

Dyalog Version 18.0



Dyalog Version 18.0 was released in June 2020 and is supported under Microsoft Windows, IBM AIX, Linux (including the Raspberry Pi) and Apple macOS. This document discusses highlights of the release.

Additional information regarding the features mentioned below is available online, from several sources:

- The [full set of release notes for version 18.0](#)
- Webinars:
 - [Introducing Dyalog version 18.0](#)
 - [Language Features of Dyalog version 18.0 in Depth \(part 1\)](#)

Overview

Dyalog 18.0 is another major release of Dyalog APL, delivering:

- Significant performance improvements
- A bridge to Microsoft's .NET Core under Windows, macOS and Linux (Intel and Raspberry Pi)
- 3 new primitive APL operators (*constant*, *atop* and *over*)
- 1 new primitive APL function (*unique mask*), and extensions to primitive functions *where* and *partition*, extending the domain from Boolean to integer
- 2 new system functions for case folding/mapping and date/time conversion
- New features to simplify the building, configuration and operation of APL systems
- Many other small improvements

Performance Improvements

Version 18.0 is another very significant release in terms of performance. As usual, we have worked on improving the performance of many of the most widely used primitives, focussing on set operations, searches and sorts. This time, we've also made some more sweeping improvements, removing overheads and reworking the memory manager to improve cache utilisation. According to our own performance test suite, overall performance is improved by around 10%. For more information, see [Dyalog version 18.0 Performance](#).

.NET Core Bridge

To complement the Microsoft .NET Framework bridge, which has been available under Microsoft Windows since Microsoft .NET was born in 2002, version 18.0 adds a very similar interface to the new .NET Core, which is intended to replace the .NET Framework over the next few years.

The .NET Core is an open source framework available on all platforms on which Dyalog runs (except IBM AIX). It provides a vast collection of useful APIs for application development. With a few exceptions, in particular some graphical libraries, these class libraries are identical on all the platforms, providing a huge boost to cross-platform application development and deployment.

```

=>SH 'uname -a'
Linux thor8 5.3.0-53-generic #47~18.04.1-Ubuntu SMP Thu May 7 13:10:50 UTC
MKDIR folder←home, '/stuff'
←datafile←folder, '/data.json'
/home/mkrom/stuff/data.json
50↑data←(JSON('HighRank' 'Split')) 1000 4pt12
[[1,2,3,4],[5,6,7,8],[9,10,11,12],[1,2,3,4],[5,6,7
data INPUT datafile
←zipfile←home, '/zippedstuff.zip'
/home/mkrom/zippedstuff.zip

USING←'System.IO.Compression, System.IO.Compression.ZipFile'
ZipFile.CreateFromDirectory folder zipfile

2 NINFO zipfile datafile          A File sizes
197 11001
100×1-÷/2 NINFO zipfile datafile  A Ratio of file sizes
98.2092537
3 NDELETE zipfile folder          A Clean up
|
    
```

As an example of the sort of thing that you will find in the .NET Core, the image shows the use of the System.IO.Compression library to "Zip" a folder containing a single JSON data file under Ubuntu Linux, achieving 98% compression (the data is highly repetitive). Also note the use of the 'HighRank' variant on JSON, which is new in version 18.0 and automatically splits any arrays of rank 2 or higher into lists of lists, to satisfy the constraints of the JSON notation.

In version 18.0, the .NET Core bridge allows the use of .NET Core libraries from APL. The ability to export APL classes as .NET Core libraries so that APL code can be called from other languages which support the .NET Core will follow in version 19.0.

Launching APL

Version 18.0 allows APL to directly run source files containing APL functions, operators, classes or namespaces – without requiring a binary workspace file or a SALT **.dyapp** file. If APL is started with a `LOAD=` parameter naming a text file, the interpreter will load the code into the workspace, and run it if it adheres to simple conventions like defining a function with the same name as the file, or a namespace or class containing a function/method called `Run`. The example below will work on Microsoft Windows, macOS or Linux, using the `HTMLRenderer` to display HTML that displays the name of the folder that the file was



```
>HelloWorld.aplf - Notepad
File Edit Format View Help
▼ HelloWorld dummy

load←2 ⍵NQ'.' 'GetEnvironment' 'LOAD' ⌘ The LOAD= parameter
folder↔1 ⍵NPARTS load ⌘ ⍵NPARTS "normalises" name

html←'<h1>Hello World!</h1>This was loaded from <i>',folder,'</i>'
'HR'⍵WC'HTMLRenderer'('HTML'html)('Size' 15 25)
⍵DQ'HR'
⍵OFF
▼

Command Prompt
C:\Docs\Webinars\200430\Demo>
C:\Docs\Webinars\200430\Demo>dyalog load=HelloWorld.aplf
C:\Docs\Webinars\200430\Demo>
```

Hello World!

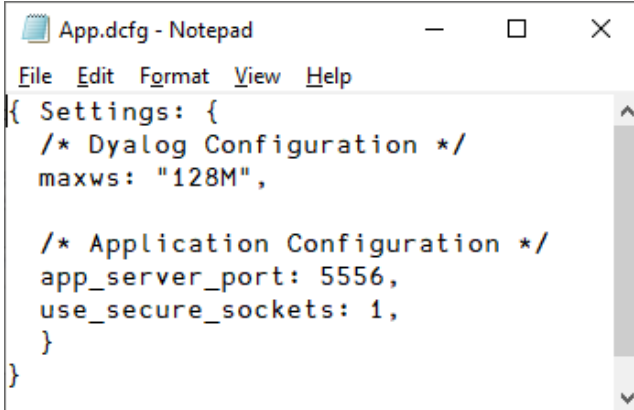
This was loaded from *C:\Docs\Webinars\200430\Demo*

located in; the screenshot was captured on Microsoft Windows.

(This is one of the examples from the webinar introducing the highlights of v18.0 – see the link at the top of this page).

Configuration Files

Traditionally, Dyalog has used configuration mechanisms that are native to the platform on which it is running, for example, environment variables (on all platforms), INI files (on DOS and Microsoft Windows) and the Registry (on Windows). To make it easier to configure applications to run on multiple platforms, and also to allow configuration to be stored in source code repositories along with the rest of the application source, version 18.0 introduces a text-based configuration mechanism, based on JSON5.



```

App.dcfg - Notepad
File Edit Format View Help
{ Settings: {
  /* Dyalog Configuration */
  maxws: "128M",

  /* Application Configuration */
  app_server_port: 5556,
  use_secure_sockets: 1,
}
}

```

New Primitives

Dyalog started out as a member of the IBM APL2/STSC NARS family of APL interpreters, closely following IBM's model of 2nd generation APL systems. Dyalog was subsequently extended with functional programming in the form of dfns and object-oriented programming, closely aligned with the Microsoft .NET CLR or C# model of OO. In recent years, we have been adding features inspired by, and in some cases directly copied from, grounded array languages systems like SHARP APL and J. Examples include the *rank*, *key* and *stencil* operators, the *interval index* function, total array ordering (TAO) and leading-axis emphasis on many primitives, allowing the extension of several primitives to higher rank arrays. In addition to adding powerful features, the grounded array features facilitate the use of simple rather than nested arrays, which provides better [mechanical sympathy with modern hardware and compiler technology](#).

In version 18.0, we have selected another set of primitives pioneered in SHARP APL and/or J, for inclusion in Dyalog:

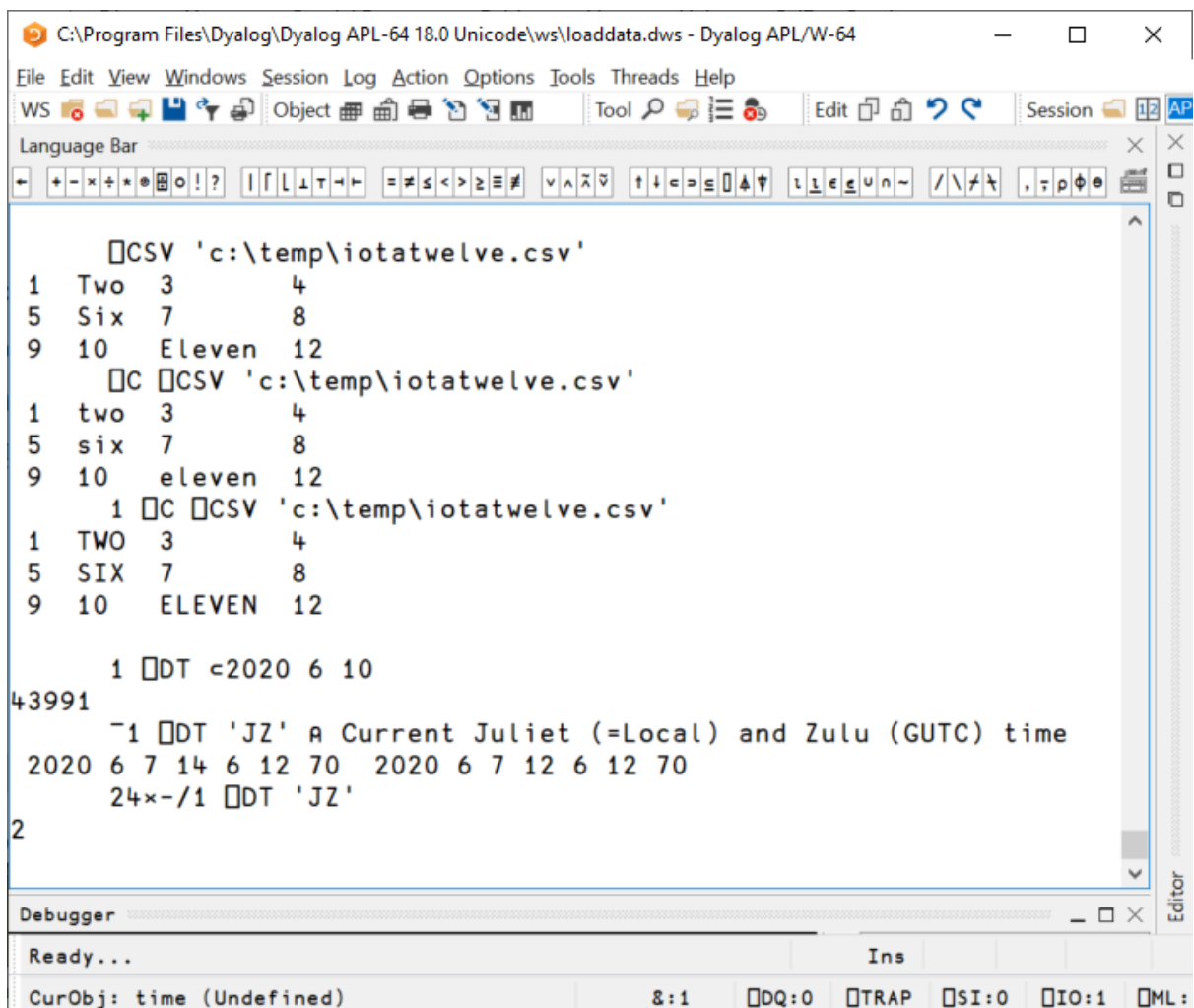
- $f \text{ over } g: \alpha (f \overset{\circ}{\circ} g) \omega \leftrightarrow (g \ \alpha) \ f \ g \ \omega$.
For example, compare the magnitudes of two arrays using $=\overset{\circ}{\circ}$, which applies absolute value to both the right and left argument before comparing them.
- $f \text{ atop } g: \alpha (f \overset{\circ}{\circ} g) \ \omega \leftrightarrow f \ \alpha \ g \ \omega$.
Useful for binding functions together in tacit expressions.

- array constant: $\alpha \sim$ effectively turns an array α into a function that can be applied in conjunction with other operators like *each* or *rank* to simplify the creation of new arrays
- unique mask: $\neq \alpha$ returns a Boolean vector of length $\neq \alpha$, indicating the first occurrence of each major cell of α .
For example, $(1\ 0\ 1\ 0\ 0 \equiv \neq 42\ 42\ 43\ 43\ 43)$

For more detail regarding the new primitives, extensions to *where* ($\underline{1}$) and *partition* (ϵ) and the system functions described in the next section, see the in-depth language feature webinars linked to at the top of this page.

New System Functions

In addition to new primitives, version 18.0 contains two new built-in library functions, one for case mapping and folding, and one for date/time conversion. The screenshot shows the use of the `⎕C` (case conversion) system function to fold or map all character elements of an array loaded from a CSV file using `⎕CSV`, leaving all other data unchanged. A left argument



can be provided to select one of the following options: 1 maps to upper case, -1 maps to lower case, -3 (the default) folds arrays for case-less comparison – mostly the same as mapping to lowercase, but with special treatment of characters like the Greek Sigma "Σ", which has two lower-case forms "σ" and "ς" – they all fold to "σ". It also shows the use of `⌈DT`, first to convert a timestamp to a day number, and finally to compute the current UTC offset. `⌈DT` is able to convert between `⌈TS`-style timestamps and more than 20 different date/time representations, including Dyalog date numbers, Julian day numbers, J, K, JavaScript, R and Excel date formats, component file time stamps, and so on.

Summary

Version 18.0 continues the tradition of continuous performance improvement. Since version 14.1 in 2005, the speed of language primitives, as measured by own performance suite, has more or less doubled. Version 18.0 has language and library enhancements to improve programmer productivity, features to simplify configuration and deployment, and an interface to Microsoft's .NET Core. There are also numerous enhancements to the Windows Integrated Development Environment (IDE), the cross-platform Remote IDE, improved tools for interfacing with source code management systems (Link), and a new framework for building web services in APL (the [Jarvis framework](#)).