Debugging with Watchpoints

A watchpoint is a variable in a function that triggers a suspension when its value changes, or when its value could change, and when a condition you specify is satisfied. Watchpoints cause the suspension in the debugger, if you have turned it on, or in the session.

You specify watchpoints as a paired entity consisting of the name of an APL variable and an expression. Whenever an APL primitive is executed that might cause a change in value to the named variable, the system evaluates the expression. If the result of the evaluation is a non-zero singleton, execution suspends in a phantom function named <WATCHPOINT>. You can resume execution by typing a branch statement in immediate execution mode, or by using the debug options.

This facility allows you to interrupt your program when a given condition occurs for a specific variable, which differs from the breakpoint facility that allows stopping on a specific line. For example, your watchpoint specification might look like:

```
1 2 ρ 'total' 'total > limit'
```

Whenever your program reaches a statement that assigns a value to the variable total, and that value is greater than the value of the variable threshold, the function suspends with a message in the session.

```
    foo
WATCHPOINT TRIGGERED
total > threshold
^
```

You can examine the state of your application in immediate execution mode or in debug mode. The stack might look like:

```
    )si
⎕WATCHPOINT[1]
foo[5]
```

and you could resume with either of the following two branch statements:

```
→ ⎕lc+1
→ 0
```

Note that you ran the function foo and the system created the function ⎕watchpoint with the watched variable names(s) prefixed the "Var:" or "Vars:". You must branch out of the system-created function or past the line that caused the suspension to continue your function.

The system variable ⎕watchpoints

Watchpoints are maintained in a system variable ⎕watchpoints or abbreviated ⎕wp. Note that the system variable is plural, while the function the system creates is singular. You can use the system variable in a statement in a function or in the session as you can any system variable. If you attempt to run the ⎕watchpoint function, the system signals a syntax error.

The system variable ⎕watchpoints is an N by 2 nested array. Each element of the array must be a character vector. Elements in the first column contain variable names. Elements in the second column contain corresponding APL expressions to be evaluated. Unlike APL+Win, multiple APL expressions can be separated by the diamond statement separator. You can invoke an input form by clicking on the Edit Watchpoints item on the Session | Debug menu, or pressing **Ctrl+W**, or you can set ⎕watchpoints directly. Here is an example:

```
    ⎕watchpoints ← 3 2 ρ 'sum' 'sum>max' (,'Z') 'alpha=0' 'oz' 'goo'
    ⎕watchpoints
 sum sum>max
 Z   alpha=0
 oz  goo
```

Note that the expression does not have to include the watchpoint variable. In the second case, the function suspends whenever the variable `Z` is assigned a value and the variable `alpha` equals zero. Also note that you must make single-character variable names into vectors; the system variable does not accept scalars. A watchpoint expression triggers a suspension if the result is a singleton with a non-zero value or if there is no result.

Warning: The ⎕WATCHPOINT function is not benign. If your watchpoint expression is the name of a function, such as in the third line of the example above, and that function returns a zero, there still may be effects. Every time the function that you originally invoked assigns a value to the variable `oz`, the function `goo` runs. Even though your original function does not suspend, whatever actions are in the function `goo` have occurred. You could use this feature constructively in debugging, for example, to record the value of `oz` each time it is updated. You could also use this feature dangerously.

Watchpoint Triggers

The following expressions trigger execution of a watchpoint expression for the variable a:

• Simple assignment:	a←3
• Strand assignment:	(a b c)←1 2 3
• Indexed assignment:	a[2]←3
• Selective Specification:	(1▯a)←5
• Control structure specification:	:for a :in ι10 (each time through the loop)

Assignment to a system variable triggers execution of a watchpoint expression for that variable:

• Quad-Names:	⎕io←0

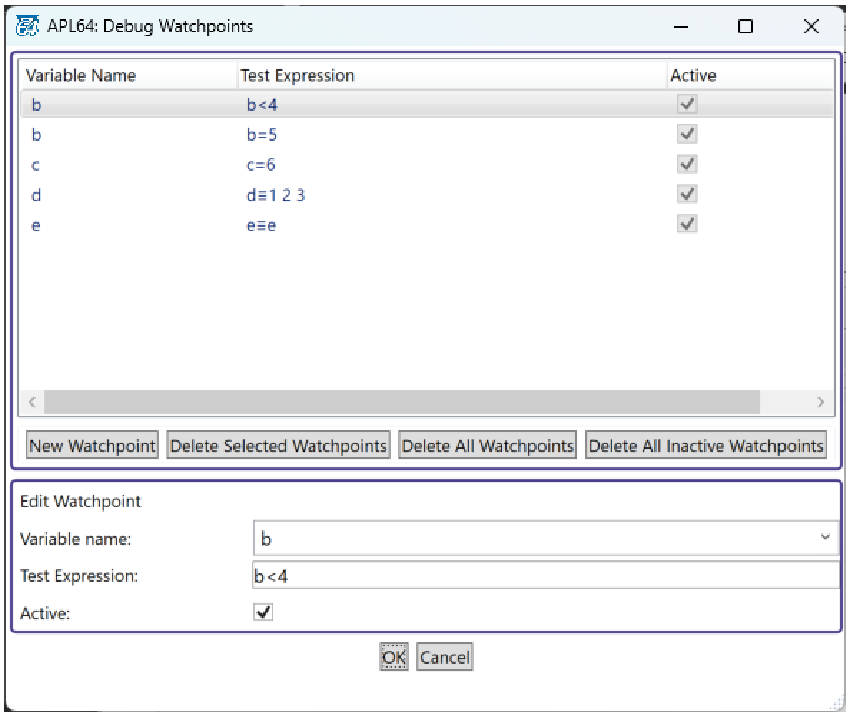
Watchpoints in the Debug Pane

To illustrate a watchpoint suspension, consider the following function and watchpoint specification:

```
   ∇ decrement a
[1]    :while a>0
[2]       (b c d) ← a
[3]       a←a-1
[4]    :endwhile
[5]    'done'
   ∇

    ⎕watchpoints
 b b<4
 b b=5
 c c=6
 d d≡1 2 3
 e e≡e
```

If the Watchpoint dialog is visible, the information displayed look approximately like this:


APL64: Debug Watchpoints

Variable Name	Test Expression	Active
b	b<4	✓
b	b=5	✓
c	c=6	✓
d	d≡1 2 3	✓
e	e≡e	✓

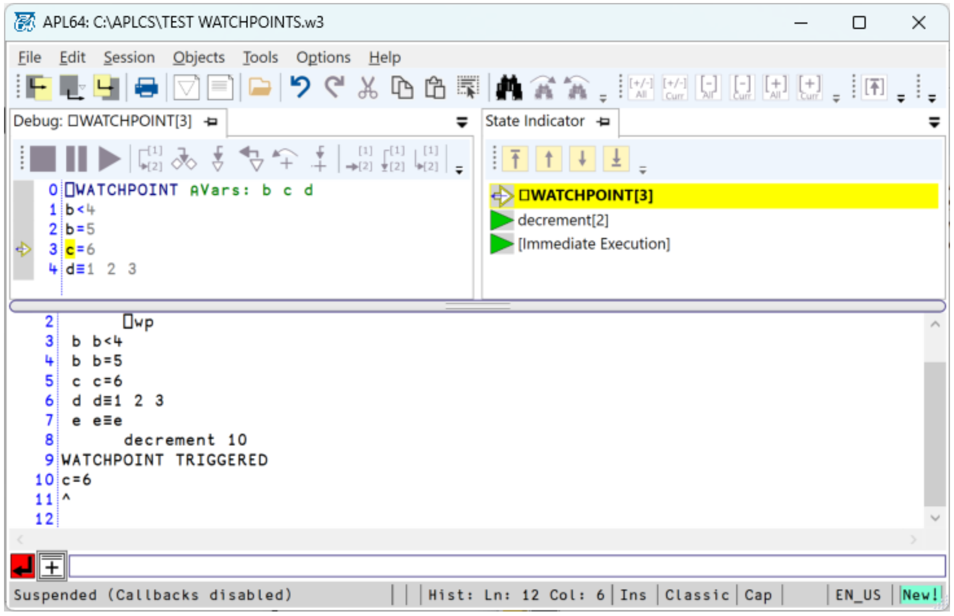
Edit Watchpoint

Variable name:	b
Test Expression:	b<4
Active:	✓

With ⎕watchpoints defined as above, run the expression decrement 10. The function suspends when the value of c becomes 6:

```
    decrement 10
WATCHPOINT TRIGGERED
c=6
^
```

If the debug pane is visible, the function and stack displays look approximately like this:


APL64: C:\APLCS\TEST WATCHPOINTS.w3
File Edit Session Objects Tools Options Help
Debug: ⎕WATCHPOINT[3]
State Indicator

```
0 ⎕WATCHPOINT ⍝Vars: b c d
1 b<4
2 b=5
3 c=6
4 d≡1 2 3
```

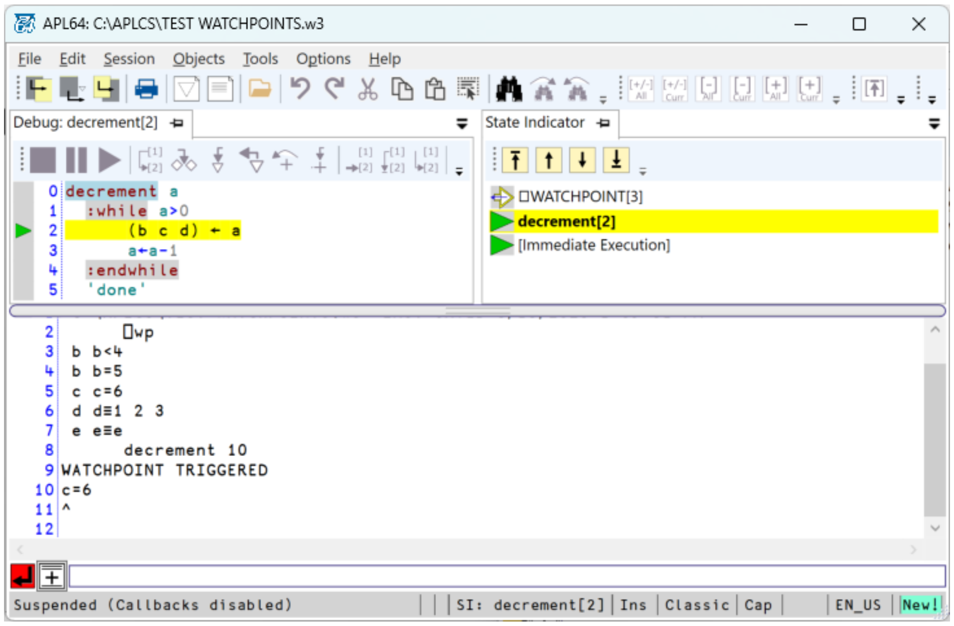
```
⎕WATCHPOINT[3]
decrement[2]
[Immediate Execution]
```

```
      ⎕wp
 b  b<4
 b  b=5
 c  c=6
 d  d≡1  2  3
 e  e≡e
      decrement 10
WATCHPOINT TRIGGERED
c=6
^
```

```
Suspended (Callbacks disabled) | | Hist: Ln: 12 Col: 6 | Ins | Classic | Cap | | EN_US | New!
```

There may be multiple watchpoint entries with the same variable name, each with a different watchpoint expression; each expression is evaluated when the variable is updated. Multiple watchpoint expressions on different variables may be executed for strand assignments. Note that only the watchpoints that are relevant to the current line appear in the debug pane. The watchpoint variable e would appear only if that variable were subject to change. The expression assigned for e would always return 1, so any function will suspend when it attempts to assign a value to e.

You can click on the > decrement[2] line in the stack display to show the function suspended on the strand assignment line. You can then use the stepping options to move through your function:


APL64: C:\APLCS\TEST WATCHPOINTS.w3
File Edit Session Objects Tools Options Help
Debug: decrement[2]
State Indicator

```
0 decrement a
1   :while a>0
2       (b c d) ← a
3       a←a-1
4   :endwhile
5   'done'
```

```
⎕WATCHPOINT[3]
decrement[2]
[Immediate Execution]
```

```
      ⎕wp
b b<4
b b=5
c c=6
d d≡1 2 3
e e≡e
      decrement 10
WATCHPOINT TRIGGERED
c=6
^
```

```
Suspended (Callbacks disabled)   SI: decrement[2] | Ins | Classic | Cap   EN_US | New!
```