Details of APL64 Project Update May 2019

This report provides detailed information pertaining to the progress of the APL64 Project in the following areas:

- APL64 Project team's approach to evaluating performance
- Operations in which the APL64 Project system outperforms APL+Win
- Improvements and enhancements in the modernized APL64 Project Programmer Session

APL64 Project Performance Tests

In most cases tested, the APL64 Project is as fast or in some instances, substantially faster than APL+Win. The APL64 Project development team was able to take advantage of the state-of-the-art development tools and superior methodology used in the APL64 Project to perform optimizations. In every case where optimization was applied, performance increased such that the APL64 Project speed was as fast or faster than APL+Win. It is our goal to optimize all the APL64 Project primitives to ensure that the performance of the APL64 Project will match or be consistently faster than APL+Win.

The optimization process is on-going. Our experience suggests we can successfully optimize all primitives to make the APL64 Project run significantly faster than APL+Win. At this time we are sharing with you the results of some APL64 Project performance tests. All of the results shown in the charts that follow are for cases where optimized APL64 Project operations run faster than APL+Win. Un-optimized cases are not shown in this report. APL2000 will provide updates on the performance of these cases as additional optimizations are completed.

In the accompanying charts, average elapsed execution time is always shown on the vertical axis in milliseconds. Lower average elapsed time values correspond to faster performance. The "winner" of each comparison is always marked with a yellow star to emphasize the better performing system, with a lower average elapsed time value. Remember in reviewing these charts that "Lower Time is Better".

Argument size is shown on the horizontal axis. In general, this indicates the number of elements in argument arrays. In a few cases this axis indicates the number of elements in the result (for example, the deal primitive, **n?m**, uses scalar values for both arguments but the result size is controlled by the value of the left argument). In all cases the horizontal axis value indicates how much work was done for the operation. As the amount of work (i.e., number of elements) increases, the corresponding time also increases for both APL+Win and the APL64 Project. In most cases there is a linear relationship between the amount of work being done and the execution time. This linear relationship between argument size and execution time is highlighted by a line of best fit on each chart.

However, in a few cases note that APL+Win and the APL64 Project values converge to roughly the same value as the argument size increases and the differences seem to be due to statistical variations. In such cases, the critical factor limiting performance is memory access time. Memory access time is workstation dependent and relatively independent of the APL implementation. The speed observed on different workstations for such cases will be heavily influenced by the amount of L1, L2, and L3 memory cache on the machine. These are hardware limitations that constrain the speed that data can be moved in memory and would be difficult or impossible to transcend for algorithms implemented in any computer language whether compiled or interpreted.

The chart showing FLOAT+FLOAT operation is a good example of the effect of memory access time for large argument sizes. Up to about 75,000 elements the APL64 Project is faster than APL+Win, but by 100,000 elements in the argument sizes the systems run at roughly the same speed because the time it takes to access memory has become the dominate factor constraining calculation throughput.

Performance results are calculated by executing a test expression over thousands of iterations and averaging the execution time. This is done to reduce statistical variations and to make sure "Garbage Collection" costs are also factored into the run times. The performance test workspace executes the expressions for varying size argument arrays of different types. For example, for the INT+INT operation, integer vectors are added together. Argument arrays are calculated using the following data values:

```
float ← count ⍴ 0.08296339080035192 2.570828471411902 0.8041284604922564
∆float ← ⌽ float
floatScalar ← ⎕first float
int ← count ⍴ 2 3 4
∆int ← ⌽ int
intScalar ← ⎕first int
bool ← count ⍴ 1 0
∆bool ← ⌽ bool
ones ← count ⍴ 1
∆ones ← count ⍴ 1
zeros ← count ⍴ 0
∆zeros ← count ⍴ 0
```

The variable `count` in the expressions above varies in a `:FOR` loop that repeats each test over the following set of argument sizes: `1 2 5 7 10 25 50 75 100 250 500 750 1000 2500 5000 7500 10000 25000 50000 75000 100000`.

When performance is measured for dyadic scalar primitives such as `INT+INT` where both arguments have the same shape and data type, the left argument uses the data variable with the value matching the data type (`int` for integer data, `float` for floating point data, etc.) and the right argument uses the data variable with a ∆ prefix (`∆int` for integer data, `∆float` for floating point data, etc.). This is done so that the calculation is accessing data at different memory addresses for the left and right arguments. This is important so that memory caching effects on performance are properly taken into consideration. While the chart for adding two integer arrays is labeled as `Operation: INT+INT` that calculation is done using the expression:

```
int + ∆int
```

For operations that mix data type such as `INT+FLOAT` we execute the non-delta variable names such as:

```
int + float
```

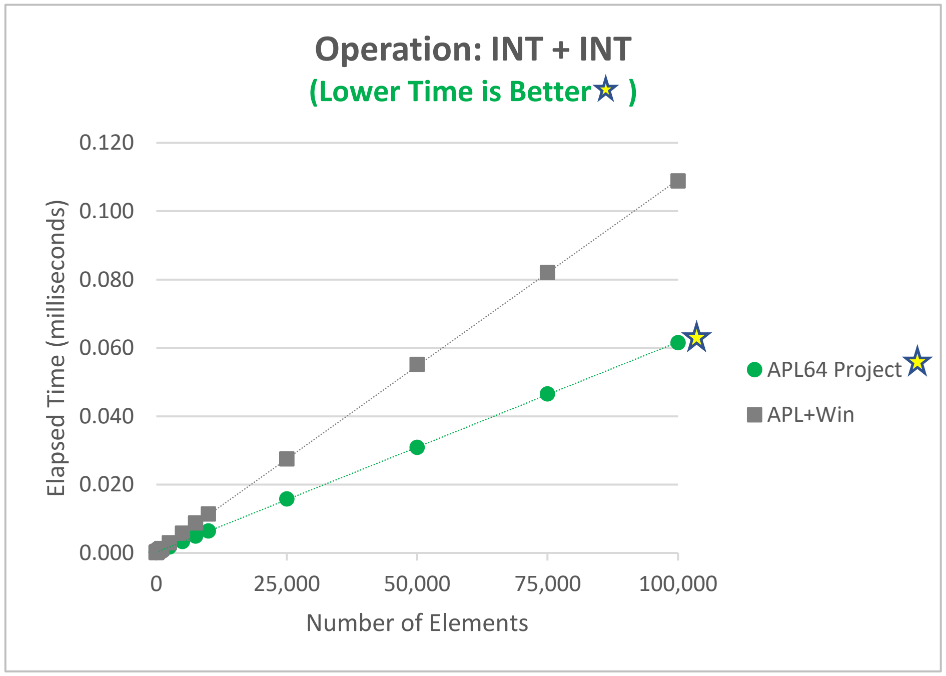
Operations that indicate one of the arguments is a scalar use the corresponding `Scalar` suffixed variable name. For example, the chart for `Operation: INT+INT Scalar` is computed using expression:

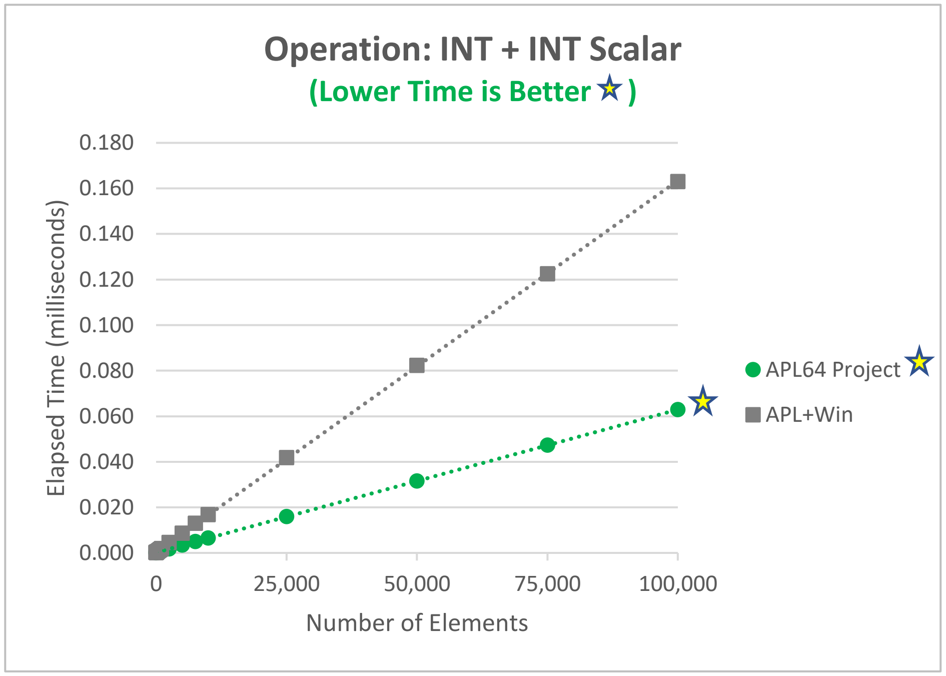
```
    int + intScalar
```

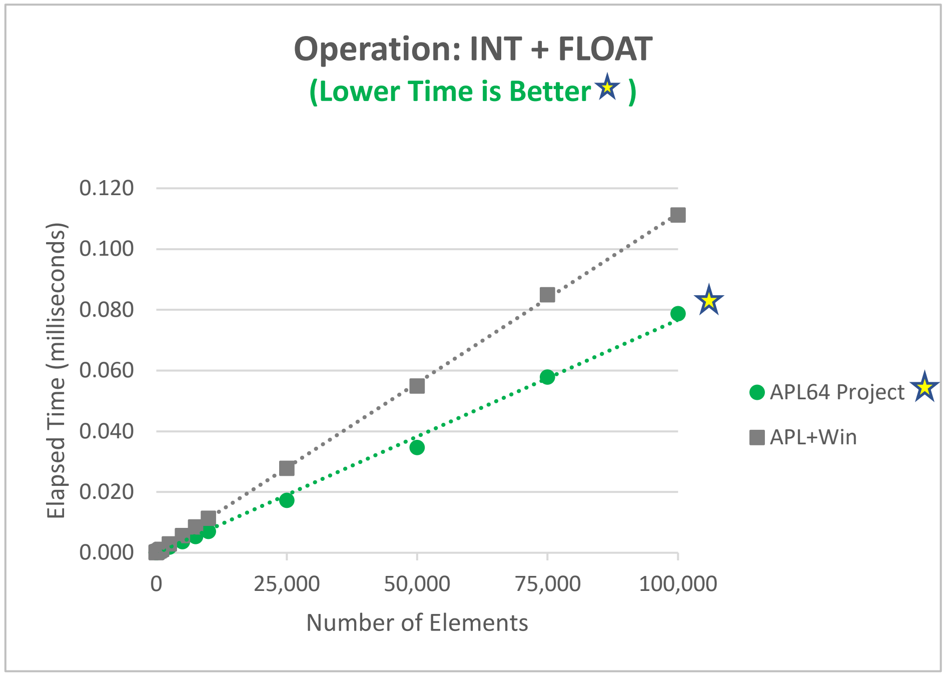
The optimizations achieved in the APL64 Project are largely dependent on the efforts of the APL64 Project team members to utilize the significant improvements in the tools and techniques available in the development environment of the APL64 Project.

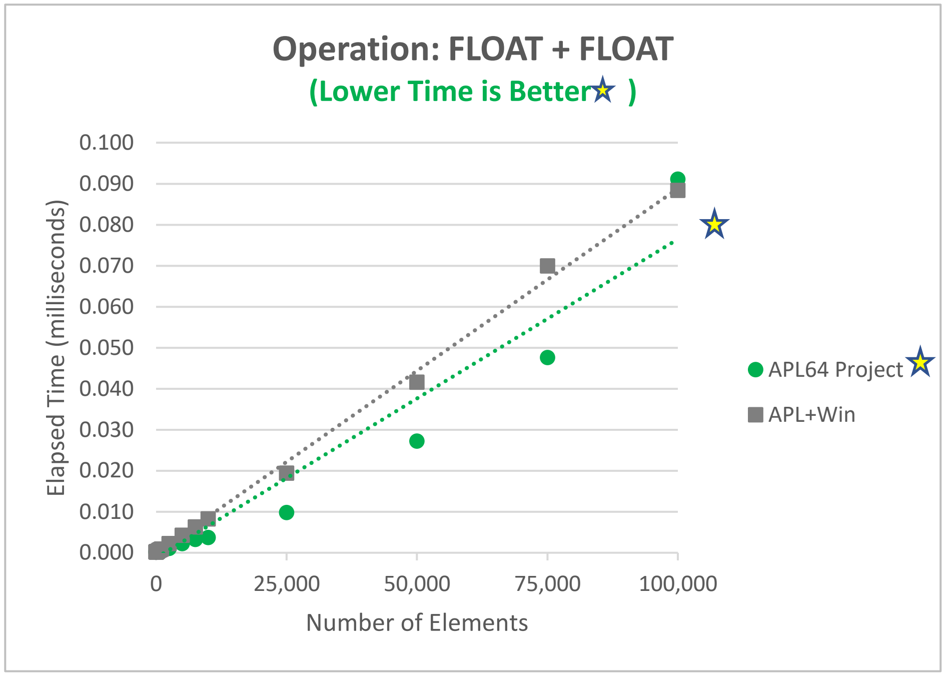
Thus far Optimized APL64 Project vs. Production APL+Win Performance results

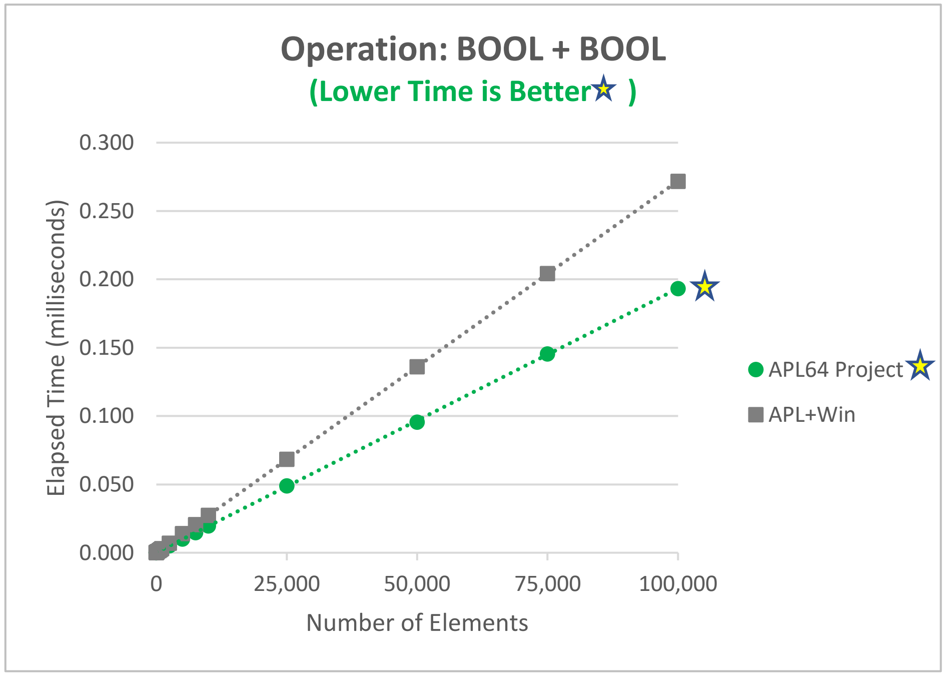
The charts below are representative of system performance for some of the APL primitives that have been optimized in the APL64 Project. They illustrate performance of the APL64 Project relative to APL+Win version 19. APL+Win data points use gray squares and APL64 Project data points use green circles in these charts.

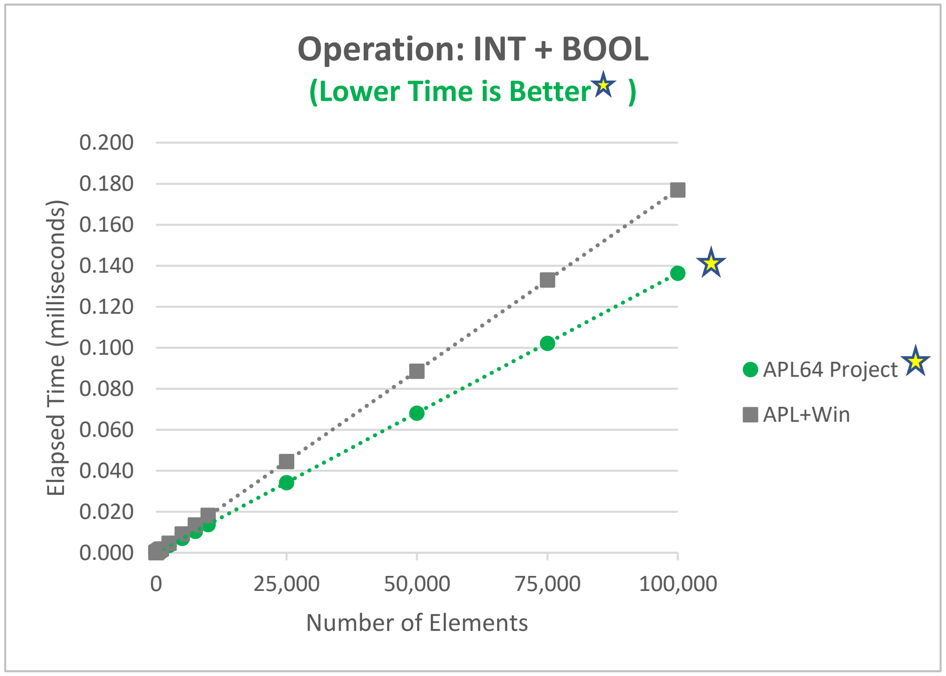


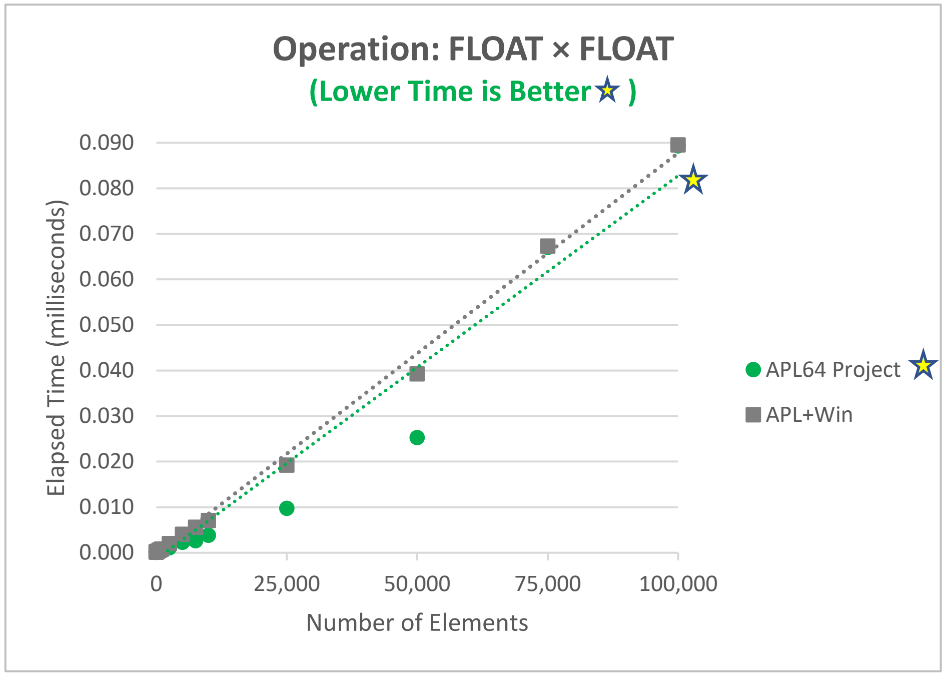


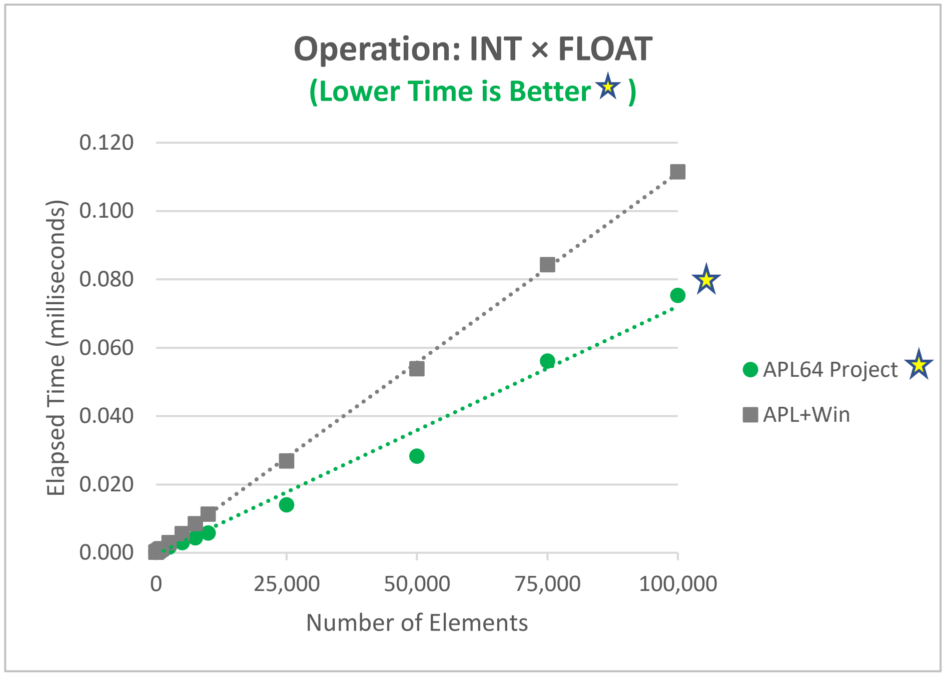

Operation: INT + FLOAT
(Lower Time is Better)
Elapsed Time (milliseconds)
0.120
0.100
0.080
0.060
0.040
0.020
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project
■ APL+Win

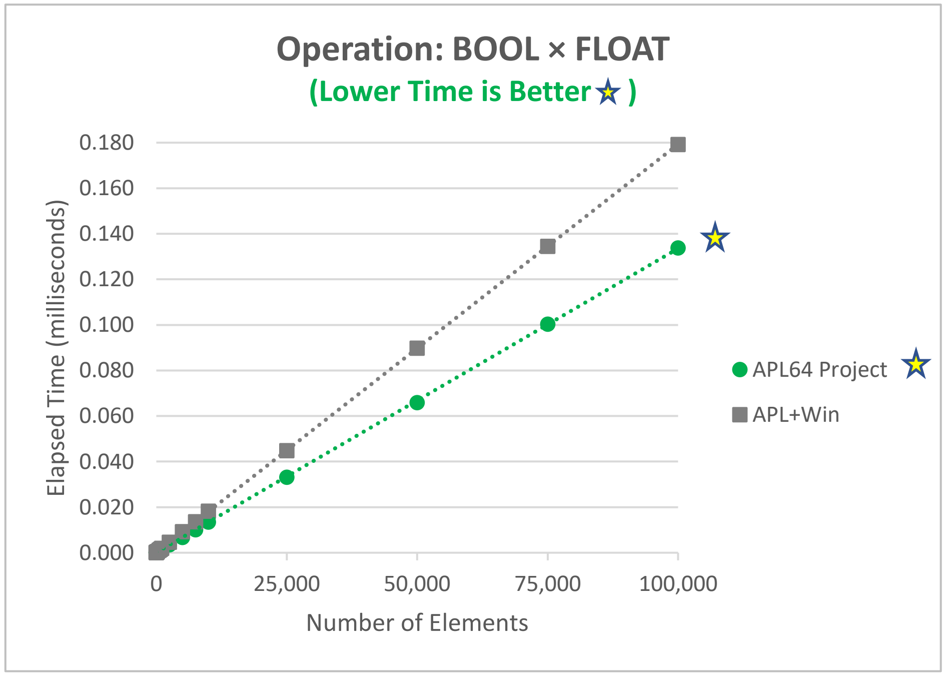

Operation: FLOAT + FLOAT
(Lower Time is Better)
Elapsed Time (milliseconds)
0.100
0.090
0.080
0.070
0.060
0.050
0.040
0.030
0.020
0.010
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project
■ APL+Win

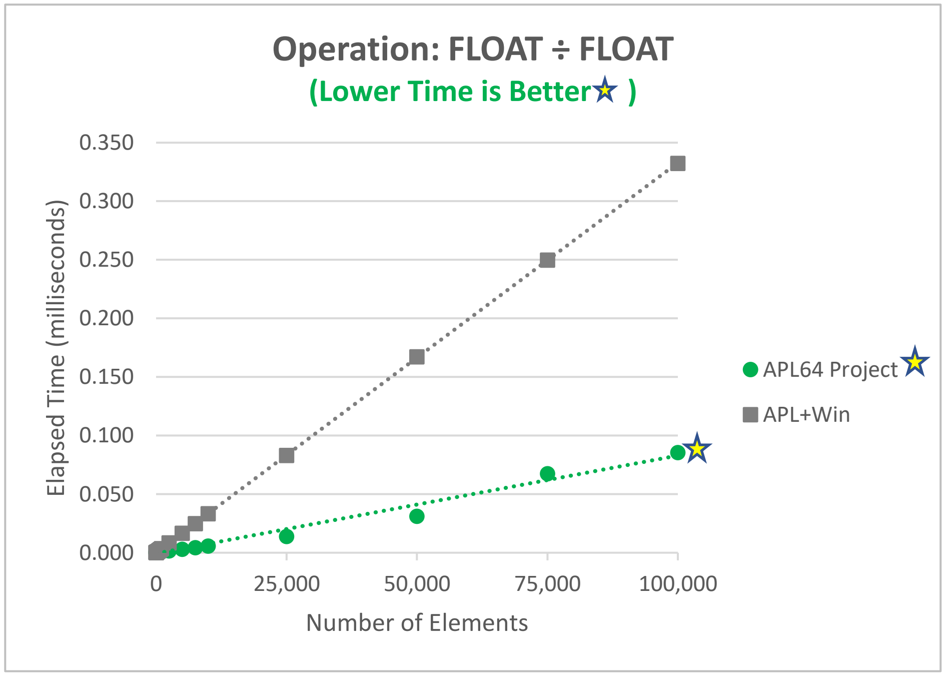

Operation: BOOL + BOOL
(Lower Time is Better)
Elapsed Time (milliseconds)
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project
■ APL+Win

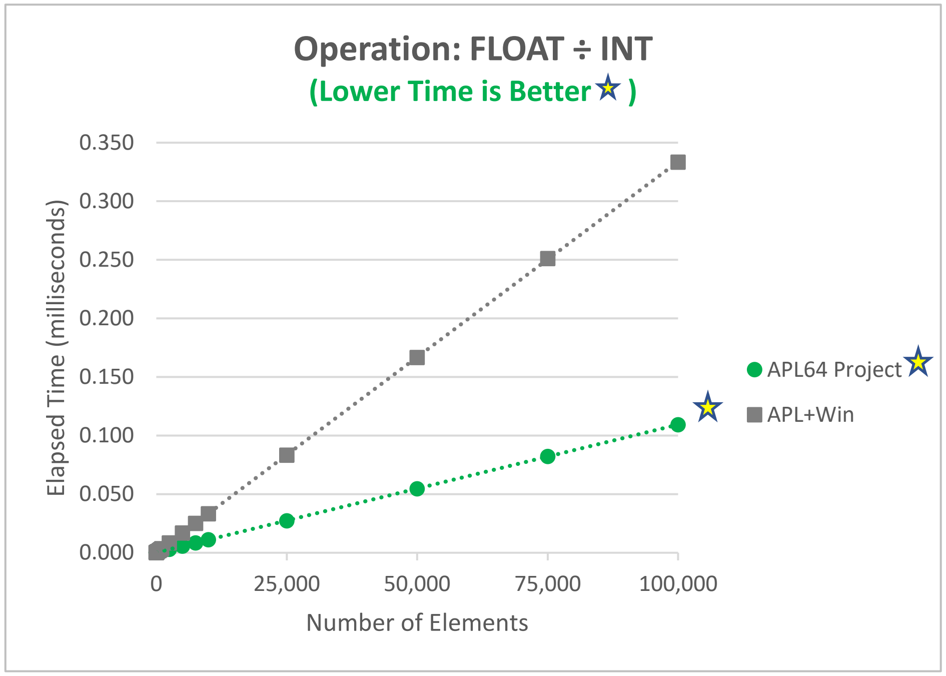

Operation: INT + BOOL
(Lower Time is Better)
Elapsed Time (milliseconds)
0.200
0.180
0.160
0.140
0.120
0.100
0.080
0.060
0.040
0.020
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project
■ APL+Win

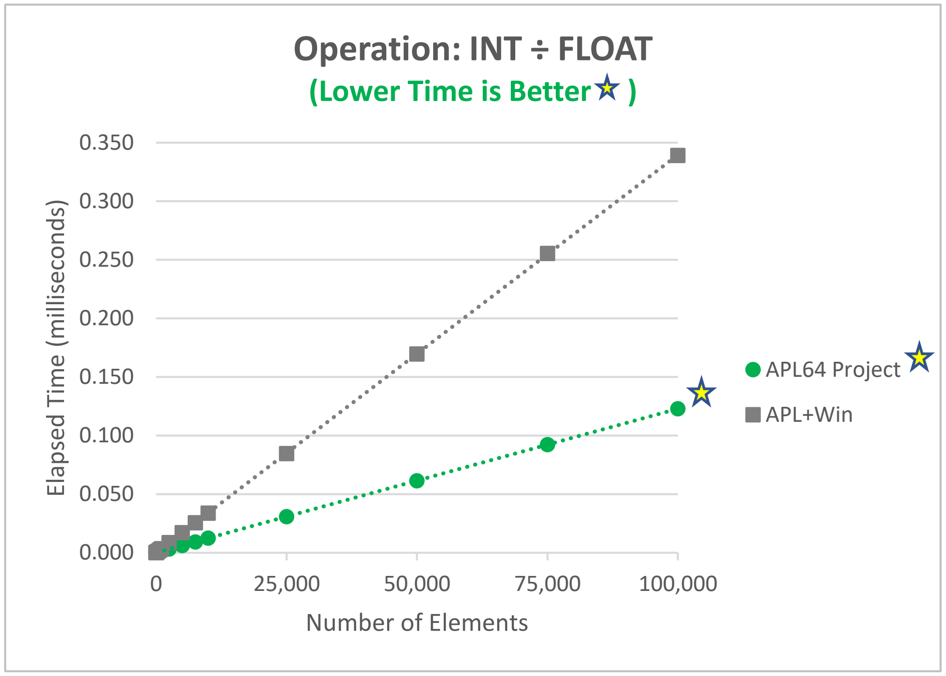

Operation: FLOAT × FLOAT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.090
0.080
0.070
0.060
0.050
0.040
0.030
0.020
0.010
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

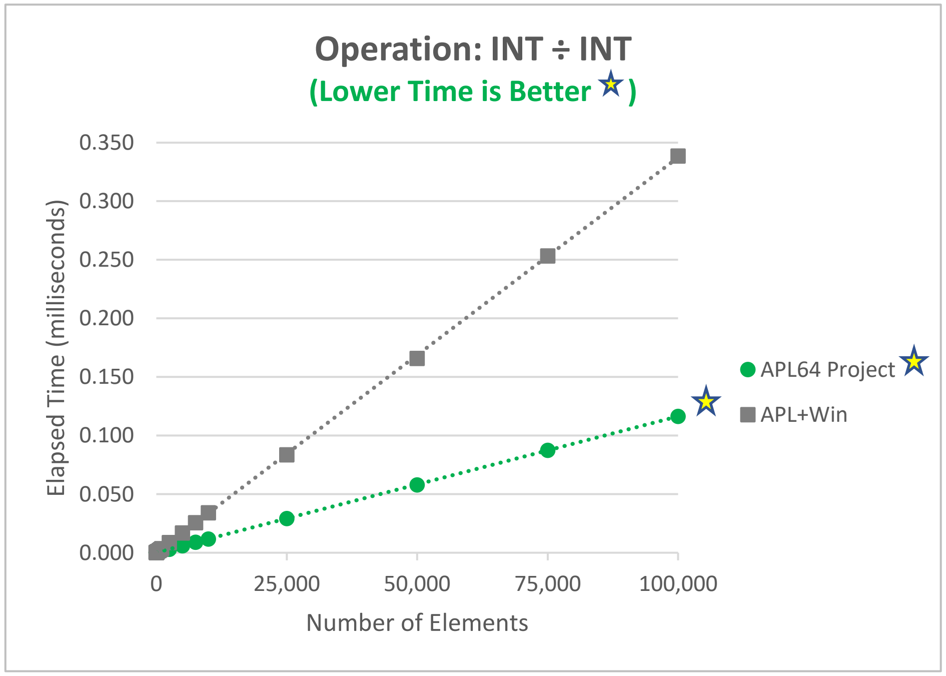

Operation: INT × FLOAT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.120
0.100
0.080
0.060
0.040
0.020
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

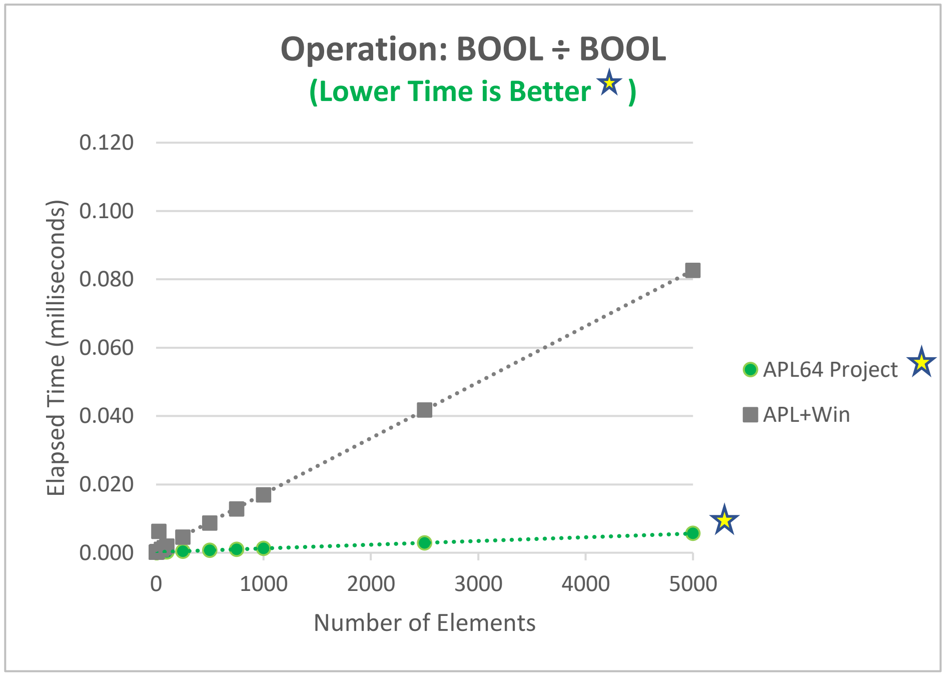

Operation: BOOL × FLOAT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.180
0.160
0.140
0.120
0.100
0.080
0.060
0.040
0.020
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

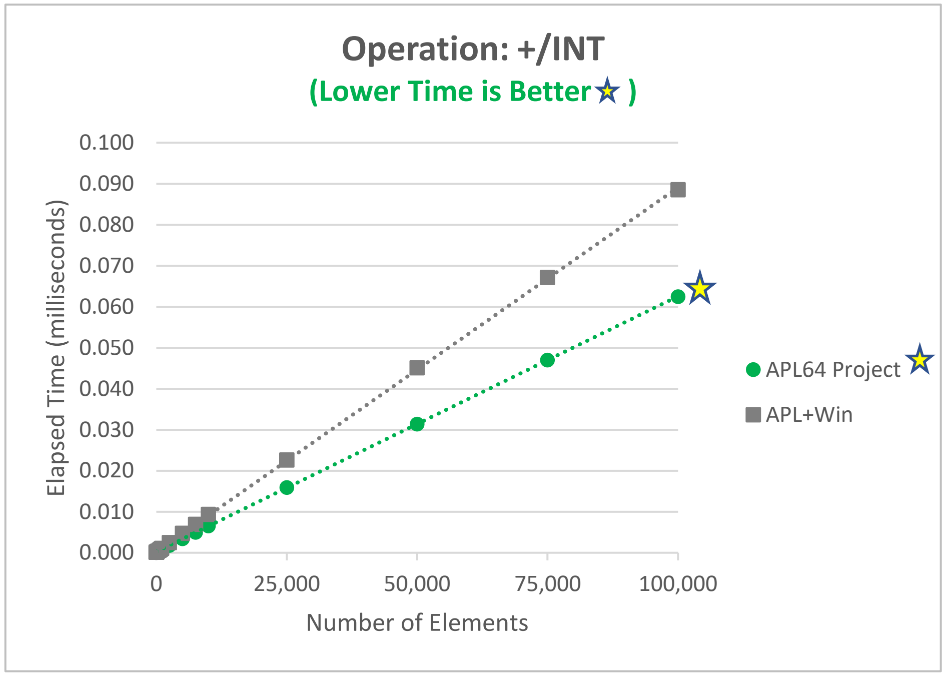

Operation: FLOAT ÷ FLOAT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.350
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

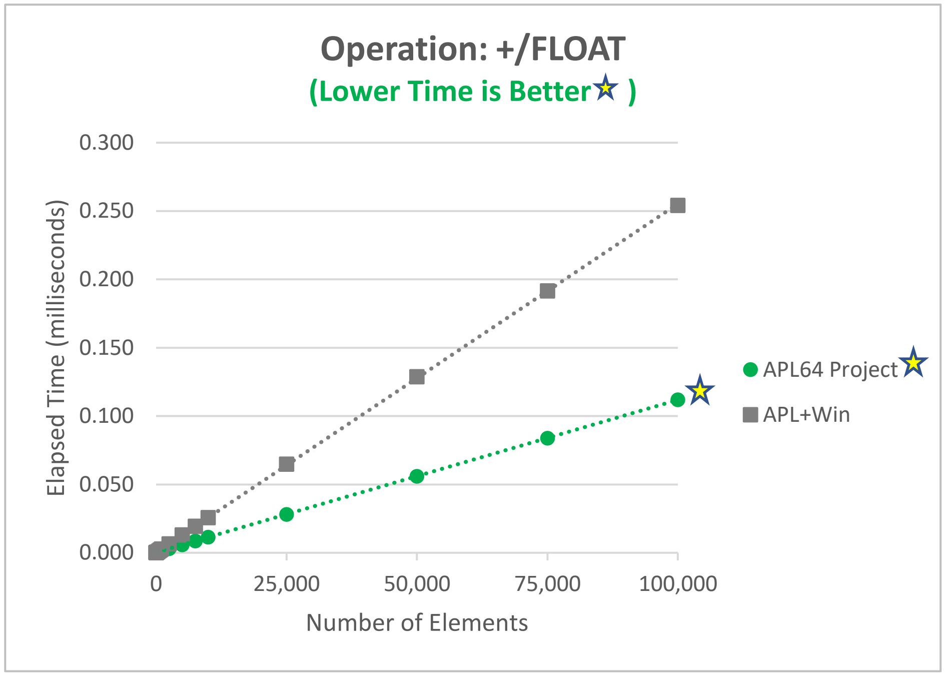

Operation: FLOAT ÷ INT
(Lower Time is Better)
Elapsed Time (milliseconds)
0.350
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
APL64 Project
APL+Win

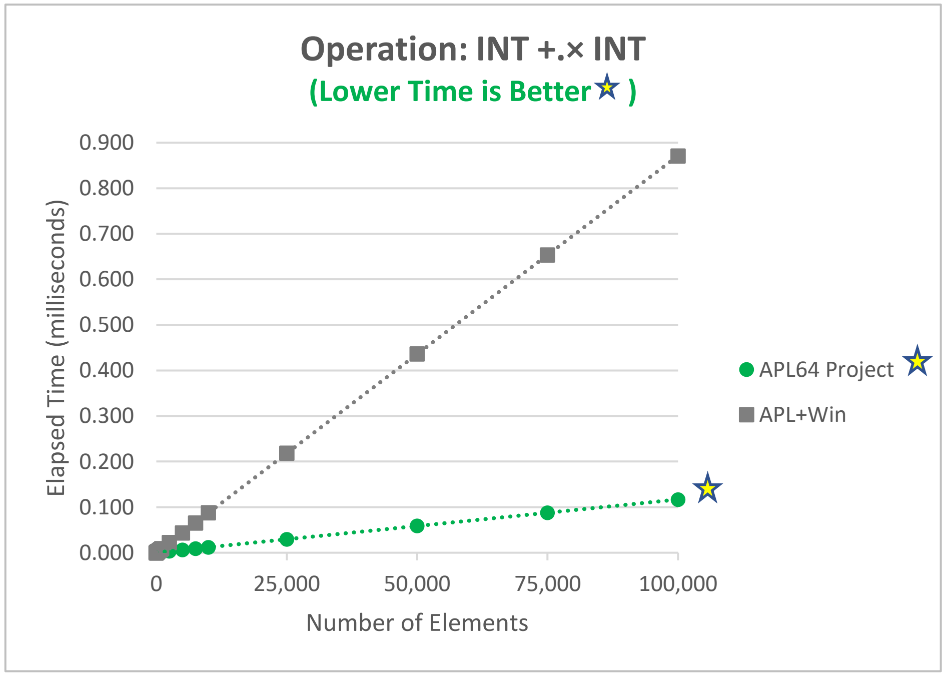

Operation: INT ÷ FLOAT
(Lower Time is Better)
Elapsed Time (milliseconds)
0.350
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
APL64 Project
APL+Win

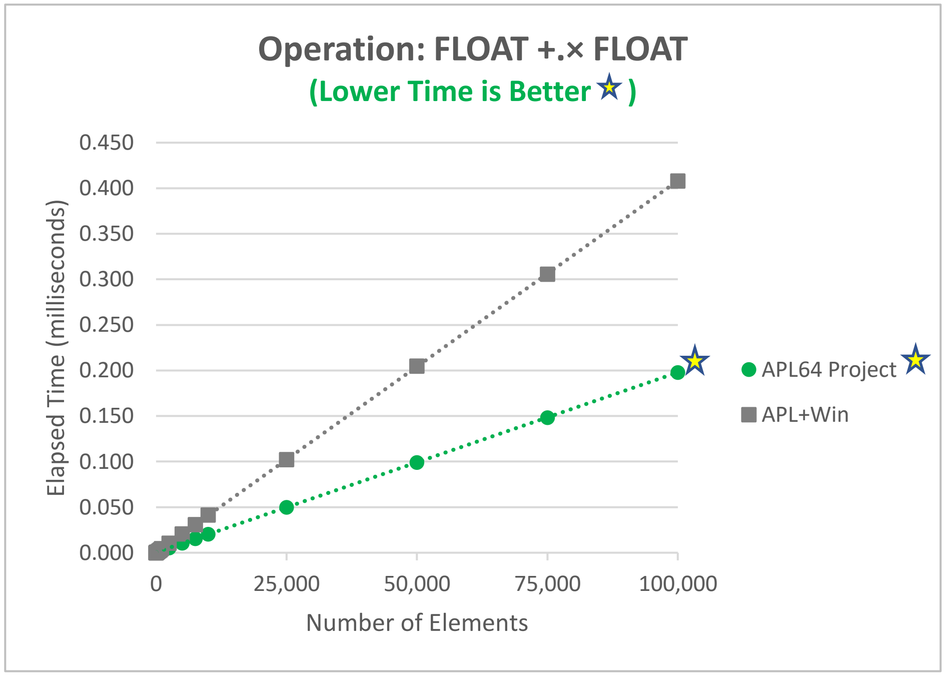

Operation: INT ÷ INT
(Lower Time is Better)
Elapsed Time (milliseconds)
0.350
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
APL64 Project
APL+Win

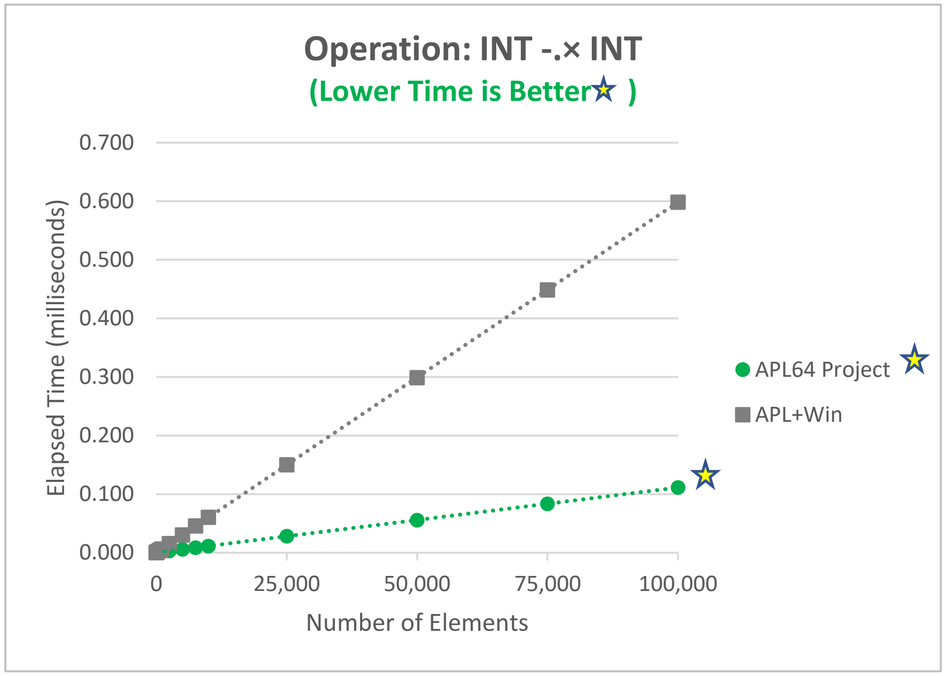

Operation: BOOL ÷ BOOL
(Lower Time is Better)
Elapsed Time (milliseconds)
0.120
0.100
0.080
0.060
0.040
0.020
0.000
0
1000
2000
3000
4000
5000
Number of Elements
APL64 Project
APL+Win

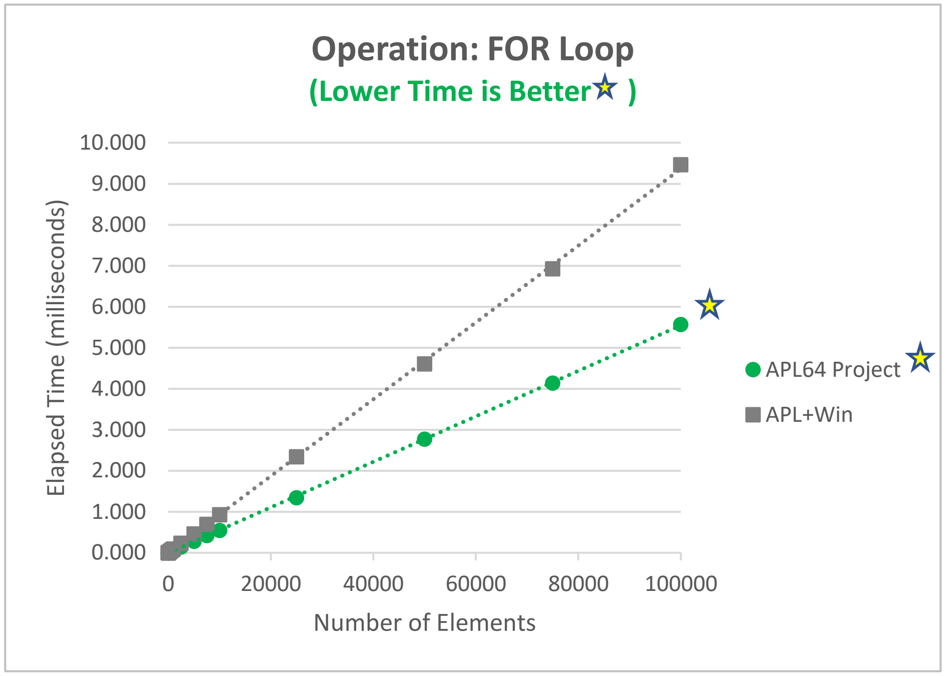

Operation: +/INT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.100
0.090
0.080
0.070
0.060
0.050
0.040
0.030
0.020
0.010
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

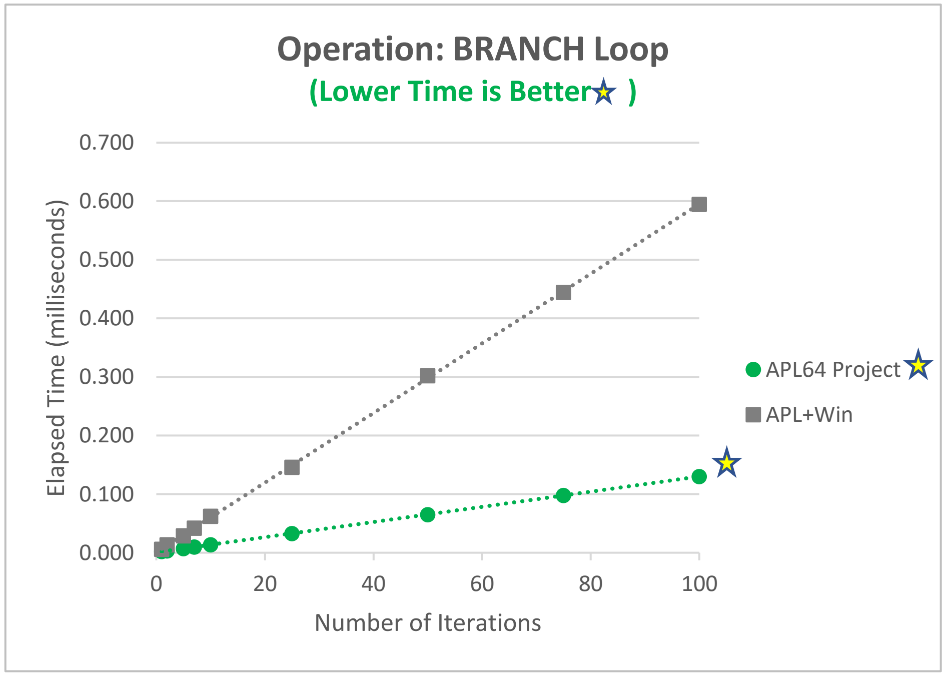

Operation: +/FLOAT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

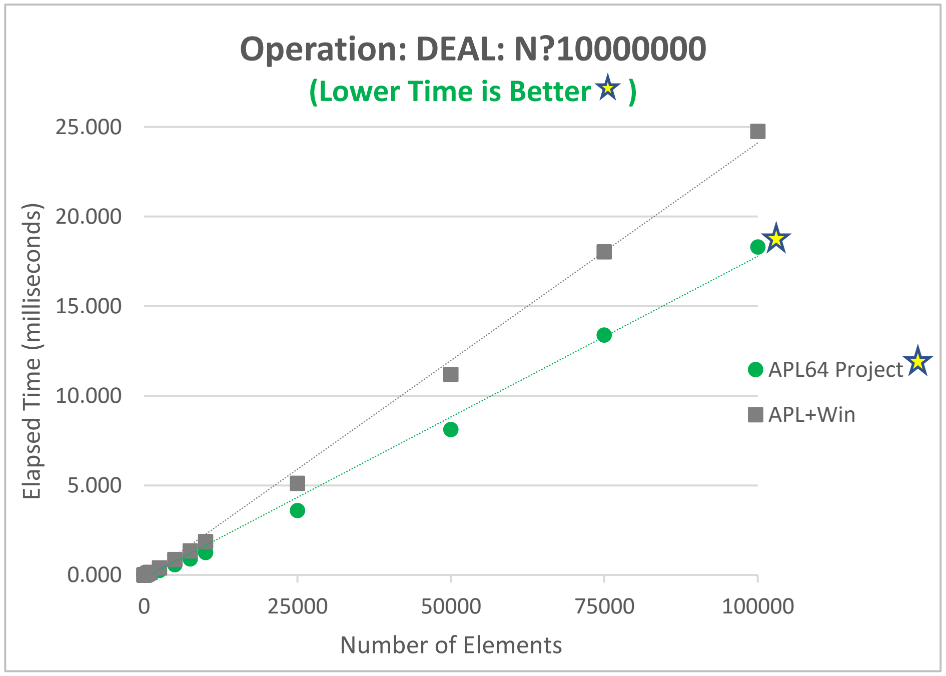

Operation: INT +.× INT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win

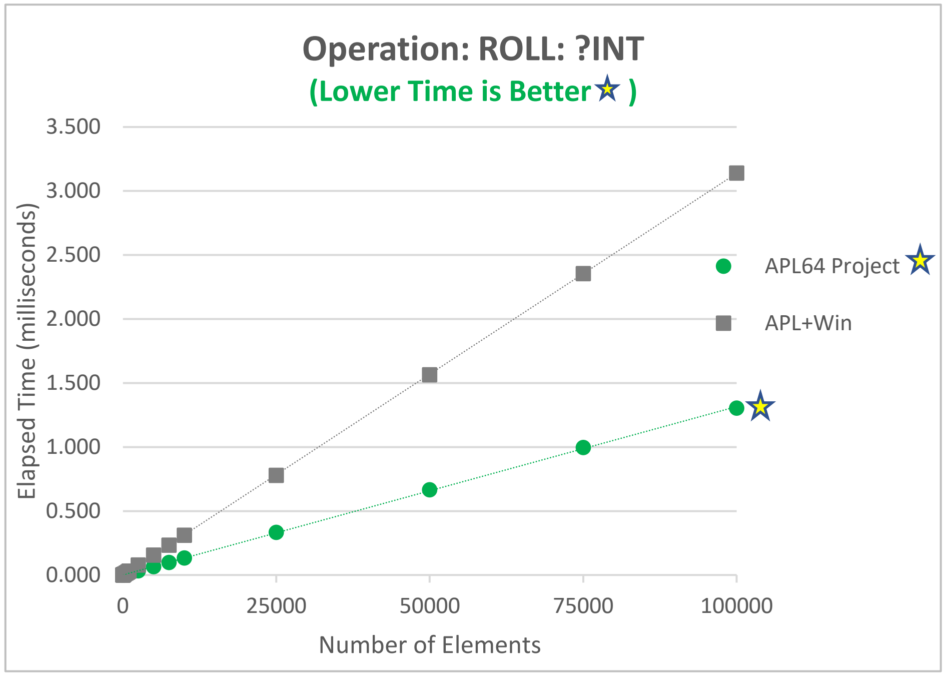

Operation: FLOAT +.× FLOAT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.450
0.400
0.350
0.300
0.250
0.200
0.150
0.100
0.050
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project ☆
■ APL+Win


Operation: INT -.× INT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
0
25,000
50,000
75,000
100,000
Number of Elements
● APL64 Project
■ APL+Win


Operation: FOR Loop
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
10.000
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0.000
0
20000
40000
60000
80000
100000
Number of Elements
● APL64 Project
■ APL+Win


Operation: BRANCH Loop
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.000
0
20
40
60
80
100
Number of Iterations
● APL64 Project
■ APL+Win


Operation: DEAL: N?10000000
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
25.000
20.000
15.000
10.000
5.000
0.000
0
25000
50000
75000
100000
Number of Elements
● APL64 Project
■ APL+Win

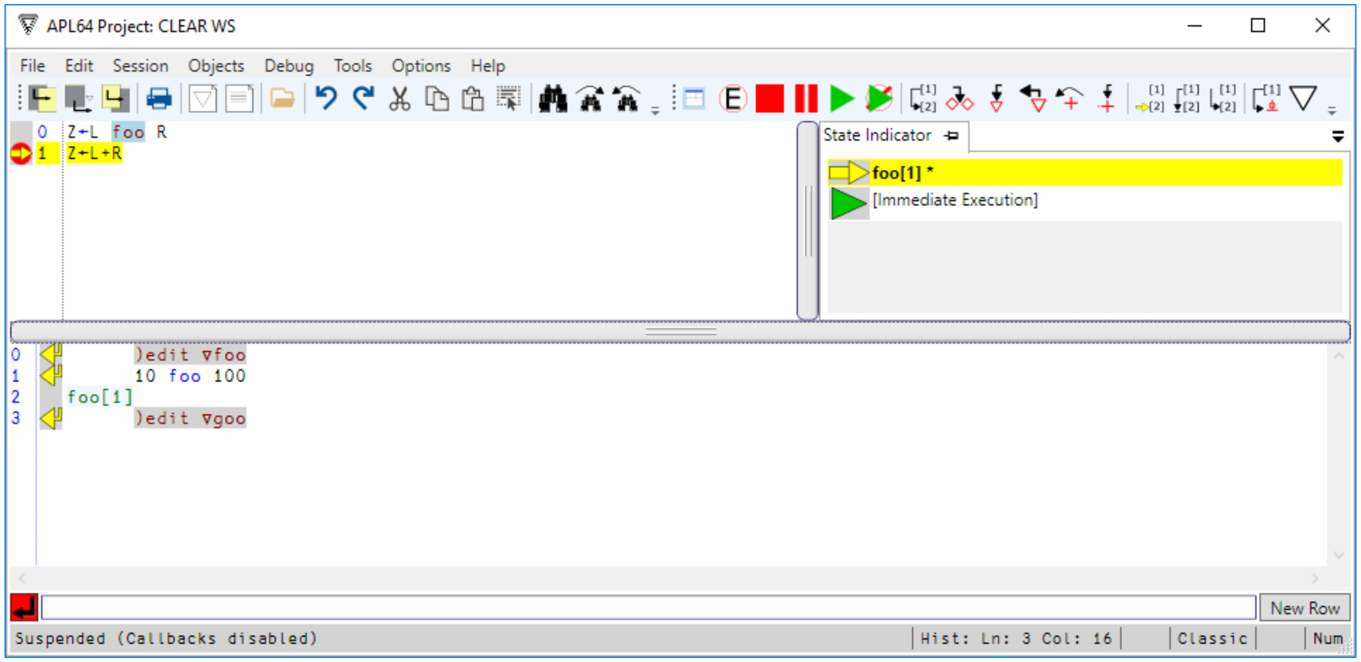

Operation: ROLL: ?INT
(Lower Time is Better ☆)
Elapsed Time (milliseconds)
3.500
3.000
2.500
2.000
1.500
1.000
0.500
0.000
0
25000
50000
75000
100000
Number of Elements
APL64 Project
APL+Win

APL64 Project Programmer Session Improvements and Enhancements

The APL64 Project Graphical User Interface (GUI) includes all the features available in APL+Win and more. This section highlights areas of the Programmer Session which have been improved by taking advantage of the up-to-date APL64 Project implementation environment.

Debugger

- The APL64 Project Debugger has been implemented:



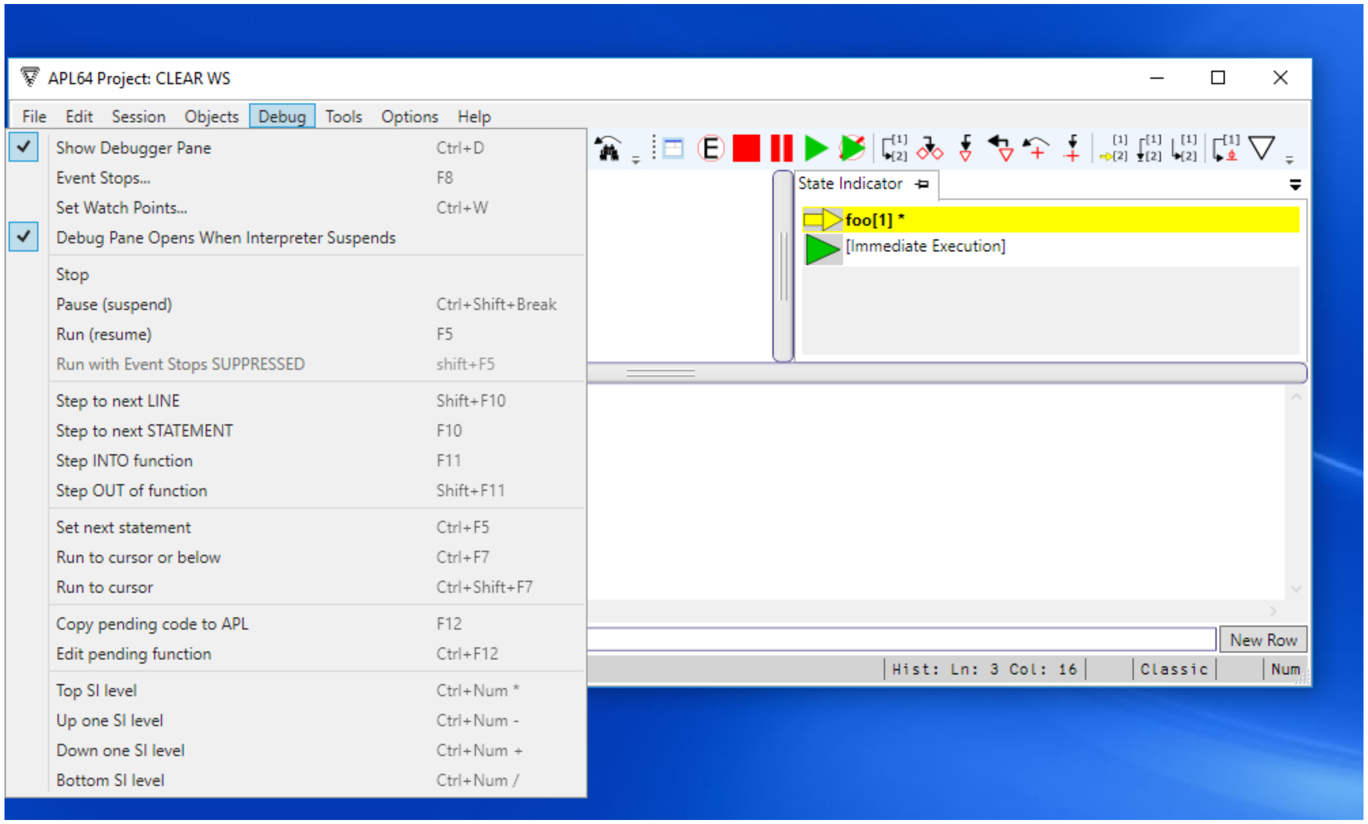
```
Z←L foo R
Z←L+R
```

```
foo[1] *
[Immediate Execution]
```

```
      )edit ∇foo
      10 foo 100
foo[1]
      )edit ∇goo
```

```
Suspended (Callbacks disabled)     Hist: Ln: 3 Col: 16     Classic     Num
```

Debugger menus and toolbar items have been implemented:


APL64 Project: CLEAR WS

File Edit Session Objects Debug Tools Options Help

✓	Show Debugger Pane	Ctrl+D
	Event Stops...	F8
	Set Watch Points...	Ctrl+W
✓	Debug Pane Opens When Interpreter Suspends	
	Stop	
	Pause (suspend)	Ctrl+Shift+Break
	Run (resume)	F5
	Run with Event Stops SUPPRESSED	shift+F5
	Step to next LINE	Shift+F10
	Step to next STATEMENT	F10
	Step INTO function	F11
	Step OUT of function	Shift+F11
	Set next statement	Ctrl+F5
	Run to cursor or below	Ctrl+F7
	Run to cursor	Ctrl+Shift+F7
	Copy pending code to APL	F12
	Edit pending function	Ctrl+F12
	Top SI level	Ctrl+Num *
	Up one SI level	Ctrl+Num -
	Down one SI level	Ctrl+Num +
	Bottom SI level	Ctrl+Num /

State Indicator

foo[1] *

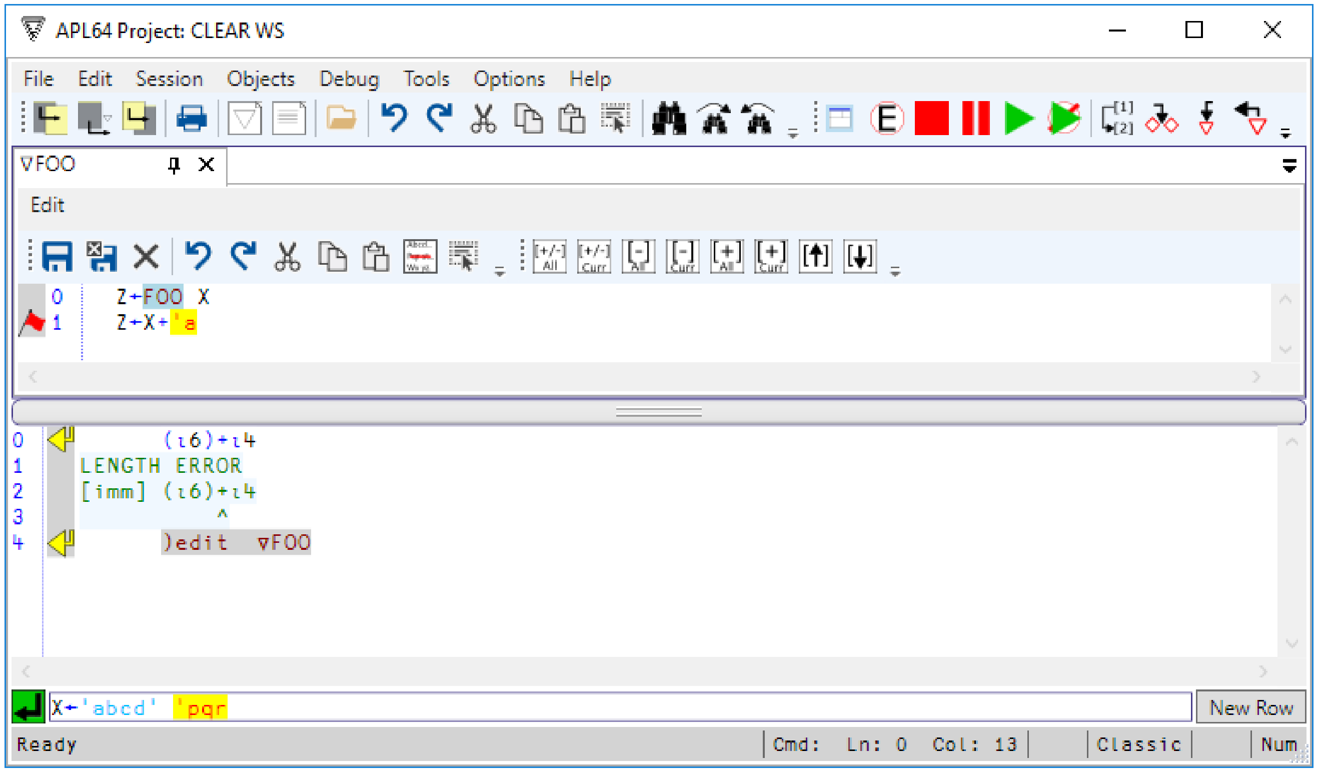
[Immediate Execution]

New Row

Hist: Ln: 3 Col: 16 | Classic | Num

Robust Syntax Checking

Syntax checking and syntax coloring have been implemented in the APL64 Project for function editors and the session command line.



```
    Z←FOO X
    Z←X+'a
```

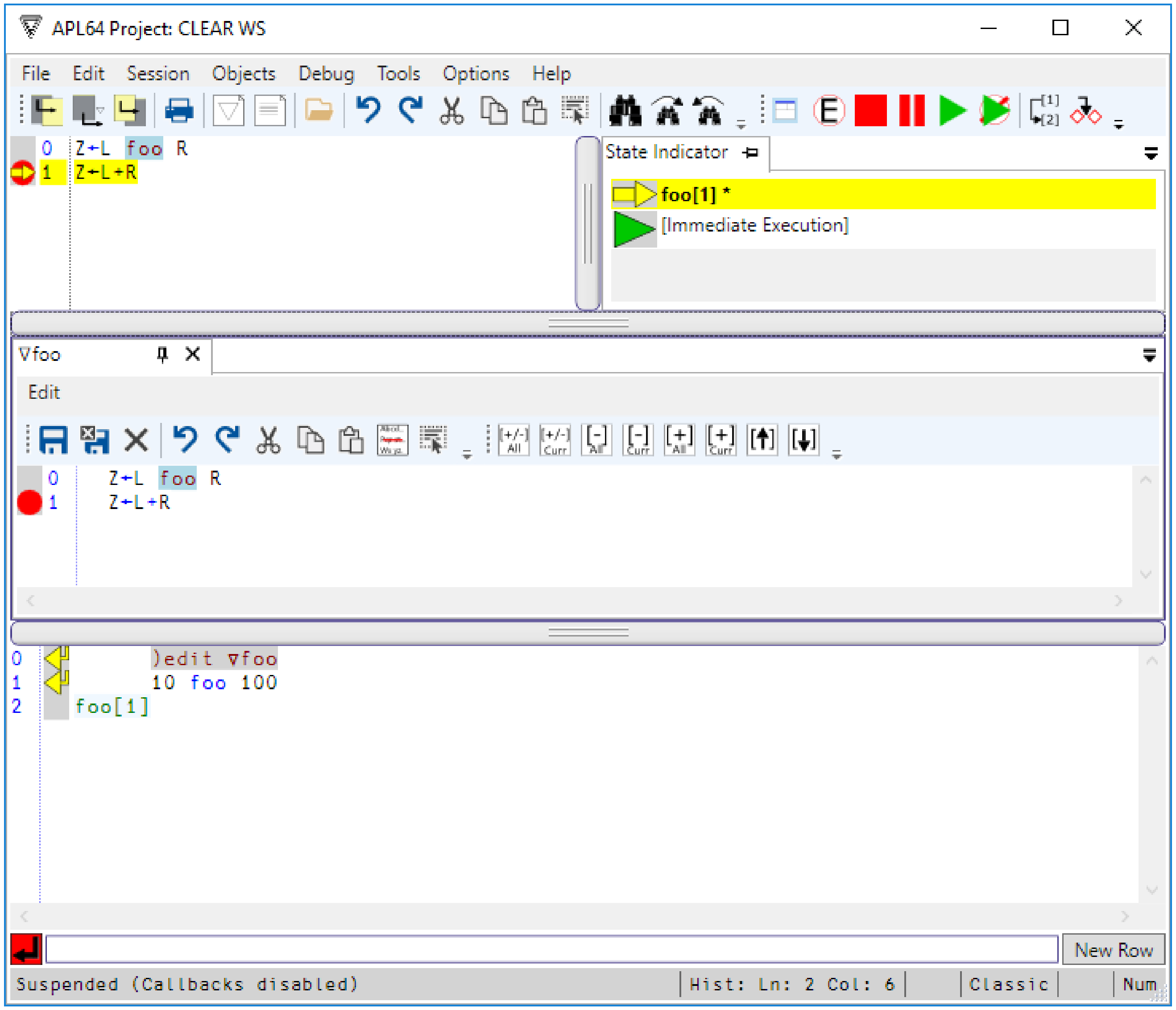
```
        (⍳6)+⍳4
LENGTH ERROR
[imm] (⍳6)+⍳4
          ^
        )edit  ∇FOO
```

```
X←'abcd' 'pqr
```

```
Ready    Cmd:  Ln: 0  Col: 13    Classic    Num
```

Flexible Session Panes Layout

- The user may float object editor and state indicator panes anywhere on any connected display, independently of the pane containing the session history, session command line and debugged function listing.
- All panes docked with object editors vertically tiled:

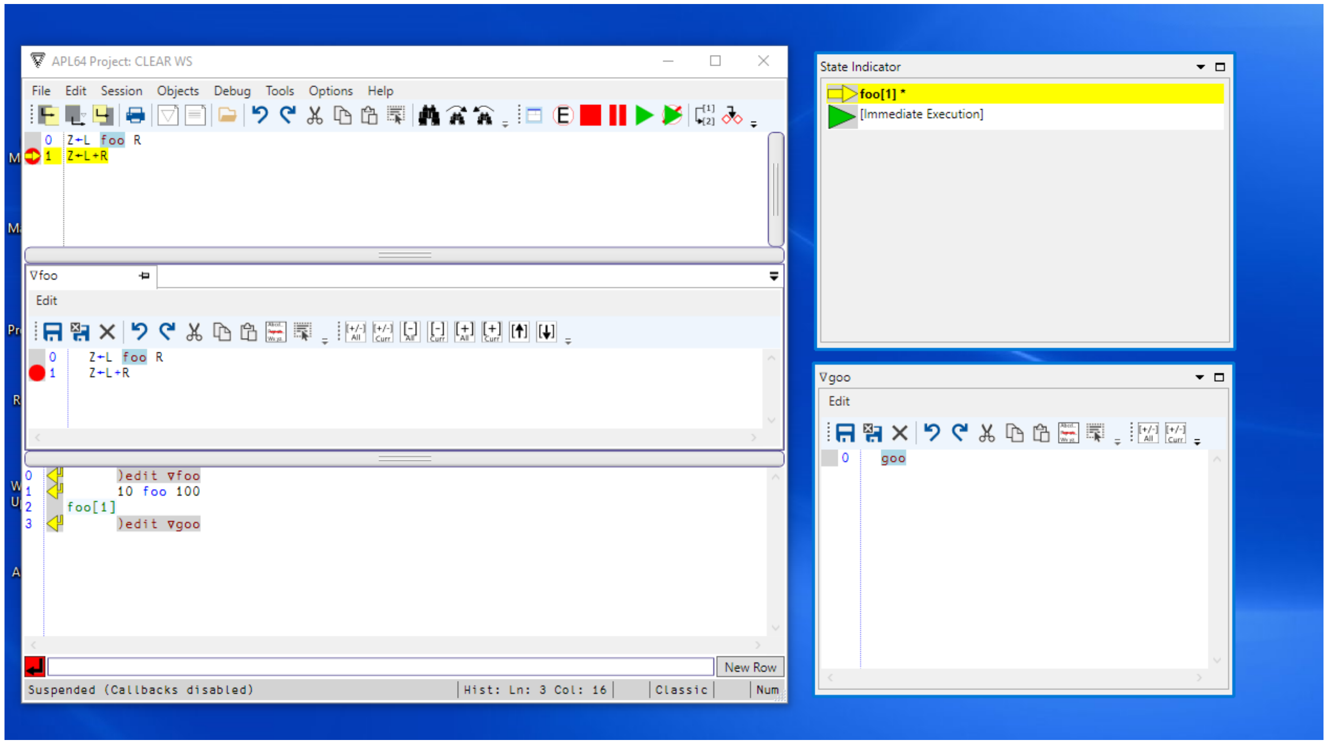


```
Z←L foo R
Z←L+R
```

```
      )edit ∇foo
      10 foo 100
foo[1]
```

```
Suspended (Callbacks disabled)    Hist: Ln: 2 Col: 6    Classic    Num
```

- Some panes Floating:



```
Z←L foo R
Z←L+R
```

∇foo:

```
Z←L foo R
Z←L+R
```

Session:

```
      )edit ∇foo
      10 foo 100
foo[1]
      )edit ∇goo
```

Suspended (Callbacks disabled) | Hist: Ln: 3 Col: 16 | Classic | Num

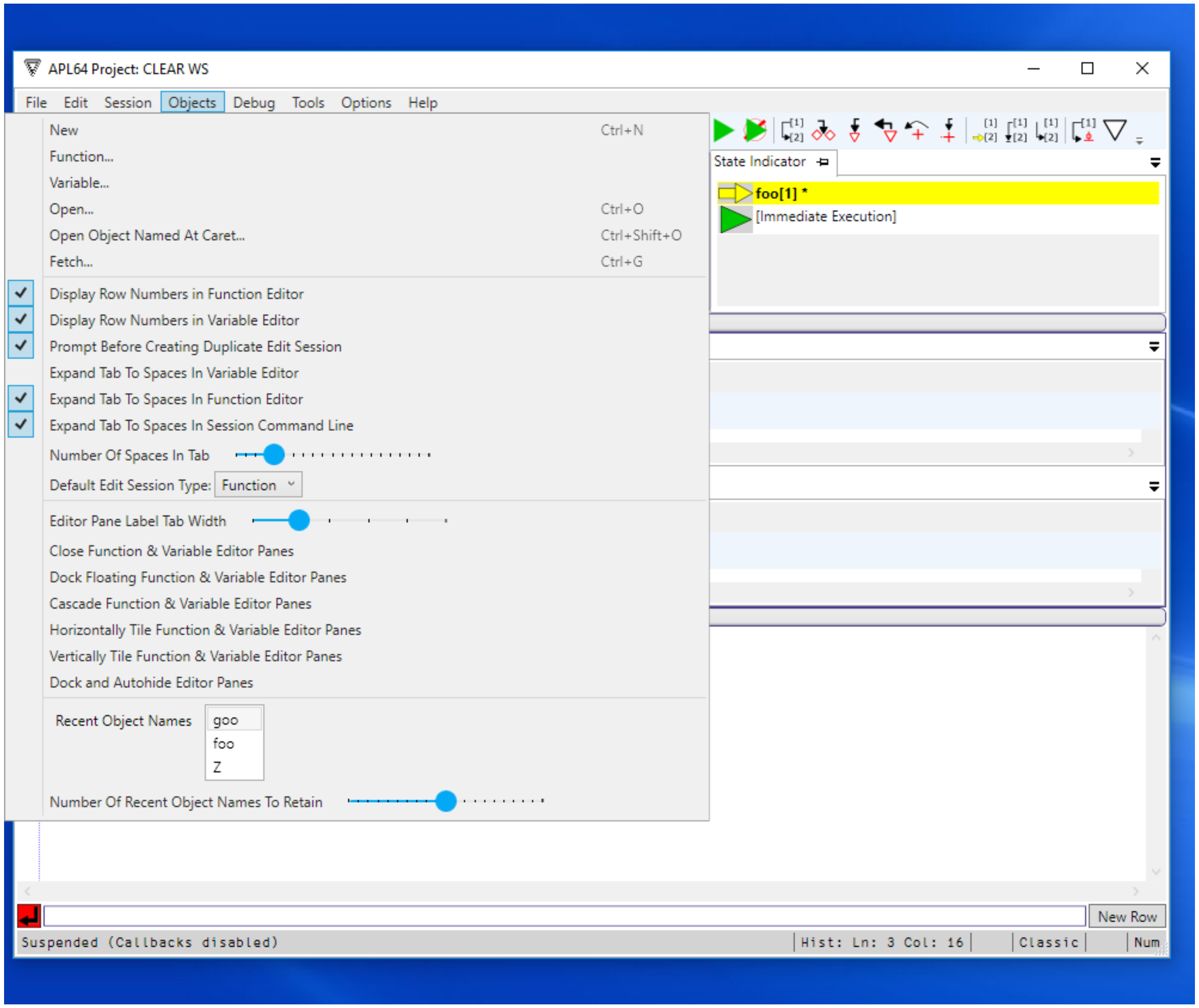
State Indicator:

```
foo[1] *
[Immediate Execution]
```

∇goo:

```
goo
```

- The Objects menu provides convenient options to close, dock, cascade, horizontal tile, vertical tile and autohide object editor panes.



APL64 Project: CLEAR WS

File Edit Session Objects Debug Tools Options Help

New Ctrl+N
Function...
Variable...
Open... Ctrl+O
Open Object Named At Caret... Ctrl+Shift+O
Fetch... Ctrl+G

✓ Display Row Numbers in Function Editor
✓ Display Row Numbers in Variable Editor
✓ Prompt Before Creating Duplicate Edit Session
Expand Tab To Spaces In Variable Editor
✓ Expand Tab To Spaces In Function Editor
✓ Expand Tab To Spaces In Session Command Line
Number Of Spaces In Tab
Default Edit Session Type: Function

Editor Pane Label Tab Width
Close Function & Variable Editor Panes
Dock Floating Function & Variable Editor Panes
Cascade Function & Variable Editor Panes
Horizontally Tile Function & Variable Editor Panes
Vertically Tile Function & Variable Editor Panes
Dock and Autohide Editor Panes

Recent Object Names: goo, foo, Z

Number Of Recent Object Names To Retain

State Indicator

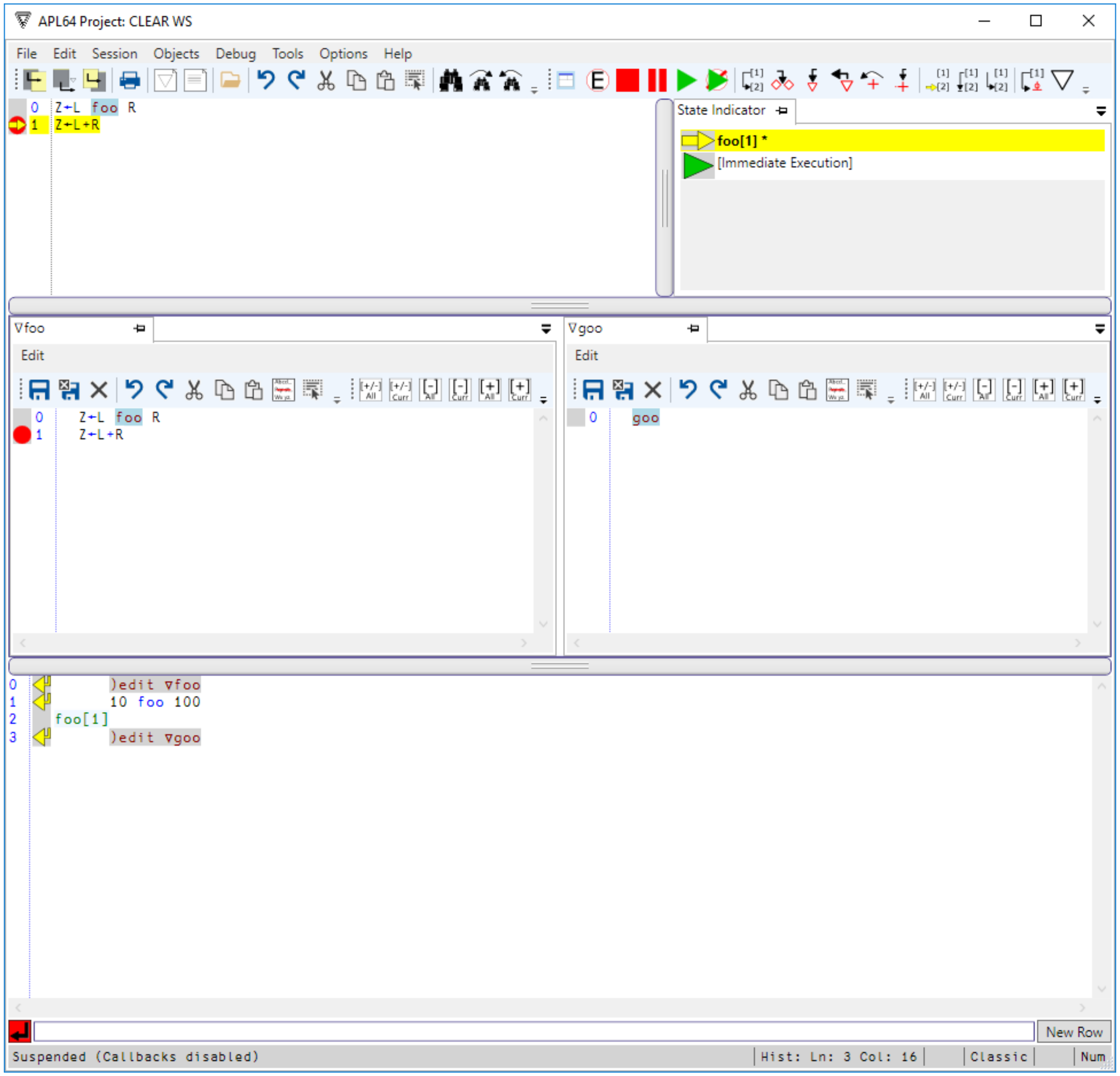
foo[1] *

[Immediate Execution]

New Row

```
Suspended (Callbacks disabled)          Hist: Ln: 3 Col: 16          Classic     Num
```

- Horizontally tiled object editors:



```
Z←L foo R
Z←L+R
```

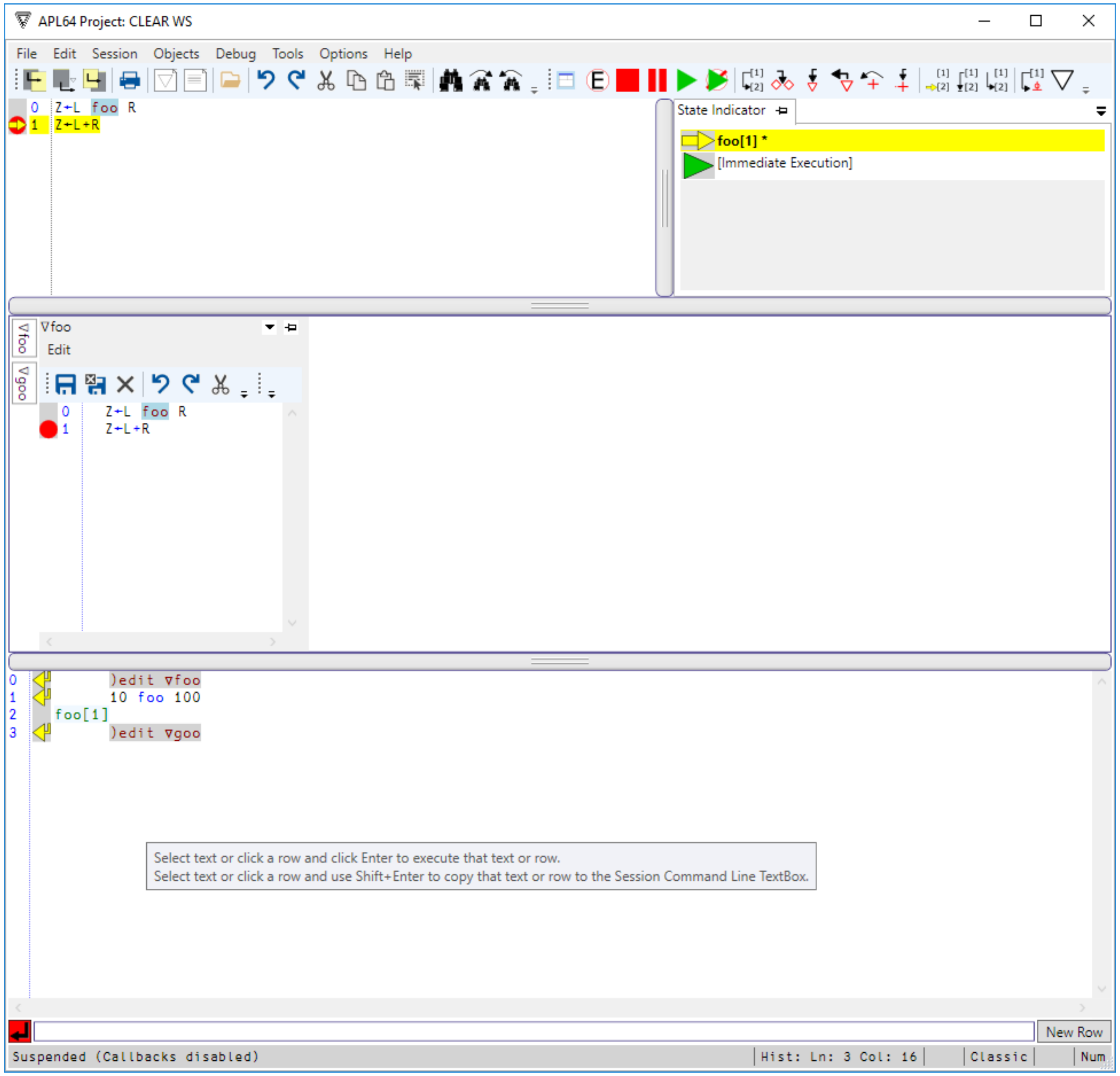
```
foo[1] *
[Immediate Execution]
```

```
goo
```

```
      )edit ∇foo
      10 foo 100
foo[1]
      )edit ∇goo
```

```
Suspended (Callbacks disabled)    Hist: Ln: 3 Col: 16    Classic    Num
```

- All Panes Docked and Object Editors in Auto-hide mode:



```
Z←L foo R
Z←L+R
```

```
foo[1] *
[Immediate Execution]
```

```
Z←L foo R
Z←L+R
```

```
      )edit ∇foo
      10 foo 100
foo[1]
      )edit ∇goo
```

Select text or click a row and click Enter to execute that text or row.
Select text or click a row and use Shift+Enter to copy that text or row to the Session Command Line TextBox.

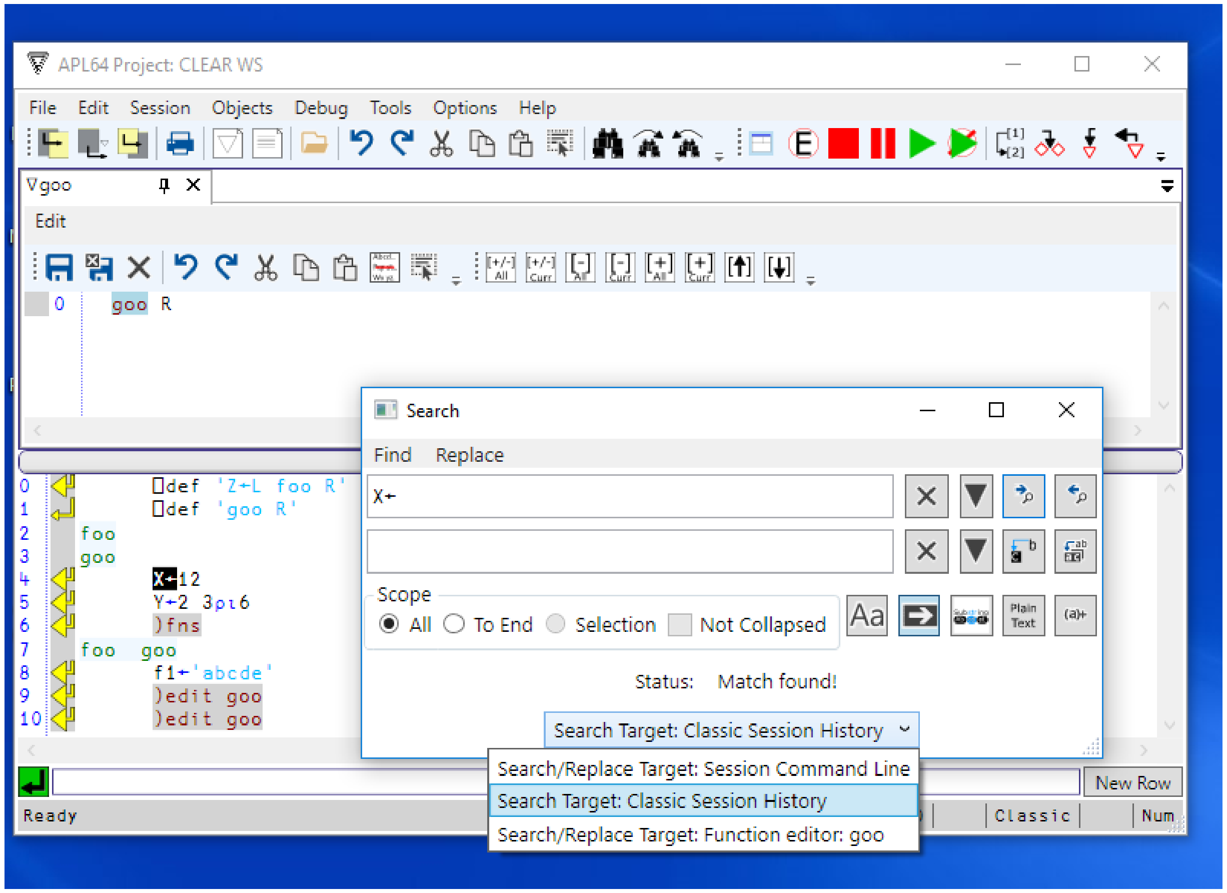
```
Suspended (Callbacks disabled)        Hist: Ln: 3 Col: 16    Classic    Num
```

Toolbars

- Debugger and Editor toolbars have been implemented

Menus

- The Load and Xload menus will transparently load the APL64 Project workspace and a previously-saved APL+Win workspace.
- The Find and Replace dialogs have been unified providing the search target option and searched text format options for plain text, wild card and regex.



```
0 goo R
```

```
            ⎕def 'Z←L foo R'
            ⎕def 'goo R'
    foo
    goo
            X←12
            Y←2 3ρι6
            )fns
    foo  goo
            f1←'abcde'
            )edit goo
            )edit goo
```

Find: X←

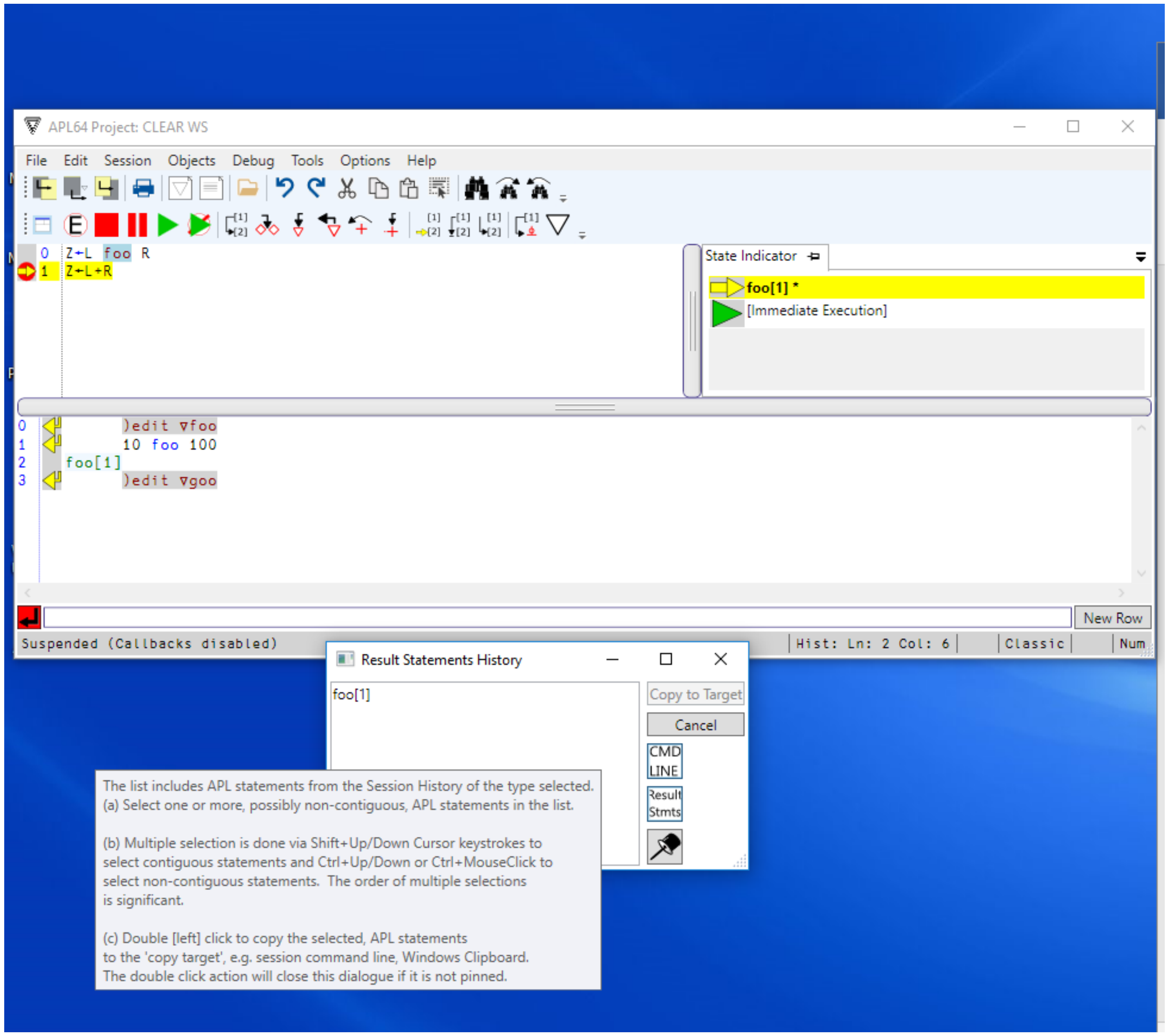
Scope: [✓ All] To End Selection Not Collapsed

Status: Match found!

Search Target: Classic Session History

- Search/Replace Target: Session Command Line
- Search Target: Classic Session History
- Search/Replace Target: Function editor: goo

- The Gather menu provides an enhanced dialog to select and use APL statements from the Session History pane. Selections can be filtered by Executed, Result and all Session History statements. The copy target can be the Session Command Line or the Windows Clipboard. The dialog may be pinned and located anywhere on a connected display.



```
0  Z←L foo R
1  Z←L+R
```

foo[1] *
[Immediate Execution]

```
      )edit ∇foo
      10 foo 100
foo[1]
      )edit ∇goo
```

Suspended (Callbacks disabled)   Hist: Ln: 2 Col: 6   Classic   Num

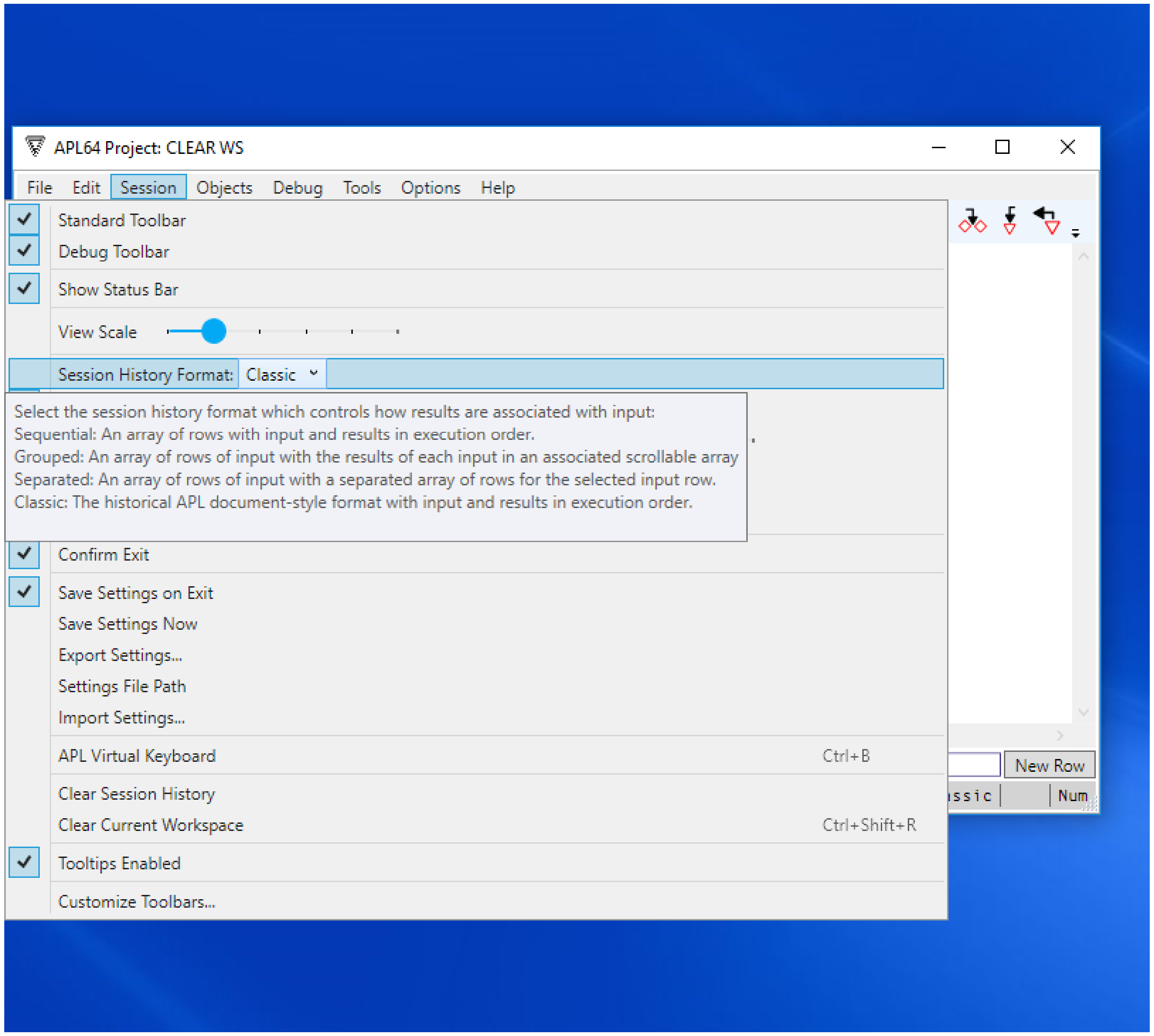
foo[1]

The list includes APL statements from the Session History of the type selected.
(a) Select one or more, possibly non-contiguous, APL statements in the list.

(b) Multiple selection is done via Shift+Up/Down Cursor keystrokes to select contiguous statements and Ctrl+Up/Down or Ctrl+MouseClick to select non-contiguous statements. The order of multiple selections is significant.

(c) Double [left] click to copy the selected, APL statements to the 'copy target', e.g. session command line, Windows Clipboard. The double click action will close this dialogue if it is not pinned.

- The Session History Formats have been renamed to Classic, Sequential, Grouped, Separated:

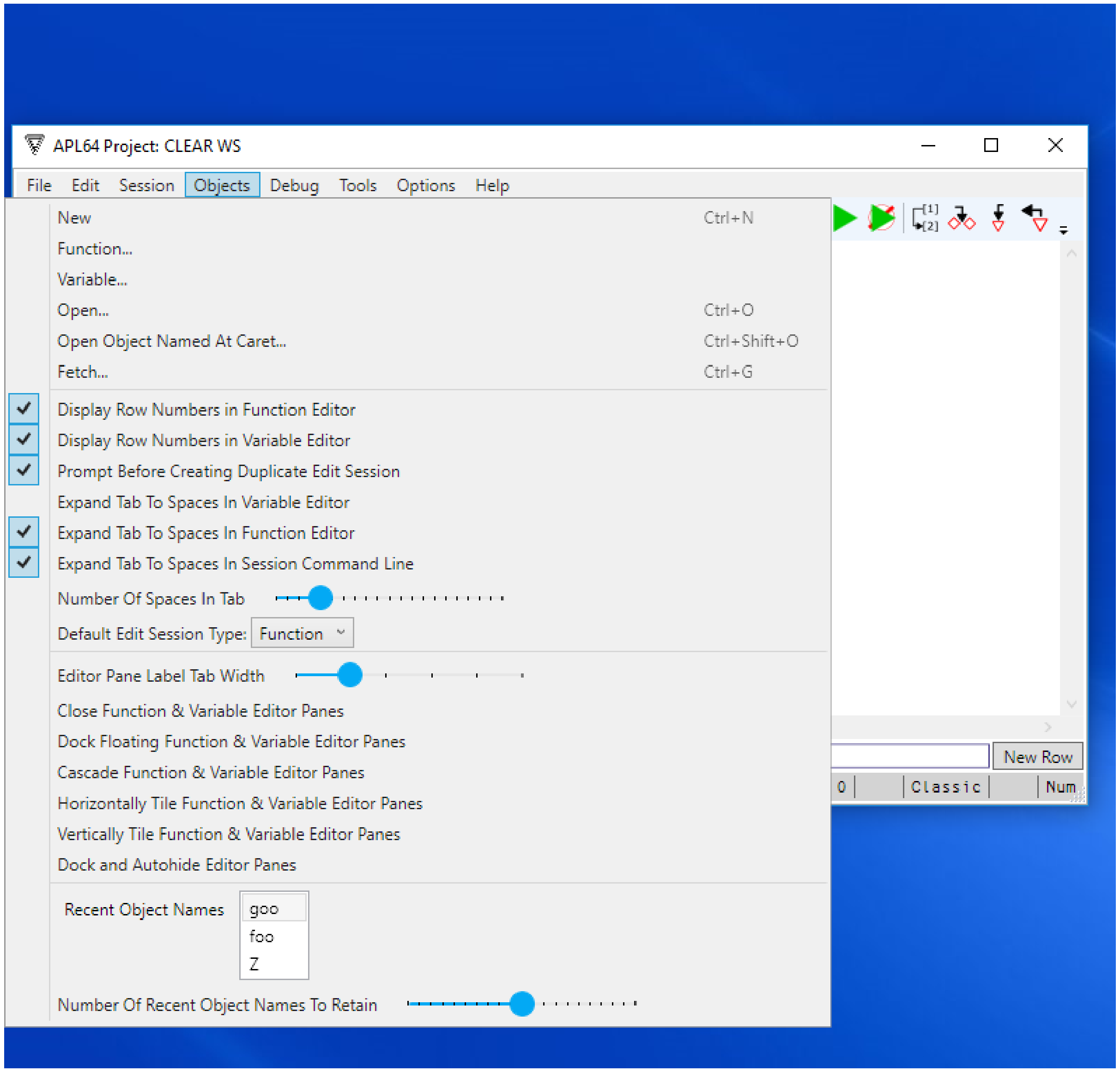


Select the session history format which controls how results are associated with input:
Sequential: An array of rows with input and results in execution order.
Grouped: An array of rows of input with the results of each input in an associated scrollable array
Separated: An array of rows of input with a separated array of rows for the selected input row.
Classic: The historical APL document-style format with input and results in execution order.

- Because the APL64 Project Session supports multi-row entries in the Session Command Line the Session Command Line Maximum Height menu item is available to control when vertical scrolling of a multi-row text in the Session Command Line will occur.

- The Tooltips Enabled menu option has been implemented to activate/de-activate tooltips which are now available for most APL64 Project Session features.

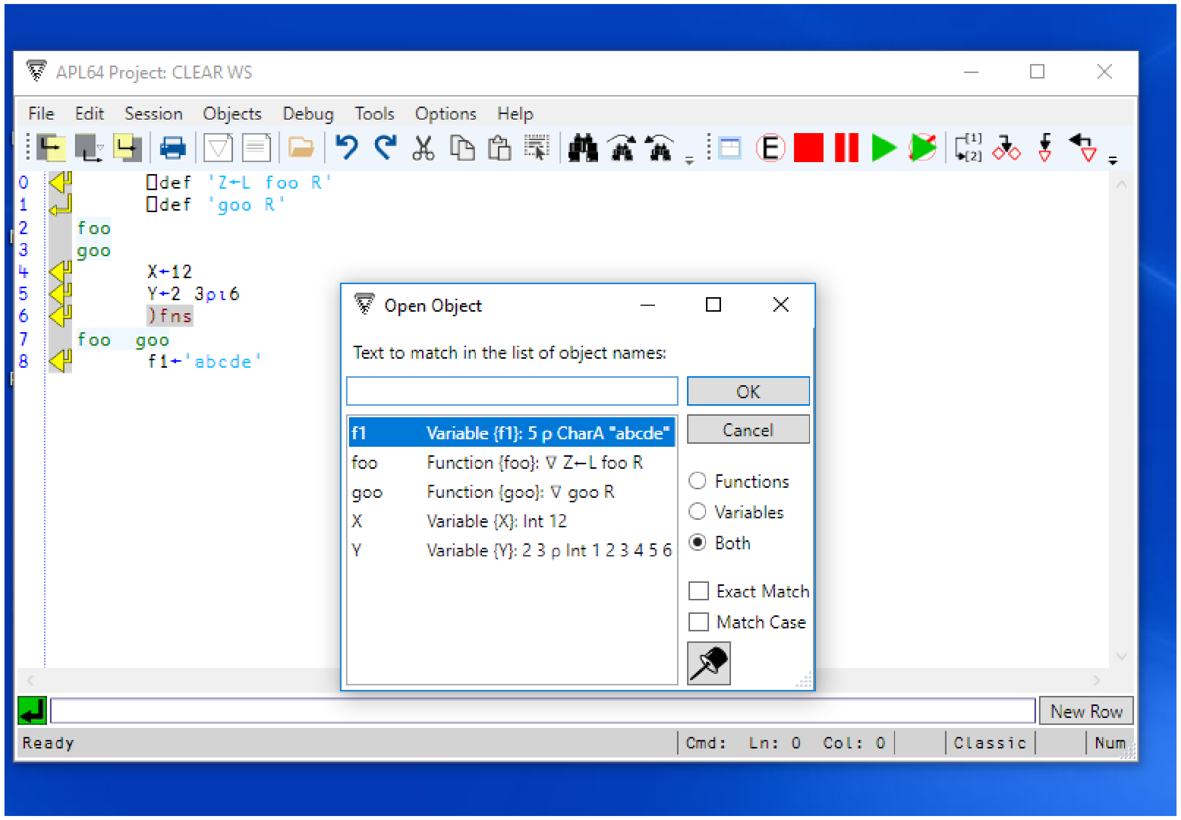
- Numerous Objects menu features have been implemented:


APL64 Project: CLEAR WS

File Edit Session Objects Debug Tools Options Help

	New	Ctrl+N
	Function...	
	Variable...	
	Open...	Ctrl+O
	Open Object Named At Caret...	Ctrl+Shift+O
	Fetch...	Ctrl+G
✓	Display Row Numbers in Function Editor	
✓	Display Row Numbers in Variable Editor	
✓	Prompt Before Creating Duplicate Edit Session	
	Expand Tab To Spaces In Variable Editor	
✓	Expand Tab To Spaces In Function Editor	
✓	Expand Tab To Spaces In Session Command Line	
	Number Of Spaces In Tab	
	Default Edit Session Type: Function	
	Editor Pane Label Tab Width	
	Close Function & Variable Editor Panes	
	Dock Floating Function & Variable Editor Panes	
	Cascade Function & Variable Editor Panes	
	Horizontally Tile Function & Variable Editor Panes	
	Vertically Tile Function & Variable Editor Panes	
	Dock and Autohide Editor Panes	
	Recent Object Names: goo, foo, Z	
	Number Of Recent Object Names To Retain	

- The Open object dialog is enhanced:
    - The list of available objects includes the objects name and value tip information.



```
        ⎕def 'Z←L foo R'
        ⎕def 'goo R'
foo
goo
        X←12
        Y←2 3⍴⍳6
        )fns
foo  goo
        f1←'abcde'
```

Open Object

Text to match in the list of object names:

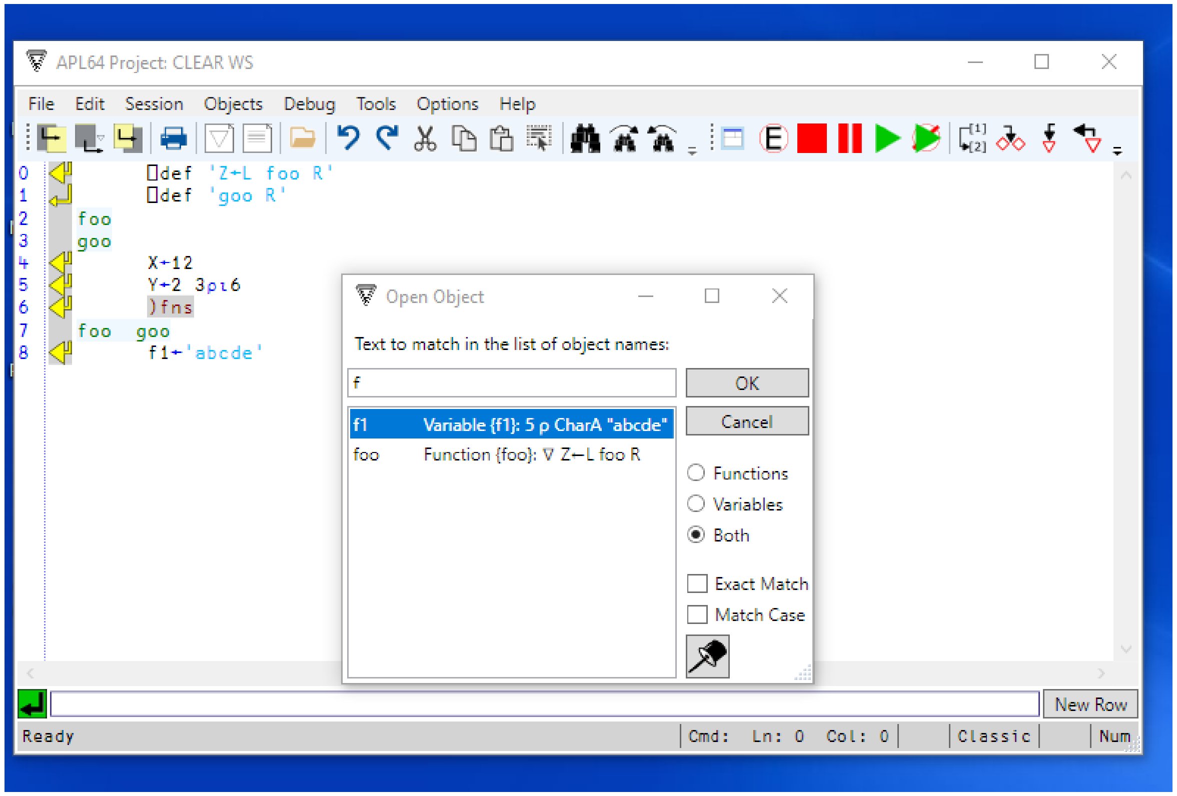
f1	Variable {f1}: 5 ρ CharA "abcde"
foo	Function {foo}: ∇ Z←L foo R
goo	Function {goo}: ∇ goo R
X	Variable {X}: Int 12
Y	Variable {Y}: 2 3 ρ Int 1 2 3 4 5 6

Functions [✓ Variables] Both — selected: Both

☐ Exact Match
☐ Match Case

Ready | Cmd: Ln: 0 Col: 0 | Classic | Num

- The object list is automatically filtered as the user types in the selection text box where filtering can be user selected among Exact Match and Match Case.



```
      ⎕def 'Z←L foo R'
      ⎕def 'goo R'
foo
goo
      X←12
      Y←2 3ρι6
      )fns
foo  goo
      f1←'abcde'
```

Open Object

Text to match in the list of object names: f

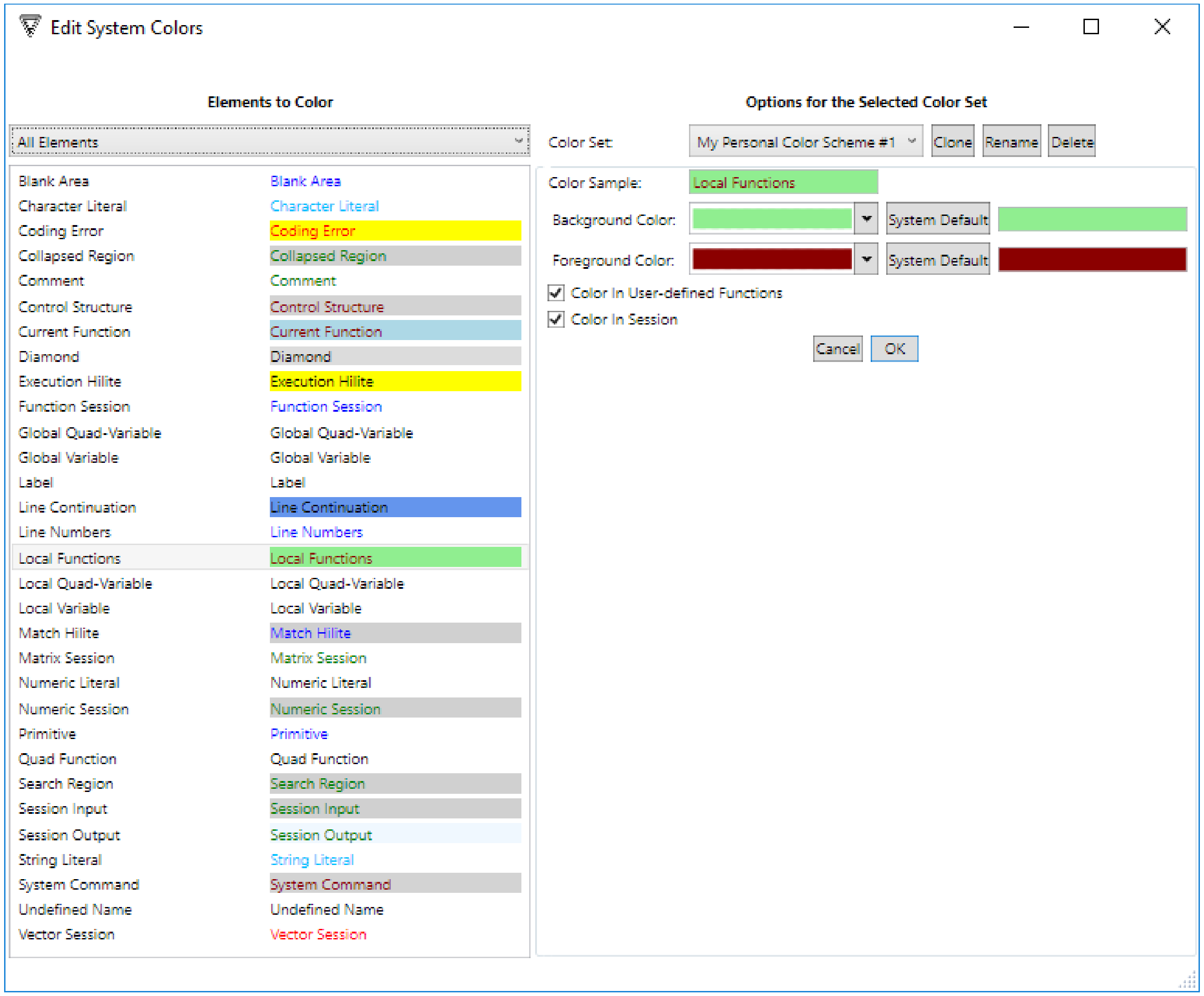
```
f1     Variable {f1}: 5 ρ CharA "abcde"
foo    Function {foo}: ∇ Z←L foo R
```

○ Functions
○ Variables
● Both

☐ Exact Match
☐ Match Case

Ready | Cmd: | Ln: 0 | Col: 0 | Classic | Num

- The Options > Colors … dialog has been enhanced:
  - Multiple stored color schemes are supported
  - Cloning of existing color schemes is supported
  - Current color scheme list includes Element Description and currently-selected color selections



Edit System Colors

Elements to Color

All Elements

Blank Area	Blank Area
Character Literal	Character Literal
Coding Error	Coding Error
Collapsed Region	Collapsed Region
Comment	Comment
Control Structure	Control Structure
Current Function	Current Function
Diamond	Diamond
Execution Hilite	Execution Hilite
Function Session	Function Session
Global Quad-Variable	Global Quad-Variable
Global Variable	Global Variable
Label	Label
Line Continuation	Line Continuation
Line Numbers	Line Numbers
Local Functions	Local Functions
Local Quad-Variable	Local Quad-Variable
Local Variable	Local Variable
Match Hilite	Match Hilite
Matrix Session	Matrix Session
Numeric Literal	Numeric Literal
Numeric Session	Numeric Session
Primitive	Primitive
Quad Function	Quad Function
Search Region	Search Region
Session Input	Session Input
Session Output	Session Output
String Literal	String Literal
System Command	System Command
Undefined Name	Undefined Name
Vector Session	Vector Session

Options for the Selected Color Set

Color Set: My Personal Color Scheme #1

Color Sample: Local Functions

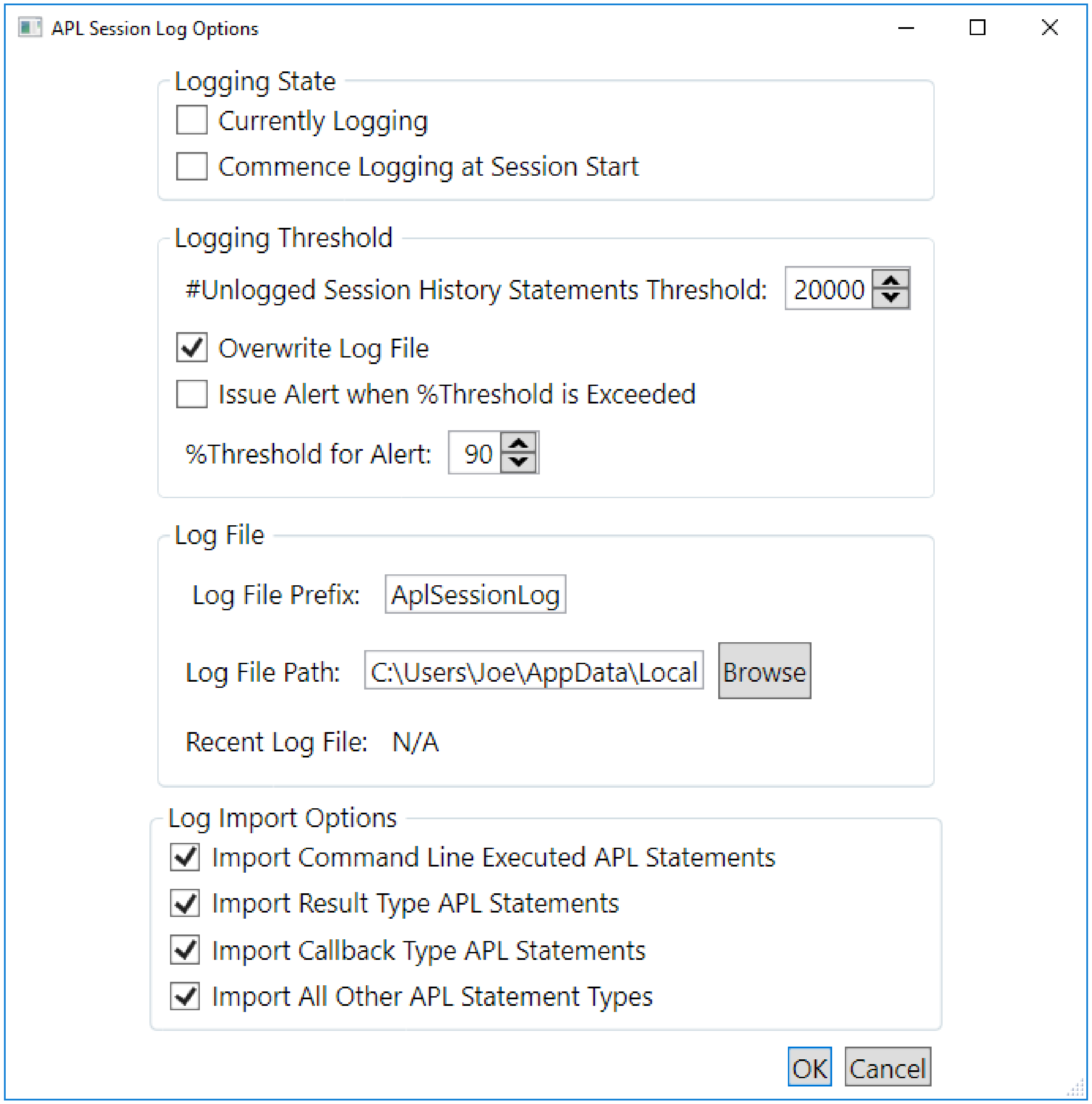
Background Color: 🟩 System Default

Foreground Color: 🟥 System Default

☑ Color In User-defined Functions

☑ Color In Session

- The APL Session Log Options dialog is enhanced:



APL Session Log Options

Logging State

- [ ] Currently Logging
- [ ] Commence Logging at Session Start

Logging Threshold

#Unlogged Session History Statements Threshold: 20000

- [✓] Overwrite Log File
- [ ] Issue Alert when %Threshold is Exceeded

%Threshold for Alert: 90

Log File

Log File Prefix: AplSessionLog

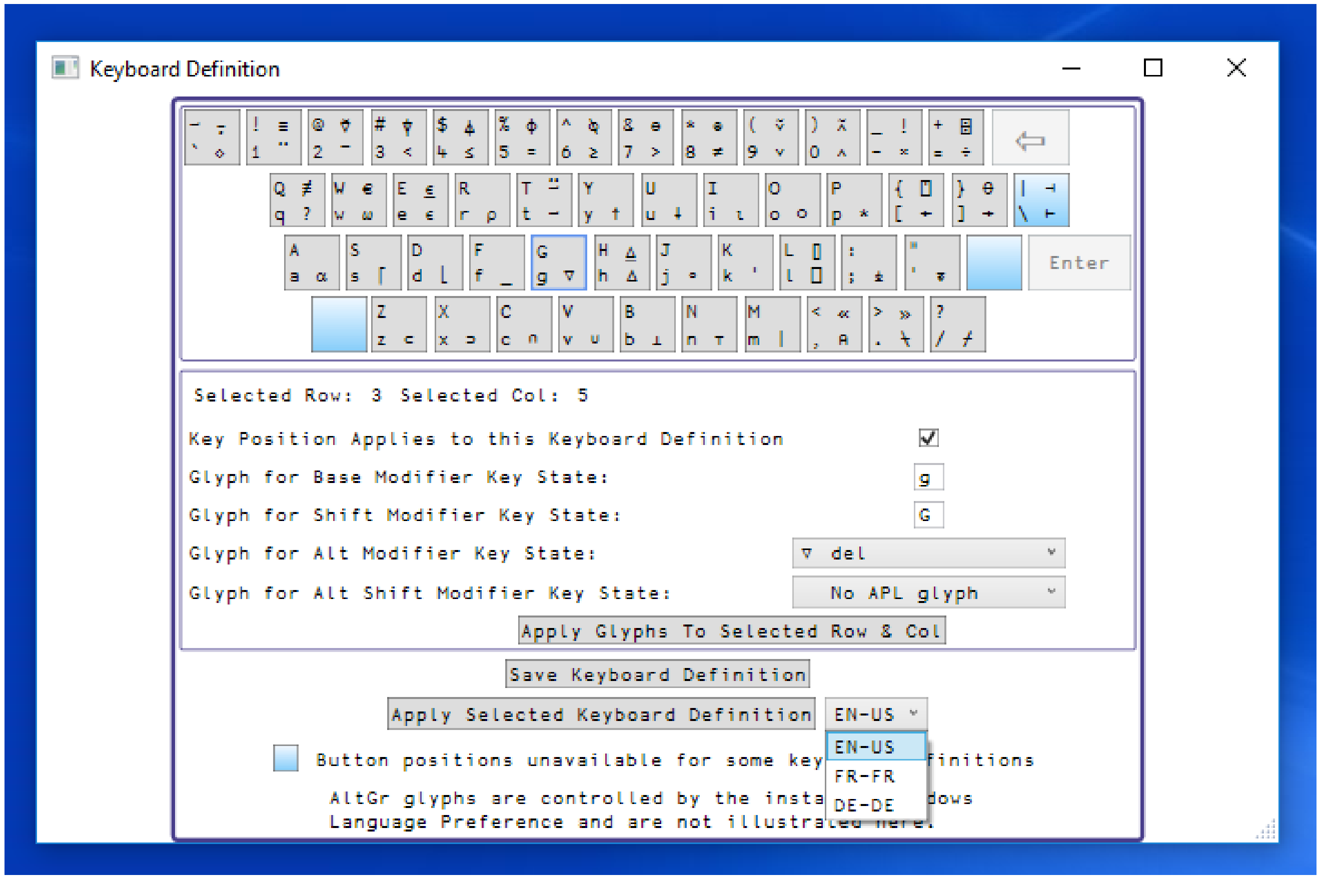
Log File Path: C:\Users\Joe\AppData\Local

Recent Log File: N/A

Log Import Options

- [✓] Import Command Line Executed APL Statements
- [✓] Import Result Type APL Statements
- [✓] Import Callback Type APL Statements
- [✓] Import All Other APL Statement Types

- The Keyboard Definition dialog has been implemented to support user specification of the Unicode glyphs associated with keyboard keys when the user has installed the associated Windows language preference component on the target workstation.


Keyboard Definition
Enter
Apply Glyphs To Selected Row & Col
Save Keyboard Definition
Apply Selected Keyboard Definition
EN-US
FR-FR
DE-DE

Selected Row: 3 Selected Col: 5

Key Position Applies to this Keyboard Definition ✓

Glyph for Base Modifier Key State: g

Glyph for Shift Modifier Key State: G

Glyph for Alt Modifier Key State: ∇ del

Glyph for Alt Shift Modifier Key State: No APL glyph

Apply Selected Keyboard Definition: EN-US

Button positions unavailable for some key finitions

AltGr glyphs are controlled by the insta dows
Language Preference and are not illustra here.