

# TPF Software News

Volume 3, Issue 2, Fall 2001

© 2001, TPF Software, Inc.

## View and edit all variables with Watch Expressions window in TPF/GI 2.6.1

TPF/GI has broken a new barrier in its quest to deliver greater power into the hands of TPF programmers.

The Watch Expressions window, new in TPF/GI 2.6.1, allows C and C++ programmers to view and edit the values of ALL variables while they debug their TPF programs.

That's right: we said *all variables*.

Previous versions of TPF/GI allowed programmers to view and edit the values of local variables only (local variables are

variables that are declared inside of functions).

---

***TPF/GI has broken a new barrier to deliver greater power to TPF programmers.***

---

Now, however, the new Watch Expressions window allows the viewing and editing of any variable that can be manipulated by the current statement when the code is

compiled with the test option. This includes extern and static variables.

Any type of variable can be viewed and edited, including classes, structs, arrays, pointers, and null-terminated strings as well as ints, chars, doubles, and other basic C/C++ types.

The values for these variables are displayed by the Watch Expressions window in human-friendly formats: numbers can be viewed in decimal as well as hex, char can be viewed as characters, and enum values can be viewed by their symbolic names rather than their ordinal

*Continued on next page*

### Table of Contents

- 1 View and edit all variables with Watch Expressions in TPF/GI 2.6.1
- 1 Data event control block support is here in TPF/GI
- 3 Local Variables window renamed and enhanced
- 4 More TPF Software enhancements
- 4 Trace Statistics window graphically provides the big picture

## Data event control block support is here in TPF/GI

TPF PUT 13 and higher support the concept data event control blocks (DECBS).

And now TPF/GI 2.6.1 provides powerful graphical tools that help programmers debug code that uses DECBS.

---

***TPF/GI 2.6.1 provides powerful graphical tools to help programmers debug DECBS.***

---

What is a DECB? DECBS act like data levels that can be created and released dynamically, that can be named, and that can grow as necessary. IBM created the DECB concept to address the archaic limitations of having only 16 data levels that cannot easily expand.

*Continued on page 3*

# Watch Expressions window new in TPF/GI

*Continued from previous page*

values only.

## Expressions, Not Just Variables

But the description so far just scratches the surface of the Watch Expressions window's power. The window actually allows you to look at *expressions*, not just variables. Expressions can be made up of *more than one* variable as well as literals and operators (see Figure 1).

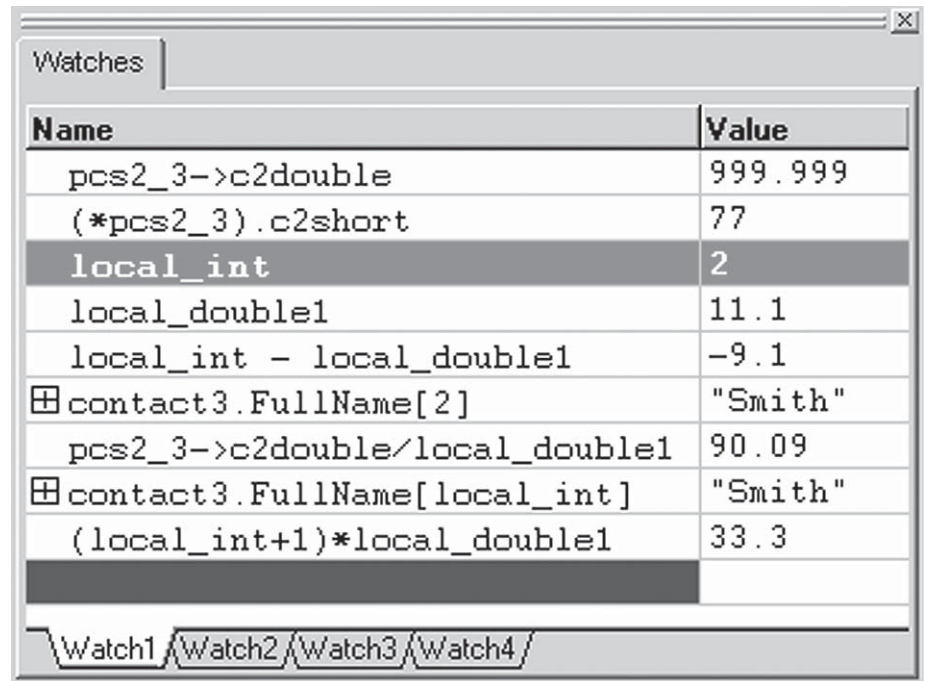
Among other things, the power to use expressions rather than simple variable names means that you can view and edit individual elements in an array and individual member variables of a struct or class without viewing the entire array, struct, or class.

For example, if *a* is an array of int, you can enter the expression *a[0]* into the Watch Expressions window to view and edit only the first integer in the array.

Even more advanced, if *i* is an int variable that your code is using to loop through the values in *a*, you can enter the expression *a[i]* into the Watch Expressions window. Each time *i* changes value, a different int from *a* will be displayed.

Other expression examples:

- **Structs and Classes.** If *s* is a struct and *m* is a member variable of *s*, *s.m* monitors the value of one member variable.
- **Pointers.** If *pn* is a pointer to an int, then the expression *\*pn* will dereference that pointer and display the value of the int. If *pc* is a pointer to a class instance and *m* is a member variable of that class, *pc->m* monitors the value of one member variable.
- **Logical Operators.** *n == i* returns 1 if the values of *n* and *i* are equal, 0 if they are not. *<*, *>*, *>=*, *<=*, *!=*, *&&*, *||*, and *!* behave as expected.
- **Mathematical and Bitwise Operators.** Operators such as *+*, *-*, *\**, */*, *%*, *&*, *|*, *<<*,



Name	Value
<code>pcs2_3-&gt;c2double</code>	999.999
<code>(*pcs2_3).c2short</code>	77
<code>local_int</code>	2
<code>local_double1</code>	11.1
<code>local_int - local_double1</code>	-9.1
<code>contact3.FullName[2]</code>	"Smith"
<code>pcs2_3-&gt;c2double/local_double1</code>	90.09
<code>contact3.FullName[local_int]</code>	"Smith"
<code>(local_int+1)*local_double1</code>	33.3

Figure 1: The new Watch Expressions window.

*>>*, and *!* work as you would expect. Parentheses group subexpressions, or the normal rules of C/C++ precedence apply. For example,  $(x + n) * a[i \% 3] / j * i$  is handled easily by the Watch Expressions window.

## Using Watch Expressions

The Watch Expressions window (Figure 1) is made up of four pages — Watch1, Watch2, Watch3, and Watch4. Each page contains a table with expressions in the left-hand column and the values for those expressions in the right-hand column.

As you step through your program, values that change are highlighted in the Watch Expression window, and values known to be invalid take on a grayed appearance.

Expressions can be added to the window in several ways. To place the name of one variable in the window, you can right click on that variable in your code and choose *Add to Watch* from the local menu that pops up.

To place an entire expression in the window (not just a single variable name), you can select the entire expression in your source code before right clicking.

You can also drag and drop a selected expression into the window.

At any time, you can type a new value into the right-hand column or a new expression into the left-hand column. Typing a new value into the right-hand column will actually change the value of that variable on the host, allowing you to interactively test how your program behaves under different conditions. Typing a new expression into the left-hand column will change which expression is being watched.

Expressions can be removed from the window by hitting the Delete key or by right clicking and selecting Delete. ❖

# Data event control block support in TPF/GI

*Continued from page 1*

To accommodate the DECB concept, TPF/GI has redesigned its graphical ECB window. In the process, support for traditional data levels has been enhanced as well.

## Redesigned ECB window

As you can see from Figure 1, a new DECB area has been added to the graphical page of the ECB window.

*The graphical ECB window has been redesigned.*

In this DECB area, each DECB is represented by a rectangle. The name of the DECB appears on the front of the rectangle. If a DECB contains a block, the rectangle representing it is red; otherwise, the rectangle is blue.

The size of the block owned by the DECB is indicated by a single character in parentheses beside the DECB name: for example, *FARE QUOTE (4)* means a 4K block is on the DECB named *FARE QUOTE*. (*H*) means a high speed block, (*L*) a large block, and (*S*) a small block.

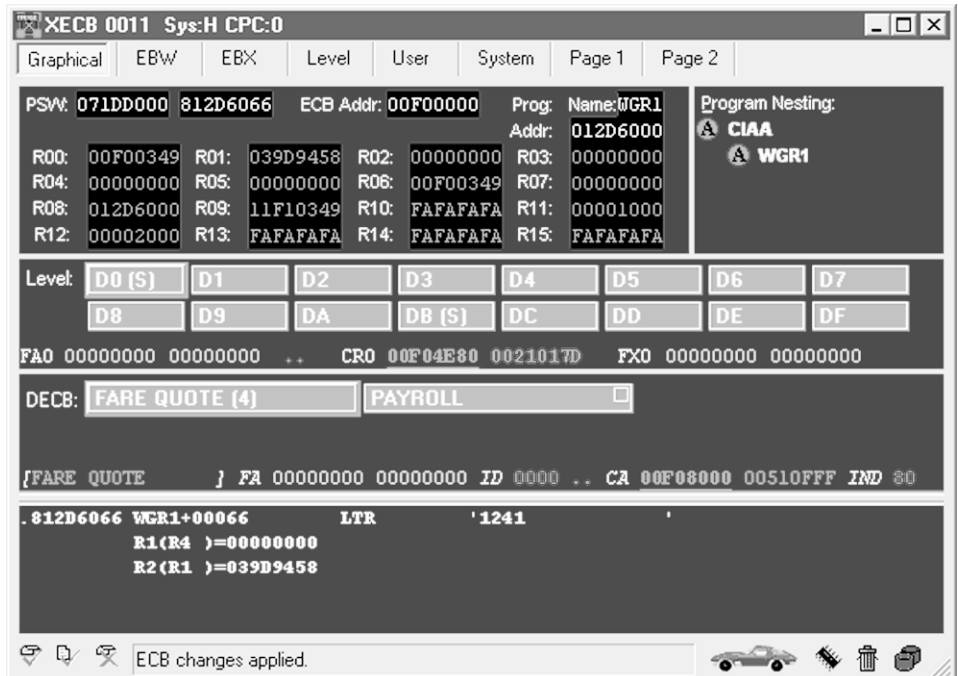


Figure 1: The graphical ECB window supports DECBs with a new DECB area and DECB detail line immediately below it.

The presence of detached blocks is signalled by a smaller red rectangle within the data level rectangle.

## Using the DECB area

To edit the contents of detached blocks, programmers simply double click the smaller red rectangle.

Similarly, to edit the DECB block itself,

programmers can double click the main DECB rectangle.

Other operations on the DECB can be performed by right clicking or by using the memory chip, trash, and database icons in the lower right corner of the ECB window.

For example, dragging the memory chip and dropping it on a data level will get core for the DECB. Dragging a DECB to the trash will release core. Dragging the database icon to the DECB will do a FINDC.

To create a DECB level, programmers can right click the DECB area and select the "Create DECB" menu item.

Selecting a DECB level causes information about that level to be displayed in the one-line detail area below the DECB rectangles.

The fields in this detail area can be double clicked or right clicked to explore DECB information in even more detail.

A similar one-line detail area has been added for the traditional data level area as well.❖

## Local Variables window renamed and enhanced

The TPF/GI window formerly known as the "Local Variables Window" has been enhanced and renamed the "Variables Window" in TPF/GI 2.6.1.

This new Variables Window has two pages. The "Locals" page displays all local and parameter variables that are visible in the current scope (to view all variables, programmers can use the new Watch Expressions window — see page 1).

The new "this" page displays the value of the hidden *this* variable when it is available. Although the Locals page also displays the "this" variable, the "this" page will display it with its member variables already conveniently expanded.❖

# More TPF Software enhancements

## New scripting objects

New TPF/GI 2.6.1 scripting objects make it easier for users to manipulate and edit core on the host. Users can read about these new objects in the TPF/GI help file.

## Microsoft Script Debugger

TPF/GI 2.6.1 takes steps to ensure that the Microsoft script debugger will automatically appear to handle programmer errors in scripts written for TPF/GI.

## Script templates

Users can now supply their own script templates to be used when they create new scripts from within TPF/GI.

## View Fixed Files enhanced

The View Fixed Files dialog box now accepts either equate names or record IDs.

## Visual Log enhanced

Users can now reopen previous Visual

Logs and append information to them.

## Trace Output viewer enhanced

The Trace Output viewer has a search filter that contains the names of all programs and all ECB IDs encountered during a trace.

The program names and ECB IDs in this filter are now automatically sorted in alphabetical order. ❖

# New Trace Statistics window graphically provides the big picture

The new Trace Statistics window allows users to see detailed information about program and file I/O, pool usage, memory and heap usage, macros, and instructions in an easy-to-understand graphical format.

To use the Trace Statistics window, programmers first set up trace options, then run a transaction.

The new window has five pages. The Program and File I/O page displays a pie chart depicting the number of programs called in from file versus the number of programs called in from core.

The Pool Usage page shows the number of pools acquired and released and can be filtered based on record size and term.

The Memory/Heap Usage page shows the amount of common and private storage for each type of memory block.

The Macro Statistics page displays a list of macros and how many times they were encountered.

The Instruction Statistics displays a table showing all instructions and the number of times each has been encountered, as well as a bar graph that compares the count for one or more individual instructions to the total number of instructions encountered. ❖

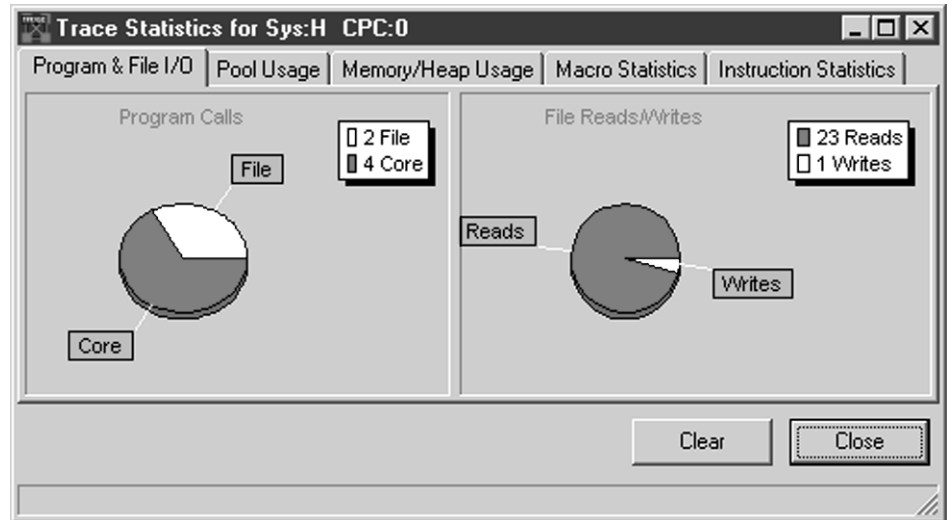


Figure 1: The Program & File I/O page of the new Trace Statistics window.



## Contact TPF Software

**Web Site** [www.tpfssoftware.com](http://www.tpfssoftware.com)

**Email** [info@tpfssoftware.com](mailto:info@tpfssoftware.com)

**Telephone** 919-676-5501

**Address** TPF Software, Inc.  
8729 Gleneagles Drive  
Raleigh, NC USA 27613-5419

**TPF Software News Editor:** Ed Jordan