Clarion for Windows Debugger

This topic describes the commands available in the Debugger. Help for dialog boxes is also available; a help topic jump appears next to each command that leads to a dialog box. You'll also find a list of "how to" topics following the command list:

File Menu

File to Debug — Allows you to choose an executable file to load and debug. Choose a file from the Open File dialog. When you start the Debugger from the IDE, it automatically loads the executable. This command allows you to start the Debugger first, then load a file into a (see also) source code window.

Load Redirection — Allows you to change the Redirection file used for the current Debugger session. The Redirection file provides a list of directories to search for libraries, source code, and debug information files.

Debug Active DLL — Adds a related dynamic link library (*.DLL) to the debug session. Choose a file from the (see also) **Active Module** dialog. This option is available for hard mode debugging only.

Sleeper Mode — Sets the Debugger into sleep mode, in which it waits for a protection violation (GPF), CTRL+ALT+SYS REQ, or an INT3. This option is available for hard mode debugging only.

You can start the Debugger in sleep mode from the command line by adding `/s` to the command line.

Add Sources — Allows you to open additional source files for the current debug session.

Exit — Closes the Debugger, and terminates the Debuggee.

Restart — Continues a previously terminated Debug session. The Debugger will restart with previous watch expressions and break points, *provided* that the source code has not changed.

Edit Menu

Find Text — Opens the (see also) **Find Text** dialog, allowing you to search for text in a source code window.

Find Next — Repeats the last text search.

Find Procedure — Opens the (see also) **Find Procedure** dialog, which allows you to choose a procedure from the list, then locate the source code for the procedure in a source code window.

Goto Line — Opens the (see also) **Goto Line** dialog, which allows you to move the selection bar to the line number you specify.

Current Line — Moves the selection bar to the currently executing line in the source code window.

Find Last Error — Moves the selection bar to the source code line where the Debugger last detected an error. See also **Setup** dialog.

Breakpoints Opens the (see also) **Breakpoints** dialog, which allows you to set conditional or unconditional breakpoints. Breakpoints allow you to automatically halt execution at the line of code at which (or near which) you think the problem occurs. Your program runs up to the breakpoint, then halts and turns control back to the Debugger. You can then check the contents of variables and expressions to identify the cause of the problem

Edit Opens the (see also) **Edit Variable** dialog, which allows you to change the value of a variable while the Debuggee is suspended. You can then resume execution to test the program with the new value.

You must first select a variable in either the **Global Variables** window or the **Active Procedures** window.

! Menus

The **Go!**, **GoCursor!**, **Step!**, and **ProcStep!** commands execute your application while the Debugger monitors it in the background. They allow you to test your application in a controlled environment which helps you identify bugs faster.

Tip: These commands are all top level menu commands. No pulldown menus appear below them; just place the cursor on the menu command and click, or press ALT plus the underlined letter to execute.

Go! Runs the Debuggee from its current state to the next breakpoint. When a source or disassembly window is active, the G key executes the command.

GoCursor! Runs the Debuggee from its current state to the currently selected source or assembler line in the source code or disassembly window. When a source or disassembly window is active, the C key executes the command.

Step! Runs the Debuggee from the currently selected source or assembler line, one line of code at a time. When a source or disassembly window is active, the S key executes the command.

ProcStep! Runs the Debuggee from the currently selected source or assembler line to the next, but skips through procedure calls without stopping. When a source or disassembly window is active, the P key executes the command.

Options Menu

Soft Mode Toggles hard and soft mode debugging.

In *soft mode*, when the Debuggee is suspended in the Debugger, part of the Debugger will attempt to simulate the behavior of the Debuggee.

In *hard mode*, when the Debuggee is suspended in the Debugger, the only window to operate is the Debugger. All other activity is suspended. One consequence of this is that the desktop is not redrawn.

Clarion Soft Mode The Debugger will use part of the runtime library to simulate the behavior of the Debuggee. This is the recommended mode for most projects.

Extended Stack Trace Debugger shows information about procedures when no debug information is available. A disassembly window opens, containing the relevant segment.

Disassembly On — The (see also) **Disassembly** window "shadows" the active source window. When you select a line of source, the cursor in the disassembly window moves to the line corresponding to it.

This menu is a toggle option. If the disassembly window is closed when you turn on the option, you can open it by double clicking on a source line, then pressing **Cancel** in the **Break Point** dialog.

Assembly Single Step — Toggles step mode for assembler breakpoints. When execution reaches an assembler breakpoint, step mode is set on. When execution reaches a source breakpoint, it turns off.

Control Panel — Displays a toolbox window with buttons corresponding to the four **Go!** commands. The next time the Debuggee is suspended, the control panel receives focus.

When debugging in hard mode, when you switch the active window to the main Debugger window, you cannot access the control panel.

Setup — Opens the (see also) **Setup** dialog

Custom Groups — Opens the (see also) **Selective Break Points** dialog, which allows you to customize the windows message groups for which you can set conditional break points.

Custom Colors — Opens the (see also) **Colors** dialog, which allows you to set the Debugger selection colors for the source code windows.

Window Menu

Cascade — Resizes the active windows, and places them in an overlapping arrangement.

Tile — Resizes the active windows, and places them in an end to end vertical arrangement.

Tile Horizontal — Resizes the active windows, and places them in an end to end horizontal arrangement.

Arrange Icons — Arranges the iconized windows and arranges them at them at the bottom of the Debugger application frame.

Registers — Opens the (see also) **Registers** window, which shows the current registers values.

Library State — Opens the (see also) Library window, which shows the return values for Clarion library functions.

Windows Messages — Opens the (see also) **Windows Messages** window, which displays up to 200 Windows messages generated by and sent to the Debuggee.

Position Debuggee — Sizes the Debuggee to the maximum desktop area not taken up by the Debugger. This has no effect when the Debugger is maximized.

Global Variables — Opens the (see also) **Global Variables** window, which shows the current value of each component of each variable.

Watch Expressions	Opens the (see also) **Watch Expressions** window, which shows the current value of variables and expressions.
Active Procedures	Opens the (see also) **Active Procedures** window, which lists the procedure currently executing, and allows you to monitor nested procedure calls.

v Additional "How To" Topics

(see also) Setting Unconditional Breakpoints

(see also) Setting Conditional Breakpoints

(see also) Editing Variables at Runtime

(see also) Preparing Your Projects for Debugging

Colors Dialog

The Debugger displays the lines within a (see also) source code window in different colors, which you can customize. Select a radio button for the line type you wish, then choose a color from the list box.

You can specify colors for the **Current Line**, **General Cursor**, **Inactive Code** and **Break Points**.

You can also specify a **Monochrome** "color" scheme, by checking the box.

Selective Break Point Groups Dialog

This allows you to set up your own custom message groups to watch. Type a name for the group, and choose the Windows messages from the listbox.

When setting conditional breakpoints in the (see also) **Break Point** dialog, you can specify your custom group.

Volume 3 of the Microsoft Windows 3.1 Programmers Reference, published by Microsoft Press, devotes over 200 pages to lists and descriptions of all Windows messages

Break Point Dialog

This dialog is where you set breakpoints.

Normally, when debugging an application, you'll identify a small part of the program which produces incorrect output, or crashes. The Debugging process for this situation will probably require running just that part of the program, and stopping it at one or more points to check its status.

Breakpoints allow you to automatically halt execution at the line of code at which (or near which) you think the problem occurs. Your program runs up to the breakpoint, then halts and turns control back to the Debugger. You can then check the contents of variables and expressions to identify the cause of the problem, and step through from that point on. When in Step mode, the Debugger executes one source code statement, then returns control to the Debugger until you issue the next (see also GENERAL_CONTEXT) **Step!** command.

You can also set conditions on the breakpoint, telling the program to continue executing if the condition is false, or turning control over to the Debugger if true. After you set a breakpoint, the source code line where the breakpoint occurs appears in magenta in the source window. For "How To information, see also (see also) Setting a Conditional Breakpoint and (see also) Setting Conditional Breakpoints

Break Kind Choose the type of break.

Always - Specifies an unconditional break. The Debugger will *always* suspend execution at this point.

Watch Expression #0 - Specifies a break based on the contents of a variable in the (see also) **Watch Expressions** window. Type the number of the variable (its place in the Watch list) in the **Watch Number** box.

Windows Message - Type in a message name, or select a message from the combo box.

Message Group - Choose a message group. A group can indicate a category of messages, such as mouse or key messages. You can also set up a custom message group in the (see also) **Selective Break Point Groups** dialog. to remember several specific messages, so that the breakpoint will occur only on one of these messages. For example, you could place a breakpoint which checks for a WM_RBUTTONDOWN message, which is the message Windows sends when the user clicks the right mouse button in your window. When you run the program, you RIGHT-CLICK inside it, and Windows sends a WM_RBUTTONDOWN message to your application. The breakpoint condition would be true.

Message Not In Group - Choose a message group to *exclude*.

Remove Removes the selected break point.

Custom Groups Allows you to define a message group comprised only of the Windows messages you want. Define them in the (see also) **Selective Break Point Groups** dialog.

Break Points Dialog

This dialog presents active break points in a list. Its main function is to allow you to select a break point, then quickly highlight the source code line containing the break.

Locate	Activates the source code window and moves the selection bar to the source code line where the break point is located.
Delete	Deletes the break point from the list.
Edit	Opens the (see also) **Break Point** dialog.

Sources to Include in Session Dialog

This dialog allows you to add source code files to the Debug session.

Only files for which you specified that the Project System includes debug information appears in the list. See also Preparing Your Projects for Debugging.

The buttons in the dialog allow you to select or de-select more than one file at a time. When you expand the dialog, two additional lists display the names of source code files selected or already in the session.

Internal Error Dialog

This dialog indicates an error has occurred in the Debug session. The Debugger provides as much of a description as possible. In some cases, it may be necessary to exit Windows and reboot the machine before trying again.

Unknown Error Dialog

This dialog indicates an error has occurred in the Debug session. Exit Windows and reboot the machine before trying again.

Load Module Dialog

Select Active Module Dialog

This dialog allows you to add a dynamic link library, if opened by the Debuggee, to the Debug session. If the .DLL contains debug information, you can view the source code.

You can specify in the (see also) **Setup** dialog whether to ignore the information in the .DLL file. Debugging the .DLL may increase the startup time for the Debug session.

Watch Expression Dialog

The **Watch Expression** dialog allows you to add or edit a variable or expression in the (see also) **Watch Expressions** window.

Type an expression in the **Expression to Evaluate** field, then press the **OK** button. When the Debugger runs the program, it will test the expression upon reaching the breakpoint, and halt if the expression evaluates true.

The **Watch Expression** dialog also contains a **Browse** button, to help create your expression faster. Press the **Browse** button to see a list of the variables local to the procedure you're currently debugging.

You can prefix the variable name with a procedure name and/or a module name. This allows you to name a variable not currently in scope, for example, a variable in another procedure that would not be visible for the current procedure.

- *To specify a procedure and variable*, prefix the variable with the procedure name plus a period (".").

  For example, "RoyalFlush.King" refers to a variable called *King* in the procedure called *RoyalFlush*.

- *To specify a module and global variable*, prefix the variable with the module name plus a dot (".").

  For example, "NewDeal.Shuffled" refers to a global variable called *Shuffled* in the module called *NewDeal*.

- *To specify a local variable in a procedure in another module*, combine the prefixes.

  For example, "Poker.RoyalFlush:King" refers to the variable called *King* in the procedure called *RoyalFlush* in the module called *Poker*.

- You may specify register names (for example, *ax*) in a watch expression.

- You may use the unary operator (@) to denote the address of a memory object.

Tip: The Debugger will guess the right prefix if the variable is unique.

The following list presents the operators and expression syntax for the **Watch Expression** dialog. The operators are language independent, derived from C/C++, and Modula 2/Pascal operators.

Key	*Function*
`+`	`add`
`-`	`subtract`
`*`	`multiply`
`/ or DIV`	`divide`
`% or MOD`	`modulus (remainder)`
`|`	`bitwise OR`
`&`	`bitwise AND`
`<`	`less than`
`<=`	`less than or equal to`
`>`	`greater than`
`>=`	`greater than or equal to`
`=`	`equal`
`!= or <>`	`not equal`
`! or NOT`	`logical NOT`
`& or AND`	`logical AND`
`| or OR`	`logical OR`
`*`	`indirection (when prefix)`
`^`	`indirection (when post-fix)`

```
->          point at member
.           select member (record field)
::={e,d}    display expression e as if it was the same type d
```

Setup Dialog

This dialog allows you to select various options which carry over from Debugger session to session.

Ignore .DLL Files Instructs the Debugger to ignore debug information in .DLL files. This reduces the startup time for the Debugger. See also **Select Active Module** dialog.

Disable Kernel Msgs If you are running the Debug version of Windows (available in the Microsoft Windows 3.1 SDK), the Debugger will automatically trap error messages posted by the kernel (one of the three main dynamic link libraries utilized by Windows). You can locate such errors with the **Find Last Error** command.

If you are *not* using the AUX device to report messages, add the line OutputTo=NUL in the [DEBUG] section of your SYSTEM.INI file.

Report Missing Source The Debugger automatically prompts for source code files it cannot locate.

Iconize When Inactive Automatically iconizes the Debugger when the Debuggee is active.

Debugger On Top The Debugger appears on top an any other open windows when active; relevant for hard mode debugging only.

Tip: When working in Hard mode, type D to bring the Debugger to the top.

Disassembly Opcodes The disassembly window contains only op codes, eliminating the space taken up by binary codes.

Smart Single Stepping When enabled, single stepping on a line with a procedure call will load the debug information for the target procedure, if available. This option extends to .DLL's with debug information.

No Horizontal Scroll Hides the horizontal Debugger scroll bars.

Global Find Text When disabled, each source window "remembers" its own search text string. When enabled, the default search text will be the same as the last search, regardless of the window.

Order Record Fields When enabled, it orders the RECORD variables by memory address.

Auto Tile Tiles the open Debugger windows.

Clean Desktop When enabled, it minimizes all other running applications (other than the debuggee) when the debugger activates.

Max Source Windows Self explanatory.

Max Disassembly Self explanatory.

Find Text Dialog

This dialog allows you to search for text within the source code document(s).

Type the text to search for in the edit box, then press the **Find** button.

About Dialog

This dialog provides a copyright screen for the Debugger.

Edit Variable Dialog

This dialog allows you to change the value of a variable at run time.

Type in a new value and press the **OK** button.

See also Editing Variables at Runtime

GoTo Line Dialog

This dialog allows you to move the selection bar to the source code line number you specify.

Type the number of the line in the edit box, then press the **OK** button.

Find Procedure Dialog

This dialog allows you to move the selection bar to the source code line at which the procedure you select begins.

Select a procedure name from the list, then press the **Locate** button.

Source Window

The source windows display the source code documents. The title bar shows the document file name. By default, the current line cursor is green. When you select a line other than the current, the cursor is light cyan.

Tip: **If the Debugger opens without listing any source code documents in the (see also) Sources to include in this session dialog, the most probable cause is that none of the source code files listed in the Project Tree contained debug information. Check for .DAD files, which contain the debug information.**

INS inserts an unconditional break point at the cursor. DEL removes one. Pressing the SPACE BAR displays the **Break Point** dialog.

Disassembly Window

The **Disassembly** window display the machine instructions for the active module.

If you run the Debugger on a program with no debug information, the **Disassembly** window automatically displays the assembly language instructions. The current instruction is selected.

DOUBLE-CLICKING (or pressing ENTER) on a line in the **Disassembly** window which contains a jump or call instruction moves the cursor to the target location. ESC returns the cursor to the original location.

INS inserts an unconditional break point at the cursor. DEL removes one. Pressing the SPACE BAR displays the (see also) **Break Point** dialog.

Global Variables Window

The **Global Variables** window shows you the current value of each component of each variable. For example, for a string variable of eighteen characters, eighteen separate lines show the contents of the each position of the string. The variables window contains tree controls, so that you can expand only the variables you want to examine.

Each variable has a tree control to expand its listing. The top level is the source code module which contains the variable. The next is the variable name

Active Procedures Window

The **Active Procedures** window lists the procedure currently executing, which allows you to monitor nested procedure calls. The window appears in tree format. The upper levels represent the names of procedures and the lower levels represent the variables. DOUBLE-CLICKING on an active procedure displays the source or disassembly. DOUBLE-CLICKING on a variable copies it to the (see also) **Watch Expressions** dialog.

When working with more than one (see also) THREAD, the **Active Procedures** window displays information for the current thread only.

Watch Expressions Window

The **Watch Expressions** window shows the current value of variables and expressions. Set a watch expression to see how a variable or expression changes as your program executes.

See also the **Watch Expression** dialog to learn the syntax for watch expressions, and how to edit an expression in the list. To add a variable to the watch list:

1. DOUBLE-CLICK on an empty line in the **Watch** window.

2. When the **Watch Expressions** dialog appears, type a variable name (as it appears in the global variables list) and press **OK**.

3. Alternatively, press the **Browse** button, select a variable from the list, then press **OK** twice.

The expression or variable appears in the watch window, and its current contents appear next to it.

You can also add a variable to the watch window by DOUBLE-CLICKING or pressing ENTER on a variable in either the global variables or the active procedures windows.

You can add a watch expression by DOUBLE-CLICKING on an empty line in the **Watch Expressions** window, then specify an expression in the **Watch Expression** dialog.

Tip: To quickly add a structured variable (such as a record, string or array) to the watch list, DOUBLE-CLICK on it in the Global Variables or Active Procedures windows, then press the Copy Variable to Watch button.

Message Window

The *Messages window* displays up to the most recent 200 message events generated by or directed to your application. The Debugger adds a separator line ("-----") to indicate a breakpoint occurred.

Every action the user takes--from mouse movement to menu commands--is first processed by Windows. If Windows determines the action is for your application, it passes the information to your application via a message. For example, if the user types the letter "A," it sends a WM_KEYDOWN message to your application, with the key code for "A" as the first message parameter.

See the **Break Point** dialog for information on selecting Windows messages to break on.

Tip: If you include DDE services in your application, we recommend testing your application with another DDE application and monitoring the DDE messages. For further information, see the *Microsoft Windows 3.1 Programmers Reference, Volume 3*, available from Microsoft Press.

Library Window

The **Library** window displays return values for Clarion library functions. These functions represent all the field and other events. Functions include (see also) ACCEPTED, (see also) SELECTED, (see also) FIELD, (see also) FOCUS, (see also) FIRST-FIELD, (see also) LASTFIELD, (see also) ERRORCODE, AND (see also) ERRORFILE.

The names listed in EQUATES.CLW and KEYCODES.CLW appear next to the return values.

Registers Window

The **Registers** window shows the current register values; the register appears in the left column, and its value to the right.

Setting Unconditional Breakpoints

Normally, when debugging an application, you'll identify a small part of the program which produces incorrect output, or crashes. The Debugging process for this situation will probably require running just that part of the program, and stopping it at one or more points to check its status.

Breakpoints allow you to automatically halt execution at the line of code at which (or near which) you think the problem occurs. Your program runs up to the breakpoint, then halts and turns control back to the Debugger. You can then check the contents of variables and expressions to identify the cause of the problem, and step through from that point on. When in Step mode, the Debugger executes one source code statement, then returns control to the Debugger until you issue the next **Step!** command.

When you set a breakpoint, the source code line where the breakpoint occurs appears in magenta in the source window.

An unconditional or "sticky" breakpoint is placed on a source code line, and stops execution whenever the program encounters that statement. To add an unconditional breakpoint:

1. Open the (see also) source code or disassembly window.

2. Locate the line of code to break on and DOUBLE-CLICK on it.

3. When the (see also) **Break point** dialog appears, select **Always** and press the **OK** button.

When you execute the **Go!** command, the Debuggee runs until it reaches the breakpoint, then stop.

Setting Conditional Breakpoints

To narrow the search for bugs, you can tell the Debugger to break only when a certain condition exists. The condition takes the form of an expression which can include program variables, operators and constants. You can also tell the Debugger to break when it detects a particular message or messages from Windows to the application.

- To set a conditional breakpoint on a change in a watch expression's value:

1. Establish a watch expression as described in the (see also) **Watch Expression** dialog, below.

2. Locate the line of code to break on and DOUBLE-CLICK on it.

3. When the (see also) **Break point** dialog appears, select **Watch Expression #0**.

4. Type the number of the watch expression (from the watch expressions window) in the **Watch#0** field.

5. Press the **OK** button.

When you execute the **Go!** command, the program runs until it reaches the breakpoint, evaluates the watch expression, then stops if the expression is true, i.e., evaluates to a non-zero value.

For example, if variable X should have a maximum value of 999, but increments to 1000 anyway, causing havoc, you can tell the Debugger to break at 999, then step through the program to see when and how it reaches 1000.

- To set a breakpoint conditional on a specific Windows message:

1. Locate a line of code to break on and DOUBLE-CLICK on it.

2. When the **Break point** dialog appears, select **Windows Message**.

3. Select a Windows message from the **Windows Message** combo box.

4. Press the **OK** button.

For example, you could place a breakpoint in a loop which checks for a WM_RBUTTONDOWN message, which is the message Windows sends when the user clicks the right mouse button in your window. When you run the program, you RIGHT-CLICK inside it, and Windows sends a WM_RBUTTONDOWN message to your application. The breakpoint condition would be true.

- To set a breakpoint conditional (or not) on receipt of one of several Windows messages:

1. Locate the line of code to break at and DOUBLE-CLICK on it.

2. When the **Break point** dialog appears, select **Message Group**; or **Message Not in Group**.

3. Select a message group from the list. This can indicate a category of messages, such as mouse or key messages. You can also set up a custom message group (by pressing the **Custom Groups** button) to remember several specific messages, so that the breakpoint will occur only on one of these messages.

4. Press the **OK** button.

For example, you could place a breakpoint in a loop which checks for a Key message. When you run the program, when you press a key, Windows sends a message to your application, and the breakpoint condition would be true.

- To set a breakpoint which breaks when you receive an unexpected message, that is, one that doesn't belong to a group you specify:

1. Locate the line of code at which you want to establish the breakpoint and DOUBLE-CLICK it.

2. When the **Break point** dialog appears, select **Message Group**.
3. Select a message group from the combo box. The breakpoint then occurs only when the application receives a message *not* in this group.
4. Press the **OK** button.

Editing Variables at Runtime

Using the Debugger, you can change the value contained in a memory variable while the program is suspended. You can then resume the program to test execution with the variable containing the new value.

To change the contents of the variable:

1. Select the variable in either the (see also) **Global Variables** or the (see also) **Active Procedures** windows.

2. Press F2, or choose **Edit** ä **Edit**.

The (see also) **Edit Variable** dialog appears.

3. Type a new value for the variable and press the **OK** button.

When you choose **Go!**, **GoCursor!**, **Step!**, or **ProcStep!**, the program resumes execution with the memory variable changed to the new variable.

Preparing Your Projects for Debugging

The Project System allows you to set the debug global options in the **Global Options** dialog, in the IDE. To create an .EXE file to debug, do the following in the IDE, before launching the Debugger:

1. Create your project file, and make it the current project.

2. Choose **Project** ä **Edit Current Project** to view the **Project Tree** dialog.

3. Select the top level of the tree, which holds the name of the project, and press the **Properties** button.

4. When the **Global Options** dialog appears, mark the **Full** checkbox in the **Debug Information** group box.

5. Optionally check the **Line Numbers** box.

6. Press the **OK** button to close the dialog.

7. Press the **Make** button on the toolbar to compile and link the application.

The application now includes the information the Debugger needs.

You can also turn on debugging information for a *single* module in the project. The advantage of this is that it reduces the overhead for the debugger. To do so, do the following in the IDE, before launching the Debugger:

1. Create your project file, and make it the current project (the *Using the Project System* chapter explains how).

2. Choose **Project** ä **Edit Current Project** to view the **Project Tree** dialog.

3. Select only the source module you need to debug, and press the **Properties** button.

4. When the **Compile Options** dialog appears, mark the **Full** checkbox in the **Debug Information** group box.

5. Check the **Line Numbers** box.

6. Press the **OK** button to close the dialog.

7. Press the **Make** button on the toolbar to compile and link the application.

This includes debug information for that module only.

The Debugger runs as a separate application, but you can start it either inside the IDE, or directly from Windows.

- To start the Debugger inside the IDE, either:

1. Either choose **Project** ä **Debug** *or* compile and link your application by pressing the **Make** button.

2. With the Compile results dialog still open, press the **Debug** button.

- To start the Debugger as a normal application.

1. Switch to Program Manager and open the Clarion group.

2. DOUBLE-CLICK the Debugger icon.

If you wish to start the Debugger from another program launcher the application file name is CLWDB.EXE.

3. With the Debugger launched, choose **File** ä **File to Debug**, then choose an .EXE file in the **Open File** dialog.

You *can* load the debugger, then debug a program which was launched even *before* you loaded the debugger. This is useful for situations where the program under development unexpectedly "misbehaves,"

but hasn't yet produced a fatal error.

To do so, start the debugger and execute the **File to Debug** command. Choose the .EXE file for the running program from the **Open File** dialog. The debugger will ask you to confirm that you wish to debug a running program.

When you launch the Debugger from the Compile Results dialog, you must confirm the name of the source code files.

- To load the source files when the Debugger appears:

1. Select the source code files in the (see also) **Sources to include in session** dialog.

The debug information file stores the files you select between debug sessions.

2. Press the **OK** button.

The Debugger windows appear.

If the application does not include debug information, the Debugger skips this step and opens a (see also) disassembly window.