APL+Win Version 15.0.08 Beta

July 27, 2015

This document contains version history information for this APL+Win v15.0.08 Beta. Please report any problems or comments pertaining to this beta to support@apl2000.com or to the APL2000 Forum discussion at http://forum.apl2000.com/viewtopic.php?f=2&t=1051.

Note: The beta will expire on or about Aug 9, 2015.

APL+Win v15.0.08 Beta

Bug Fixes

1. Fixed bug that caused the :ENDREGION keyword in the version 15.0.06 beta to require an argument or else report the `OUTER SYNTAX ERROR` error message when saving the function.

2. Fixed bug that caused APL+Win to crash during the copy operation of a workspace when APL+Win improperly allocated memory for the workspace in low memory instead of high memory (above the 2 GB address space). This bug is present in APL+Win versions 10.0 to 15.0.

3. Fixed bug that caused APL+Win to hang when executing “`⎕IT 'AuditRefcountsS`” after a workspace copy operation where the resulting workspace occupied both the lower (< 2GB) and upper (> 2GB) workspace memory. This bug is present in APL+Win versions 10.0 to 15.0.

4. Fixed bug that caused the new random number generator in APL+Win v15.0 to generate orphan objects (bad referenced pointer objects) in the workspace. To check for the presence of orphan objects in the workspace, start with making a backup copy of the workspace. Next start APL+Win with the entry below in the APLW.INI file:

   [Config]
   NoOrphanClean=7

   Then execute the following statement:

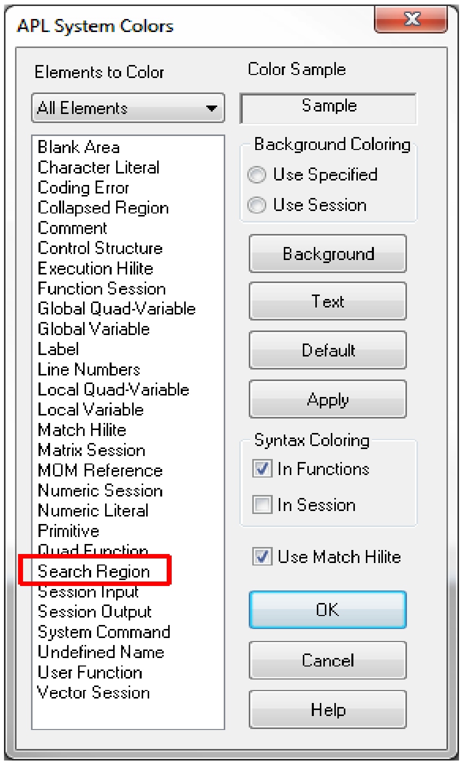
   ```
   ⎕it 'AuditRefcountsS'
   ```

   The result 3x2 array, when not all zeros, means orphan objects are present in the current workspace. To clear the workspace of the orphan objects, execute the following statement until the result array is all zeros prior to resaving the workspace:

   ```
   ⎕it 'AuditRefcountsC'
   ```

Update to APL System Colors Dialog

New “Search Region” option in the “APL System Colors” dialog accessed in the Tools > Colors menu. This option controls syntax color highlighting of the temporary selection (highlight) that is visible during the Find and Replace operations.



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

Syntax Coloring: [✓ In Functions] In Session

[✓ Use Match Hilite]

Fig 1. Updated APL System Colors Dialog

APL+Win v15.0.06 Beta

Bug Fixes

1. Dyadic iota produced an incorrect negative result instead of correctly reporting the `LIMIT ERROR` error message. Example,

```
      ⎕io
1
      (2147483647/1)⍳0
¯2147483648
```

2. An APL+Win failure occurred when the internal symbol table had overflowed instead of correctly reporting the `SYMBOL TABLE FULL` error message.

3. An APL+Win crash occurred when specifying rank>2 empty arrays with an empty trailing dimension for ⎕CM, ⎕CV, ⎕CN, ⎕FX and ⎕DEF. The example below produced an APL+Win crash in versions 15.0 and the version 15.0.04 beta:

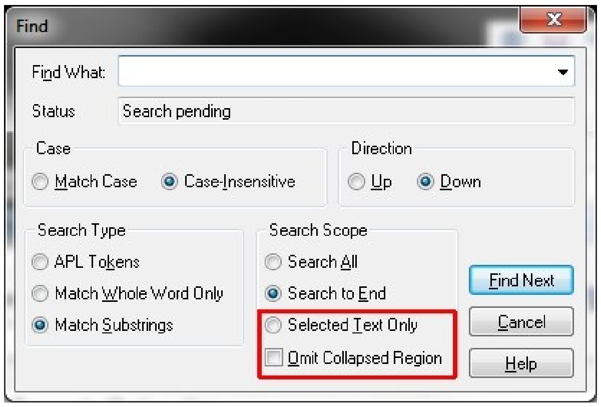
```
      ⎕fx 2 10 0⍴''
```

Update to :ENDREGION keyword

Added support for unexecuted comments at the end of the :ENREGION keyword just as in the :REGION keyword.

Update to Find/Replace Dialogs

New in the ‘Search Scope’ region in the Find and Replace dialogs are the options: 1) Selected Text Only and 2) Omit Collapsed Region. The ‘Selected Text Only’ option restricts the operation to the currently selected text. The ‘Omit Collapsed Region’ option is only available in the function editor and excludes the operation for any collapsed region in the function editor. Note that in the Replace dialog the button labelled ‘Done’ in prior versions is relabelled Cancel.



Find

Find What:

Status: Search pending

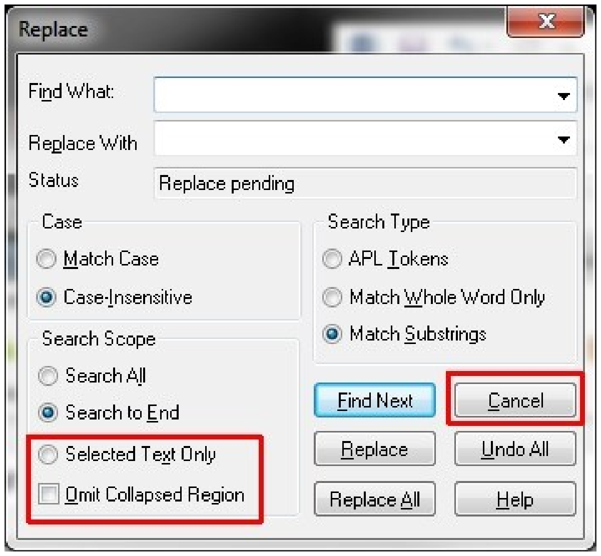
Case: Match Case [✓ Case-Insensitive]

Direction: Up [✓ Down]

Search Type: APL Tokens, Match Whole Word Only [✓ Match Substrings]

Search Scope: Search All [✓ Search to End], Selected Text Only, Omit Collapsed Region

Fig 1. Updated Find Dialog



Replace

Find What:

Replace With

Status: Replace pending

Case: Match Case [✓ Case-Insensitive]

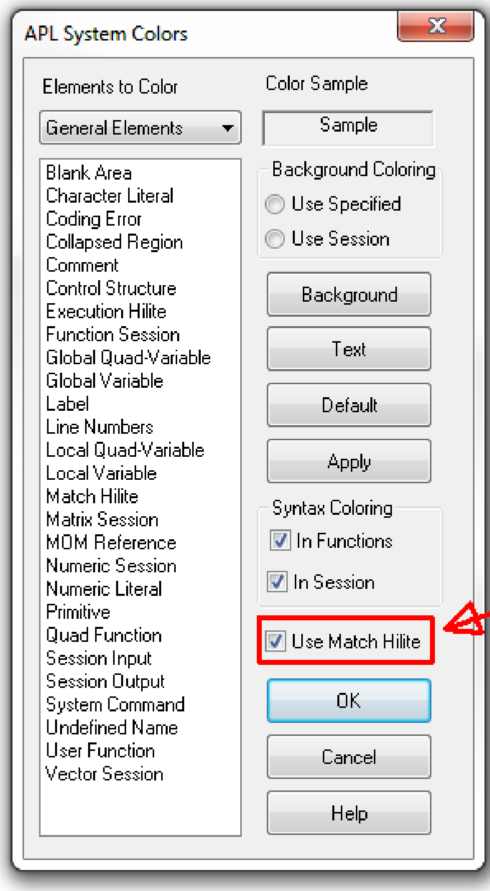
Search Type: APL Tokens, Match Whole Word Only [✓ Match Substrings]

Search Scope: Search All [✓ Search to End], Selected Text Only, Omit Collapsed Region

Fig 2. Updated Replace Dialog

Update to APL System Colors Dialog

New “Use Match Hilite” option in the “APL System Colors” dialog accessed in the Tools > Colors menu. This option controls syntax color highlighting of matching elements (opening and closing) and user defined names introduced in the prior beta release. (Note: The final version (15.1) will also include an option for changing the colors for the temporary selection that is visible during the Find and Replace operations).


APL System Colors

Elements to Color: General Elements

Blank Area
Character Literal
Coding Error
Collapsed Region
Comment
Control Structure
Execution Hilite
Function Session
Global Quad-Variable
Global Variable
Label
Line Numbers
Local Quad-Variable
Local Variable
Match Hilite
Matrix Session
MOM Reference
Numeric Session
Numeric Literal
Primitive
Quad Function
Session Input
Session Output
System Command
Undefined Name
User Function
Vector Session

Color Sample: Sample

Background Coloring
- Use Specified
- Use Session

Syntax Coloring
- [✓] In Functions
- [✓] In Session

[✓] Use Match Hilite

Fig 3. Updated APL System Colors Dialog

APL+Win v15.0.04 Beta

New System Function: ⎕CN - Character Nested-Array

The ⎕CN system function is used to normalize any character argument into a nested vector of character arrays.

Syntax: res ← ⎕CN array

Argument: array is an array of characters arrays or scalars.

Result: result is a nested vector of character arrays.

Example:
```
     DISPLAY ('M' ('F',  ('C1' (3 3⍴'ABCDEFGHI'))),⊂'abc', ⊂'def')
+→-----------------------------------+
|   +→-------------+ +→------------+ |
| M |   +→-+ +→--+ | |       +→--+ | |
| - | F |C1| ↓ABC| | | a b c |def| | |
|   | - +--+ |DEF| | | - - - +---+ | |
|   |        |GHI| | +∊------------+ |
|   |        +---+ |                 |
|   +∊-------------+                 |
+∊-----------------------------------+
     DISPLAY ⎕CN ('M' ('F',  ('C1' (3 3⍴'ABCDEFGHI'))),⊂'abc', ⊂'def')
+→-------------------------------------------+
| +→+ +→+ +→-+ +→--+ +→--+ +→--+ +→--+ +→--+ |
| |M| |F| |C1| |ABC| |DEF| |GHI| |abc| |def| |
| +-+ +-+ +--+ +---+ +---+ +---+ +---+ +---+ |
+∊-------------------------------------------+
```

New support for syntax color highlighting of matching elements

This enhancement adds support for syntax color highlighting of matching pairs (opening and closing) of the punctuation marks including single quotes, double quotes, parenthesis, and index brackets in both the session and the function editor and highlighting of control structure block statements that are at the same nesting level and user defined names (including line labels) in the function editor.

When the caret is adjacent to an opening or closing parenthesis, index bracket, single quote, or double quote, then both the opening and closing matched pair of punctuation marks will be highlighted or similarly distinguished visually. By adjacent we mean that the caret should be immediately to the right or left of the punctuation mark. If there is ambiguity, such as when the caret is between two adjacent punctuation marks, then this highlighting will apply to the punctuation mark on the right of the caret. Highlighting of syntax pairs is not done for elements that occur inside of quoted strings or comments.

If exactly one such punctuation mark is selected, rather than simply having the cursor adjacent to the punctuation mark, then the corresponding, matching opening or closing punctuation mark will be highlighted as described before, but the selected punctuation mark will be highlighted for normal selected display. If multiple characters are selected, or a single character is selected but it is not one of the paired punctuation marks, then the rules for matched pair highlighting are NOT applied to any character and only normal selection display applies.

The behavior described above is intended to be similar but not identical to the behavior of Visual Studio 2013. In particular, in Visual Studio, when the caret is to the immediate left of a parenthesis and the immediate right of a user defined name, both the name and the parenthesis and its matching paired parenthesis are highlighted. APL+Win will only highlight the element (and its associated matching element(s)) in such cases for the element that it to the immediate right of the caret, not both elements to the left and right of the caret. In Visual Studio, paired syntax elements such as parenthesis, get highlighted even when selection extends past the parenthesis, such as into leading or trailing whitespace or some other characters. APL+Win does not follow their pattern here either. Selection must select only a single character in order for its paired element to be highlighted.

Highlighting is also done for the keywords of multi-part control structures. For example, when the caret is adjacent to or within an :IF keyword, all corresponding :ELSEIF, :ELSE, :ENDIF, and/or :END keywords that are at the same nesting level as the :IF statement are highlighted. Nested :IF statements at deeper or higher levels of containment are not highlighted. Similarly, if the caret is adjacent to or within the :ELSEIF, :ELSE, :ENDIF, or :END keyword that are part of an :IF statement, the corresponding other elements of the same control structure nesting level will all be highlighted as if the caret were in the :IF statement as described above. Note that inline control structure keywords are not highlighted and also the :IN keyword associated with a :FOR or :FOREACH statement.

If any part of a highlight-able multi-part control structure keyword such as described above is selected and the selection does not extend before colon at the beginning of the keyword nor past the last character of the keyword, then all matching keywords at the same nesting level will be highlighted as described above.

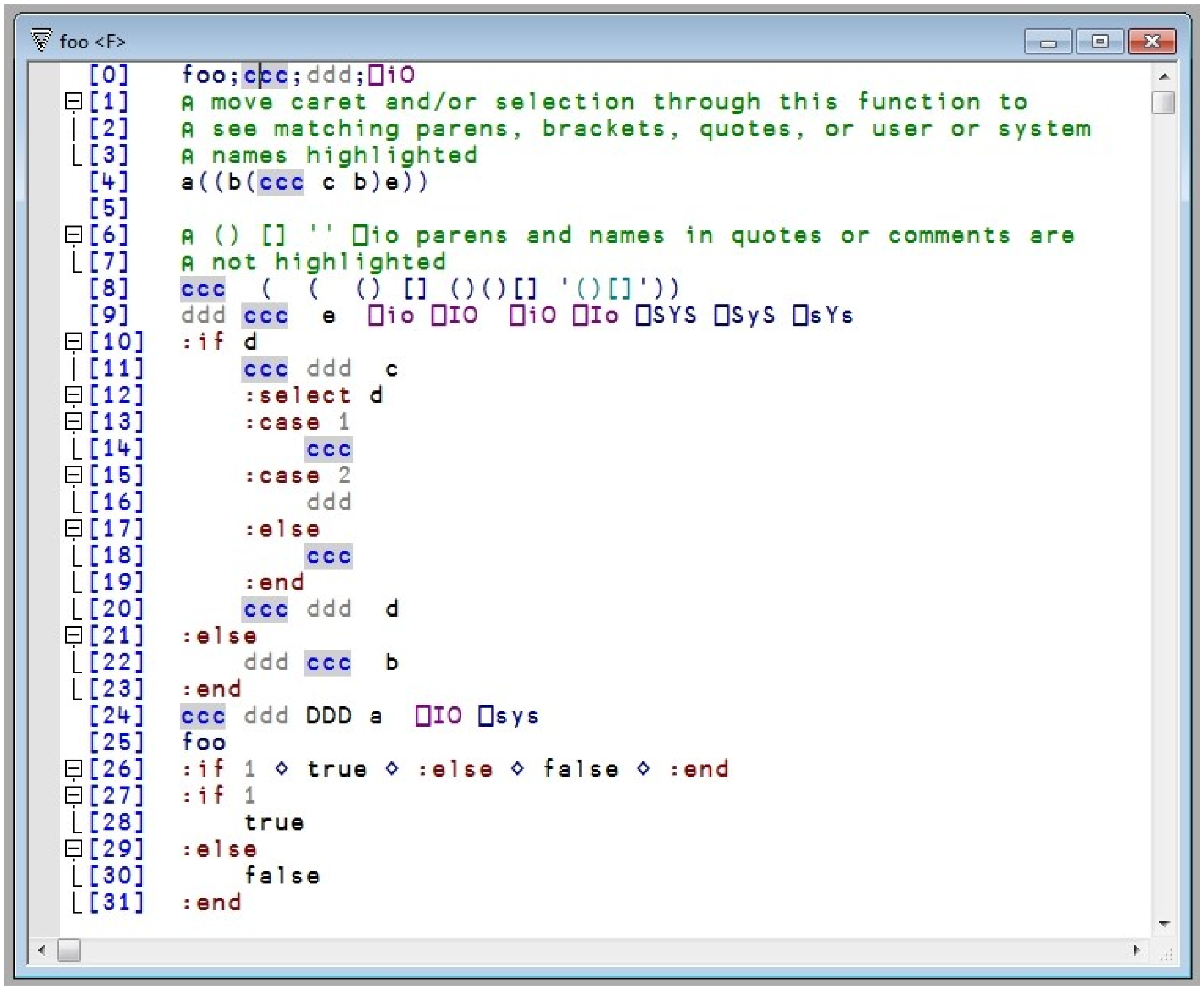
The following multi-part control structures will participate in the highlighting described above. In all cases where structure specific ending keywords such as :ENDIF or :ENDSELECT are listed below, aliases such as :END will be considered surrogates for the fully-qualified ending keyword and will be similarly highlighted. Each line below specifies which keywords are part of each control structure sequence that begin with the first keyword on the line and are followed by some number of the other keywords on the line, ending with the ending keyword or one of its surrogates.

```
:IF :ELSEIF :ELSE :ANDIF :ORIF :ENDIF

:SELECT :CASE :CASELIST :LIKE :ELSE :ENDSELECT

:WHILE :ENDWHILE

:WHILE :UNTIL

:REPEAT :ENDREPEAT

:REPEAT :UNTIL

:FOR :ENDFOR

:FOREACH :ENDFOREACH

:TRY :CATCHIF :CATCH :CATCHALL :FINALLY :ENDTRY

:TRYALL :ENDTRYALL

:TEST  :PASS  :FAIL :ENDTEST

:IFDEBUG :ENDIFDEBUG
```

Finally, user defined names will be highlighted. If the caret is adjacent to or inside any user defined name then all occurrences of that name in the current function edit session will be highlighted.

Similarly if any portion of the name is selected and the selection does not extend before the first letter of the name or last character of the name, then all occurrences of that name will be highlighted.

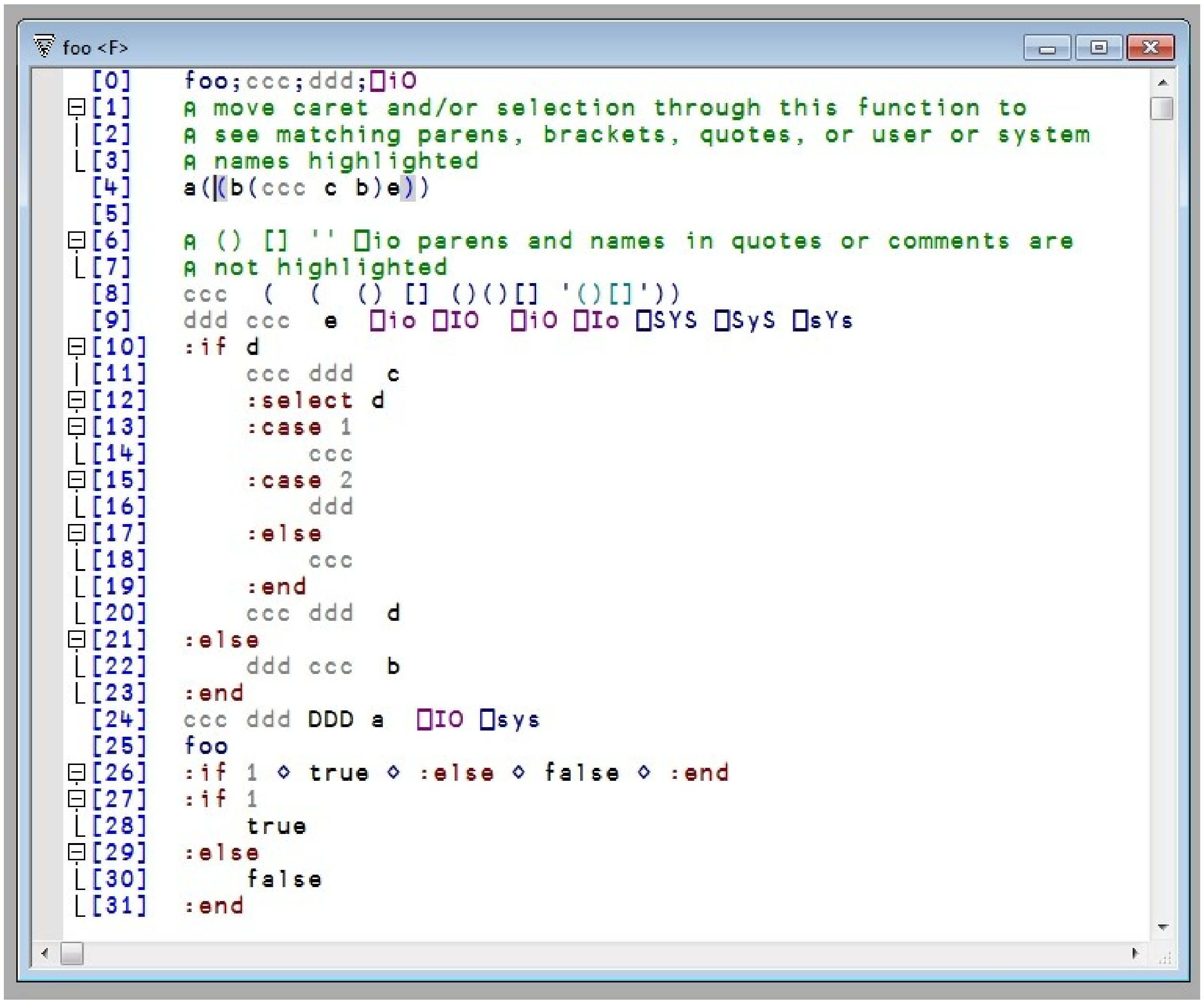
In all cases when the caret is in an ambiguous position adjacent to (between) two syntax elements (such as between a name and a parenthesis or between two parenthesis) the syntax element on the right will be highlighted along with all other elements that are associated with that element.



foo <F>

```
foo;ccc;ddd;⎕iO
⍝ move caret and/or selection through this function to
⍝ see matching parens, brackets, quotes, or user or system
⍝ names highlighted
a((b(ccc c b)e))

⍝ () [] '' ⎕io parens and names in quotes or comments are
⍝ not highlighted
ccc  (  (  () [] ()()[] '()[]'))
ddd ccc  e  ⎕io ⎕IO  ⎕iO ⎕Io ⎕SYS ⎕SyS ⎕sYs
:if d
    ccc ddd  c
    :select d
    :case 1
        ccc
    :case 2
        ddd
    :else
        ccc
    :end
    ccc ddd  d
:else
    ddd ccc  b
:end
ccc ddd DDD a  ⎕IO ⎕sys
foo
:if 1 ⋄ true ⋄ :else ⋄ false ⋄ :end
:if 1
    true
:else
    false
:end
```

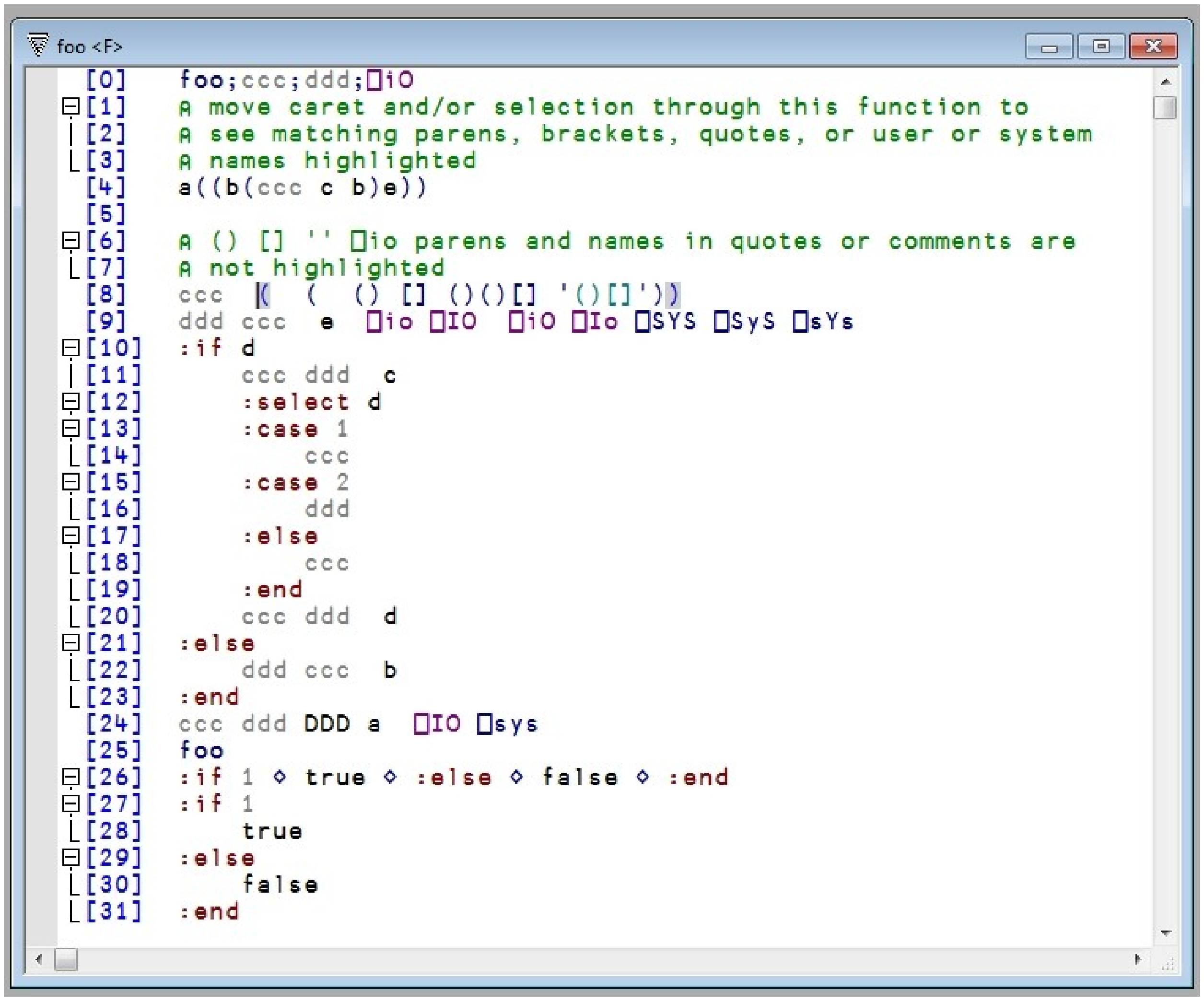
Figure 4: Highlight variable named ccc



foo <F>

```
[0]   foo;ccc;ddd;⎕iO
[1]   ⍝ move caret and/or selection through this function to
[2]   ⍝ see matching parens, brackets, quotes, or user or system
[3]   ⍝ names highlighted
[4]   a((b(ccc c b)e))
[5]
[6]   ⍝ () [] '' ⎕io parens and names in quotes or comments are
[7]   ⍝ not highlighted
[8]   ccc  (  (  () [] ()()[] '()[]'))
[9]   ddd ccc  e  ⎕io ⎕IO  ⎕iO ⎕Io ⎕SYS ⎕SyS ⎕sYs
[10]  :if d
[11]      ccc ddd  c
[12]      :select d
[13]      :case 1
[14]          ccc
[15]      :case 2
[16]          ddd
[17]      :else
[18]          ccc
[19]      :end
[20]      ccc ddd  d
[21]  :else
[22]      ddd ccc  b
[23]  :end
[24]  ccc ddd DDD a  ⎕IO ⎕sys
[25]  foo
[26]  :if 1 ⋄ true ⋄ :else ⋄ false ⋄ :end
[27]  :if 1
[28]      true
[29]  :else
[30]      false
[31]  :end
```

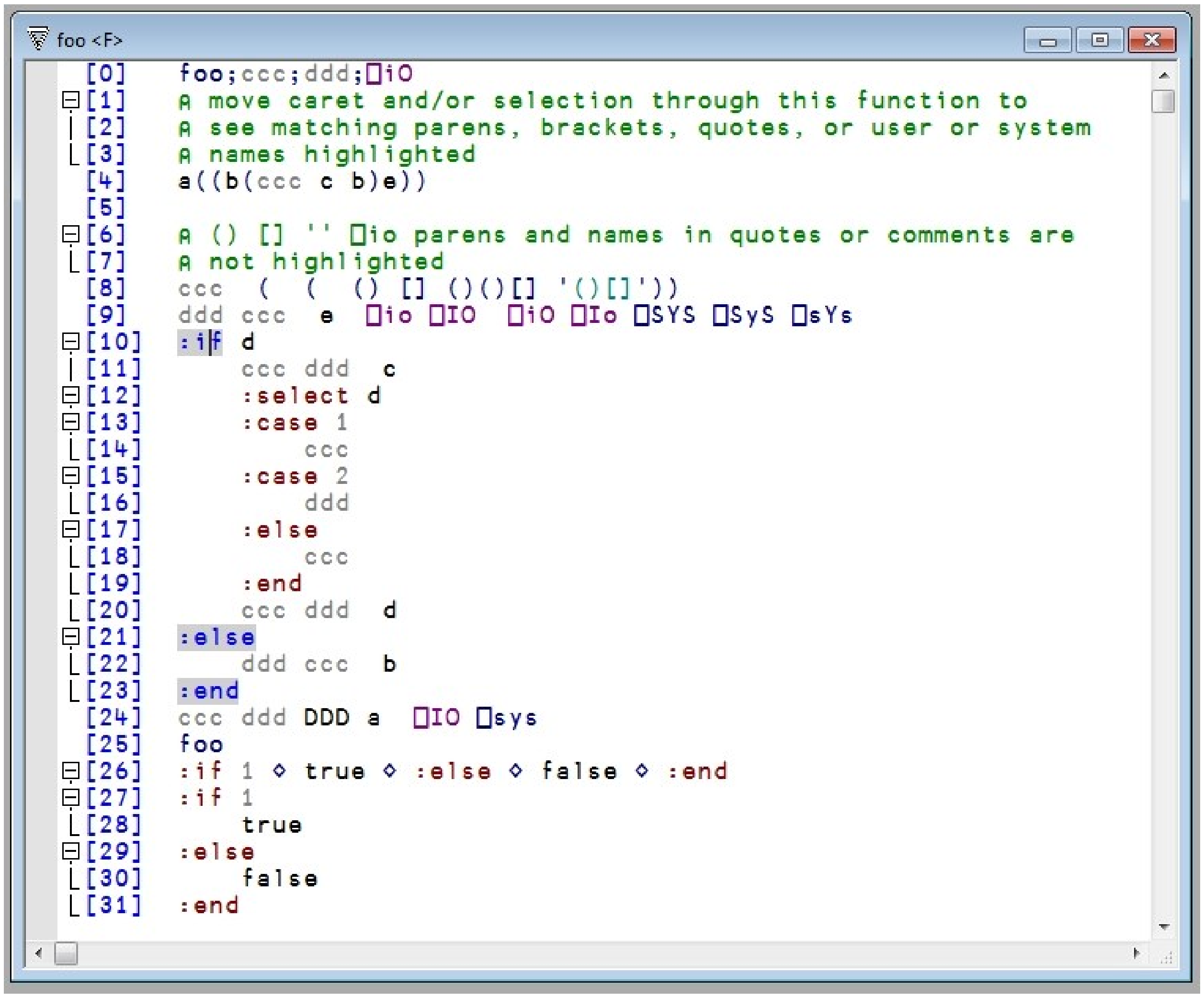
Figure 5: Highlight second pair of parenthesis on line 4



foo <F>

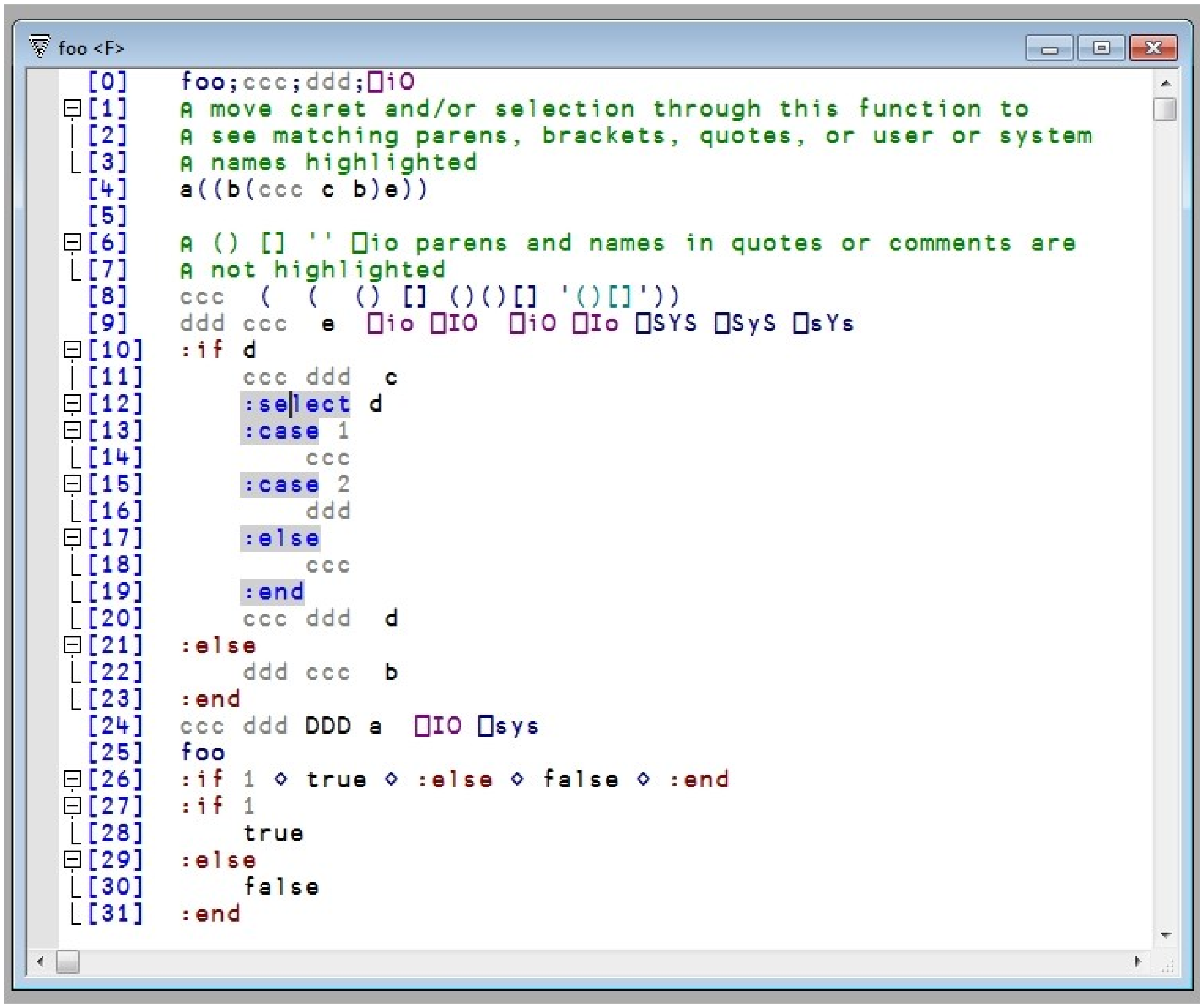
```
[0]     foo;ccc;ddd;⎕iO
[1]     ⍝ move caret and/or selection through this function to
[2]     ⍝ see matching parens, brackets, quotes, or user or system
[3]     ⍝ names highlighted
[4]     a((b(ccc c b)e))
[5]
[6]     ⍝ () [] '' ⎕io parens and names in quotes or comments are
[7]     ⍝ not highlighted
[8]     ccc  (  (  () [] ()()[] '()[]'))
[9]     ddd ccc  e  ⎕io ⎕IO  ⎕iO ⎕Io ⎕SYS ⎕SyS ⎕sYs
[10]    :if d
[11]        ccc ddd  c
[12]        :select d
[13]        :case 1
[14]            ccc
[15]        :case 2
[16]            ddd
[17]        :else
[18]            ccc
[19]        :end
[20]        ccc ddd  d
[21]    :else
[22]        ddd ccc  b
[23]    :end
[24]    ccc ddd DDD a  ⎕IO ⎕sys
[25]    foo
[26]    :if 1 ⋄ true ⋄ :else ⋄ false ⋄ :end
[27]    :if 1
[28]        true
[29]    :else
[30]        false
[31]    :end
```

Figure 6: Highlight first pair of parenthesis on line 8


foo <F>

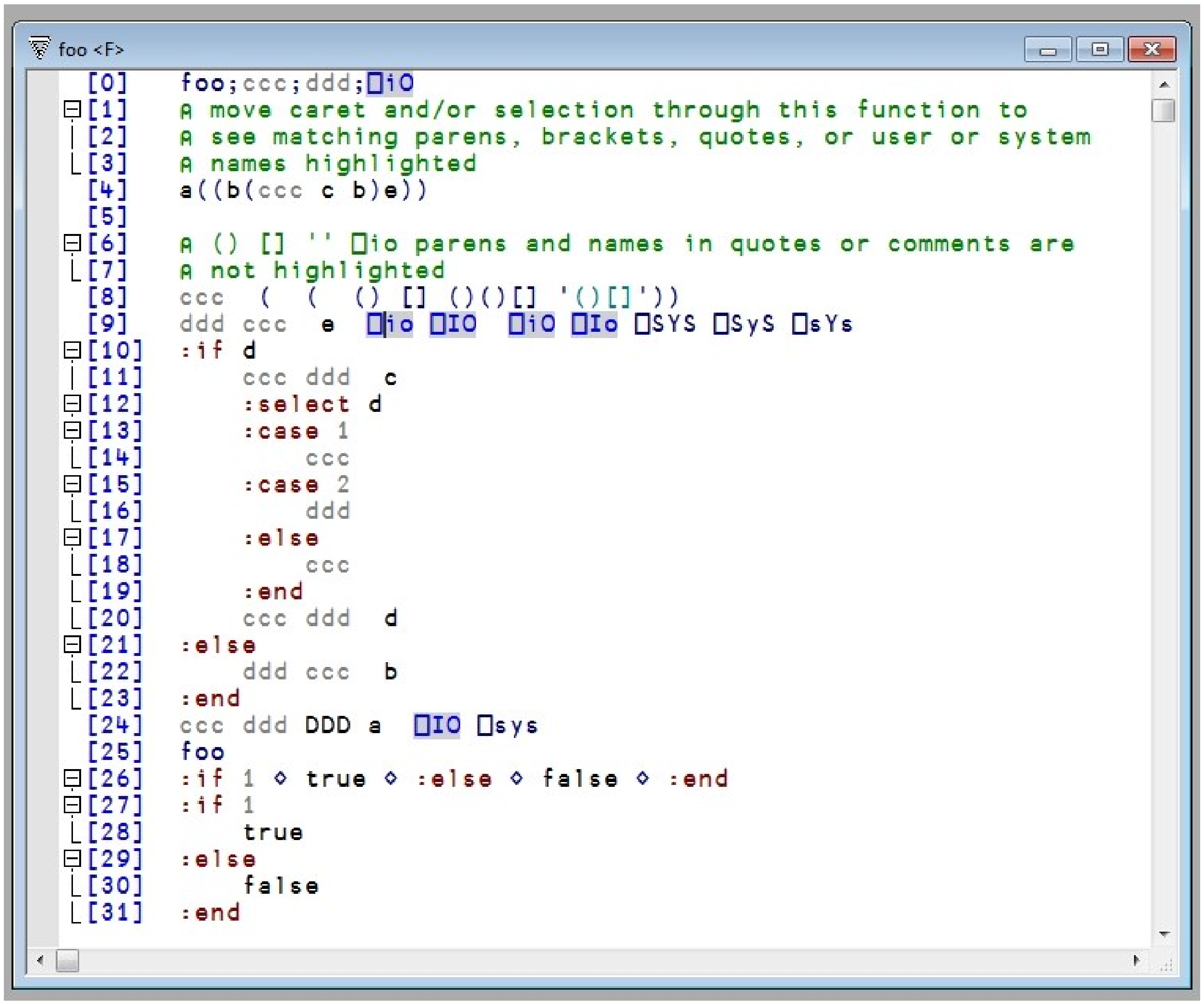
```
 [0]   foo;ccc;ddd;⎕iO
⊟[1]   ⍝ move caret and/or selection through this function to
|[2]   ⍝ see matching parens, brackets, quotes, or user or system
└[3]   ⍝ names highlighted
 [4]   a((b(ccc c b)e))
 [5]
⊟[6]   ⍝ () [] '' ⎕io parens and names in quotes or comments are
└[7]   ⍝ not highlighted
 [8]   ccc  (  (  () [] ()()[] '()[]'))
 [9]   ddd ccc  e  ⎕io ⎕IO  ⎕iO ⎕Io ⎕SYS ⎕SyS ⎕sYs
⊟[10]  :if d
|[11]      ccc ddd  c
⊟[12]      :select d
⊟[13]      :case 1
└[14]          ccc
⊟[15]      :case 2
└[16]          ddd
⊟[17]      :else
└[18]          ccc
└[19]      :end
└[20]      ccc ddd  d
⊟[21]  :else
└[22]      ddd ccc  b
└[23]  :end
 [24]  ccc ddd DDD a  ⎕IO ⎕sys
 [25]  foo
⊟[26]  :if 1 ⋄ true ⋄ :else ⋄ false ⋄ :end
⊟[27]  :if 1
└[28]      true
⊟[29]  :else
└[30]      false
└[31]  :end
```

Figure 7: Highlight the :IF/:END control structure statements on lines 10 - 23


foo <F>

```
[0]   foo;ccc;ddd;⎕iO
[1]   ⍝ move caret and/or selection through this function to
[2]   ⍝ see matching parens, brackets, quotes, or user or system
[3]   ⍝ names highlighted
[4]   a((b(ccc c b)e))
[5]
[6]   ⍝ () [] '' ⎕io parens and names in quotes or comments are
[7]   ⍝ not highlighted
[8]   ccc  (  (  () [] ()()[] '()[]'))
[9]   ddd ccc  e  ⎕io ⎕IO  ⎕iO ⎕Io ⎕SYS ⎕SyS ⎕sYs
[10]  :if d
[11]      ccc ddd  c
[12]      :select d
[13]      :case 1
[14]          ccc
[15]      :case 2
[16]          ddd
[17]      :else
[18]          ccc
[19]      :end
[20]      ccc ddd  d
[21]  :else
[22]      ddd ccc  b
[23]  :end
[24]  ccc ddd DDD a  ⎕IO ⎕sys
[25]  foo
[26]  :if 1 ⋄ true ⋄ :else ⋄ false ⋄ :end
[27]  :if 1
[28]      true
[29]  :else
[30]      false
[31]  :end
```

Figure 8: Highlight the :SELECT/:END control structure statements on lines 12 - 19

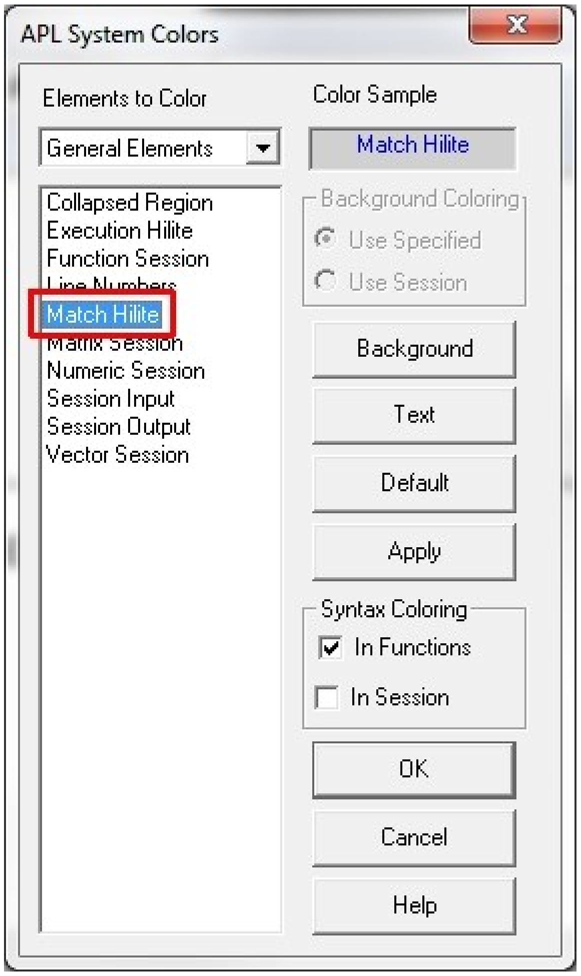


```
foo;ccc;ddd;⎕IO
⍝ move caret and/or selection through this function to
⍝ see matching parens, brackets, quotes, or user or system
⍝ names highlighted
a((b(ccc c b)e))

⍝ () [] '' ⎕io parens and names in quotes or comments are
⍝ not highlighted
ccc  (  (  () [] ()()[] '()[]'))
ddd ccc  e  ⎕io ⎕IO  ⎕iO ⎕Io ⎕SYS ⎕SyS ⎕sYs
:if d
    ccc ddd  c
    :select d
    :case 1
        ccc
    :case 2
        ddd
    :else
        ccc
    :end
    ccc ddd  d
:else
    ddd ccc  b
:end
ccc ddd DDD a  ⎕IO ⎕sys
foo
:if 1 ⋄ true ⋄ :else ⋄ false ⋄ :end
:if 1
    true
:else
    false
:end
```

Figure 9: Highlight ⎕IO system variable

Updated APL System Colors Dialog

Match Hilite - the (name of the color to control) syntax coloring choice for how syntax pairing is highlighted. The default is blue on grey.

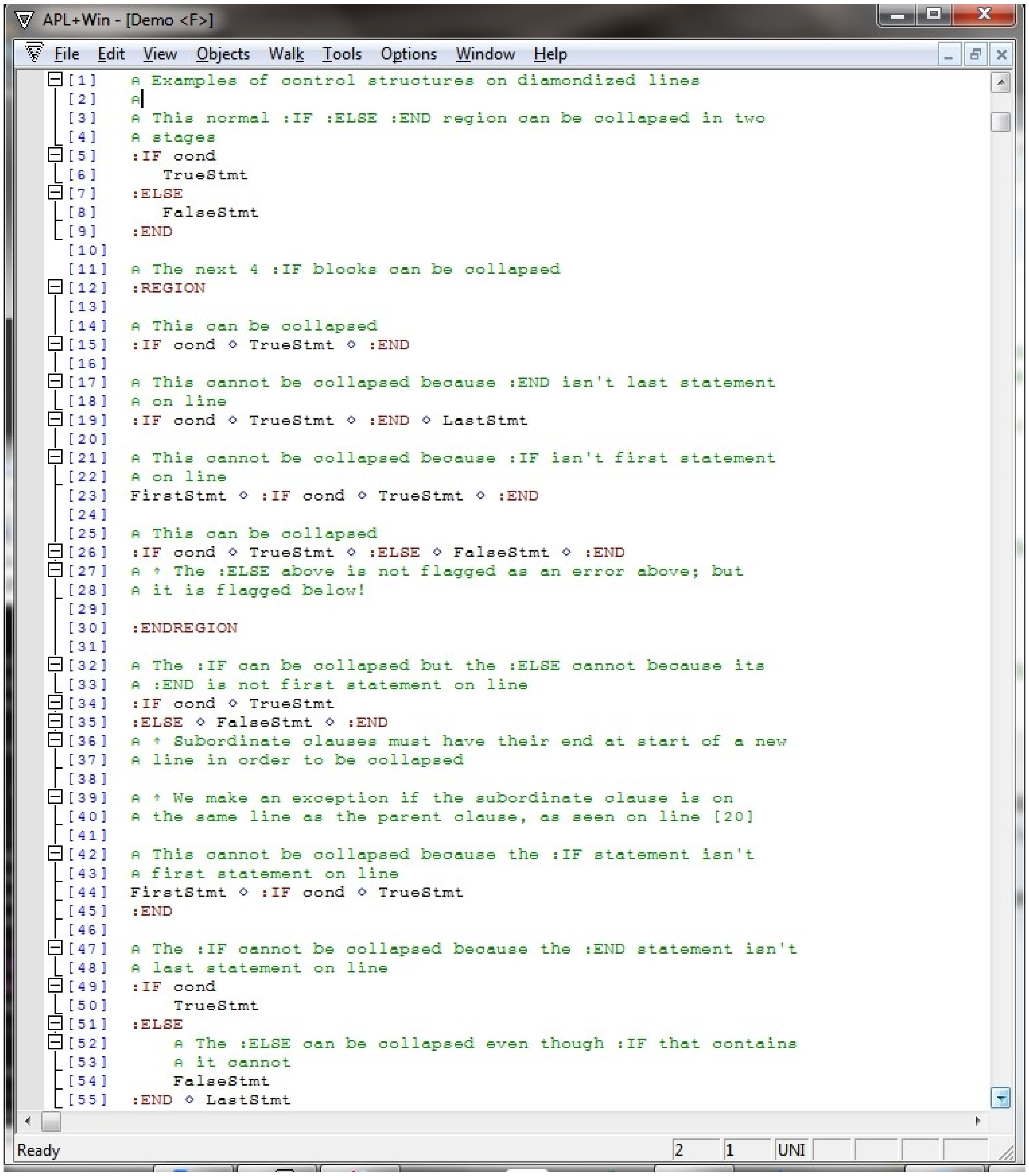


Background Coloring: [✓ Use Specified] Use Session

Syntax Coloring: [✓ In Functions] In Session

Figure 10: Match Hilite color settings

New support for collapsing & expanding of comment and control structure elements in the function editor

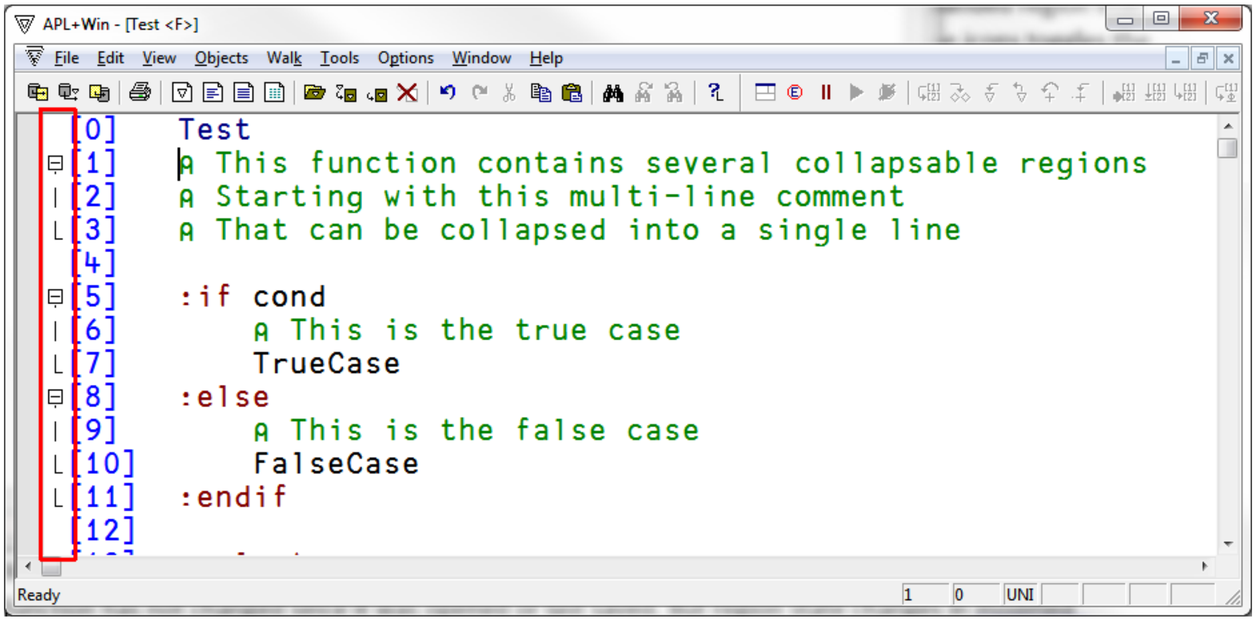


```
[1]   ⍝ Examples of control structures on diamondized lines
[2]   ⍝
[3]   ⍝ This normal :IF :ELSE :END region can be collapsed in two
[4]   ⍝ stages
[5]   :IF cond
[6]      TrueStmt
[7]   :ELSE
[8]      FalseStmt
[9]   :END
[10]
[11]  ⍝ The next 4 :IF blocks can be collapsed
[12]  :REGION
[13]
[14]  ⍝ This can be collapsed
[15]  :IF cond ⋄ TrueStmt ⋄ :END
[16]
[17]  ⍝ This cannot be collapsed because :END isn't last statement
[18]  ⍝ on line
[19]  :IF cond ⋄ TrueStmt ⋄ :END ⋄ LastStmt
[20]
[21]  ⍝ This cannot be collapsed because :IF isn't first statement
[22]  ⍝ on line
[23]  FirstStmt ⋄ :IF cond ⋄ TrueStmt ⋄ :END
[24]
[25]  ⍝ This can be collapsed
[26]  :IF cond ⋄ TrueStmt ⋄ :ELSE ⋄ FalseStmt ⋄ :END
[27]  ⍝ ↑ The :ELSE above is not flagged as an error above; but
[28]  ⍝ it is flagged below!
[29]
[30]  :ENDREGION
[31]
[32]  ⍝ The :IF can be collapsed but the :ELSE cannot because its
[33]  ⍝ :END is not first statement on line
[34]  :IF cond ⋄ TrueStmt
[35]  :ELSE ⋄ FalseStmt ⋄ :END
[36]  ⍝ ↑ Subordinate clauses must have their end at start of a new
[37]  ⍝ line in order to be collapsed
[38]
[39]  ⍝ ↑ We make an exception if the subordinate clause is on
[40]  ⍝ the same line as the parent clause, as seen on line [20]
[41]
[42]  ⍝ This cannot be collapsed because the :IF statement isn't
[43]  ⍝ first statement on line
[44]  FirstStmt ⋄ :IF cond ⋄ TrueStmt
[45]  :END
[46]
[47]  ⍝ The :IF cannot be collapsed because the :END statement isn't
[48]  ⍝ last statement on line
[49]  :IF cond
[50]      TrueStmt
[51]  :ELSE
[52]      ⍝ The :ELSE can be collapsed even though :IF that contains
[53]      ⍝ it cannot
[54]      FalseStmt
[55]  :END ⋄ LastStmt
```

Figure 11: Example of expanded code

The :REGION and :ENDREGION keywords defines the bounds of collapsible regions. The :REGION keyword can have an argument which is free text that is not executed. It is simply used as a label for the region both when the region is expanded and collapsed.

A new column of "region icons" has been added between the Function Line Markers ("gutter") column (in the Preference dialog) where breakpoints can be set, and the function line number column. This appears in the red rectangle in the figure below. The icons in the region column display a [-] for expanded region that can be collapsed or a [+] for collapsed regions that can be expanded. Clicking on these icons toggles the expanded/collapsed state of the region.



```
[0]    Test
[1]    ⍝ This function contains several collapsable regions
[2]    ⍝ Starting with this multi-line comment
[3]    ⍝ That can be collapsed into a single line
[4]
[5]    :if cond
[6]        ⍝ This is the true case
[7]        TrueCase
[8]    :else
[9]        ⍝ This is the false case
[10]       FalseCase
[11]   :endif
[12]
```

Fig. 12: New "gutter" region with expanded/collapsed icons

Allow most control block structures such as :IF :ELSEIF :ELSE :ENDIF, :SELECT :CASE :CASE :ELSE :ENDSELECT, etc. to be collapsible.

Some complex structures such as :IF :ELSEIF ... :ELSE :ENDIF have multiple segments and the topmost statement (the :IF statement) can toggle partial collapse of just the top section of the block (up to the first :ELSEIF or :ELSE) or of the whole block all the way to the :ENDIF. This requires a third icon state which looks like [=] indicating partially collapsed state.

In addition to the icon to the left of the numbers column, collapsed regions also get painted with an ellipsis "..." enclosed in a box drawn at the end of the first line of the collapsed region.

Allow comment blocks containing more than one comment line to be collapsed.

Modify function definition in workspace so that collapsed state can be saved between edit sessions. Region state changes are automatically stored into the workspace without saving the function, if the function has not changed since it was opened or last saved. But region state changes in modified functions are not saved into the workspace until (and unless) the modifications are saved. Region state changes are NOT traced in the undo/redo buffers.

Updated APL System Colors Dialog

Collapsed Region – the syntax coloring choices for how collapsed regions and their ellipsis block is displayed. These are currently colored the same as local variable names.

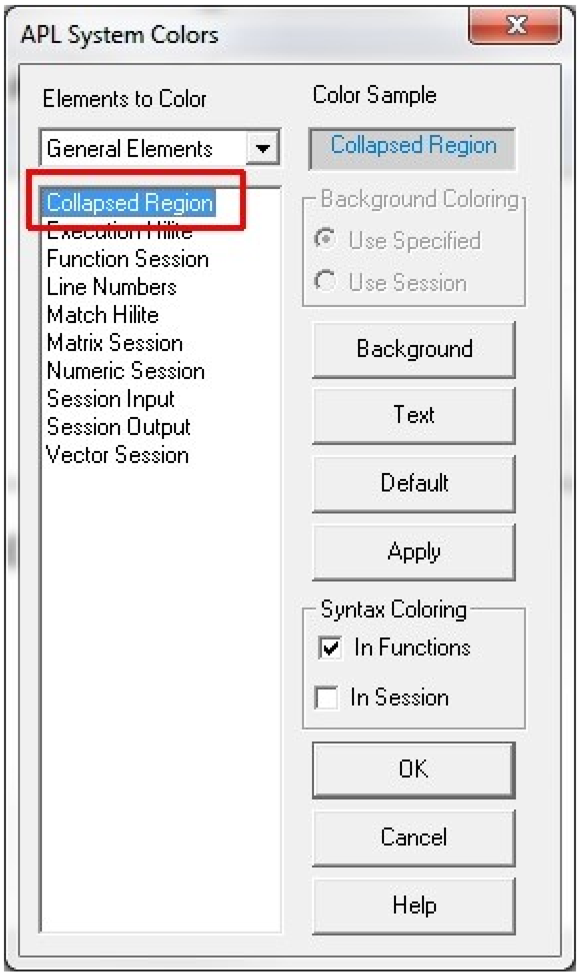


Figure 13: Collapsed Region color settings

⎕DEF system function supports nested vector of character vectors or scalars

The ⎕DEF system function has been improved to support an argument that is a nested vector of character vectors or scalars. This is identical to the enhancement to ⎕FX in v15.0. E.g.,

```
      ⎕DEF 'res←left MyFn right' 'res←left + right' 'res←res,left * right'
MyFn
```

is equivalent to the line below in prior versions of APL+Win

```
      ⎕DEF ⊃[2] 'res←left Foo right' 'res←left + right' 'res←res,left * right'
```

New support for Hexadecimal and floating point numbers

Add support for directly entering hexadecimal integer and floating point numbers in APL+Win.

A hexadecimal integer is prefixed by 0x or 0X followed by 1-16 hex digits (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). For example, 0x00100000 is equal to 1048576. Hex numbers with 1-8 digits are ALWAYS integers in APL. They do not overflow into floating point numbers when the high order bit is set. For example, the hex number 0xFFFFFFFF is -1 rather than 4294967295. A hexadecimal float can be prefixed by 0x or 0X followed by 9-16 hex digits or by a 0r or 0R prefix followed by 1-16 hex digits.

Hex integers can also be specified in binary notation with a 0b or 0B prefix followed by 1-64 binary digits (0 or 1). For example, 0b1111 is 15. In the case of a 0r or 0R prefix, the following 1-16 hex digits are the hex representation of the bit pattern for an IEEE 64-bit float. For example, 0r3FF0000000000000 = 1.0. If less than 16 digits are specified, the value is padded with trailing zeros to 16 hex digits. Thus 0r3FF is the same as 0r3FF0000000000000.

A relatively easy to read summary of IEEE-754 floating point specification can be found here: http://en.wikipedia.org/wiki/IEEE_754-1985. The formal IEEE-specification can be found here: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4610935&filter%3DAND%28p_Publication_Number%3A4610933%29

Here are some other examples:

0x00	=	0
0x01	=	1
0xFF	=	255
0xFFFFFFFF	=	-1
0x0FFFFFFFF	=	4294967295
0b1100	=	12
0b11111111	=	255
0b10101010	=	170

0r3FF0000000000000	=	1.0	
0r4000000000000000	=	2.0	
0rC000000000000000	=	-2.0	
0r4010000000000000	=	4.0	
0r3FE0000000000000	=	0.5	
0r4	=	2.0	Note that 0r4 = 0r4000000000000000
0rC	=	-2.0	Note that 0rC = 0rC000000000000000

NOTE: In order for hex numbers to be accepted in any of these contexts they must be enabled via the [Experimental]EnableHex property. Possible values are 0=disabled; 1=allow 0X and 0B; 2=allow 0R; 3=allow 0X, 0B, or 0R notation. The default is 1. The intention is to eventually eliminate the EnableHex property and have hex value always enabled. It is only used because this is an experimental feature at this time.

Be careful to avoid specifying bit patterns using "R" notation that represent infinities or NaNs. APL+Win does not expect to encounter these and can exhibit unexpected behaviors. For example, you can hang APL+Win by trying to display the value of 0r7FF (positive infinity). This is the reason that 0R notation is not enabled by default. So be especially careful to avoid generating them.

Example:

```
      (↑645 ⎕dr 82 ⎕dr 110619 ¯2147483648)
```

Could be more directly entered in hex like this (noting the 0x prefix):

```
      (↑645 ⎕dr 0x0001B01B 0x80000000)
```

Or even more directly as a floating point 64-bit hex like this (noting the 0r prefix):

```
      0r800000000001B01B

      ⎕VI '0xFF'
1
      ⎕FI '0xFF'
255
      ⎕DR '0xA'
82
      ⎕DR 0xA
323
```