APL+Win Version 17.0.01 Beta

Apr 26, 2017

This document contains version history information for this APL+Win 17.0.01 Beta. Please report any problems or comments pertaining to this beta to support@apl2000.com.

Note: This beta is programmed to expire on or about May 7, 2017.

Enhancements and Changes

The new style property value 65536 has been added to the Button and UButton objects to allow their caption to wrap when the width of the caption is wider than the width of these objects or at the `⎕tcnl` and `⎕tclf` characters embedded in the caption string all while the button maintains the newer visual style (when APL is started with a manifest file).

Bug Fixes

- Addressed bug in version 12.0 to 16.2 that resulted in wrong values being displayed in some cases due to rounding errors in the system. For example, notice how the first value for `N` is formatted in the result with trailing `...9990` rather than `...9998` in the example below:

```
      N←9.999999999999998
      ⎕PP←16
      (N) (1 2 3)
9.999999999999990  1 2 3

      2 2⍴(N) (1 2 3) 9.99 (3 2 1)
9.999999999999990  1 2 3
9.990000000000000  3 2 1
```

  Here is the corrected behaviour:

```
      N←9.999999999999998
      ⎕PP←16
      (N) (1 2 3)
9.999999999999998  1 2 3
      2 2⍴(N) (1 2 3) 9.99 (3 2 1)
9.999999999999998  1 2 3
9.990000000000000  3 2 1
```

  Here's another formatting issue:

```
'K17G<ZZZZZZZZZZZZZZZZZ9.ZZZZZZZZZZZZZZZZZ>' ⎕FMT 100987
       100986.9999999999
```

which is corrected by this fix:

```
'K17G<ZZZZZZZZZZZZZZZZZ9.ZZZZZZZZZZZZZZZZZ>' ⎕FMT 100987
       100987
```

- Addressed bug that caused the icons in the APL session's toolbar to partially display when Windows is displayed in a resolution greater than 1920x1080 and when doing any of the following:
  1. After resuming Windows after being put in Sleep or Hibernate mode
  2. When maximizing the APL session from a non-maximized size
  3. After changing the Windows dpi scaling value
  4. After undocking and then docking the APL session's toolbar
- Addressed bug that prevented APL from successfully starting in Windows 10 when APL was started without a manifest (file) or with a manifest that explicitly specified dpiAware=false and for certain (not all) custom (dpi) scaling settings. This type of failure would follow with the display of the following error message:

  Application manifest specifies invalid dpiAware setting (must be true or false; Per-Monitor not supported)" error message.

Previous Beta (Released Apr 12, 2017)

Enhancements and Changes

- New Support for DPI-awareness in APL+Win

  APL enables system-wide DPI scaling in the session manager window and the ⎕wi (Windows Interface) system objects (excluding any ActiveX controls and objects) when started with a manifest file (e.g., aplw.exe.manifest) in the same folder specifying the entry colored in red below:

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <asmv3:application xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
        <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
          <dpiAware>true</dpiAware>
        </asmv3:windowsSettings>
        </asmv3:application>
  ...

</assembly>
```

When true is replaced with false, APL allows the operating system to do the image stretching for Windows that are drawn as if running at 100% DPI. This is the same behavior as when starting APL without a manifest file present in the same folder. Note: The DPI scaling percentage of the system is the represented by the slider setting for the "*Change the size of text, apps, and other items*" selection for the primary monitor in the Windows Display settings window.

When APL is running in dpiAware=true mode, most things "just work" automatically, but there are a few things that might not work as expected. In all of the examples below, the system-wide DPI scaling is set to 200%. What this means is that everything drawn by APL will be twice as tall and twice as wide (in pixel units) as it was when running at 100% DPI scaling. But all of the scale settings used on ⎕wi objects automatically scale to this larger size. So, if you normally work in character units (scalemode = 1), you will find that 10 units of height will now correspond to twice as many pixels of height as before. But since you normally run 200% DPI scaling on a monitor with twice as many pixels as at 100% scaling, the objects you create will look about the same size as they did before, when running in a DPI unaware application. Traditionally one character unit was 16 pixels high by 8 pixels wide. So a form that was sized at 10 20 will be 10 20 x 16 8 = 160 160 pixels in size at 100% DPI. But at 200% DPI this form that is sized at 10 20 character units will be 320 320 pixels in size. If you are used to working in points (scalemode = 3) units then you are used to each point being 1.3333 pixels on the screen, but at 200% DPI each point will be 2.6667 pixels on the screen. If you used to work in pixels (scalemode = 5) these units are also affected by the DPI scaling. So if you create a ⎕wi object that is sized at 200 by 400 pixels high and wide, then when running at 200% DPI this will actually be drawn at 400 x 800 pixels in size. All of the traditional scaling units scale by the DPI percentage you are running at. If you reference the '#' ⎕wi 'units' property, this will show you exactly how large each unit is in terms of pixels. So for example, while in previous versions of APL you always saw the following:

```
      '#' ⎕wi 'units'
 16             8
  2             2
  1.333333333   1.333333333
  0.06666666667 0.06666666667
  1             1
```

When running this version at 200% DPI you'll find you now get the following reported for the units of the system:

```
       '#' ⎕wi 'units'
 32            16
```

```
4              4
2.666666667    2.666666667
0.1333333333   0.1333333333
2              2
1              1
```

Note carefully that in past versions, the units property returned 5 rows of data corresponding to character units, dialog units, points, twips, and pixels. But in this version units returns 6 rows corresponding to character units, dialog units, points, twips, virtual pixels, and true pixels. The last row corresponds to the new scalemode = 6 mode setting that allows you to work in real pixels rather than virtual scaled pixels that you get from traditional scalemode = 5. You can override the virtualization of scalemode = 5 so that it also refers to real pixels (the same as scalemode = 6) by setting VirtualPixels=0 in the [Config] section of the INI file. However, you are advised to use caution in doing so since this will cause visual elements of your application whose layout is defined using pixel units (scalemode = 5) to not scale proportionally with other visual elements specified in other scale units.

Because all traditional scalemode units (1-5) are virtualized by the current DPI scale setting, this means that all ⎕wi layout calculations work exactly as they did before, relative to each other. But their correspondence to actual pixels on the screen has changed. But since fonts have scaled in exactly the same way as character units, for most applications, most layout related things work exactly as they did in the past. But there are some places where applications can experience issues. One example is if they specify a font using the LOGFONT notation (14 element argument to the font property). In that case, the LOGFONT is specified in terms of pixels, and these pixel units are not scaled. They correspond to the Windows LOGFONT data structure which expects all size measurements to be in terms of pixels. Similarly, if you use low level Windows APIs with ⎕wcall using window handles, any windows you access are in terms of actual pixels, not virtualized (scalemode = 5) pixels. So you are on your own in terms of scaling these values to the current DPI setting. Similarly, some ActiveX objects created may expect you to specify column widths, row heights, font heights in pixel units. And if they do, those are real pixels, not virtual pixels. The APL.Grid object is such an ActiveX object. It does not automatically scale the pixel units you specify for cells or font sizes. So you must compensate manually by scaling pixel setting by `('#' ⎕wi 'dpi')[2]÷100`. Even though the APL.Grid ActiveX object is distributed with APL, because we have no control over other ActiveX objects with regard to how they behave with respect to DPI scaling, the decision was made to not support automatic scaling of any ActiveX objects.

A final DPI issue involves scaling of images such as bitmaps, icons, and cursors used in the application. As with ⎕wi object layout units and font scaling, this too happens automatically, with rare exceptions that are noted below. Icons and cursor are the easiest to describe. Windows automatically scales them to an appropriate size as a function of system-wide DPI scaling. So they are drawn at the default icon and cursor size that Windows chooses for these UI (User Interface) elements as a function of DPI setting. However, bitmaps are more complex. In general, any bitmaps that you reference by name or DLL image resource notation are automatically scaled appropriately. If a bitmap used in a toolbar imagelist is designed as 10 pixels high and 20 pixels wide per image, and has 5 images, its total dimension will be 10 pixels high by 100 pixels wide, in the bitmap file or DLL image resource. But when this bitmap is used to in an APL application that is running at 200% DPI, it will automatically be scaled such that it is 20 pixels high by 200 pixels wide, and the width of each image will be 40 rather than 20 pixels wide. You would normally use such a bitmap in an Imagelist object and specify its size with the imagesize property, to tell the Imagelist how wide each image is. The imagesize returns pixel size of the image as corresponding to the bitmap as it existed in the bitmap file or DLL image resource. So while imagesize continues to be 10 20 for this particular example, internally the image has been stretched to 20 40 in size. If you think of the pixel units used in the imagesize property as being in scale 5 units, you'll be correct in thinking of them as being virtualized internally to the correct number of DPI scaled real pixels required to render the image to appear as it always has in relation to other element sizes (that have correspondingly been scaled by the DPI setting by exactly the same amount).

In some rare cases you might work with bitmap handles that you obtain from other means, perhaps from some ActiveX object that creates and draws into a bitmap for you. In this case, you will often be working with a bitmap that was scaled as it should be displayed at 100% DPI. That's the default assumption APL makes for working with such bitmap handles. If you pass a bitmap handle to the AddImages method of an Imagelist object, APL will assume the bitmap was drawn for 100% DPI viewing and scale it up automatically to the actual DPI scale that the application is running at. The same thing happens if you use the Draw method with the Bitmap operation that uses a bitmap handle as an argument (rather than a bitmap file name or DLL resource specification). In that case, the bitmap will be scaled up to the actual DPI scale and draw onto the object you specified.

There may be times when automatic re-scaling of bitmap images is not what you want to happen. You can control this image scaling behavior by using the imagescale property. All Window based ⎕wi objects have this property (such as Form, Picture, Label, etc.) as well as the Imagelist object. The imagescale property specifies the assumed scaling of the image source file, DLL image resource, or bitmap handle that you are using to draw or load images from. The default is 100 which means that APL will assume that images are scaled for 100% DPI rendering. In this case, when running at a different DPI scale than 100%, the system will automatically re-scale the image for the DPI scale you are running at.

You can disable automatic image scaling by specifying imagescale as 0. In that case, APL will not re-scale any bitmaps it draw or loads for that particular object that specifies imagescale = 0. Use this setting to suppress automatic image scaling for a particular object. If you have an image that it actually drawn for DPI viewing scale other than 100%, you can specify that percentage, and then if you are running APL at a different scale, it will be automatically scaled up or down as needed to the DPI level you are running at.

There is an important exception for the Picture object's image property. When you reference this property it returns a matrix of color codes corresponding to each actual pixel in the picture. If you are working in scalemode = 6 (true pixels) the shape of the image property and the value of the imagesize property will be the same. But for other scale units, the imagesize property returns the scaled units corresponding to the actual pixel size of the image drawn on the Picture object. In the past this was true for scalemode = 5 (virtual pixels) and imagesize. But now that scale 5 units are virtualized, the shape of the image property's return value will not match the imagesize property value unless you are running at 100% DPI. The only time the scale size matches the shape of the value returned by referencing the image property is when you are running with scale 6 units. The imagescale setting does not have any effect on the value returned by the image property because the image property is always related to the actual image pixels on the screen.

But there is a tricky issue that can arise if you use the image property to save images into a numeric matrix. Let's say you draw the image during one APL session and save it (into an integer matrix) and then in another session you set the image property on the Picture to restore the previously drawn image. This all works perfectly unless the DPI scaling in effect when you saved the image is different than when you restore the image. Let's say you were running at 200% DPI and created a Picture object that was 10 x 20 character units high and wide. This could correspond to an image value of shape 320 320 (because that's the number of pixels that correspond to 10 x 20 characters at 200% DPI). So let's say you save that image matrix into a file. Later you start APL running at 100% DPI, perhaps on a different computer or simply because you changed the system DPI scaling. When your application now runs your Picture object will still (most likely) be 10 x 20 character units in size (assuming you are work in character units) but because of the change in DPI scaling, it will now only be 160 x 160 pixels in size. If you set the Picture object's image property with the shape 320 320 matrix you saved from the previous session, you will only see the top-left quadrant of the image, and the rest of the image will be off screen. If you have enabled scrolling on the Picture, you'll be able to view the other parts of the image. Or if you have set the style 4 for the Picture, the image will be stretched to fill the available display area, and you'll be able to see it completely.

So, as described above the imagescale doesn't have any effect on the image property insofar as referencing the property or when setting it to a numeric matrix value corresponding to image color values. However, you can also specify the image property as a bitmap resource name. In that case the image will be scaled normally by the imagescale property, as described earlier. However, when you then reference the image property, it will return a matrix value containing the image color matrix pixel values. So in this case, the imagescale does have an effect on the image property, but only when the property is set as a character vector bitmap image resource specification. Otherwise the image property ignores the imagescale setting.

There is a similar behavior with respect to the Picture object's bitmap property. You can specify it as a color matrix similar to the image property, and if you do that, the imagescale property is ignored by the bitmap property when loading the bitmap into the Picture background, or when you reference the bitmap property. But when you specify the bitmap as a character vector bitmap resource specification, in that case the imagescale does affect how the background bitmap is scaled when loading into the Picture. But in this case, when you reference the bitmap property, it returns the character vector you set rather than returning the color matrix.

- New dpi property

  Added the read-only system object property dpi that returns a vector with the following values:

  [1] returns 1 if APL+Win is running in DPI aware mode; 0 if DPI un-aware mode.
  [2] returns the system-wide DPI scaling percentage of the system (100 by default, 200 for 200% scaling, etc.).
  [3] returns the dots per inch of the system-wide DPI scaling (96 at 100% scaling, 192 at 200% scaling, etc.).
  [4] returns the pixel height of the default font.

- New scalemode element added to scale property

  The scale property has been extended to support scalemode = 6 to return true pixels.

- Change to existing scalemode =5 element in the scale property

  The scalemode = 5 element has changed from returning pixels units to virtual pixel units.
  Note: Available is a new APLW.INI configuration setting [Config]VirtualPixels that controls whether scalemode = 5 units are virtualized for DPI scaling or unscaled (true) pixels; 1 (default) means virtual pixels are scaled to the current DPI level and 0 means not scaled.

- New defaultFont property to system (#) object

  This enhancement improves the default font selection process for the ⎕wi system when running in DPI aware mode. Instead of allowing Windows to choose a font size for the ⎕wi system, APL does actively participate in the font selection process when running in dpiAware=true mode. This dramatically improves font size versus window layout behavior when running at high DPI scaling levels.

  Font selection is controlled by testing several strings to see how they render at the selected DPI settings and comparing this to how they render at dpiAware=false setting. APL will tweak the font size smaller and smaller until none of the test strings come out wider than they would render if the dpiAware=false rendering were simply stretched to the DPI. This assures that text will not be wider than it was in the previous systems where dpiAware=false was the norm and was always rendered at 100% DPI in an 8x16 box size.

  Here's an example that queries the property, showing its default value:

```
      '#' ⎕wi 'defaultFont'
16 8 [MS Shell Dlg] 13 {Label Caption} 65 {WMXwmxiiii} 56
```

Here's an explanation of all the defaultFont values:

```
  16 8 [MS Shell Dlg] 13 {Label Caption} 65 {WMXwmxiiii} 56 ... {MoreTestPhrases} MoreTestWidths
   ^ ^     ^           ^   ^              ^   ^           ^
   | |     |           |   |              |   |           |
   \ /     |           |   |              |   |           |
    A      B           C   D              E   F           G
```

A - Optional character box size at 100% scaling (height width); (defaults to 16 8)
B - Optional font name (defaults to [MS Shell Dlg])
C - Optional standard font height at 100% scaling; (defaults to 81.25% of character box height in A)
D - Test Phrase whose width is to be checked
E - Width of {Label Caption} test phrase at 100% scaling
F - Test phrase whose width is to be checked
G - Width of {WMXwmxiiii} test phrase at 100% scaling

An empty font name "[]" implies the standard font name "[MS Shell Dlg]". In order to specify the standard font height you must specify a font name or a "[]" placeholder for the standard font name before the font height. In other words you cannot specify the font height like either of the following:

```
13
16 8 13
```

But it can be set like any of the following:

```
[] 13
16 8 [] 13
[Some Font Name] 13
16 8 [Some Font Name] 13
```

You can specify the first three fields without any test phrases, but it is recommended to always include one or more test phrase. They are used as font sizing constraints to make sure the font isn't selected that would cause these phrases to appear relatively wider at high DPI settings than they do at 100% scaling. The font height specification is only a hint about the proper font size to use. The system scales the specified font height specification by the DPI scaling level and uses this scaled height as a starting point for font sizing. For example, if the standard font height is 13 but Windows is running at 200% DPI scaling, APL will scale the 13 by 200% and begin with a 26 pixel high font. APL then tests each of the Test Phrases to see how many pixels wide they render with this proposed font height. If any of the test phrases render wider than their width constraints (after scaling the test width constraint by the DPI scaling factor) APL rejects that font size and pick another one that is one pixel shorter. This is repeated until APL finds a font size where all of the test phrase constraints are satisfied, and that becomes the font size APL actually uses. In the default case, APL has the following two test phrases and width constraints:

```
Test Phrase    Width
-------------  -----
Label Caption  65
WMXwmxiiii     56
```

At 200% DPI scaling APL scales these width constraints up by 200% to 130 and 112, respectively. Then APL tests to make sure that the width of the test phrases do not exceed these scaled-up width constraints. If they do, APL picks a smaller font and repeat until a font size is found where all constraints are satisfied. Note that bogus width constraints could in theory force the font size to be too small to read, so APL has a lower limit of 5 pixels as the minimum font size that will ever be picked. This is too small to read, especially at high DPI settings, but that's the limit at which APL stops searching. This is just a safety limit, not a practical one. If you change the defaultFont, you are responsible for picking appropriate width constraints that are realistic. The best way of doing this is to run APL at 100% scaling and let APL compute the widths for you, as the test phrases would render at 100% scaling. The technique for doing this is illustrated below.

Here's an example that sets the default font property to a different application-specific value:

```
  '#' ⎕wi 'defaultFont' '{TestPhraseOne} 74 {TestPhraseTwo} 75'
```

In the case above some fields (such as character box size, font name, and tentative size) are omitted and only the width constraint fields are specified. If you query the property after setting it like this, the system will return the full specification as shown below:

```
      '#' ⎕wi 'defaultFont'
16 8 [MS Shell Dlg] 13 {TestPhraseOne} 74 {TestPhraseTwo} 75
```

The defaultFont property should be set before any ⎕wi objects are created. Any ⎕wi object created before setting the defaultFont property may exhibit unexpected behavior if a different value is set after they have been created. APL does not preventing you from setting defaultFont when ⎕wi Forms are already in existence, but you are discouraged from doing so.

Now the only remaining question is how to compute proper width constraints for application-specific test phrases. This is easily done by setting the defaultFont property using test phrases that don't specify width constraints, like this :

```
      '#' ⎕wi 'defaultFont' '{TestPhraseOne} {TestPhraseTwo} {TestPhraseThree}'
```

Then query defaultFont to discover the actual widths these strings would be at 100% DPI.

```
      '#' ⎕wi 'defaultFont'
16 8 [MS Shell Dlg] 13 {TestPhraseOne} 74 {TestPhraseTwo} 75 {TestPhraseThree} 82
```

In addition to the dynamically settable defaultFont property of the system object, the default font can also be specified statically by the [Config]DefaultFont parameter in the APLW.INI file. You can use the system object defaultFont property to dynamically experiment with width constraints (letting APL compute default widths for the test strings as they would be drawn at 100% DPI) but then store these settings in the application INI file so that subsequent APL session will configure the default system font the same way for subsequent sessions. If an application experiences font scaling problems at high DPI levels, it can easily resolve them by picking the offending string that is rendering too wide at some DPI settings and adding that string as a test phrase as outlined above. Then use this as a new application-specific [Config]defaultFont setting. After restarting the application, it will now pick a font that is guaranteed not to be too large to create that problem in the future. Applications should try to pick the best test constraints they can to achieve good scaling behaviors. But in doing so they should keep in mind that some users might experience problems that are not uncovered during pre-release testing. That's because different devices behave differently with respect to font mapping. If a user of the application reports a problem, along with a screen shot of the offending issue, the developer can easily compute new test phrase constraints and have the user add the new constraints to their INI file in order to resolve the scaling issue for that specific user, or perhaps add that constraint to the next release of the application so that all users can benefit from this in the future.

Note: The defaultFont property can be set with missing TestPhrase width specifications at any DPI. APL simply selects the standard height (unscaled) font and uses it to compute test phrase widths. For example, set the defaultFont property using test phrases that don't specify width constraints, like this:

```
      '#' ⎕wi 'defaultFont' '{TestPhraseOne} {TestPhraseTwo} {TestPhraseThree}'
```

Then query defaultFont to discover the actual widths these strings.

```
      '#' ⎕wi 'defaultFont'
16 8 [MS Shell Dlg] 13 {TestPhraseOne} 74 {TestPhraseTwo} 75 {TestPhraseThree} 82
```

These test phrases can then be put into the INI file like this, in [Config] section:

```
  [Config]
  DefaultFont={TestPhraseOne} 74 {TestPhraseTwo} 75 {TestPhraseThree} 82
```

Note that you don't need to include the character box size, font name, or default font height in the [Config]DefaultFont setting. Those are all default values in this case, and so don't need to be specified. You can just supply the test phrases and their constraint widths. But if you prefer, you can do the whole spec like this:

```
  [Config]
```

```
DefaultFont=16 8 [MS Shell Dlg] 13 {TestPhraseOne} 74 {TestPhraseTwo} 75 {TestPhraseThree} 82
```

Next time you start APL, these settings will be in effect and will determine what font size is selected for the session as a function of system-wide DPI scaling.

- Improved placement of Find and Replace Dialogs

  Corrected placement of the Find and Replace dialogs when using multiple displays. This means the Find and Replace dialogs appears where they were last positioned instead of the primary display.

- New opaque property for ⎕wi APLGUI objects

  The opaque property defines the behaviour of the opaque feature in most ⎕wi APLGUI objects. The opaque property can have a value of 0 or 1. The default value is 1. A value of 0 causes the background of the object to be transparent such that the parent window behind it is visible through the background area of the control. For example, a Check control that's a child of a Picture control will display the image assigned to the Picture control as the background of the Check control.

- New usereplstr property for the ⎕nfe system object

  This property controls when the APL+Win carriage return, ⎕tcnl, needs replacing with Windows carriage return in text transferred from APL+Win to a Windows file and vice versa. By default, the value of the usereplstr property is set to 1 to perform the character replacement.

- The ⎕nfe encoding property supports hyphenated encoding names

  For example, UTF-8 will be treated the same as UTF8 by the encoding property.

Bug Fixes

- Addressed bug that caused APL, started without enabling DPI awareness, to shrink in size when starting an external C# application that did DPI awareness enabled.
- Addressed bug that prevented pasting or entering more than 32767 and 30000 characters (their default limits) in the RichEdit and Edit controls, respectively, when the controls were first opened and the limit properties had not previously been set to 0 or a value exceeding the default limits.
- Addressed several display issues with the row and column headers font in the session’s numeric editor:
  - the font was too small at high DPI scaling levels

- the background of the font in the row and column header cells and the color of the headers cell did not match
- the column header width was sized too narrow to read all column headers for a very large number of columns containing small numbers such as the example below:

  v←1 1000000ρ0

- Addressed bug that did not permit multi-lined strings in the Write and Read methods in the ⎕nfe system object.
- Addressed the problem with the display of the CommandBar control in a Frame control when the Form control was created hidden or closed. In this case, the CommandBar appeared with a black background color when APL+Win is run with a manifest file.
- Addressed bug when Checkbox and Option controls are nested inside a style=7 (transparent) Frame control that is also nested inside a CommandBar control. In these cases, the background color of the Checkbox and Options controls were not painted correctly (gray rectangle behind them) when APL+Win is run with a manifest file.
- Addressed bug when multiple style=0 Frame controls are nested inside each other. In this case the caption did not display in the same font for all the Frame controls after the top-level Frame control when APL+Win is run with a manifest file.
- Addressed bug when a CommandBar control is a child of a style=0 Frame. In this case, various painting issues occurred when APL+Win was started with a manifest file, like the CommandBar control appearing with a black background color.
- Addressed bug when executing ⎕cn and ⎕cm with a space as the left argument and the letter 'q' in the right argument. In these cases, APL+Win could crash.
- Addressed bug in the function editor where a control structure block in a single line did not expand while collapsed when a search word was found in the collapsed statement.
- Addressed bug in dyadic thorn (⍕) with an empty left argument that sometimes reported a DOMAIN ERROR and sometimes reported a random result instead of correctly reporting a LENGTH ERROR.

Example,

```
      (0ρ0)⍕10 20
************************ 20
      (0ρ0)⍕10 20
```

```
DOMAIN ERROR
      (0ρ0)⍕10 20
           ^
```

Important Note: Any enhancements, features or changes described in this document is subject to change in the production release of APL+Win 17.0.