Dyalog v21.0



Dyalog v21.0 was released in November 2026 and is supported under Microsoft Windows, IBM AIX, Linux for x64 and ARM (incl. Raspberry Pi), and Apple macOS.

This document discusses highlights of the Dyalog v21.0 release. For additional information regarding the features mentioned below, see:

- the release notes for Dyalog v20.0 (https://docs.dyalog.com/21.0/release-notes/)
- the release notes for Link v4.2 (https://dyalog.github.io/link/4.2/ReleaseNotes42/)
- the list of issues resolved in Ride v4.7 (https://github.com/Dyalog/ride/milestone/6?closed=1)

Introduction

Dyalog v21.0 is a relatively small release in terms of features that are visible to the user, with a lot of internal work being done to prepare for future improvements. However, it does include several new features that are intended to increase developer productivity, especially if your application needs to run on more than one platform.

Video introducing Dyalog v21.0 from Dyalog Ltd's Technical Director (CTO) Morten Kromberg: https://www.youtube.com/watch?v=cHGA_zMPPIo

Key Enhancements in Dyalog v21.0

Key enhancements in Dyalog v21.0 include the following:

- ⎕DT has been extended to validate and parse text-format datetimes.
- ⎕SYSTEM provides information about Dyalog and the environment, including the operating system, important library versions, and the current/initial/temporary directories.
- .NET generics are now fully supported for both .NET and .NET Framework.
- A complete Software Bill of Materials is now included.

Parse and Format Dates using ⎕DT

⎕DT has been extended to validate and parse text-format datetimes. It now also includes formatting features prototyped as 1200⌶.

Example:

```
  'YYYY:MM:DD hh:mm' ¯1 ⎕DT⊂'2026:09:07 12:20'
2026 9 7 12 20 0 0

      1 'Ddd YYYY:MM:DD "at" hh:mm'⎕DT 47000.33334
Tue 2028:09:05 at 08:00
```

Inspect Platform Attributes using ⎕SYSTEM

⎕SYSTEM is intended as a one-stop-shop for information that applications typically need during initialisation so that they can adapt to the current execution environment. In Dyalog v21.0, this new system namespace contains nearly 50 items of information about the Dyalog executable, the current process and the command line it was invoked with, important library versions, the operating system, and the current, initial, and temporary directories.

Examples:

```
      ⎕SYSTEM.⎕NL ¯9
(
'CommandLine'
'Directories'
'Executable'
'Features'
'Host'
'OS'
'Process'
)

      ⎕SYSTEM.Executable
(
Bits:64
BuildTarget:'win'
BuildType:'opt'
Path:'C:\Program Files\Dyalog\Dyalog APL-64 21.0 Unicode\dyalog.exe'
RideConnected:0
Runtime:0
Unicode:1
Vendor:'Dyalog'
Version:21 0 54393
VersionMoniker:'210U64'
VersionNumber:21
)
```

Refer to Other Settings in Any Configuration Parameter

A configuration parameter can refer to the value of another system parameter by naming it within square brackets, such as in **[DYALOG]/MyFile**. Prior to Dyalog v21.0, this only worked within configuration files; it now works wherever a setting is defined. On Microsoft Windows, the new **[=DOCUMENTS]** reference stands for the user's **Documents** directory. Dyalog's own settings use this, which means that they work even if a tool (for example, OneDrive or Google Drive) relocates that folder, and Dyalog does not need to be reinstalled.

APL Syntax for .NET Generics – Built for .NET 10

Dyalog v21.0 is built for .NET 10; this is the current Long-Term Support version of .NET, scheduled to be supported until November 2028. The .NET Framework continues to be supported, and there are no plans to retire Dyalog's .NET Framework interface.

The APL language has been enhanced to provide a natural way to make use of generic methods and classes – this support is available for both the .NET and the .NET Framework interfaces.

Example:

```
      ⎕USING←'System'
      types←Char Int32
      dict←⎕NEW Collections.Generic.Dictionary[types]
      dict.Add'L' 42
      dict.Add'P'(⌊22÷7)
      dict['P']
3
```

Software Bill of Materials Included

If you distribute solutions built on Dyalog, you can now use the "SBOM" delivered with Dyalog v21.0 to document any of our files that you redistribute as part of your application.

Experimental Features

Dyalog v21.0 contains several experimental features. If you would like to explore these and provide feedback on them, please contact us!

Thread Scheduling

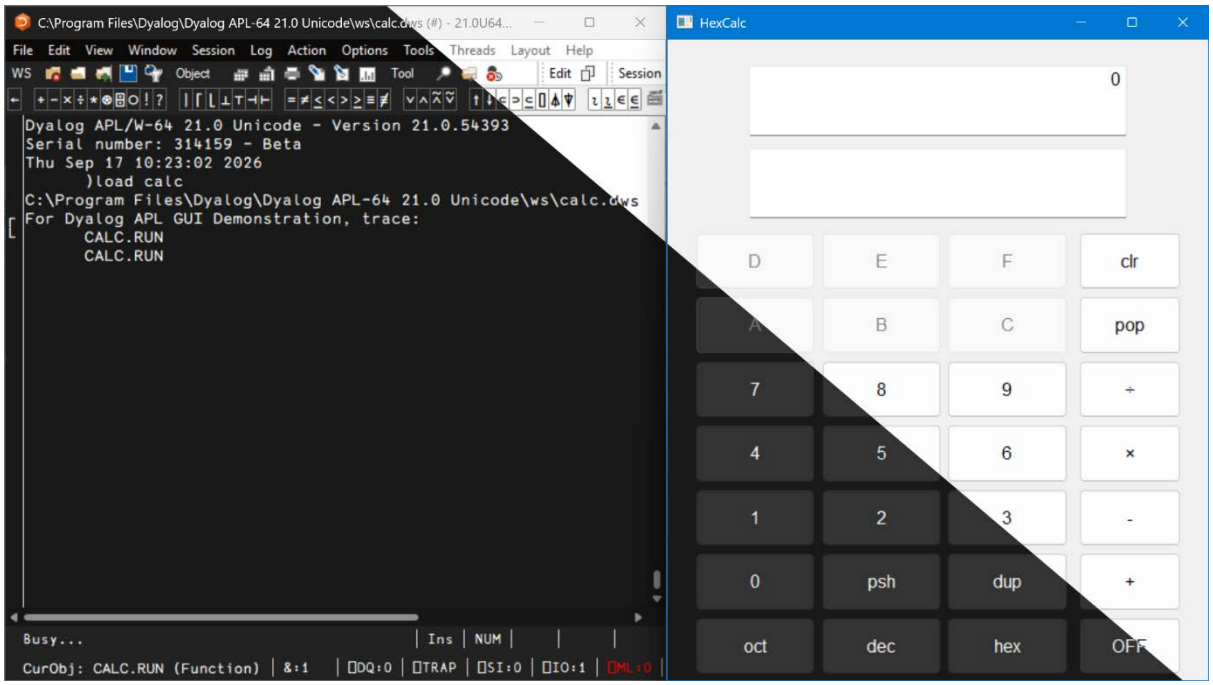
Changes have been made to the thread handling code in preparation for future improvements to the efficiency and "fairness" of thread scheduling. Although these changes are not visible to the end user, we would be grateful if you could test any of your code that uses multiple threads and report any issues to support@dyalog.com.

The APL thread scheduler checks that its own record of each thread's scheduling status is consistent. The temporary I-beam `4061⌶0` returns a two-element vector: the number of inconsistent thread statuses and the number of missed thread wakeups that have been detected. A non-zero count means that the interpreter detected an inconsistency in its own scheduling, not that the application is at fault.

In Dyalog v21.0, the interpreter will perform consistency checks to verify that new algorithms (which will be implemented in Dyalog v22.0) perform correct scheduling. In the unlikely event that discrepancies are detected, the interpreter will write to the Status window or, if it is enabled, the log file set using `109⌶`; again, we would again appreciate it if you could send such information to support@dyalog.com.

Dark Mode (Microsoft Windows only)

Dyalog v21.0 for Microsoft Windows includes experimental support for dark mode, and we are keen to have feedback on this new mode. For information on how to enable dark mode, please send a message to darkmode@dyalog.com briefly describing the experiments that you intend to perform. This feature is still under development; the implementation (and thus the appearance of individual elements) could change at any time throughout the life of Dyalog v21.0.



Visit the Documentation Centre for the documentation for Dyalog v21.0.