

The Toolbar

The Toolbar is a row of buttons at the top of the main window which represent application commands. Clicking one of the buttons is a quick alternative to choosing a command from the menu. Buttons on the toolbar activate and deactivate according to the state of the application.

Button	Action	Menu Equivalent
	Open File to Disassemble.	Disassembler <u>O</u> pen
	Save Disassembly to a Text File.	Disassembler <u>S</u> ave
	Find Text. Search <u>F</u> ind	
	Copy Selected Disassembly Text Lines.	Disassembler <u>C</u> opy Selected Lines
	Goto Code Starting Location of Code.	Goto <u>G</u> oto Code Start Point
	Goto Program Entry Point in the Code.	Goto <u>G</u> oto Program Entry Point
	Select and Goto Specified Page .	Goto <u>G</u> oto Page
	Select and Goto Specified Location .	Goto <u>G</u> oto Code Location
	Execute Jump Instruction.	Execute Text <u>E</u> xecute Jump
	Return From Last Jump. Execute Text	<u>R</u> eturn From Last Jump
	Execute Call Instruction. Execute Text	<u>E</u> xecute Call
	Return From Last Call. Execute Text	<u>R</u> eturn From Last Call
	List Import Functions. Functions	<u>I</u> mports
	List Export Functions. Functions	<u>E</u> xports
	Display of Data Object in Hex Format.	Hex Data <u>H</u> ex Display of Data Objec/Segmentd
	Display of Code Data in Hex Format.	Hex Data <u>H</u> ex Display of Code Data
	List and Search Menu References.	Refs <u>M</u> enu References
	List and Search Dialog References.	Refs <u>D</u> ialog References
	List and Search String Data References.	Refs <u>S</u> tring Data References
	Print the Disassembler Text.	Disassembler <u>P</u> rint

Contents

General Information About W32Dasm Version 8.x Series

W32Dasm Information

**Do you have the most recent DEMO version of W32Dasm?
For the latest W32Dasm update information.
Visit the W32Dasm WEB page at:**

<http://www.expage.com/page/w32dasm>

THIS IS THE DEMO VERSION 8.xx OF W32DASM

**CERTAIN FEATURES ARE DISABLED AND THE NUMBER OF
OPERATIONS PER SESSION IS LIMITED. TO GET LATEST
PRICES AND DETAILS ON HOW TO ORDER THE FULL
VERSION WRITE URSOFT AT:**

**URSoft
44 Sachem Dr.
Glastonbury Connecticut 06033**

OR Email

urbie@msn.com

**Suggested price of full version is \$75.00 US for Email Delivery and \$80.00 US for
Regular Mail on a 3.5" Floppy Disk**

Above Price Valid until Sept 30,1997

No EuroChecks or Charge Cards Accepted

**US Bank Checks and International Money Orders
Must Be Made Payable To: **P J Urbanik****

If Sending Cash, Please Use Registered Mail

Debugger Tutorial

Tutorial

Tutorial - 1.0 Getting Started

Tutorial - 2.0 Saving The Disassembly Text and Creating a Project File

Tutorial - 3.0 Open an Existing Project File

Tutorial - 4.0 Disassembly Text Navigation

Tutorial - 5.0 Printing/Copying Disassembly Text

[Tutorial - 6.0 Loading a 32 Bit Disassembly into the Debugger](#)
[Tutorial - 7.0 Running, Pausing, & Terminating a Program](#)
[Tutorial - 8.0 Single Stepping a Program](#)
[Tutorial - 9.0 Setting & Activating Program BreakPoints](#)
[Tutorial - 10.0 Attaching to an Active Process](#)
[Tutorial - 11.0 Modifying Registers, Flags, Memory, & Instructions](#)
[Tutorial - 12.0 Exploring Called Modules \(DLLs\)](#)
[Tutorial - 13.0 The WIN API Detail Feature](#)

Commands

[Disassembler Menu](#)

[Project Menu](#)

[Debug Menu](#)

[Search Menu](#)

[Goto Menu](#)

[Execute Text Menu](#)

[Functions Menu](#)

[HexData Menu](#)

[Refs Menu](#)

[Help Menu](#)

Toolbar

[Toolbar Commands](#)

For information on how to use Help, press F1 or select Using Help from the Help menu.

W32Dasm Version 8.x Series General Information:

This is the **Demo Version 8.x** Series of W32Dasm. It may be **copied** for distribution and **downloaded** as shareware. The demo version has some functions disabled and limits the number of operations per session.

Platform Notes:

W32Dasm Version 8.xx Series requires **Windows 95**. Or Windows NT 4.0

It will load and run on Windows NT 3.51 if a Windows 95 or Windows NT compatible IMAGEHLP.DLL file is placed in the same directory as W32Dasm. With out the newer file, an error message will appear during the initial load. Also, some dialog formats are not completely supported by NT 3.5 and some dialog controls will not display properly.

It will also load on Windows 3.11 with W32s but the debugger operation will be unpredictable.

Comments, Questions, & Problems:

If you have any Comments (Good & Bad), Questions About or Problems with W32Dasm Ver 8.x, send them to me via Email. (urbie@msn.com)

If you encounter a problem, please provide as much detail as you can about your computer system, file you were disassembling, operation you were performing, etc. in your description of the problem.

W32Dasm Series 8.xx New Functionality:

* * W32Dasm now runs on Windows NT 4.0. (See Platform Notes Above)

New in Version 8.5

* W32Dasm will now disassemble the Intel MMX instruction set.

New in Version 8.7

* Added a Code Patch Function (Instruction Assembler) to the 32 bit debugger.

Windows 32 Bit PE Format Files:

W32Dasm Version 8.x will disassemble 32 Bit Windows programs that are in the Portable Executable (PE) Format.

For valid Windows 32 Bit PE formats, an assembly code listing will be produced that has header information describing all Imported and Exported functions in the file. **Exported Functions, Imported Functions** and **String Data** references are **color** coded on the screen listing.

Search functions are available to aide in finding text, functions, code, string data, etc.

Imported Functions, Exported Functions and **String Data References** are listed

alphabetically in special List Boxes with a search feature.

See:

[Imports](#)
[Exports](#)
[String Data Reference](#)

A [Goto Program Entry Point](#) command is available to quickly find the program starting point. (Starting points for windows programs are not necessarily at the start of the code listing).

A [Goto Code Location](#) command is available to quickly goto any valid code location reference.

Commands for [Execute Jump](#), [Execute Call](#), [Return From Last Jump](#), and [Return From Call](#) are also available.

[Windows 16 Bit NE Format Files:](#)

W32Dasm Version 8.x will disassemble 16 Bit Windows programs that are in the New Executable (NE) Format.

For valid Windows 16 Bit NE formats, an assembly code listing will be produced that has header information describing all Imported and Exported functions in the file. **Exported Functions, Imported Functions** and **String Data** references are **color** coded on the screen listing.

Search functions are available to aide in finding text, functions, code, string data, etc.

Imported Functions, Exported Functions and **String Data References** are listed alphabetically in special List Boxes with a search feature.

See:

[Imports](#)
[Exports](#)
[String Data Reference](#)

A [Goto Program Entry Point](#) command is available to quickly find the program starting point. (Starting points for windows programs are not necessarily at the start of the code listing).

A [Goto Code Location](#) command is available to quickly goto any valid code location reference.

Commands for [Execute Jump](#), [Execute Call](#), [Return From Last Jump](#), and [Return From Call](#) are also available.

NOTE: W32Dasm Ver 8.x will automatically find **Imported Function Names** if the import DLL files exist either in the **Windows System Directory** or the **Current Directory** of the file being disassembled. If the Import DLL file is **not found** or does **not contain function names**, only the Imported Function **Ordinal** will be displayed.

[Windows 32 Bit LE Format Files:](#)

W32Dasm Version 8.x will disassemble 32 Bit Windows programs that are in the Linear Executable (LE) Format. (ie: Vxd files and others).

For valid Windows 32Bit LE formats, an assembly code listing will be produced that has header information describing the file and its objects. Limited Search functions are available. Commands for execution of jump, return from jump, call, and return from call instructions are also available.

Other File Formats:

W32Dasm Version 8.x can also disassemble the byte data in any any file. If the file opened for disassembly is not in the Windows 32Bit PE, 32Bit LE, or 16Bit NE format, the user will be given the option to disassemble the opened file from a user specified starting byte offset into the file.

NOTE: The user has a choice of treating the data in the file as 32 bit assembly code or 16 bit assembly code when disassembling via a checkbox on the popup dialog box that appears when raw disassembly is the option.

NOTE: W32Dasm automatically detects .COM files and sets the 16 bit option and first code location value to :0001.0100.

NOTE: Disassemblies of files (Other than .COM files disassembled as 16 bit) not in the Windows 32Bit PE, 32Bit LE, or 16Bit NE format will not have Data references.

NOTE: Disassemblies of files not in the Windows 32Bit PE, 32Bit LE, or 16Bit NE format will not have Import or Export references.

Disassembler Menu

The Disassembler menu provides commands for opening files for **disassembly**, **saving** disassembler text and **creating** Project files, **printing** disassembler text, changing printer /screen **fonts**, setting **disassembler options** and **exiting** the application.

Open File to Disassemble

Opens an existing file for disassembly.

To enable the selection of any file extension for opening, select All Files(*.*) from the List Files of Type list box in the Open File Dialog Box.

Save Disassembly Text and Create Project File

Saves the disassembler text to a ASCII file and creates a .hpi Project file. Project files are used to save time when repeatedly opening large files for disassembly. The default filename for the text and project files are the original disassembly filename with an .alf (Assembly List File) extension and .hpi (Hex Project) respectively.

Print Preview

View a sample printout of the current disassembler text.

Print

Sends the disassembler text to a printer. User can select specific pages, lines, or print the whole listing.

Print Setup

Set printer characteristics, ie Landscape, Portrait, etc.

Copy Selected Lines

Copies Selected Disassembler Text Line/s to the Clipboard.

Select Font

Displays Font Select Dialog from which the user can select fonts.

Save Default Font

Sets the currently selected font to the default startup font.

Clear All Trace Marks

Clears all red trace marks from the screen listing.

Disassembler Options

Sets various disassembler default options.

Exit

Exits the W32Dasm application.

Project Menu

Open Project File

Text

Debug Menu

The Debug menu provides commands to **Load** or **Attach To** a 32 bit Executable Process and **DEBUG** by using **Single Step** and **Breakpoint** Commands.

Load Process

Loads a disassembled process into the debugger

Attach to an Active Process

Disassembles an active process, then attaches it to the debugger.

BreakPoint Toggle

Toggles a breakpoint at the Highlighted Code Line. This function is active only when the debugger is active.

Run Process

Runs the process currently loaded in the debugger.

Pause Process

Pauses a process that is running in the debugger

Goto Current Eip

Sets the disassembly to the current instruction pointer. (EIP)

Single Step Thru

Single Steps over calls and Repeat functions.

Single Step Into

Single Steps into Calls and Repeat functions. NOTE: Many API Calls are automatically stepped over by the debugger and cannot be stepped into.

Auto Single Step Thru

Animates the Single Step Thru function.

Auto Single Step Into

Animates the Single Step Into function

Terminate Process

Terminates the process currently loaded in the debugger

Debugger Options

Sets various default options for the debugger.

Search Menu

The Text Search menu provides commands to find and mark selected text.

Find Text

Finds a pattern of text and marks in red on the screen display. This is useful for finding specific functions in the disassembly text. All code location begin with a semicolon (ie :004123CB (32 Bit) or :0004.345D (16 Bit)) to uniquely identify them in searches. Searches can be performed up or down from the current screen location by selecting the appropriate button in the Find Dialog Box. The text Case Match is defaulted OFF but may also be activated in the Find Dialog Box.

Find Next

Repeats search of last find operation. The F3 key can also be used for "Find Next" operations.

Goto Menu

Goto Code Start Sets Disassembled text to the start of the Code Listing. The F7 or Toolbar Button can also be used for this function.

Goto Program Entry Point Sets Disassembled text to Program Entry Point Code. The F8 or Toolbar Button can also be used for this function.

Goto Page

Sets Disassembled text to the Page Selected from the Goto Page Dialog Box. The F9 or Toolbar Button can also be used for this function.

Goto Code Location

Sets Disassembled text to the Code Location Offset Selected (32 Bit) or Code Location Segment.Offset (16 Bit) from the Goto Code Location Dialog Box. The F10 or Toolbar Button can also be used for this function.

Execute Text Menu

The Execute Text Menu provides access to the generic jump, call, and return from call commands.

Execute JumpSets listing to the location specified by the jump instruction that is at the top line of the screen display.

Return From Last JumpSets listing to the location of the last executed jump instruction.

Execute CallSets listing to the location specified by the call instruction that is at the top line of the screen display.

Return From CallReturns listing to the location of the most recently executed call (See Execute Call).

Functions Menu

The Functions Menu provides access to the Import and Export Function List Boxes.

Imports Displays the Imported Functions List Box.

Exports Displays the Exported Functions List Box.

Hex Data Menu

The Hex Data menu provides access to the Data Object/Segments Hex Display and Hex display of the code currently displayed on the screen..

Hex Display of Data Object/Segments

Displays the Data Object/Segments in hexadecimal form.

Hex Display of Code Data Displays the current screen code as Hex data.

Refs Menu

Menu Reference Displays the Menu Reference List Box.

Dialog Reference Displays the Dialog Reference List Box.

String Data Reference Displays the String Data Reference List Box.

Help Menu

The Help menu provides access to the help system and the about dialog.

Contents Help topic contents.

Context Help

Context sensitive help.

About

About W32Dasm.

Exiting

To exit W32Dasm, choose File|Exit from the menu. You will be prompted to save any unsaved disassembly text before exiting.

Printing

There are three commands on the File menu which support printing of disassembled text from W32Dasm. File|Print Setup is used to select and configure a printer device. File|Print Preview displays a special preview window which shows how the text will appear when printed. File|Print allows for printing all or selected pages of the current displayed disassembler text.

File Exit Command

The File|Exit command exits W32Dasm. If you've modified documents without saving, you'll be prompted to save before exiting.

File Open Command

The File|Open command displays the Open a File dialog box so you can select a file to disassemble. If the file selected is not in the Windows 32 bit PE format, 32 bit LE format, or Windows 16 bit NE format, an option to interpret and disassemble the file as raw 32 bit assembly code or raw 16 bit assembly code will be enabled. Use the "List Files of Type" list box to choose the file extensions that will displayed in the file selection box. All Files(*.*) can be chosen to list all files in the selected Directory.

NOTE: The last selected Directory and FileType selected will be the default the next time the File Open Dialog is selected.

[See Tutorial 1.0](#)

Disassembler Print Command

The Print|Print command prints all, selected lines, or selected pages of the disassembler text. Use File|[Print Preview](#) to see how the text will be laid out on printer pages. Use File|[Print Setup](#) to select a printer, and to set printer options.

([See Tutorial 5.0](#)) more more details.

Disassembler Print Preview Command

Disassembler|Print Preview opens a special window that shows how the disassembled text will appear when printed. The preview window shows one or two pages of the active document as they would be laid out on printer pages. If the Printer is set up for Landscape printing, only one page is available in the print preview window. Controls on the window allow you to page through the pages of the text.

Disassembler Print Setup Command

The Print|Printer Setup command displays the Printer Setup dialog box which allows you to select and configure the printer to be used to print documents in the application.

Disassembler Copy Selected Lines Command

The Disassembler|Copy Selected Lines command copies the Selected Disassembly Text to the Windows Clipboard thus allowing the user to Paste this text into any other Windows compatible program.

Text lines are selected in the same way as for printing selected lines of text. (See Tutorial 5.2) for details on how to select lines of text.

Disassembler Save Command

The File|Save command saves the disassembler text to an ASCII text file. The default name of the saved file will be the original name/path of the disassembled file with an “.alf” (Assembler List File) extension. This file can be used in a word processor such as MS Word for further editing, searching and formatting. The default filename/path can be overridden by the user.

Text Search Find Command

The Search|Find command searches the disassembled text for a pattern. The command displays the Find dialog which controls the search process. Options in the dialog determine whether the case of characters is significant, and whether the search should be conducted forwards or backwards through the document. As each match is found, it is highlighted in red in on the screen.

NOTE: Text searches on large files may take a long time -- be patient.

Text Search Next Command

The Search|Find Next command repeats the last Find operation.

Goto Goto Code Start Command

The Goto|Goto Code Start Command sets the disassembled text to the start of the code listing. This is not necessarily the start point for the execution of the subject program. (See Goto Program Entry Point)

Goto Goto Program Entry Point Command

The Goto|Goto Program Entry Point Command sets the disassembled text to the program entry point code. This is the start point for the execution of the subject program. When using a debugger program, a breakpoint may be set here to trap beginning of the program execution.

If the code location (ie. :XXXXXXXX or :XXXX.XXXX) of the program entry point is set to the top line of the display by using the Scroll Bar or pressing the Line Down key, the status bar at the bottom of the screen will give information as to the hex offset in the program file where this code exists. Patching the code byte of this location with a hex editor to a "CC" value will place an "INT 3" instruction which will cause the program to "break" automatically when used with a debugger. Of course the original value of the patched byte would then have to be reinstated to continue the debug.

NOTE: Program Entry Points are only valid for 32 bit PE and 16 bit NE files. This command is disabled if no valid PEP exists.

Goto Goto Page Command

The Search|Goto Page Command sets the disassembled text to the page selected from the Goto Page Dialog. The Current Page is displayed when the Dialog is initialized. Page numbers entered that are greater than the total pages available will goto to the last page.

Goto Goto Code Location Command

The Goto|Goto Code Location Command sets the disassembled text to the Code Location Offset (32 Bit) or Code Location Segment.Offset (16 Bit) selected from the Goto Code Location Dialog. The Last Code Location goto point that was entered is displayed when the Dialog is initialized. The initialization value for the first time the Dialog Box is opened after a file is disassembled is the lowest valid Code Location goto Value. Code goto points entered that are greater than or less than the lowest and highest valid values are automatically clamped. Valid in-range values that are entered that are not valid Code Locations in the listing are automatically set to the nearest lowest valid value.

Functions Imports

The Functions|Imports displays a List Box with all the disassembled programs identified Imported Functions listed alphabetically.

Imported Functions are functions that are required to run a program but reside in files other than the subject program file. Imported Functions in the Disassembled Listing, are references to Calls to the function. There can be more than, and usually is, one reference. Imported Functions are usually the result of calls made to Dynamic Link Library files (DLLs).

If there are no Imported Functions identified, the command is disabled. To search for an Imported Function in the disassembled text, Double Click the Left Mouse Button on the desired function in the List Box.

NOTE: If a Imported Reference is not found, it is most likely due to the fact that the disassembler could not properly decode the location of the import reference due to a mixture of data and code in the object being decoded.

NOTE: Files that are not in the Windows NE or PE format will not have Function Import Data.

Functions Exports

The Functions|Exports displays a List Box with all the disassembled programs identified Exported Functions listed alphabetically.

Exported Functions are functions that are available to other programs. Exported Functions in the Disassembled Code Listing show the actual function code. There should be only one reference in the Code portion of the Disassembler Listing per Exported Function. DLL files usually have many exported functions. Program {exe} files usually have few if any.

If there are no Exported Functions identified, the Command is disabled. To search for an Exported Function in the disassembled text, Double Click the Left Mouse Button on the desired function in the List Box.

NOTE: If a Exported Reference is not found, it is most likely due to the fact that the disassembler could not properly decode the location of the export reference due to a mixture of data and code in the object being decoded.

NOTE: Files that are not in the Windows NE or PE format will not have Export Data.

Execute Execute Jump

The Execute|Execute Jump command sets the screen listing to the location specified by the jump instruction that is on the top line of the screen listing. All direct jumps to the code object are executed. The command is automatically enabled if a valid jump instruction is in the top line position. In the case that the jump location is invalid, a message will be posted after a unsuccessful attempt is made.

Execute Return From Last Jump

The Execute|Return From Last Jump command sets the screen listing to the location of the last executed Jump. The Hot Keys for this function is the Left Arrow Cursor Key. This command is only enabled if a valid jump instruction was executed.

Execute Execute Call

The Execute|Execute Call command sets the screen listing to the location specified by the call instruction that is on the top line of the screen listing. All direct calls to the code object are executed. The command is automatically enabled if a valid call instruction is in the top line position. In the case that the call location is invalid, a message will be posted after a unsuccessful attempt is made. After a call command is executed, the Execute|Return command can be used to return to the original call location. Call commands can be stacked, and Return commands will set locations in the order of the calls.

Execute Return From Call

The Execute|Return From Call command sets the screen listing to the location of the last executed Execute|Execute Call command. Call commands are stacked, and Return commands will set locations in the reverse order of the calls. The command is automatically enabled when valid call commands are executed.

Data String Data References

The Data|String Data References displays a List Box with all the disassembled programs identified string data references listed alphabetically.

If there is **no** String Data identified, the Command is **disabled**. To **search** for a string data item reference in the disassembled text, **Double Click** the Left Mouse Button on the desired string data text in the List Box.

NOTE: Long String Data References are **abbreviated** in the String Data List Box but are **fully displayed** as wrapped text in the disassembler listing.

NOTE: Only Files that are in the Windows **NE** format, **PE** format, or **.COM** files disassembled as 16 bit, are queried for String Data. This command is **disabled** for all other file types.

Data Hex Display of Data Object Command

The Data|Hex Display of Data Object displays a List Box with all the Data Object/Segment data in hexadecimal format. The data is grouped into 1024 byte pages which are selectable from the list box. The number of pages is dependent on the size of the data Object/Segment. This command is enabled after a valid disassembly is executed.

NOTE: If there is more than one Page of data in the Data Object/Segment, the Page Select buttons are automatically enabled.

NOTE: For NE files that have more than one Data Segment, Segment Select buttons are automatically enabled.

Code Hex Display of Code Data Command

The Data|Hex Display of Code Data displays a List Box with the current page code data in hexadecimal format. The starting location is the same as the code location specified at the top line of the screen. This command is disabled if a valid code location is not available on the top line of the screen.

NOTE: If a page reference is at the top of the screen, the next line will activate this command if it is a valid code location.)

Window Help table of contents

The Help|Contents displays the help contents page.

Font Select Font Command

The Font|Select Font Command displays a font selection Dialog Box from which the user may select a text font. W32Dasm uses the **Courier New, Regular** Style, Size **8**, font as a starting default. This default can be changed using the Save Default Font. command.

Font Save Default Font Command

The Font|Save Default Font Command sets the current selected font (See [Select Font](#)) as the program default font. W32Dasm uses the **Courier New, Regular** Style, Size **8**, font as a starting default.

Window Context Sensitive Help

The Help|Context Help creates a cursor that calls up help when clicked over any toolbar or menu item. The context help cursor is automatically deleted upon return from help or it can be canceled by using the right mouse button or pressing the keyboard escape key {Esc}.

Window About W32Dasm

Displays dialog box with information about the W32Dasm Application.

Load Process

Loads a disassembled Executable Process into memory and halts execution at the process's Program Entry Point. Only executable processes can be loaded.

Non executables, such as .dll files can be debugged by first loading or attaching (See [Attach to an Active Process](#)) to an executable process that calls the dll, and then opening the appropriate dll disassembly.

[See Tutorial 6.0](#)

Attach to an Active Process

Attaches a disassembled Executable Process to a currently active process. Only 32 bit processes can be attached to. Also, certain processes that are part of the Windows operating kernel are prohibited from being attached to.

Non executables, such as .dll files can be debugged by first loading (See [Load Process](#)) or attaching to an executable process that calls the dll, and then opening the appropriate dll disassembly.

[See Tutorial 10.0](#)

BreakPoint Toggle

Toggles a breakpoint at the code line in the highlight bar. The breakpoint is noted by a bright yellow highlight when active and dark blue when deactivated (See Below) The F2 key can also be used to toggle a breakpoint.

Breakpoints can be activated/deactivated from the breakpoint list box by selecting the breakpoint address and then right clicking.

[See Tutorial 9.0](#)

Run Process

Runs a process that is currently loaded in the debugger. The process will run until a breakpoint is reached or the Pause or Single Step functions are called.

NOTE: Running processes may not stop immediately when the Pause or Single Step functions are called. Often a Window message must be generated for this to occur since many programs run in a **Message Loop** when idle. Moving the mouse in the process window being debugged will generate a Window message and cause the process to **break**.

[See Tutorial 7.0](#)

Pause Process

Pauses a process running in the debugger

NOTE: Running processes may not stop immediately when the [Pause](#) or Single Step functions are called. Often a Window message must be generated for this to occur since many programs run in a “[Message Loop](#)” when idle. Moving the mouse in the process window being debugged will generate a Window message and cause the process to “break”.

[See Tutorial 7.0](#)

Goto Current Eip

Resets the disassembly to the current instruction pointer when a process is paused or at a breakpoint condition..

[See Tutorial 9.2](#)

Single Step Thru

Single Steps over Call and Repeat functions. Use this function if you want to bypass going thru the code of a Called function or Iterations of a Repeat function.

[See Tutorial 8.0](#)

Single Step Into

Single Steps into Call and Repeat functions. Use this function if you want to examine the code of a Called function or see the results of iterations of a Repeat function.

[See Tutorial 8.0](#)

Auto Single Step Thru

Animates the Single Step Thru Function.

[See Tutorial 8.0](#)

Auto Single Step Into

Animates the Single Step Into Function.

[See Tutorial 8.0](#)

Terminate Process

Terminates the process currently loaded in the debugger.

[See Tutorial 7.0](#)

Debugger Options

Sets defaults for various debugger options.

Clear All Trace Marks

As a process is debugged, code lines that have been single stepped to and codes lines that have been stopped at due to breakpoints are all marked (traced) in red. This function will clear all the red trace marks from the disassembly screen display..

Disassembler Options

Sets defaults for various disassembler options..

[See Tutorial 1.0](#)

Open Project File

Opens a project file that was previously created by the user. Project files are automatically created when the user saves the disassembly text. Project files are useful for saving disassembly time when a file is repeatedly being examined by the user.

Menu References

The Menu References displays a List Box with all the disassembled programs identified menu references listed alphabetically.

If there are **no** Menu References identified, the Command is **disabled**. To **search** for a menu reference in the disassembled text, **Double Click** the Left Mouse Button on the desired menu reference text in the List Box.

Menu References

The Dialog References displays a List Box with all the disassembled programs identified dialog references listed alphabetically.

If there are **no** Dialog References identified, the Command is **disabled**. To **search** for a dialog reference in the disassembled text, **Double Click** the Left Mouse Button on the desired dialog reference text in the List Box.

W32Dasm Ver 8.xx Series Debugger Tutorial

Preface

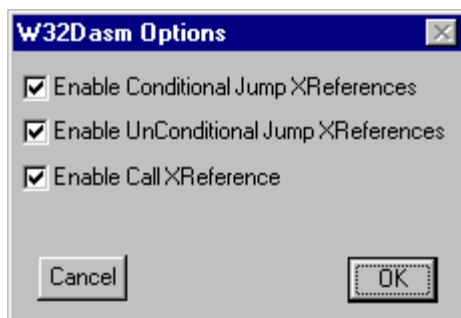
This tutorial will take you step by step thru a typical debugging session using the W32Dasm Ver 8.xx Series Software. The best way to learn is by doing and it is suggested that you follow this tutorial by actually performing the example steps on your computer.

NOTE: Your computer screen **Resolution** should be set to **1024 x 768 x 256** colors at a minimum to provide enough comfortable window viewing space while running the W32Dasm debugger.

1.0 Getting Started (Disassemble File Calc.exe)


- 1.1 Start W32dasm.
- 1.2 Select "**Disassembler Options**" item from the Disassembler Menu.

A dialog Box will appear



which has enable option check boxes for Call Cross References, Conditional Jumps, and Unconditional Jumps. Enabling or disabling these options will will include or not include cross references in the disassembly text on each line of code that is referenced by a call or jump instruction.

NOTE: Enabling the cross reference features will substantially increase the amount of time it takes to disassemble a file. However if the Project File (Exercise 2.0 & 3.0) feature is used for subsequent disassemblies, this will not be a significant factor.

- 1.3 Select the "**Open File to Disassemble**" item from the **Disassembler** Menu **OR** Press the  toolbar button.
- 1.4 Select the Windows 95 file "Calc.exe" which should be found in your WINDOWS Directory and Press Open to begin disassembly of the file.

W32Dasm will now disassemble Calc.exe. Note the time it takes for the

disassembly process for comparison with the time that will be noted in Exercise 3.0.

NOTE: If the disassembly text appears as "Garbage Characters" on the screen, you need to select and save a default font which works on your system. To do this, select Font from the Disassembler Menu. A secondary menu choice will appear that has both a *Select Font* and *Save Default Font Choices*. Use the *Select Font* to find a font which works best with your system. When you have the proper font selected, use the *Save Default Font* to make it your automatic default.

2.0 Save The Disassembly Text and Create A Project File

In this exercise you will save the disassembly text to an ASCII ".alf" file and create a Project ".hpj" file which can be used to quickly input the disassembled text into W32Dasm without going thru the process of disassembly. The Project files are especially useful when you are repeatedly analyzing large files which take a long time to disassemble.

2.1 Select "*Save Disassembly Text File and Create Project File*" from the *Disassembler* Menu OR press the  toolbar button.

2.2 The Save As Dialog Box will appear with the default save filename set to the disassembled file name with an .alf extension. The default Project File Directory **DRIVE:\W32DASM DIRECTORY NAME\WPJFILES** should also be set.

It is recommended that you use the default .alf file name and default project directory for your files. When the .alf file is created, a .hpj file will also be created automatically. These two files should always reside in the same directory in order to work properly. Also, the .alf and .hpj files should not be modified by the user. If you want to edit the disassembler text that resides in the .alf file, make a copy and rename the file.

Press the OK button to save the disassembly and project files.

3.0 Open an Existing Project File


In this exercise you will open the Project ".hpj" file which was saved in Exercise 2.0.

3.1 Select "*Open Project File*" from the *Project* menu. The file calc.hpj should be a selection in the Dialog Box. Select this file, and press OK.



3.2 The disassembly of calc.exe will now load. You will notice that the time to load the project file is much faster than the time it took to initially generate the disassembly in Exercise 1.0.

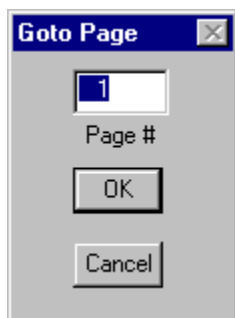
4.0 Disassembly Text Navigating


In this exercise you will learn how to navigate thru the disassembly text listing using W32Dasms search and goto functions.

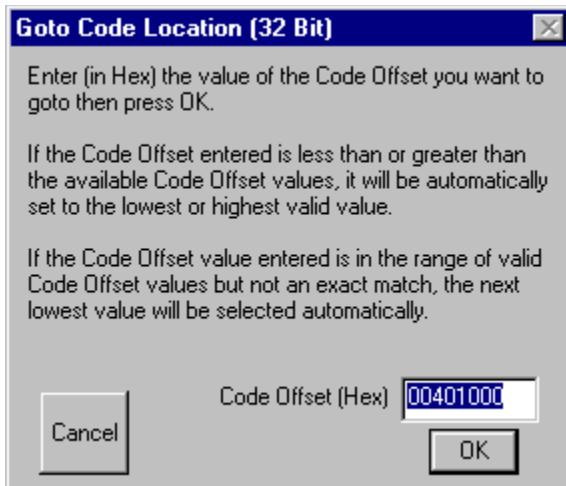
- 4.1 **Goto Code Start:** By pressing the  toolbar button (4th from the left on the toolbar) OR selecting the "**Goto Code Start**" from the Goto Menu OR Pressing **Ctrl S**, the disassembly listing will be set to the beginning of the code listing. The **light blue highlight bar** is where the listing is focused. The user can change the position of the highlight bar by double clicking on any text line or by holding the **Shift** Key while pressing the **Up** or **Down** arrow keys. It is suggested to set the highlight bar somewhere near center of the display in order to view the code before and after the selected line.

Note that the beginning of the code listing is not necessarily where the code begins execution. The execution start point is called the **Program Entry Point** and will be discussed next.

- 4.2 **Goto Program Entry Point:** By pressing the  toolbar button (5th from the left on the toolbar) OR selecting the "**Goto Program Entry Point**" from the Goto Menu OR Pressing **F10**, the disassembly listing will be set to the Programs Entry Point. This is where the programs main execution begins. It is also where the debugger will automatically stop when it first is loaded with a program.
- 4.3 **Goto Page:** By pressing the  toolbar button (6th from the left on the toolbar) OR selecting the "**Goto Page**" from the Goto Menu OR Pressing **F11**, a dialog box will appear enabling the user to enter a page number to goto in the disassembly listing.

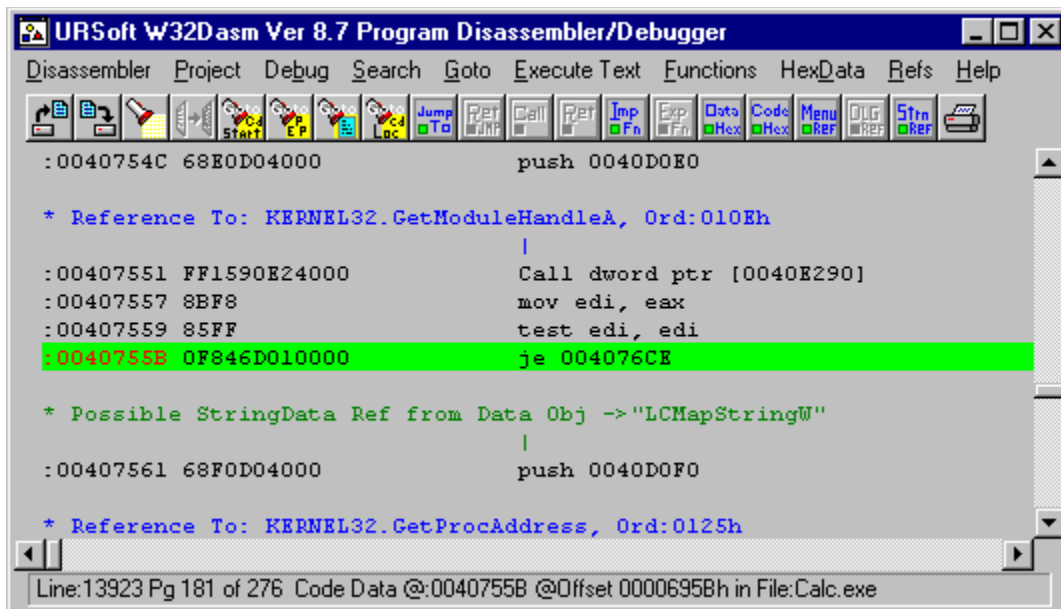


- 4.3 **Goto Code Location:** By pressing the  toolbar button (7th from the left on the toolbar) OR selecting the "**Goto Code Location**" from the Goto Menu OR Pressing **F12**, a dialog box will appear enabling the user to enter a code address to goto in the disassembly listing.



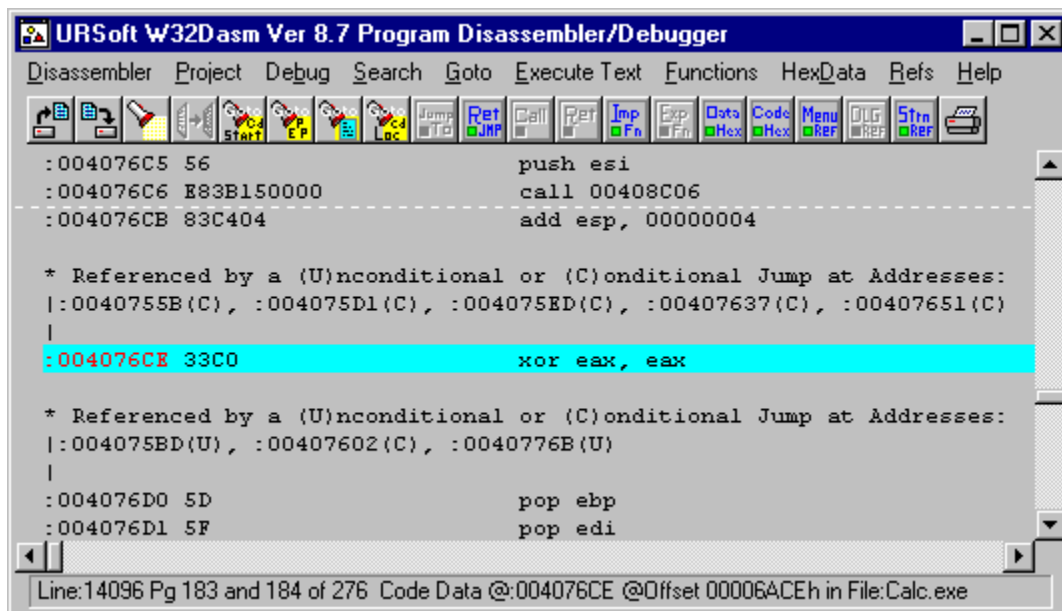
The address format is in hexadecimal. With calc.exe loaded in the disassembler, set the location in the dialog box to **40755B** and Press OK. The listing should now be set at code address :0040755B which is a je 004076Ce instruction. Notice also that the highlight bar turns green. The green color indicates that the je instruction is a valid jump instruction to execute a **TEXT JUMP** (See Exercise 4.4).

- 4.4 **Execute Text Jump:** The Execute Text Jump function is only active when a valid jump instruction is positioned on the highlight bar. The highlight bar will turn green and the **Jump To** toolbar will activate when a valid condition exists. If exercise 4.3 was done properly you should now be at code location 0040755B in calc.exe and the highlight bar should be green.






By pressing the **Jump To** toolbar button (8th from the left on the toolbar) **OR** selecting the **"Execute Jump"** item from the Execute Text Menu **OR** pressing the **Right Arrow** key, the disassembly listing will goto the location specified by

the jump instruction. In this example the listing should goto code location :004076CE which is a xor eax, eax instruction.



To return to the original jump location see Exercise 4.5.

4.5 **Return From Last Jump:** The Return From Last Jump function is only active when a Execute Jump action was performed. (See Exercise 4.4). The  toolbar will be activated when this condition exists. By pressing the  toolbar button (9th from the left on the toolbar) **OR** selecting the "Return From Last Jump" item from the Execute Text Menu **OR** pressing the **Ctrl Left Arrow** key, the disassembly listing will return to the location of the last executed text jump.


4.6 **Execute Text Call:** The Execute Text Call function is only active when a valid call instruction is positioned on the highlight bar. The highlight bar will turn green and the  toolbar will activate when a valid condition exists. Use the goto code location function to set the listing to code address 0040751D. The listing should now be set at code address :0040751d which is a call 004073D4 instruction. The highlight bar should also be green and the Call toolbar button should be active.

```

|:0040749A(C) , :004074A9(C)
|
:0040750E 66C1EE04      shr si, 04
:00407512 6A00             push 00000000
:00407514 FF750C          push [ebp+0C]
:00407517 0FBFF6         movsx esi, si
:0040751A FF7508          push [ebp+08]
:0040751D E8B2FEFFFF     call 004073D4
:00407522 DD5DF8         fstp qword ptr [ebp-08]
:00407525 83C40C         add esp, 0000000C
:00407528 81EEFE030000   sub esi, 000003FE

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040748C(U) , :0040750C(U)

```

By pressing the  toolbar button (10th from the left on the toolbar) **OR** selecting the **"Execute Call"** item from the Execute Text Menu **OR** pressing the **Right Arrow** key, the disassembly listing will goto the location specified by the call instruction. In this example the listing should goto code location :004073D4 which is a push ebp instruction.



```

|:004073AB(U) , :004073B6(U) , :004073C1(U) , :004073CC(U)
|
:004073D3 C3             ret


* Referenced by a CALL at Addresses:
|:00407501 , :0040751D
|
:004073D4 55             push ebp
:004073D5 8B44240C       mov eax, dword ptr [esp + 0C]
:004073D9 8BEC          mov ebp, esp
:004073DB 83EC08        sub esp, 00000008
:004073DE 8B4D08        mov ecx, dword ptr [ebp+08]
:004073E1 8945FC        mov dword ptr [ebp-04], eax
:004073E4 8B4510        mov eax, dword ptr [ebp+10]

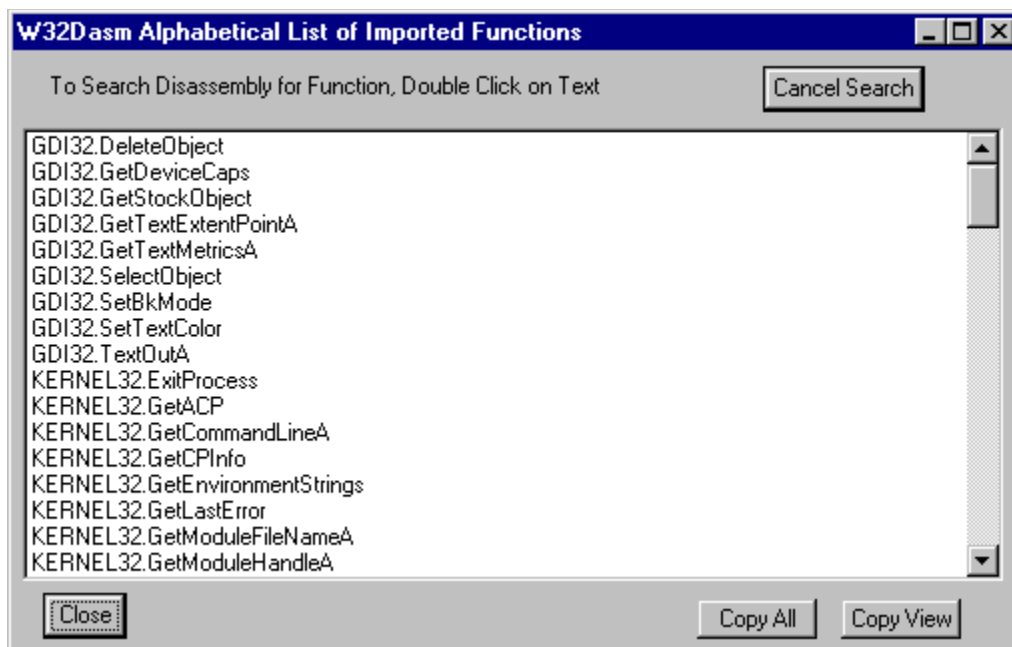
```

To return to the original call location see Exercise 4.7.

4.7 **Return From Last Call:** The Return From Last Call function is only active when a Execute Call action was performed. (See Exercise 4.6). The  toolbar will be activated when this condition exists. By pressing the  toolbar button (11th from the left on the toolbar) **OR** selecting the **"Return From Call"** item from the Execute Text Menu **OR** pressing the **Left Arrow**


key, the disassembly listing will return to the location of the last executed text call.

- 4.8 **Finding Imported Functions:** To search the listing for imported functions press the  toolbar button (12th from the left on the toolbar) **OR** select "**Imports**" item from the Functions Menu. A dialog box



will appear which lists all the import function references found by W32Dasm in the code. By double clicking on any of these items, the disassembly will be set to the location of the function reference. NOTE: there can be more than one reference for an imported function in the listing. Double clicking repeatedly on the same function will cycle thru all the found references for that function. In this example, double click on the first item in the list which should be GDI32.DeleteObject. The listing should go to code location :00402ABF which is a call instruction.


The Import Reference dialog box also has **Copy View** and **Copy All** buttons if you wish to transfer the visible or entire contents of the list box to a text editor of your choice. After pressing the copy button, go to your text editor and use the paste function to transfer the text.

- 4.9 **Finding Exported Functions:** To search the listing for exported functions press the  toolbar button (13th from the left on the toolbar) **OR** select "**Exports**" item from the Functions Menu. A dialog box will appear which lists all the export function references found by W32Dasm in the code. By double clicking on any of these items, the disassembly will be set to the location of the function reference.

Note: Calc.exe does not have any exported functions therefore the  is

not active. Try disassembling a .DLL file which should always have exported functions.

The Export Reference dialog box also has **Copy View** and **Copy All** buttons if you wish to transfer the visible or entire contents of the list box to a text editor of your choice. After pressing the copy button, go to your text editor and use the paste function to transfer the text.

4.10 **Finding Other References:** To search the listing for **MENU**, **DIALOG** or **STRING DATA** references press the ,



, or




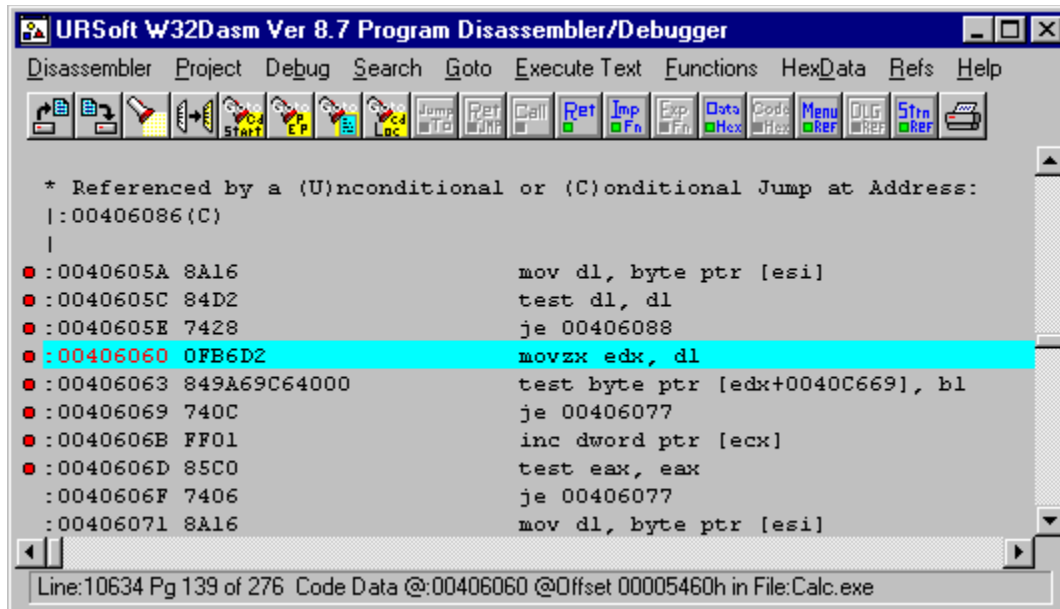
, toolbar button **OR** select the desired item from the item from the Reference Menu. A dialog box will appear which lists all the references found by W32Dasm in the code. By double clicking on any of these items, the disassembly will be set to the location of the reference. **NOTE:** there can be more than one reference for any reference item in the listbox. Double clicking repeatedly on the same reference will cycle thru all the found references.

Each Reference dialog box also has **Copy View** and **Copy All** buttons if you wish to transfer the visible or entire contents of the list box to a text editor of your choice. After pressing the copy button, go to your text editor and use the paste function to transfer the text.


In this exercise you will learn how to navigate thru the disassembly text listing using W32Dasms search and goto functions.

5.0 Printing/Copying Disassembly Text

- 5.1 **Standard Printing:** By pressing the  toolbar button (the last one on the toolbar) **OR** selecting "**Print**" from the Disassembler Menu, a standard Windows print dialog box will appear which enables the user to print selected pages or all of the disassembly listing.
- 5.2 **Printing/Copying Selected Lines of Text:** W32dasm also allows printing or copying of individual lines of disassembler text. This is accomplished by left clicking on the far left margin of the line of text you wish to print. A small red dot should appear in the left margin. To select multiple lines, left click again on the other extreme of the range of lines you wish to print while holding the **SHIFT** key. A series of red dots indicating the range of selected lines will appear in the left hand margin.



To Print:

Select **Print** via the toolbar  or Disassembler Menu and select be sure the **Selection** checkbox is checked. Press OK to print.

To Copy:

Select **Copy Lines of Text** via the toolbar  or Disassembler Menu. You can now use the Paste function in any Windows compatible program to transfer the copied text.

6.0 Loading a 32 Bit Disassembly into the Debugger.

In this exercise you will load and initialize calc.exe into the debugger.

- 6.1 Disassemble the program calc.exe.
- 6.2 Select the "**Load Process**" item from the Debug Menu **OR** Press **Ctrl L**. A dialog box will appear which allows inputting an optional command line parameter to the program being loaded. You may enter optional command line text at this time. For now, just Press the **LOAD** button.

Calc.exe will now load into the debugger and the main W32Dasm window will move and resize while two new debugging windows (See Figs 1 & 2) are be created. These Windows will be referred to as the **Lower Left Hand Debugger Window** and **Lower Right Hand Debugger Window**. After initializing the calc.exe program the program and disassembly listing will be halted and set at the Program Entry Point code location. For calc.exe the PEP

should be set to Code Address :0040534E.

The **Lower Left Hand Debugger Window** has various List boxes that contain CPU Register Data, CPU Flag Data, Breakpoints, Active DLLs, Code Segment Registers, Trace History, Event History, and User Set Memory Addresses, and Data Displays. Data Display 1 has four data format buttons (DWORD, WORD, BYTE, and Instruction) from which you can choose the way data is displayed. The source of the data is selected by the buttons to the left of the display. Data Display 2 shows all four data formats at once and also has a button to select the source address from the Data Display 1. Two user input address boxes are also available to allow the user to choose specific memory addresses to look at. This window also has some debugger function buttons which will be explained later in this tutorial.

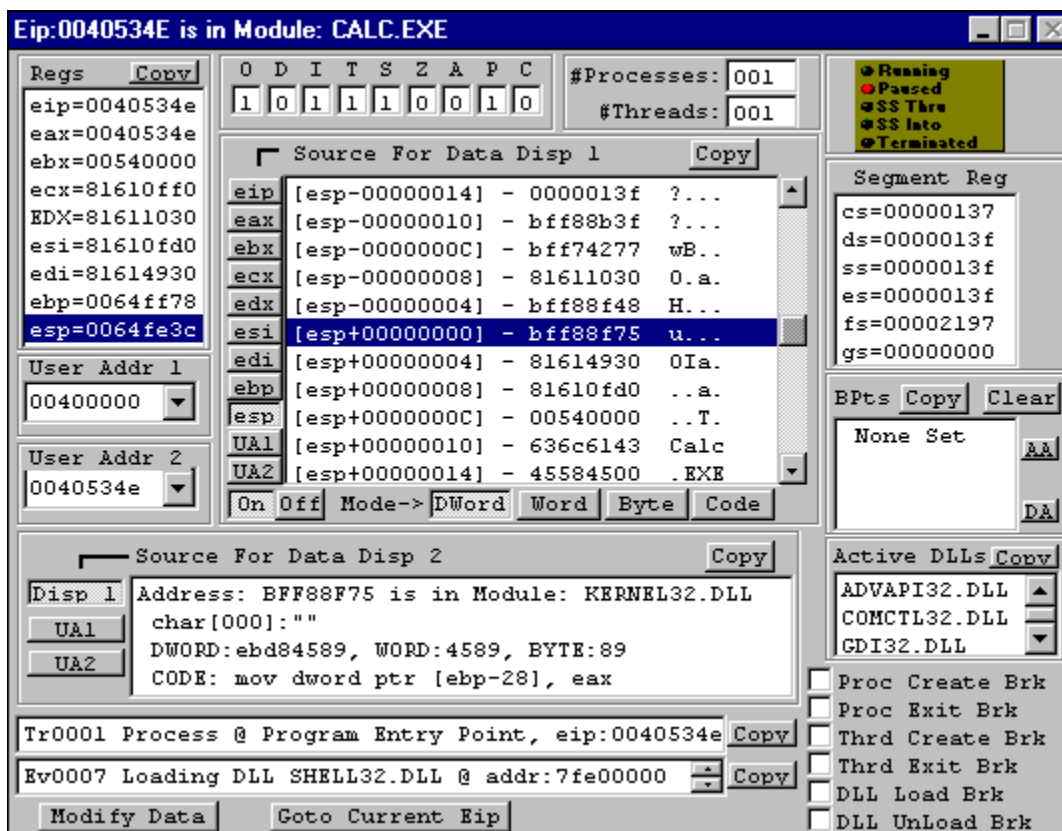


Fig 1. **Lower Left Hand Debugger Window**

The **Lower Left Hand Debugger Window** has a secondary code display plus various debugger control buttons which will be explained later in this tutorial.

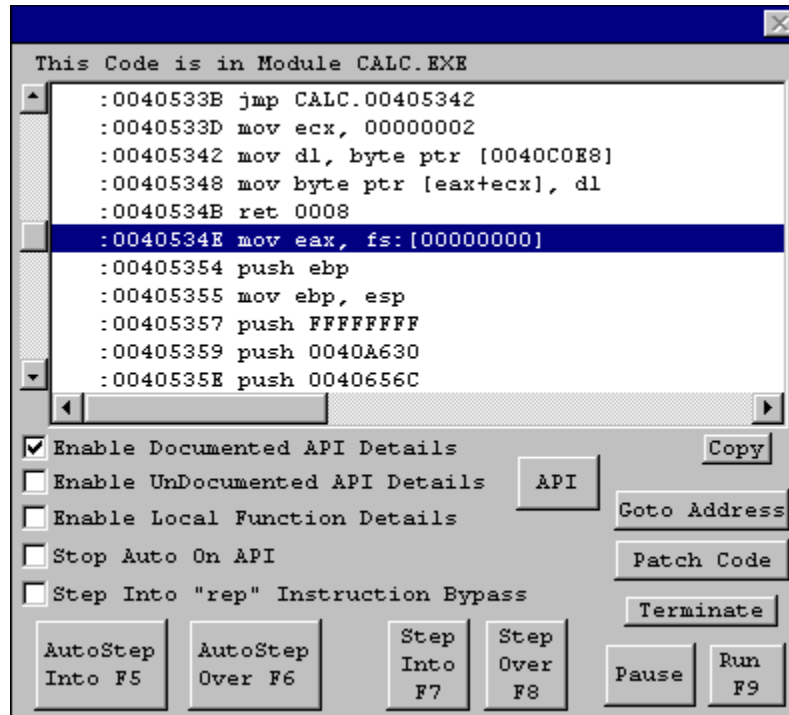


Fig 2. Lower Right Hand Debugger Window

You are now ready to use the debugger.

7.0 Running, Pausing & Terminating a Program.

In this exercise you will run calc.exe from the debugger, pause it, then terminate it.

- 7.1 To **RUN** calc.exe from the debugger, Press the **RUN** button on the lower right hand Debugger Window **OR** Press **F9**. The calc.exe program window should appear and be fully functional. **Memory, CPU Register and Flag** data is displayed in the **Lower Left Hand Debugger Window** along with various options on how data is displayed. Active DLLs, Breakpoint info, Trace History, Event History and Debugger Status is also displayed in this window.

To **PAUSE** the calc.exe program, Press the **Pause** button on the **Lower Right Hand Debugger Window** **OR** Press the **Space Bar**. The action will stop program execution somewhere in the programs message loop if the program was idle at the time of the pause. Exactly where the program pauses in the code is highly dependent on the program and the function it is performing when it is paused. Once a program is paused, you can use the single step debugger features (See Exercise 8.0).

To **TERMINATE** the calc.exe program you can use the programs normal exit procedure or you can Press the **Terminate** button on the **Lower Right Hand Debugger Window** **OR** Press **Ctrl T**. When terminating the program using the normal exit procedure, the Debugger Windows will remain open and display the last debugger event data which is pertinent to how the program exits. When using the **Terminate** feature of the debugger, you will be

given a warning message about the termination from which you can cancel. If you choose to terminate, the program will exit and the debugger windows will automatically close.

8.0 Single Stepping a Program.

In this exercise you will use the four single step functions of the debugger.

- 8.1 Reload the calc.exe program into the debugger.
- 8.2 After the program loads and is stopped at the Program Entry Point, you can execute each program instruction one at a time by Pressing the **Single Step Into** or **Single Step Over** buttons on the **Lower Right Hand Debugger Window**. The **F7** and **F8** keys can also be used. The difference between Single Step Into and Single Step Over is that Single Step Into will execute every instruction in sequence whereas Single Step Over will "jump" over repeat functions such as rep movsb, and also jump over code in Call functions. NOTE that many API functions will be jumped over by default no matter if Single Step Into or Over is used in Windows 95. Windows NT will step thru most all APIs. To avoid stepping into repeat instructions while still using the Single Step Into buttons, you can check the box marked **Step Into rep Instruction Bypass** on the **Lower Right Hand Debugger Window**.
- 8.3 As you single step thru the program you can observe changes to CPU registers and memory as each instruction is executed.
- 8.4 The **Auto Single Step Into (F5)** and **Auto Single Step Over (F6)** buttons Animate the single step functions. To stop the Auto functions you can either press the **Pause**, **Single Step Into**, **Single Step Over**, or **Run** buttons.

9.0 Setting & Activating Program Breakpoints

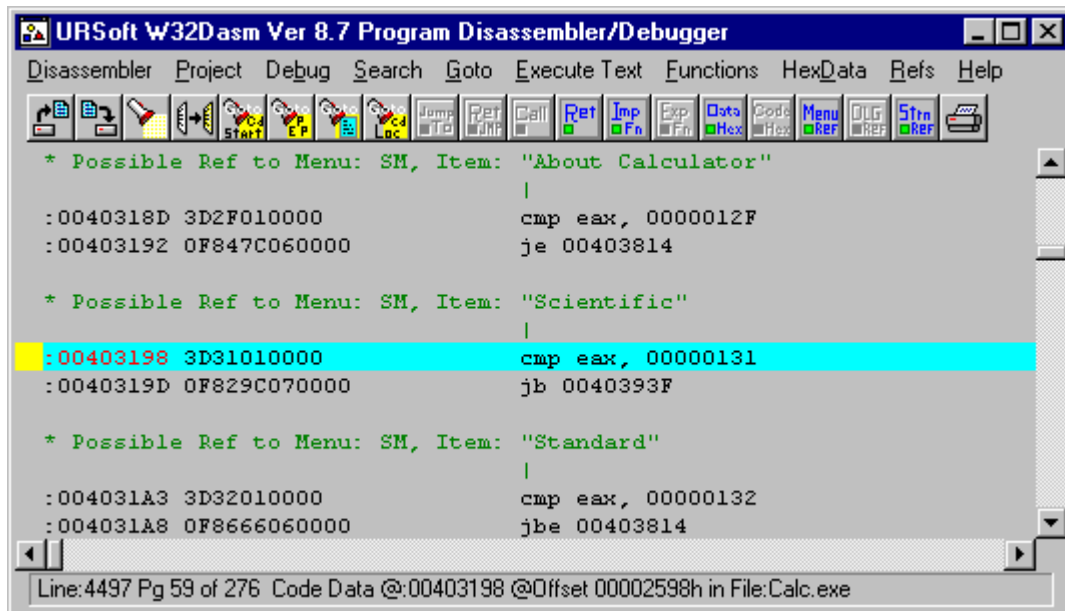
In this exercise you will learn how to set breakpoints and activate and deactivate them.

- 9.1 Load the program calc.exe into the debugger.
- 9.2 Using the goto code location function set the disassembly to code location 403198. This location has a cmp eax, 00000131 instruction in it. You could also have gotten to this instruction by using the Menu Reference search function by double clicking on the **"Menu: SM, Item: "Scientific"** ref. (See Tutorial 4.10).

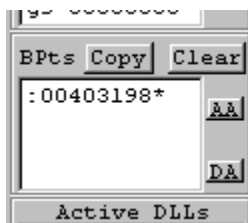
This code is part of the routine that determines what menu function is selected from the calc.exe **VIEW** Menu. While the highlight bar is on the code line for address :00403198, you can set a program breakpoint by Pressing the **F2** key **OR** holding the **Ctrl Key** while **Left Clicking** the mouse in the far left hand margin of the code line. Using the mouse does not require that the desired code line be in the highlight bar.

When the breakpoint is set the far left hand margin will turn Yellow,

indicating it is a breakpoint.



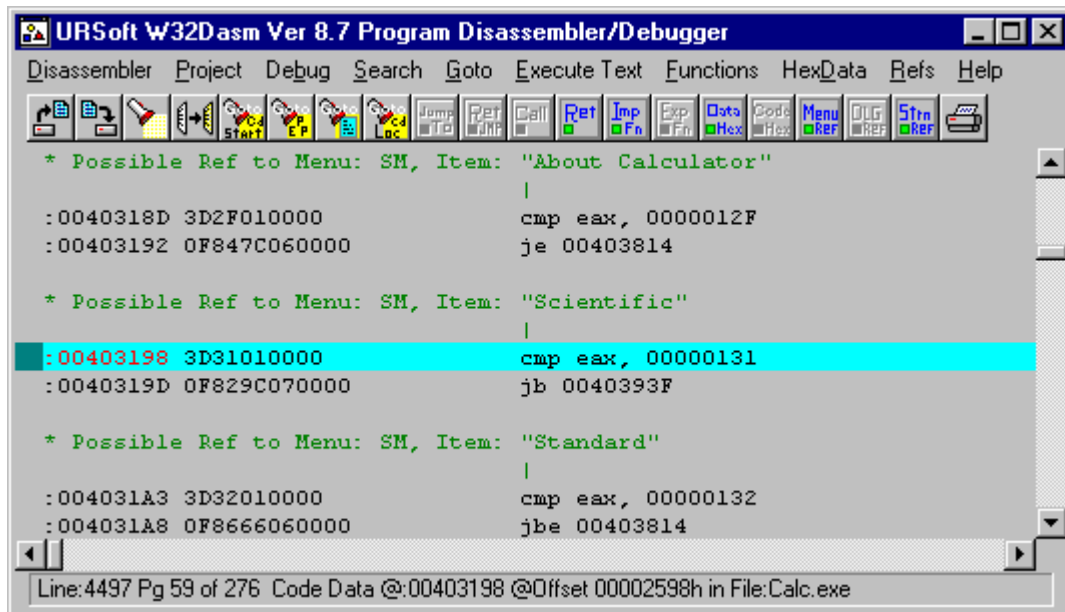
If the breakpoint line is not in the highlight bar, the whole line will be displayed as yellow. Also note that the Breakpoint Address is now displayed in the Breakpoint display window in the lower left hand debugger window. It should have a * symbol next to it which indicates it is an active breakpoint.



Pressing the **F2** Key while a breakpoint line is in the highlight bar will remove the breakpoint. Also, the mouse function can be used to toggle a breakpoint on and off by left clicking the appropriate code line while holding the Ctrl Key. For the next step, leave the breakpoint set at code location :00403198. Reset the disassembly listing to the current instruction (which is the Program Entry Point in this case) by pressing the **Goto Current Eip** button which is at the bottom of the lower left hand debugger window. This button provides a quick way to get back to the current instruction after navigating around the disassembly listing.

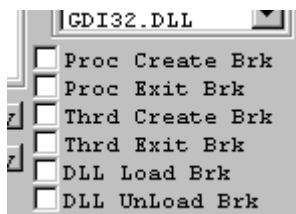
- 9.3 Now that a breakpoint is set, Press the **RUN** button (**F9**) to start calc.exe running. Now select the "**Scientific**" item from the calc.exe **View** Menu. The debugger should now break at code location :00403198 and halt program execution.
- 9.4 Breakpoints can also be left at any address in a inactive state. This is accomplished by selecting the breakpoint address in the breakpoint display list with the mouse and right clicking on the selection. When a breakpoint is deactivated, the * symbol will disappear from the address

in the list and the disassembly listing line will turn from yellow to dark green.

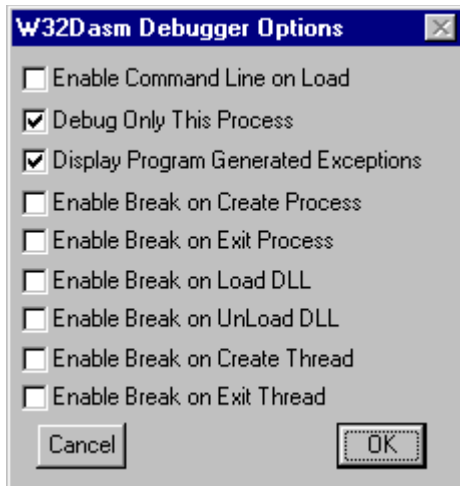


The breakpoint can be reactivated by repeating the above procedure. If you need to deactivate all the breakpoints at once you can press the **DA** button which is just to the right of the breakpoint display list box. Press the **AA** button to activate all the breakpoints. If you want to delete (Clear) all breakpoints, press the **Clear** button. A warning message will appear to verify the Clear action.

- 9.5 **GOTO BREAKPOINT LOCATION:** You can goto breakpoint locations quickly in the disassembly listing by **Left Double Clicking** the mouse on the Breakpoint Address in the debugger window Breakpoint listbox. The breakpoint can be Active or Not Active for this feature to work.
- 9.6 **AUTOMATIC BREAKS ON EVENT:** You can have the debugger automatically break the program execution for certian events such as when a **DLL is Loaded or Unloaded**, When a **Thread is Created or Exited**, or when a **Process is Created or Exited**. This is accomplished by checking the appropriate checkboxes on the Lower Left Hand Debugger Window.



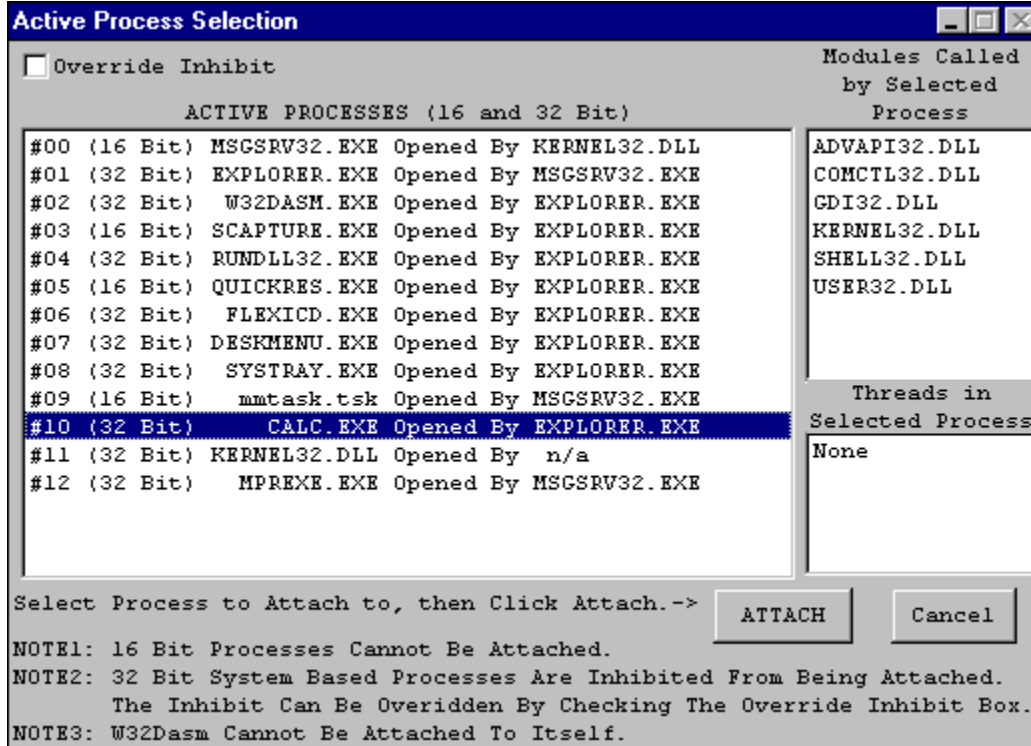
These check boxes can be set to user defaults by selecting the **Debugger Options** item from the Debug Menu.



10.0 Attaching to an Active Process.

In this exercise you will learn how to attach the debugger to a process that is already running in Windows.

- 10.1 Start calc.exe running in Windows
- 10.2 Select the "**Attach to an Active Process**" item from the Debug Menu **OR** Press **Ctrl L**. A dialog box



will appear which lists all of the active processes currently running on your computer. This list contains references to both 16 bit and 32 bit

programs. Only 32 bit programs can be attached to. Also, some 32 bit programs are inhibited from being attached to because they are Windows system critical and attaching the debugger to them may cause the computer to crash. However if you want to play explore debugging these programs, there is an Override Inhibit checkbox in the upper left hand corner of the Dialog that will allow you to attach to these programs when the box is checked. See the **WARNING** below.

The Attach Dialog box also shows what Modules (DLLs) each running process has called and what Threads each process has created as you select a process from the main list.

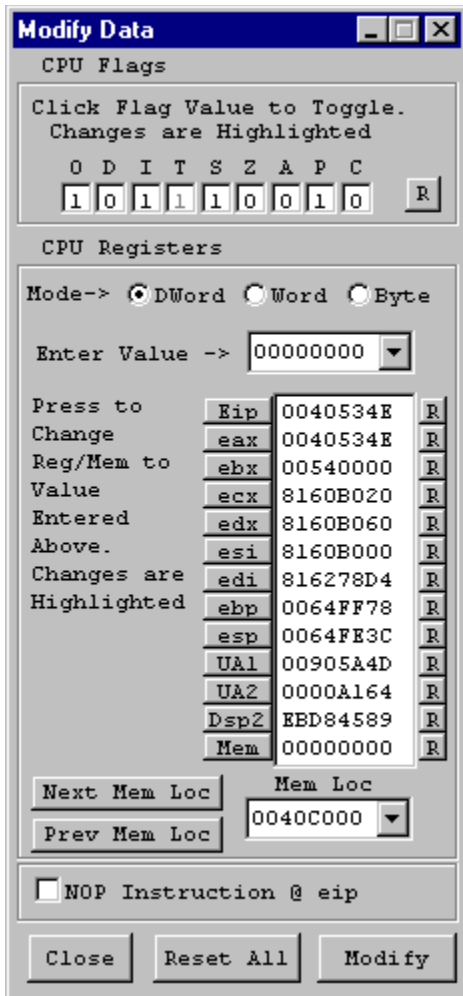
- 10.3 Find calc.exe in the list and select by clicking on the line.
- 10.4 Press the **Attach** button. Calc.exe will now load into the debugger and the main W32Dasm window will move and resize while two new debugging windows are be created. The main difference between attaching and loading a program into the debugger is that attaching does not stop the program at the Program Entry Point because this code was already executed to get it running in Windows. You can break into the program by pressing any of the **Single Step** buttons or the **Pause** button.

WARNING: Once a program is attached to, it will be terminated if W32Dasm is closed. When attaching to system critical programs, be warned that pausing them or terminating them will most likely cause Windows to not operate properly. If this happens, reboot the computer.

11.0 Modifying Registers, Flags, Memory, & Instructions

In this exercise you will learn how to modify register data, CPU Flag data, Instructions, and Memory Data with the debugger

- 11.1 Load calc.exe into the debugger. Do Not RUN Program
- 11.2 Press the **Modify Data** button which is in the lower left hand debugger window. A Modify Data Dialog will popup.



- 11.3 **Modify Flags:** To change the value of a CPU flag, left click on the desired flag value. It will toggle from 0 to 1 on each click. When the value of the flag is not the same as the original value, it will be highlighted in blue. No modifications actually take place until you Press the **Modify** button on this dialog.

RESETTING FLAG DATA: At any time before you press the **Modify** button, you can **RESET** all the flags to their original value by pressing the **R** button in the Flags data area.

NOTE: The **T** flag cannot be modified.

- 11.4 **Modify CPU Register Values:** To change the value of a CPU register, enter the desired value in the **Enter Value->** List Box. Depending on the Mode Selected by the **Mode** checkboxes, you have the option of changing the **DWORD** register (ie: Eax) or **WORD** register (ie: ax) or **BYTE** register (ie: ah, or al). Press the button of desired register to change its value. Values changed from the original are highlighted in blue. No modifications actually take place until you Press the **Modify** button on this dialog.

RESETTING REGISTER VALUES: Each register has a **R** button to reset it to

its original value. You can **RESET** the data at any time before you press the **Modify** button.

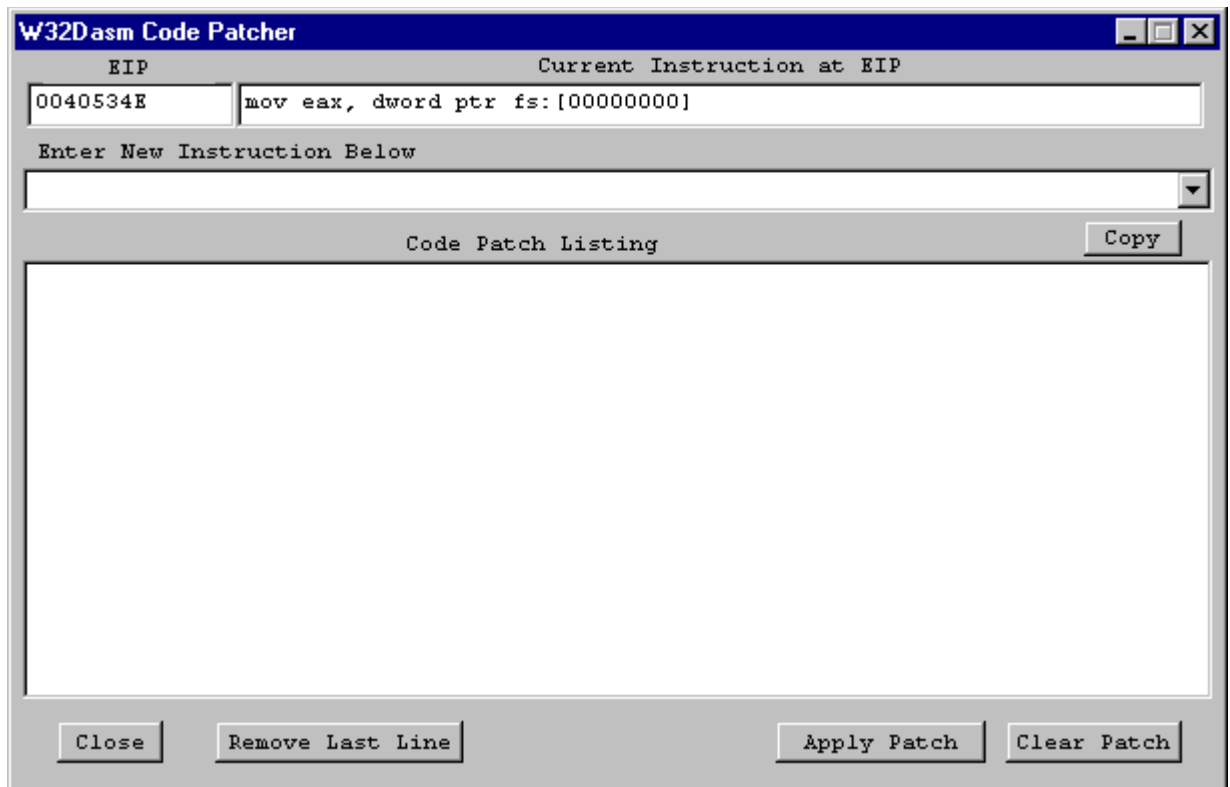
- 11.5 **Modify Memory Values:** To change the (DWORD, WORD, or BYTE) value of a memory location, enter the desired value in the **Enter Value->** List Box and enter the desired memory address in the **Mem Loc** List Box. Then press the Mem button to modify the data. You can also adjust the Memory Location by pressing the Next Mem Loc or Prev Mem Loc buttons. The value of the Memory Location will increase or decrease by a **DWORD, WORD, or BYTE** depending on the Mode selected in the Mode checkboxes. No modifications actually take place until the **Modify** button is pressed.

Data addresses specified by the **User Addr 1** and **User Addr 2** and **Display 2** List boxes on the lower left hand debugger window may also be modified by pressing the appropriate buttons on the Modify Data dialog box.

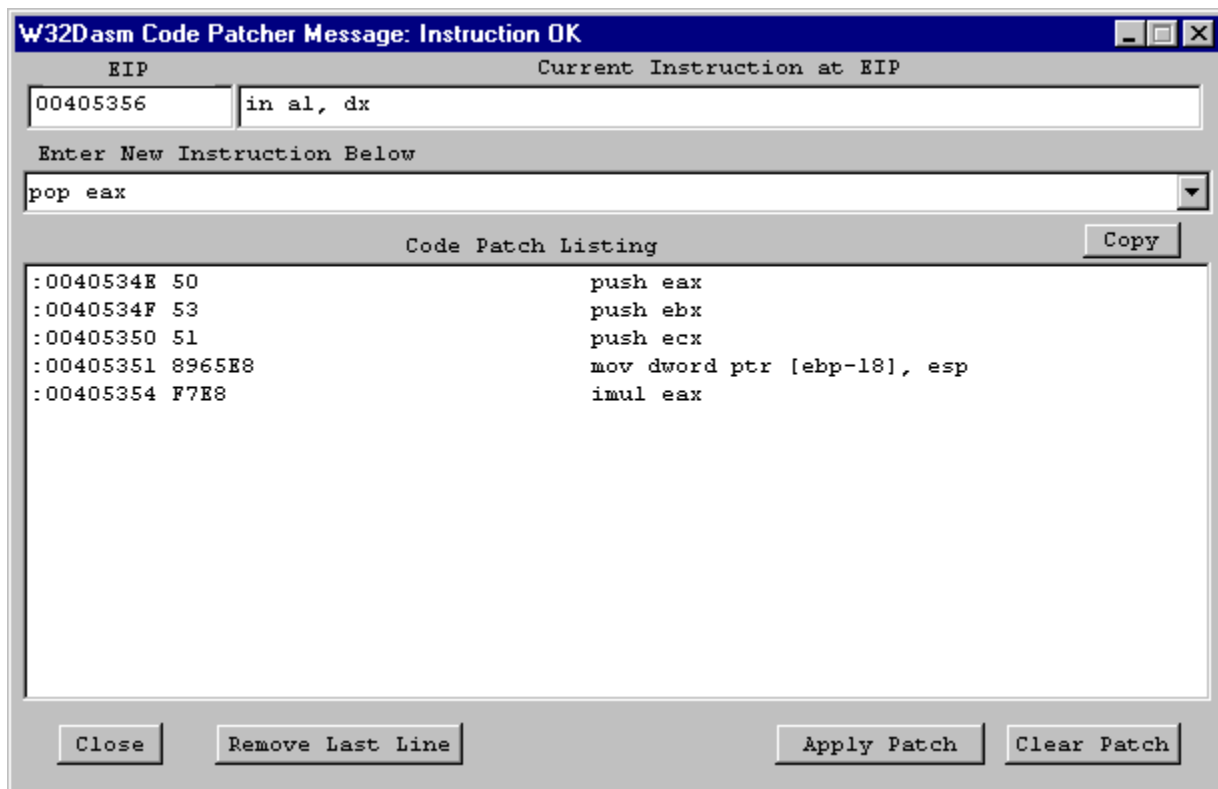
RESETTING MEMORY ADDRESS VALUES: At any time before the Modify button is pressed, you may **RESET** to the original value by pressing the corresponding **R** button.

- 11.6 **Modifying Instructions:** You can **NOP** any instruction at the selected Eip Address by checking the NOP Instruction @ eip before pressing the **Modify** button. W32Dasm automatically determines how many NOP instructions to insert to completely NOP the code line.

To **Modify Instructions to any other Code**, use the **Code Patch** Button located on the **Lower Right Hand Debugger Window**. This will open the Code Patcher Dialog Box.

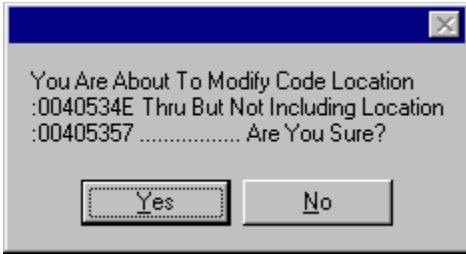


The address of the code location/s to patch is determined by the highlighted location in the Lower Right Hand Debugger Window. You can change the address by (1.) Single Stepping the program, (2.) Use the **Goto Address** button or (3.) Use the up/down scroll bars on the Lower Right Hand Window Code Listing Display. Once you have the code address you want to change, you can type instructions in the List Box titled **Enter New Instruction Below**. When you hit the Enter key, the new instruction will be listed in the Code Patch Listing display. If the instruction is invalid or improperly formatted, it will not be entered. As a general guideline for proper format, look at the main disassembly listing for examples. Some instructions require that a size (ie: "dword ptr" or "byte ptr") be used. All numerical values are to be in HEX notation. Leading zeros are not required. To clear the entire patch listing, press the **Clear Patch** button. To erase only the last line in the listing, press the **Remove Last Line** button.



When modifying code, you need to be sure that instructions being replaced are covered by the same number of bytes by the new code. Use the NOP instructions to fill in bytes that are not used by the new instruction.

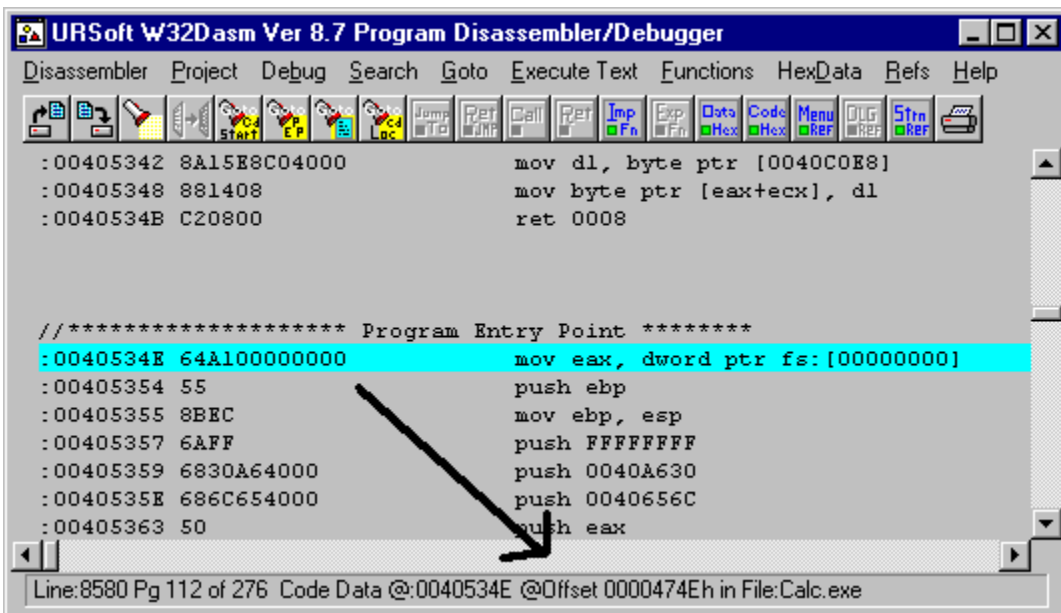
When the Code Patch has been composed, you may modify the program code by pressing the **Apply Patch** button. You will get a Message Box confirming your action. If you decide "Yes" the program code will be modified.



NOTE 1: Code modifications cannot be done on Write Protected Memory Areas such as where the KERNEL32.DLL code is placed.

NOTE 2: Code modifications are only displayed in the lower right debugger window. The main disassembly listing only shows the original data for the code.

NOTE 3: Code patches are only temporary changes. The original executable file or DLL file is not modified. To make permanent changes to the files, use a HEX editor to change the files data bytes at the proper location. File locations can be determined by looking at the status bar on the main disassembly window while the highlight bar is on the code location of interest. The status bar will display the value of the hex offset into the file where the byte data for that code is located.

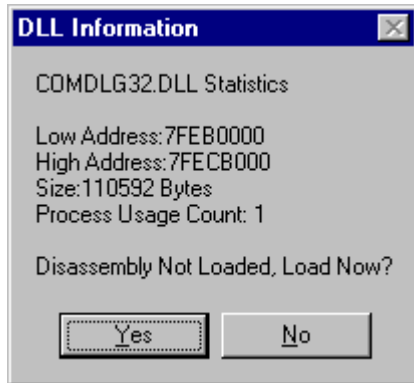


12.0 Exploring Called Modules (DLLs).

In this exercise you will learn how to debug DLL files. DLL files cannot be loaded by themselves. Only executables can be loaded by the debugger. However, any DLL called by the executable can be debugger after the executable is loaded into the debugger.

12.1 Load notepad.exe into the debugger and press **F9** to run notepad.exe.

- 12.2 Look at the List Box called Active DLLs in the **Lower Left Hand Debugger Window**. This List Box has the names of all the DLLs currently active in the loaded program. If you left Double Click on a name, another dialog box will popup




giving details about the DLL. This dialog also gives you the option of disassembling the DLL and attaching it to the debugger.

- 12.3 Press the **YES** button on the dialog box that pops up. COMCTL.DLL will now disassemble and load. If COMDLG32.DLL was previously disassembled and saved as a project file, the project file will be called. This saves the time it takes to disassemble the file.

- 12.4 You can now use the Debugger on the DLL.

NOTE: Certain System DLLs such as KERNEL32.DLL DLL cannot accept breakpoints or have their Memory modified because they are in READ ONLY protected memory areas.

- 12.6 Press the  toolbar and double click on **GetOpenFileNameA**. The disassembly listing should goto code location **:7FEB1162** which is a **push 00000017** instruction. This code is the beginning of the COMDLG32.DLL function **GetOpenFileNameA**. Now Press **F2** to set a breakpoint at this location.

- 12.7 Now goto the Notepad.exe programs **File** Menu and select **Open**. The debugger should now halt notepad.exe execution and break at the **GetOpenFileNameA** code location **:7FEB1162** in COMDLG32.DLL.

13.0 The WIN API Detail Feature

In this exercise you will learn how to use the WIN API Detail feature of W32Dasm.

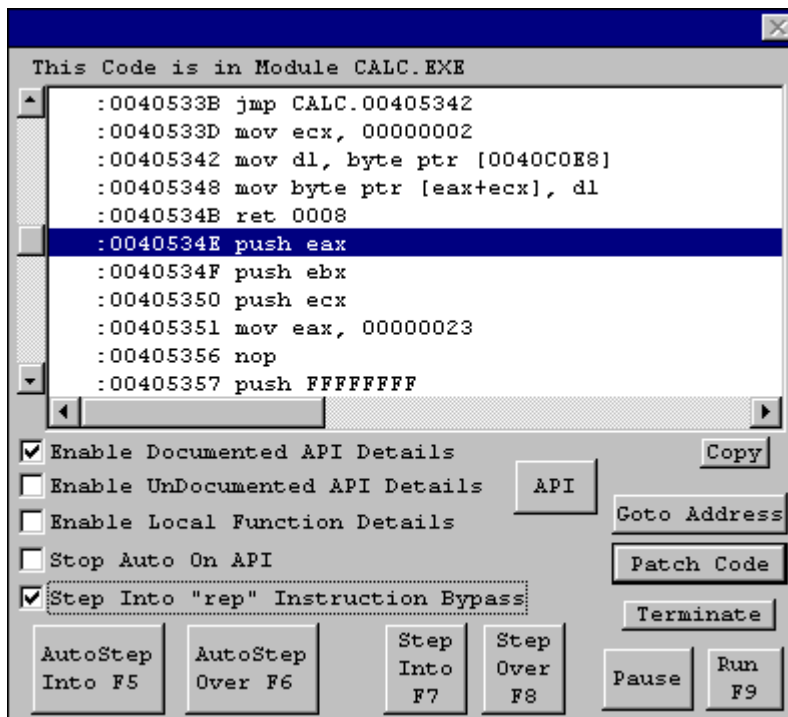
- 13.1 W32Dasm has a feature which allows the user to look at the details of many WIN API Functions. Almost all of the generally used function from KERNEL32, ADVAPI32, SHELL32, COMCTL32, COMDLG32, USER32, and GDI32 are available. Also, there are options to display **UnDocumented API Details**

and Process **Local Function Details** with a generic type data display. The generic data shows both Hex Data and String Pointer Data if appropriate.

Load calc.exe into the debugger, but don't RUN it.

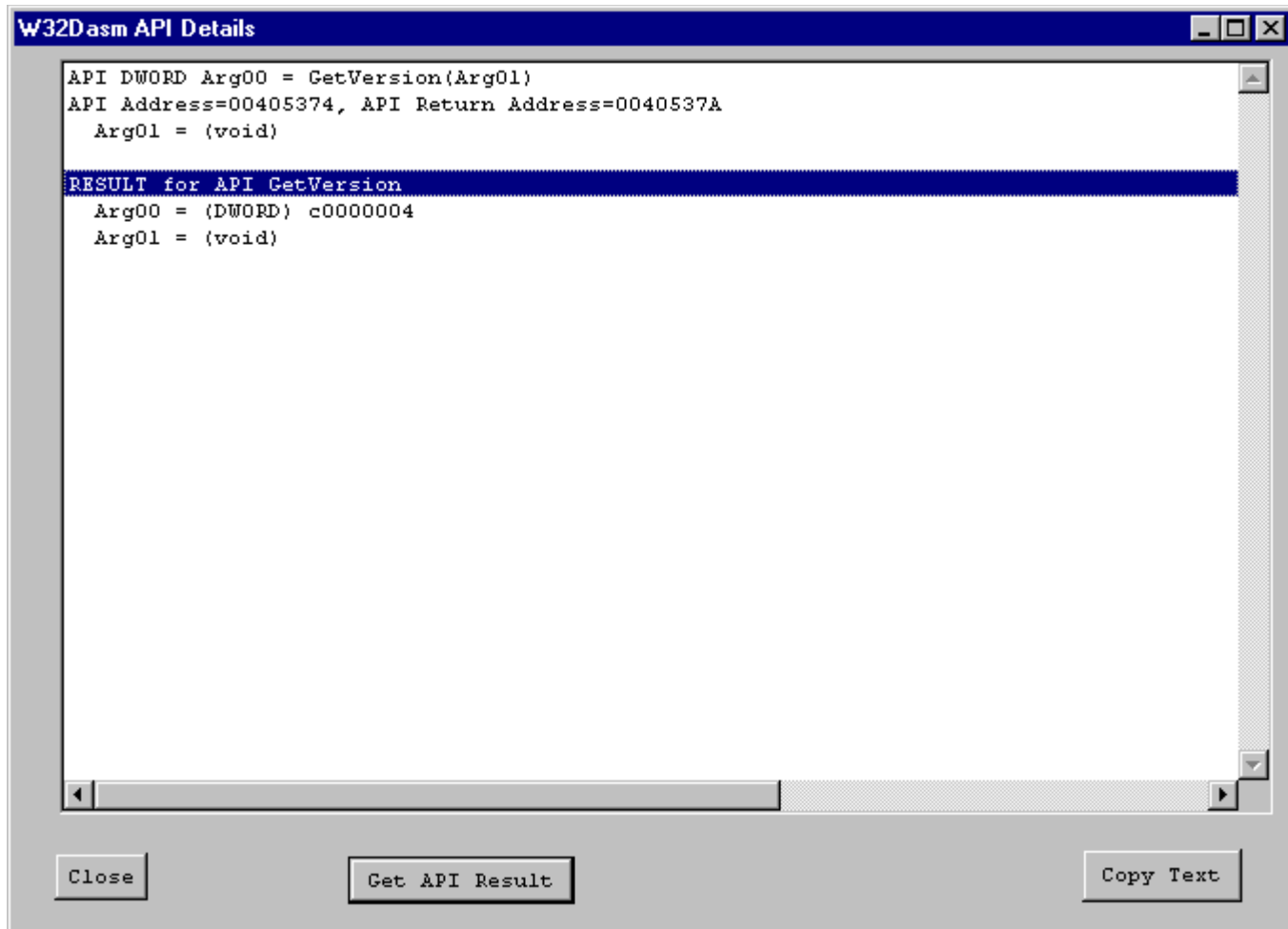
- 13.2 Make sure that the **Enable API Details** checkbox is checked on the **Lower Right Hand Debugger Window**. (This should be checked by default). Also put a check in the **Stop Auto on API** checkbox. If this box is not checked, the auto single step function will continue without stopping when an API function is reached.

NOTE: To activate **UnDocumented API** or **Local Function Details**, check the appropriate box/es.



- 13.3 Now Press **F5** to activate the Auto Single Step Into Function. (**You can also use the manual single step F7, to step thru the code until a API function is reached**).

- 13.4 The debugger should now auto single step into until the first WIN API function is reached which is **KERNEL.GetVersion** at code location **:00405374**. A Dialog box will appear that shows the API and type of data and values input to the API function. If the **Get API Result** button in this dialog is pressed, the API function will run and return the API results in the dialog list box.



You can use the **Copy Text** button to Paste the dialog text into a text editor.

- 13.5 Pressing **F5** again or manually single stepping will cause the API Details dialog box to minimize until the next API function is reached. The **API Details** function can be disabled by unchecking the EnableAPI Details checkbox on the **Lower Right Hand Debugger Window**.
- 13.6 If the instruction pointer is on a API call function but the **API Enable Details** checkbox was not checked, you can activate the API List Window by checking the **API Enable Details** checkbox and then press the **API** button in the **Lower Right Hand Debugger Window**.

