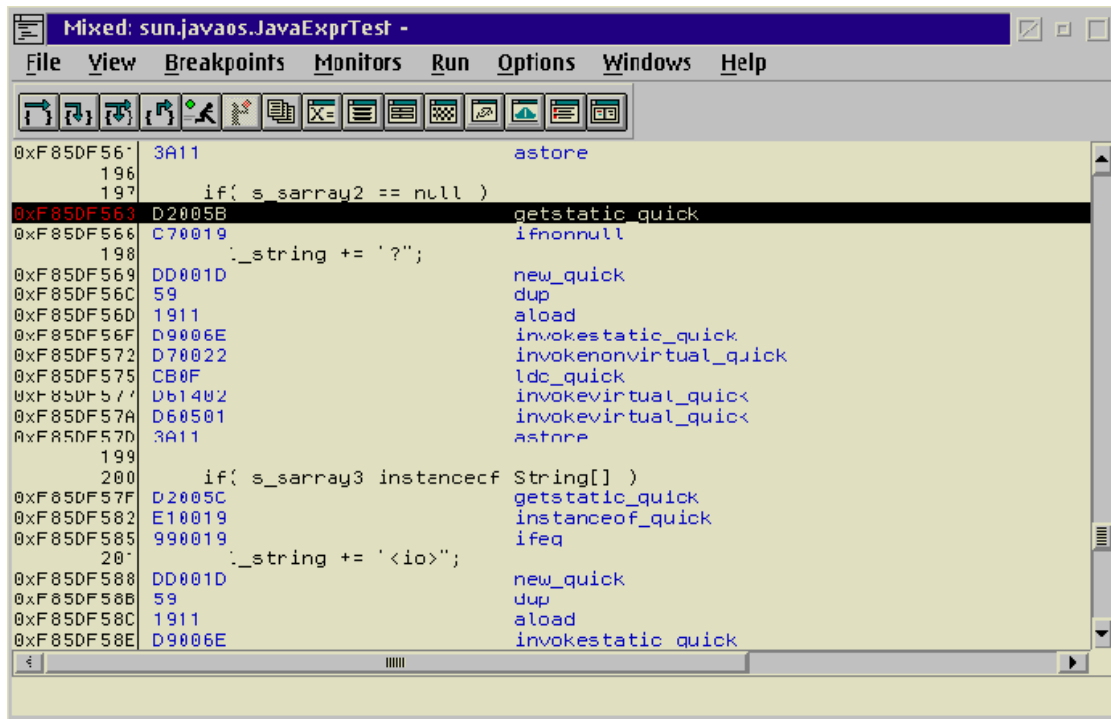


Figure 11. Mixed Window

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

- **File** menu—**Save to 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 in 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 in 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 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
- Other Monitor windows

These windows are also accessible from the tool bar of the source windows. See “Tool Buttons” on page 11 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.

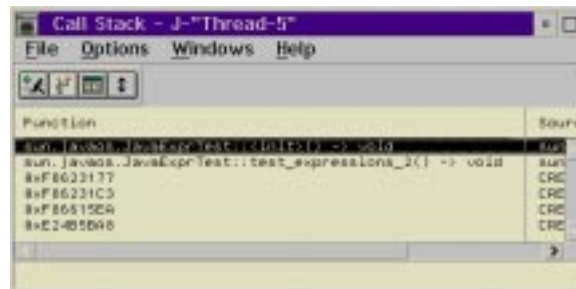


Figure 12. Call Stack Window

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.

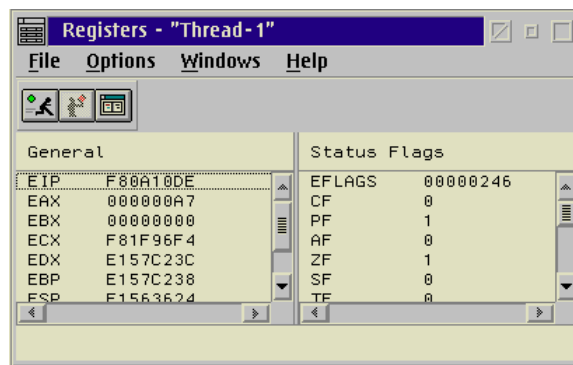


Figure 13. Registers Window

The contents of all of the registers except floating-point registers are displayed in hexadecimal. 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.

Note: The Registers window shown in Figure 13 displays PowerPC 603 registers. The debugger also supports the Intel® processor.

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.
- 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.

Drag-and-Drop Function

The Registers window can be used as a source or a target for drag-and-drop operations. All register values that are dragged and dropped are treated as hex numbers. This register will be updated with the value from the drag operation.

See “Using the Drag-and-Drop Function” on page 13 for more information about the drag-and-drop function.

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 14. 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:
 - Select the font to be displayed in the window.
 - Select the items you want displayed in the window.
 - Restore the defaults.
 - Enable or disable the tool bar.
 - Fill memory with a specific character or hexadecimal pattern.
 - 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.

Drag-and-Drop Function

The Storage window can be used as a source or a target for drag-and-drop operations. The item that is dragged is the item in the column under the mouse. You cannot drag a portion of the column. When items are dropped in the address portion of the Storage window, the Storage window adjusts to show memory at this new address. When items are dropped in the content fields of the Storage window, the item in the column under the mouse is updated. Columns that display hex values can have only hex values dropped on them. Columns that display strings can have only strings dropped on them.

See “Using the Drag-and-Drop Function” on page 13 for more information about the drag-and-drop function.

Monitoring Local Variables

You can monitor 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.

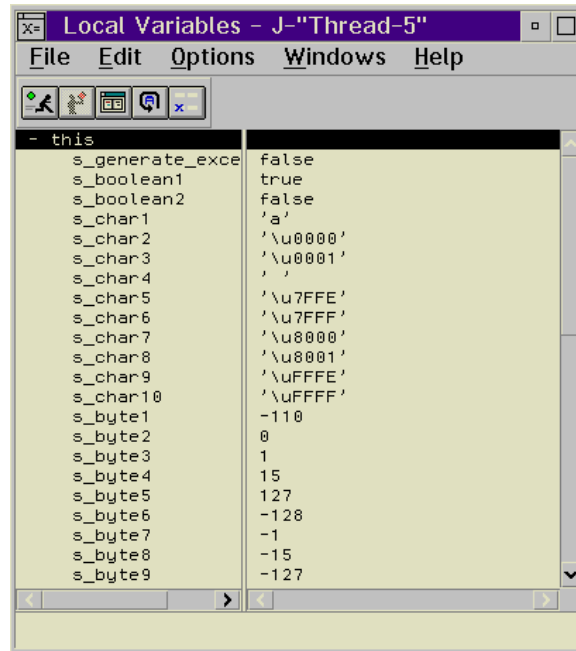


Figure 15. Local Variables Window

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.

Viewing Messages

Use the Messages window to view printf's that are emitted by the JavaOS for Business kernel (diagnostic) code. The contents of this window can be updated even while the debugger has control and your program is quiesced!

To display the Messages window, click **Messages** located within the **Monitors** menu or click the **Mes-**
sages button  located on the tool bar.

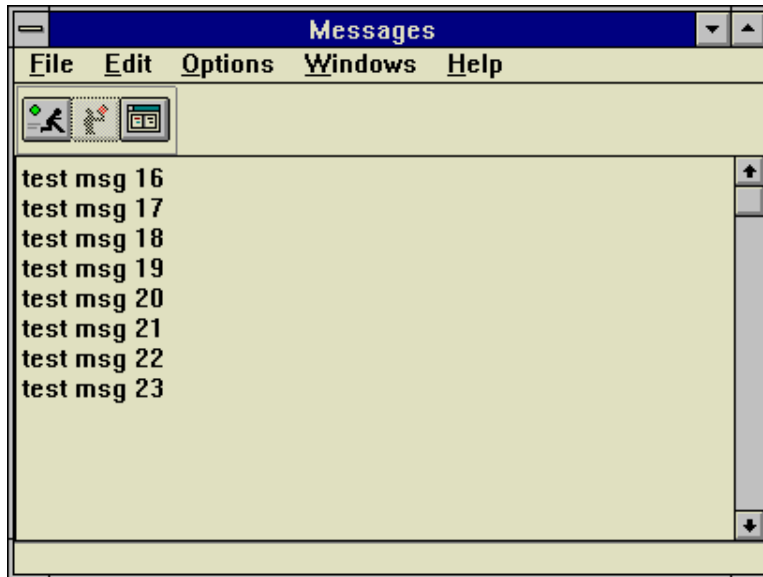


Figure 16. Messages Window

Menus

From the messages window you can:

- End the debugging session.
- Save the contents of the Message window in a file.

Choose the file name by clicking **Options** from the menu and then **Display Style**. Type the name of the file in the File Name field.

- Select the font used to display the information in the window.
- Clear the current collection of printfs.
- Enable or disable the tool bar.
- Select whether you want hover help to be shown.
- 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 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 in which you might be interested.

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.

Drag-and-Drop Function

Monitor windows can be used as a source or a target for the drag-and-drop operations. You can drag either the variable/expression or the value of a monitor.

Note: The context is not dragged with the monitor, so some variables/expressions might not be accepted in other windows. When an item is dropped on a monitor item, that monitor item is updated with the new value. When an item from the Source window is dragged onto a monitor in white space, that variable/expression is added to the monitor. Adding variables/expressions this way is supported only in the Private and Program monitors.

See “Using the Drag-and-Drop Function” on page 13 for more information about the drag-and-drop function.

Hardware Monitor

You can monitor the physical memory, JVM heap statistics, processor type, version, and build information of your target network computer’s hardware from the Hardware Monitor window.


To display the Hardware Monitor window, click **Hardware Monitor** located within the **Monitors** menu or click the **Hardware Monitor** button  located on the tool bar.



Figure 17. Hardware Monitor Window

Expressions Supported

This section describes the expression language supported by the debugger, which is a subset of C and Java. 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 in C/C++ is 1.8E308. The smallest floating-point is 2.23E-308.• A string constant, enclosed in quotation marks (" ").• A character constant, enclosed in single quotation 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 are displayed in mixed-mode expressions, general purpose registers are treated as unsigned arithmetic items with a length appropriate to the register. Note: Register expression operands are supported only in a C/C++ context. |

If you monitor an enumerated variable, a comment is displayed 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 are displayed as an empty field.

The comment (empty or not) lets you distinguish between a valid enumerated value and a value that is not valid. A value that is not valid 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 is updated.

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

Note: Expressions are evaluated in context. Some expressions are C/C++ specific; others are Java specific.

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

| <i>Table 1. Supported Expression Operators</i> | |
|--|---|
| Operator | Coded as |
| Subscripting | <i>a[b]</i> |
| Member selection | <i>a.b</i> or <i>a->b</i> (<i>a->b</i> for C/C++ only) |
| Size | <i>sizeof (a)</i> or <i>sizeof (type)</i> (C++ only) |
| Logical not | <i>!a</i> |
| One's complement | <i>~a</i> |
| Unary minus | <i>-a</i> |
| Unary plus | <i>+a</i> |
| Dereference | <i>*a</i> (C/C++ only) |
| 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> |
| Instance of | <i>a instanceof b</i> (Java only) |
| 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> |
| Unsigned Right Shift | <i>a > > b</i> (Java only) |

Supported Data Types

C/C++

You can monitor an expression that uses the following data types:

- 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 number
- 64-bit floating-point number
- 128-bit floating-point number

- Pointers
- User-defined types
- Structures
- Arrays
- Classes (C++ only)

Java

You can monitor an expression that uses the following data types:

- Boolean
- 8-bit signed byte
- 16-bit character
- 16-bit signed integer
- 32-bit signed integer
- 64-bit signed integer
- 32-bit floating-point number
- 64-bit floating-point number
- Null reference
- Object reference
- Array reference

Notices

First Edition (May, 1998)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication was developed for products and services offered in the United States of America. IBM may not offer the products, services, or features discussed in this document in other countries, and the information is subject to change without notice. Consult your local IBM representative for information on the products, services, and features available in your area.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

© Copyright International Business Machines Corporation 1998. 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.

Disclaimers

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 that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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:

IBM Director of Licensing

IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

IBM
OS/2
PowerPC
PowerPC 603

The following terms are trademarks of other companies:

Intel and Pentium are registered trademarks of the Intel Corporation.

Java, JavaOS, JavaOS for Business, Sun, Sun Microsystems and the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries, and are used under license. The JavaOS for Business technology is a result of the collaboration of Sun and IBM.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.