# IBM C/C++ Tools: Debugger Introduction

## 2.0

Document Number S61G-1184-00

IBM C/C++ Tools:

S61G-1184-00

# Debugger Introduction

2.0

IBM C/C++ Tools:

S61G-1184-00

# Debugger Introduction

2.0

**Note!**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v.

**Second Edition (March 1993)**

This edition applies to Version 2.0 of the IBM C/C++ Debugger (Programs 61G1176 and 61G1426) and to all subsequent releases and modifications until otherwise indicated in new editions.

Requests for publications and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, Ontario, Canada. M3C 1V7

You can also send your comments by facsimile to (416) 448-6057 addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to **torrcf@vnet.ibm.com**; IBMLink, to **toribm(torrcf)**; IBM/PROFS, to **torolab4(torrcf)**; IBMMAIL, to **ibmmail(caibmwt9)**

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

# Contents

# Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

## Trademarks and Service Marks

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks and service marks of IBM Corporation in the United States and/or other countries:

IBM
OS/2
WorkFrame/2
Operating System/2
Presentation Manager

## Highlighting Conventions

The following highlighting conventions are used in this book:

| Font | Convention |
|---|---|
| **Bold** | Names of windows, menus, menu choices, push buttons, and fields and selections in action windows. |
| `Monospace` | Text displayed in the sample program, text or commands that you type, or text that you select. |
| *Italics* | Words that are used for emphasis and variables in command strings. |

## Related Books

You should be familiar with the following books before you use this manual.

- *Presentation Manager Programming Reference Volume I*, 10G6264, which describes the Operating System/2&trade. (OS.2*) functions.
- *Presentation Manager Programming Reference Volume II*, 10G6265, which details the OS/2 function calls.
- *Presentation Manager Programming Reference Volume III*, 10G6272, which describes window processing, messages, and other OS/2 features.

The following publications provide information about the IBM C/C++ Debugger:

- *IBM C/C++ Tools: Installation*, S10G-4443, which describes how to install the product.
- *IBM C/C++ Tools: Programming Guide*, S10G-4444, which describes how to write, compile, and link programs for the debugger.
- *IBM WorkFrame/2: Introduction*, S10G-4475, which describes how to install the WorkFrame/2.
- *IBM C/C++ Tools: License Information*, S10G-4442, which provides a summary of the features and warranty information.

# Introducing the IBM C/C++ Debugger

The IBM* C/C++ Debugger (hereafter called the debugger) uses Presentation Manager* (PM) window services to help detect and diagnose errors in code developed in IBM 32-bit C/C++.

The debugger provides three program view windows that display an image of the program you are debugging. You can execute your program, set breakpoints, examine message queues, and monitor variables, registers, storage, and stacks.

This chapter describes the new and enhanced features of the OS/2* debugger. It also describes how to access the online contextual help and online tutorial that are provided by the debugger.

## Understanding the New and Enhanced Features

The debugger now supports the C++ language. A summary of the new features added to support C++ are as follows:

| | |
|---|---|
| **Template support** | Allows you to debug template functions. When an action is ambiguous, the debugger prompts you for the correct instance of a function. |
| **Inheritance view** | Displays class hierarchy in a graphical format. |
| **Class details view** | Displays information about a class. This information includes type information for data members, member functions and parameters, access information and relationship data. |
| **Expression evaluation** | Evaluates expressions that use the new C++ operators for reference and scope. Full casting is supported for C++. |

In addition to the features supporting C++, there are new and enhanced features. Descriptions of these features are as follows:

| | |
|---|---|
| **Window analysis** | Displays PM windows in a three-dimensional view. Window characteristics and window relationship data are also displayed. |
| **Message queue monitor** | Monitors PM messages that are related to specific PM windows. |
| **Include file support** | Debugs include files that contain executable code. You can browse the source code of an include file and set breakpoints. |
| **Exception handling** | Displays options when an exception occurs during execution of your program. |
| **Jump to location** | Resets the extended instruction pointer (EIP) to a location in your program without executing any statements. |
| **Buttons in the title bar** | Provides quick access to step commands and various utility windows. |
| **Breakpoints** | Manipulates breakpoints using a notebook format. This allows easier access to different types of breakpoints. |
| **Stack** | Views a more complete picture of the stack. |
| **Debug Session Control** | Displays a hierarchical representation of the relationships among all components, such as executable files, object files, and functions. |
| **Default data representation** | Allows you to change the default representation of data. For example, integers can be displayed as either decimal or hexadecimal. |

| | |
|---|---|
| **Locate function** | Displays source code for a given function. The debugger lists overloaded functions and lets you choose the correct function. |
| **Registers** | Views and alters the individual flags in the processor's EFLAGS register, coprocessor floating point control word, and floating point status word registers. |

## Using the Help

The debugger provides online help as contextual help and an online tutorial.

## Using the Contextual Help

Documentation is available as context-sensitive help. You can ask for help on any menu choice or window to discover how to use that particular item.

You can access the help screens in one of the following ways:

- Select a choice from the **Help** menu.
- Press F1 in any debugger action window.
- Press F1 while highlighting any menu-bar choice.
- Select the **Help** push button in any action window.

## Using the Online Tutorial

To become familiar with the debugger features, use the online tutorial.

The online tutorial guides you through sample debugging sessions in several different sample programs, demonstrating basic C debugging features, debugging features specific to C++, and features for PM programs.

To access the tutorial, select **Tutorial** from the **Help** menu in the **Debug Session Control** window, or any of the program view windows (**Source, Disassembly,** or **Mixed**).

# Before You Begin

Before you run the debugger on your program, there are considerations you need to be aware of and preparatory items you should complete. This chapter explains how to complete these tasks.

## Writing Code that the Debugger Supports

Using C and C++, you can write your program code with stylistic features that are not supported by the debugger. Multiple statements on the same line are difficult to debug. None of the individual statements can be accessed separately when you set breakpoints[1] or when you use step commands.

## Compiling and Linking Your Program

To use the debugger, compile and link your program with the following options:

`/Ti+` Compiles your program to produce an object file that includes line number information and a symbol table, in addition to the source code.

`/O-` Compiles your program with optimization off. This is the default.

`/Oi-` Compiles your program with inlining off. This is the default.

`/Debug` Links your program to produce an executable file that includes line number information and a symbol table, in addition to the executable code.

For more information about compiling and linking your program, refer to *IBM C/C++ Tools: Programming Guide*.

[1] Breakpoint is a defined location or condition in a program that, when it is met, stops the execution of the program.

## Customizing the Environment

There are two environment variables that you can set: PMDPATH and PMDTAB. For more information, refer to the online help.

# Getting Started

This chapter explains how to start and end a debugging session.

You can start the debugger either from the OS/2 command prompt or the WorkFrame/2 environment.

## Starting the Debugger from an OS/2 Prompt

To start the debugger from the OS/2 command prompt, enter the command `ipmd` and the following parameters, in the order they are listed:

1. Any debugger parameters that you want to use.
2. Name of the program you want to debug.
3. Any input parameters that you want to pass to the program.

For example, type the following:

```
ipmd /x myprog xyz
```

where */x* represents a debugger parameter, *myprog* represents your program name, and *xyz* represents the program parameter you want to pass to the program.

The debugger parameters are:

*/n* Do not use any restart information.

*/i* Start the debugger in the system initialization routine so that you can debug initialization code.

If you type `ipmd` only and press Enter, the **Startup Information** action window is displayed. To continue, complete the entry fields as follows:

1. In the **Program** entry field, type the name of the program you want to debug or click on the list button at the end of the entry field to select a program. You can also select the **File List** push button.
   If you select **File List**, the **Select Program** action window is displayed. From this action window, select the program you want and select **OK**. The **Startup Information** action window is dis-

played again with the program name you selected displayed in the **Program** entry field.
2. In the **Parameters** field, type any parameters that you want to pass to your program. You must separate multiple parameters with spaces.
3. Select **OK** to accept the information entered, close the action window, and start the debugger.

**Note:** If you are starting the debugger for the first time, the **Change Location** action window is displayed, prompting you to type the location where you want to store the profile information file. The debugger uses the profile information file to store session settings. Type the full path name and then select **OK**. The profile information file, IPMD.@2S, is created and stored in the directory that you typed. You can also select the **Default** push button and the profile information file is stored in the default directory.

## Starting the Debugger from the WorkFrame/2* Environment

Before you start the debugger from the WorkFrame/2* environment, you must create a project for the program you want to debug. To be able to compile and link a target program with debugging information, you must set the debugger options that the WorkFrame/2 environment uses for creating a project. For information on creating a project, setting options, and starting the debugger, refer to *IBM WorkFrame/2: Introduction*.

## Ending the Debugging Session

To end the debugging session, select **Close debug** from the **File** menu in a debugger window. The **Close Debug** action window is displayed. Select one of the following choices:

- Select **Yes** to end your debugging session and save the restart information.
- Select **No** to end your debugging session without saving the restart information.
- Select **Cancel** to return to the previous screen without exiting the debugger or changing any information.

*Restart information* is used to restore the debugger windows and breakpoints when debugging a program more than once. It is stored separately for each program debugged.

You can also end the debugging session by pressing F3 in the program view windows or in the **Debug Session Control** window.

# Using the Debugger Windows

This chapter describes the **Debug Session Control** window, program view windows, the basic debugger windows, and the C++ windows. The **Debug Session Control** window and the program view windows are the main debugger windows.

## Using the 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 displays the threads and components for the program you are debugging. From this window, you can enable or disable threads and select program components for viewing. Selecting program components for viewing provides you with the capability to access any part of your program.

The **Threads** list box shows the state of the threads that have been started by your program.

The **Components** list box shows the executable files (EXEs and DLLs) for the program you are debugging. To display a list of object files (OBJs) contained within an executable file, click on the plus icon to the left of the executable file name. To open a program view of an object file, double-click on the object file name. To display a list of functions for a specific object file, click on the plus icon to the left of the object file name. To open a program view to a specific function, double-click on the function name.

**Note:** For C++ programs, the function name includes the class name and parameters.

To determine the characteristics of the displayed view, select the **Source Window Properties** action window as described in “Using the Source Window Properties” on page 22.

## Using the Program View Windows

The program view windows display the source, if available, for the program you are debugging. The following program view windows are displayed by the debugger:

**Source** Used to view the source file.

**Disassembly** Used to view the lower-level machine instructions and the use of the registers.

**Mixed** Used to view the source statements of your program combined with the disassembly instructions.

The **Notebook** choice is available for the **Source** and **Mixed** windows. This choice supports easy access to executable statements that are in the include files. If your program contains executable code in include files and you enable **Notebook**, the program view is displayed in a notebook format. To enable or disable **Notebook**, select the **View** cascaded menu from the **File** menu in any of the program view windows.

All program view windows have a prefix area located at the left of each window. In the **Source** window, each line is prefixed with a line number. In the **Disassembly** window, each line is prefixed with an address. In the **Mixed** window, each source line is prefixed with its line number, as it is in the **Source** window and each disassembled line is prefixed with an address, as it is in the **Disassembly** window.

To switch from one view to another, select a different view choice from the **View** cascaded menu in the **File** menu of the **Debug Session Control** window or from any of the program view windows.

## Displaying Other Source Files

If you have multiple source files in your program, only one source file is initially displayed. However, you can display additional source files.

To switch from one source file to another source file, use the following steps:

1. Select **Open new source** from the **File** menu of the **Debug Session Control** window or from any of the program view windows. The **Open New Source** action window is displayed.
2. In the **Source** entry field, type the source file name.
3. Select **OK**.

## Using Buttons on the Title Bar

Buttons have been provided for easier access to frequently used functions. The following buttons are located in the title bar of the program view windows.

**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** allows you to start and halt the program. When the push button is green, you can start the program. When the push button is red, you can stop the program.

**View** changes the current view to one of the other views (**Source, Disassembly** or **Mixed**).

**Stack** displays the **Stack** window. See “Using the Stack Window” on page 16 for more information.

**Register** displays the **Register** window. See “Using the Register Window” on page 16 for more information.

**Program Monitor** displays the **Program Monitor** window. See “Using the Data Popup and Monitor Windows” on page 17 for more information.

**Storage** displays the **Storage** window. See “Using the Storage Window” on page 17 for more information.

**Debug Session Control** displays the **Debug Session Control** window. See “Using the Debug Session Control Window” on page 11 for more information.

## Executing a Program

You can execute your program by using step commands or the **Run** command.

**Step commands** Step commands control the execution of the program. The execution of the line of code is reflected in all open views, and is performed in the thread specific to the view.

To single step your program, click mouse button two. This executes the current line in the program.

**Run command** The **Run** command runs the program until a breakpoint is encountered, the program is halted, or the program ends.

When you execute your 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.

## Setting Breakpoints

You can control how your program executes by setting breakpoints. A breakpoint stops the execution of your program at a specific location or when a specific event occurs. To set breakpoints, select the **Breakpoints** menu from the **Debug Session Control** window or from any of the program view windows. You can also set a simple line breakpoint by double-clicking in the *prefix area* of an executable statement in any of the program view windows. The prefix area is the area on the left of the program view window where line numbers or addresses are displayed. The prefix area turns *red* indicating that the breakpoint has been set.

The following is a list of the types of breakpoints that can be set in your program. For detailed information on setting breakpoints, refer to the online help.

- Line
- Function
- Address
- Change address
- Load occurrence.

## Handling Exceptions

During execution of the program by the debugger, it is possible that the program can generate an exception. If this happens, the debugger suspends execution of the program and indicates the location within the code where the exception occurred. The **OS/2 Application Exception** action window is displayed with the following choices:

**Examine/Retry** You can investigate the cause of the exception and retry execution of the line that caused the fault.

**Step Exception** The debugger steps into the first registered exception handler, which is tracked by OS/2. Execution stops at the first executable line of code in that exception handler.

**Run Exception** The debugger runs the exception handlers.

## Using the Register Window

The **Register** window lists all the processor and coprocessor registers for a particular thread. The contents of all of the registers except ST0 through ST7 are displayed in hexadecimal. To update a register, type over the contents that are displayed in the register. To toggle the value of a 1-bit flag, double-click on it or place the cursor on it and press Enter.

To display the processor registers and flags, including the math coprocessor information, select **Registers** from the **Windows** menu or select the **Register** button on the title bar.

## Using the Stack Window

The **Stack** window lists all of the active functions for a particular thread including the PM calls. The functions are displayed in the order that they were called.

To display the **Stack** window, select **Stack** from the **Windows** menu or select the **Stack** button on the title bar.

## Using the Storage Window

The **Storage** window shows the storage contents and the address of the storage. 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, type over the contents of the field in the **Storage** window.

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

To display the **Storage** window, select **Storage** from the **Windows** menu or select an already displayed storage monitor from the list. You can also select the **Storage** button on the title bar to display a new **Storage** window.

## Using the Data Popup and Monitor Windows

The debugger has four windows that you can use to monitor variables:

- **Data popup**
- **Local variables**
- **Program monitor**
- **Private monitor**.

A **Data Popup** window monitors single variables or expressions. The variables or expressions can be transferred either to the **Program Monitor** window or the **Private Monitor** window.

The **Local Variables** window monitors the local variables (static, automatic, and parameters) for the current execution point in the program. The contents of the **Local Variables** window change each time your program enters or leaves a function.

The **Program Monitor** and the **Private Monitor** windows are used as collectors for individual variables or expressions you might be

interested in. Variables and expressions may be created in these monitors or may be transferred to them from a **Data Popup** window.

The difference between the **Private Monitor** window and the **Program Monitor** window is the length of time that they remain open. The **Program Monitor** window is a main debugging window and remains open for the entire debugging session.

The **Private Monitor** window is associated with the program view window from which it was opened and closes when its associated view is closed.

### Displaying Data Constructs

The debugger shows the value of classes, structures, and arrays in a graphical hierarchy. When a class, structure, or array is displayed in a monitor window, you can select the associated icon to see the contents of each element. After expanding the element, you can contract it by clicking on the icon. This leaves more space to expand the contents of other classes, structures or arrays.

## Using C++ Debugger Windows

This topic describes the debugger windows available for C++ programs. The **Inheritance View** window and the **Class Details** window provide information pertaining to classes in your program.

### Using the Inheritance View Window

The **Inheritance View** window displays a list of C++ classes and a graphical depiction of the C++ class hierarchy. This provides a method of visualizing the relationships of the classes in your program.

To display the **Inheritance View** window for the classes in a C++ object, highlight the name of the object in the **Debug Session Control** window, then select **Inheritance** from the **Windows** menu.

You can browse the total class picture for an object as a graph that shows the inheritance and multiple inheritance relationships of the classes.

To view the member data and member functions for the class, double-click on the class name to open the **Class Details** window.

## Using the Class Details Window

The **Class Details** window allows you to view more information about a particular class definition in a C++ program. This view contains a list of the base and derived classes for a particular class. It also contains the type information and attributes for data members.

To display the **Class Details** window, select **Class details** from the **File** menu in the **Inheritance View** window.

# Using the Debugger Options

Use the debugger options to control the operation of the debugger. Some of the more commonly used options are described in this chapter.

## Using Session Settings and Window Settings

The debugger settings are divided into two categories: session settings and window settings.

The session settings options can be accessed from any of the program view windows or from the **Debug Session Control** window by selecting choices from the **Session settings** cascaded menu in the **Options** menu. These settings affect the behavior of the debugger and remain in effect for the duration of the session. If you select **Save session settings** on any of these windows, these settings apply to future debugging sessions.

The window settings options are accessed from those windows that have additional formatting requirements and by selecting choices from the **Window settings** cascaded menu in the **Options** menu. The settings for the window options apply only to the particular window in which they are available. If a particular window has only one window setting option, then the **Window settings** cascaded menu does not display. However, the available option is displayed in the **Options** menu.

## Using the Initial Window Placement

The **Initial Window Placement** action window lets you select the initial position and size for all the primary windows that the debugger creates. To display the **Initial Window Placement** action window, select **Session settings** → **Initial window placement** from the **Options** menu.

## Using the Monitor Properties

The **Monitor Properties** action window lets you save the options for the monitor. To display the **Monitor Properties** action window, select **Session settings** → **Monitor properties** from the **Options** menu.

## Controlling the PM Debugging Mode

When you are debugging a PM program, you can set the *synchronous* or *asynchronous* debugging mode by selecting **Session settings** → **PM debugging mode** from the **Options** menu. For more information about setting the PM debugging mode, refer to "Using PM Debugging Mode" on page 25.

## Controlling Window Repainting

You can control the repainting of windows from the **PM Debugging Mode** action window that is displayed when you select **Session settings** → **PM debugging mode** from the **Options** menu. The choices under the **Application windows** group heading lets you control the interaction between the application windows and PM, while the program you are debugging is stopped. The color you select from the **Invalid area color** combination box is used for the repainting.

## Using the Source Window Properties

You can control how windows are displayed and closed from the the **Source Window Properties** action window.

The **Source Window Properties** action window is displayed when you select **Session settings** → **Source window properties** from the **Options** menu.

## Setting the Animation Rate

Use the **Animation Rate** action window to modify the rate that the debugger single steps when you select **Animate** from the **Run** menu from any of the program view windows.

To display the **Animation Rate** action window, select **Session settings** → **Animation rate** from the **Options** menu.

## Changing the Default Data Representation

The **Default Data Representation** action window lets you select a different data type. For example, in a C program, you can display an integer as a decimal or hexadecimal.

To display the **Default Data Representation** action window, select **Default data representation** from **Session settings** from the **Options** menu.

## Changing the Font Selection

The **Font Selection** action window lets you select alternative fonts for the current windows. To display the **Font Selection** action window for your current window, select **Fonts** from the **Options** menu.

# Using the PM Debugger Features

Because the debugger runs in the OS/2 environment (and, more specifically, the PM environment) it offers some features that allow you to debug programs written for PM. These features are described in this chapter.

## Using PM Debugging Mode

PM is a message-based system, meaning that as program events are encountered by PM applications, the applications communicate with each other by passing messages and by receiving user input through input messages. When a PM program encounters an enabled breakpoint, the input queue can become blocked and dependent program events or processes can become blocked also. For example, the input queue can become blocked when your program is halted at a breakpoint that has been triggered by an input event.

The debugger provides two modes of operation by which PM messages can be processed while the debugger has control. These two modes are synchronous and asynchronous.

### Understanding the Synchronous Mode

When the debugger is operating in synchronous mode, the messages passed between PM applications are answered by their target applications in the order that they were created. The messages that are passed within the debugger, however, take priority over any other messages that are passed in the system.

When the program being debugged is stopped and the debugger is in synchronous mode, other PM applications are locked, leaving the debugger free to operate. In synchronous mode, you are not able to use any other applications that are running.

## Understanding the Asynchronous Mode

When the debugger is operating in asynchronous mode and the program you are debugging is stopped, the debugger immediately responds to messages sent to the program being debugged on the program's behalf. The debugger answers the messages with a simple default response, freeing up other processes to operate while the debugger has control. When you are running the debugger in asynchronous mode, other PM applications running in the system are not blocked when the program being debugged stops.

**CAUTION:**
**Do not operate the debugger in asynchronous mode if the PM application you are debugging requires appropriate responses to its messages. For example, a dynamic data exchange (DDE) message would require the appropriate response.**

## Setting the PM Debugging Mode

By selecting **Session settings** → **PM debugging mode** from the **Options** menu in the **Debug Session Control** window, you can set the mode of the debugger to asynchronous or synchronous. This lets you control how PM messages are processed while the debugger has control.

---

## Using the Message Queue Monitor Window

The **Message Queue Monitor** window displays PM messages associated with a PM application. It presents formatted messages in a list as they occur. Using the message queue monitor, you can control:

- How the information is displayed for each message.
- How message parameters are formatted.
- Which messages are monitored.
- Which windows have their messages monitored.
- Which message queues have their messages monitored.
- How the user generated messages are displayed.

## Using the Window Analysis Window

The **Window Analysis** window gives you an understanding of PM application windows. It presents both graphical and textual information about your program's windows and lets you observe the relationships between windows.

It allows you to view a three-dimensional image of your program's windows. When this image is displayed, you can rotate the image to visually separate the windows, select a window on the image, and look at the detailed information pertaining to that window.

The two secondary windows of **Window Analysis** are **Window Characteristics** and **Parent and Z-Order Tree**.

To display the **Window Analysis** window, stop the program, then select **Window Analysis** from the **Windows** menu.

## Using the Window Characteristics Window

The **Window Characteristics** window shows the window characteristics of the debuggee windows, such as the size and the window handles.

The windows listed in the **Window Characteristics** window at one time reflect the debuggee windows on the current page of the **Window Analysis** window. The characteristics are displayed in a column format with each row representing a different window. To specify which characteristics are displayed, select **Display style** from the **Options** menu in the **Window Characteristics** window.

## Using the Parent and Z-Order Window

The **Parent and Z-Order Tree** window shows the parent and z-order relationships between the debuggee windows and some non-debuggee windows. This allows you to see which windows are drawn in front of other windows. The non-debuggee windows shown in the **Parent and Z-Order Tree** view are the desktop and desktop-object windows and their children which are not debuggee windows. To specify which characteristics are displayed, select **Display style** from the **Options** menu in the **Parent and Z-Order Tree** window.

# Appendix A. Expressions Supported

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

**Note:** You can display and update bit fields for OS/2 C/C++ code only. You cannot look at variables that have been defined using the `define` preprocessor directive.

## Using the Supported Expression Operands

You can monitor expressions or set conditional breakpoints using the following operands:

**Variable** A variable

**Constant** The constant can be one of the following types:

- Floating-point

  **Note:** The largest floating-point number supported is `1.8E3 8`. The smallest floating-point number supported is `2.23E-3 8`.
- String (enclosed in " ")
- Character (enclosed in ' ')
- Segment: Offset address specification.

  When you are specifying a segment offset address for monitoring in a variable monitor window, specify the offset address in the format `x : x`.
- Integer.

**Register** One of the following general register names: AX, BX, CX, DX, SP, BP, SI, DI, AL, BL, CL, DL, AH, BH, CH, DH, EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP, CS, DS, ES, FS, GS, SS, and EFLAGS.

One of the following floating-point registers: ST0 through ST7, FPCW, FPSW, FPTW, FPEIP, FPCS, FPEDP, and FPDS.

If there are 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, the register attributes are:

- General purpose registers are represented as 32-bit pointers
- Floating-point registers are represented as 80-bit float operands.

## Using the Supported Expression Operators

You can monitor expressions or set conditional breakpoints using the following operators:

*Table 1 (Page 1 of 2). Supported Expression Operators and Their Coding in C*

| Operator | Coded as |
|---|---|
| Complement | ~*a* |
| Bitwise negate | ¬*a* |
| Logical negation | !*a* |
| Dereference | **a* |
| Address of | &*a* |
| Multiply | *a* * *b* |
| Divide | *a* / *b* |
| Modulo | *a* % *b* |
| Add | *a* + *b* |
| Subtract | *a* - *b* |
| Shift left | *a* << *b* |
| Shift right | *a* >> *b* |
| Less than | *a* < *b* |
| Greater than | *a* > *b* |
| Less than or equal to | *a* <= *b* |
| Greater than or equal to | *a* >= *b* |
| Equal | *a* == *b* |
| Not equal | *a* != *b* |
| Bitwise AND | *a* & *b* |

*Table 1 (Page 2 of 2). Supported Expression Operators and Their Coding in C*

| Operator | Coded as |
|---|---|
| Bitwise OR | *a* \| *b* |
| Bitwise exclusive OR | *a* ^ *b* |
| Logical AND | *a* && *b* |
| Logical OR | *a* \|\| *b* |
| Structure element | *a*.*b* |
| Array element | *a*[*b*] |
| Subfield select | *a*->*b* |
| Scope * | a::b |
| Cast | *(dt)(exp)*, where dt is one of the data types listed under "Using Supported Data Types" on page 31, and exp is an expression that evaluates to one of these data types. |
| **Note:** * C++ operator only. | |

## Using Supported Data Types

Operations on the following data types and references to the following data types are supported:

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

®

Part Number: 61G1184
Program Number: 61G1176
61G1426

Printed in U.S.A.