

## Dyalog v20.0



Dyalog v20.0 was released in November 2025 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 v20.0 release. For additional information regarding the features mentioned below, see:

- the release notes for Dyalog v20.0 (<https://docs.dyalog.com/20.0/release-notes/introduction/>)
- the release notes for user commands v2.7 ([https://docs.dyalog.com/20.0/files/User\\_Commands\\_v2.7\\_Release\\_Notes.pdf](https://docs.dyalog.com/20.0/files/User_Commands_v2.7_Release_Notes.pdf))
- the release notes for Link v4.1 (<https://dyalog.github.io/link/4.1/ReleaseNotes41/>)
- the list of issues resolved in Ride v4.6 (<https://github.com/Dyalog/ride/milestone/9?closed=1>)

### Introduction and Key Enhancements

Dyalog v20.0 is one of the most significant developments of both the Dyalog APL language and the Dyalog IDEs in recent years. Joining such releases as Dyalog v8.1 (1997) and Dyalog v14.0 (2014), which introduced *dfns* and *trains* respectively, Dyalog v20.0 introduces a new way to write arrays, with the introduction of *array notation*. On the IDE front, Dyalog v20.0 is comparable to Dyalog v5.1 (1988) and Dyalog v13.2 (2013), which introduced the windowed Tracer/Editor and David Liebttag's array editor respectively, with the ability to reformat array notation in the editor (replacing the array editor) and the introduction of *inline tracing*.

Key enhancements in Dyalog v20.0 include the following:

- Array notation, a literal syntax for most arrays (including nested and high-rank arrays) and namespaces, has been introduced. Array notation allows you to write literal arrays in a way that was not possible before.
- Namespace manipulation and data extraction has been made easier with the addition of `⊖VGET` and `⊖VSET`, and extensions to `⊖NS`. These changes are particularly useful when dealing with namespaces generated by applying `⊖JSON` to JSON requests from webservices, as default values can be specified to mitigate the effects of missing members.
- The two common patterns `( f Y ) g Y` and `( f X ) g Y` can now be represented by the new compositional operator *behind* (`⊖`).
- `⊖SHELL` enables execution of external programs with more control and options than those provided by `⊖SH`/`⊖CMD`.
- Expressions can now be evaluated primitive-by-primitive using inline tracing, while accessing arguments and results of each function along the way.

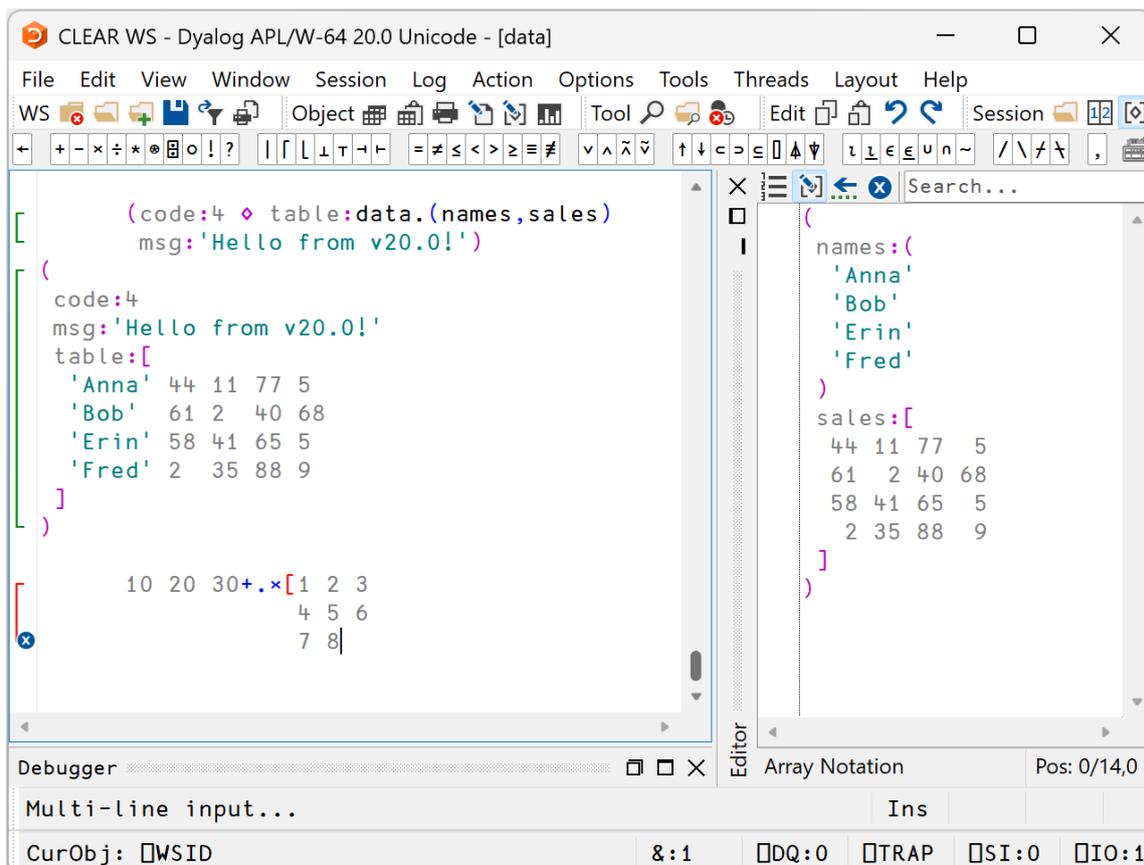
## Platform News

We are always working to stay aligned with new developments in the hardware scene, and for Dyalog v20.0 we have expanded our ARM64 support to include Linux and Raspberry Pi. We have also published example Docker containers for these platforms to help you get started with running on cloud services that use these systems.

In 2023, Apple Inc discontinued the last Intel-based Mac computer, and the last version of macOS with Intel support has just been released. In line with this, Dyalog v20.0 is the first Dyalog version that does not have Intel support.

## Array Notation

First suggested by Phil Last in 2015, APL array notation means that most arrays can be written literally, optionally over multiple lines. Unlike strand notation, it can also be used to write arrays of rank higher than 1. Array notation comprises a vector notation written with parentheses, ( ), a high-rank notation using square brackets, [ ], and a namespace notation using name:value pairs in parentheses.



The screenshot shows the Dyalog APL/W-64 20.0 Unicode IDE. The main editor displays a function definition using array notation:

```
(code:4 ♦ table:data.(names,sales)
msg:'Hello from v20.0!')
(
code:4
msg:'Hello from v20.0!'
table:[
'Anna' 44 11 77 5
'Bob' 61 2 40 68
'Erin' 58 41 65 5
'Fred' 2 35 88 9
]
)
10 20 30+.×[1 2 3
4 5 6
7 8]
```

The right-hand pane shows the internal representation of the array notation:

```
(
names:(
'Anna'
'Bob'
'Erin'
'Fred'
)
sales:[
44 11 77 5
61 2 40 68
58 41 65 5
2 35 88 9
]
)
)
```

The bottom status bar shows the current object is `WSID` and the cursor is at position `0/14,0`.

Array notation has multiple benefits, including:

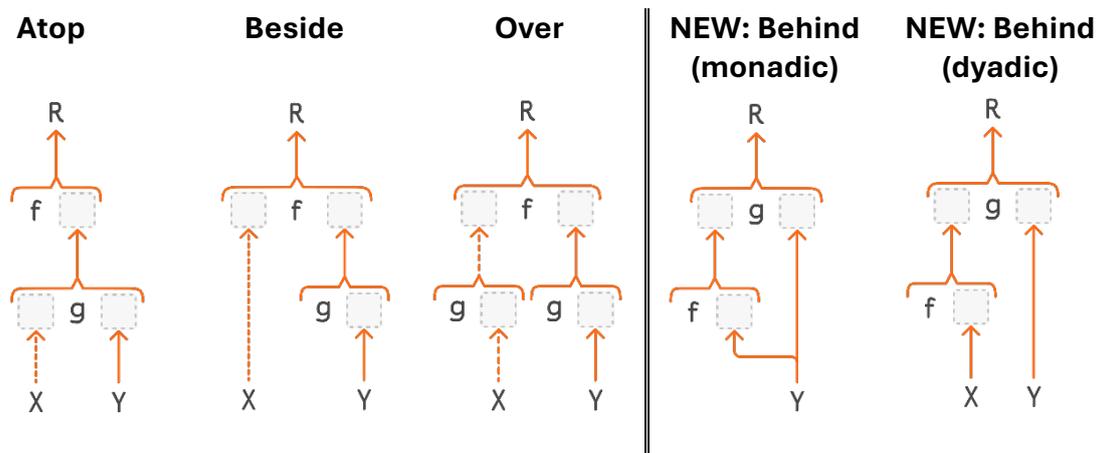
- allowing intuitive representation of matrices.
- writing vectors of character vectors in a convenient form over multiple lines.
- creating namespaces for function calls that resemble usage of named-arguments.

## Namespace Manipulation

Namespaces are increasingly being used as data structures, especially for sharing data between APL and other systems. Dyalog v20.0 provides three powerful tools to manipulate such data – new system functions `⊖VGET` and `⊖VSET` and an enhanced `⊖NS`. Applying these should not only make your code cleaner and more robust, it will also mean that you can eliminate most of the uses of `⊖` that were previously required, leading to safer and more performant code.

## Tacit Programming

*Behind* (`⊖`) completes the set of compositional operators, and thus our full support for tacit programming. The journey began with Dyalog v1.0 in 1983. Unusually for APLs at the time, it included the *compose* operator (`∘`) from STSC's experimental Nested Arrays System ([NARS](#)); three years later, Dyalog v4.0 added the ability to assign derived functions. Almost three decades then passed before Dyalog v14.0 introduced trains in 2014; six years after that, *compose* was renamed to *beside*, and the *atop* (`∘`) and *over* (`∘`) compositions were added, together with *constant* (`∘`), which is also useful in tacit programming.



Syntax of the full set of functional composition operators

*Behind* offers a convenient way to write two common patterns:

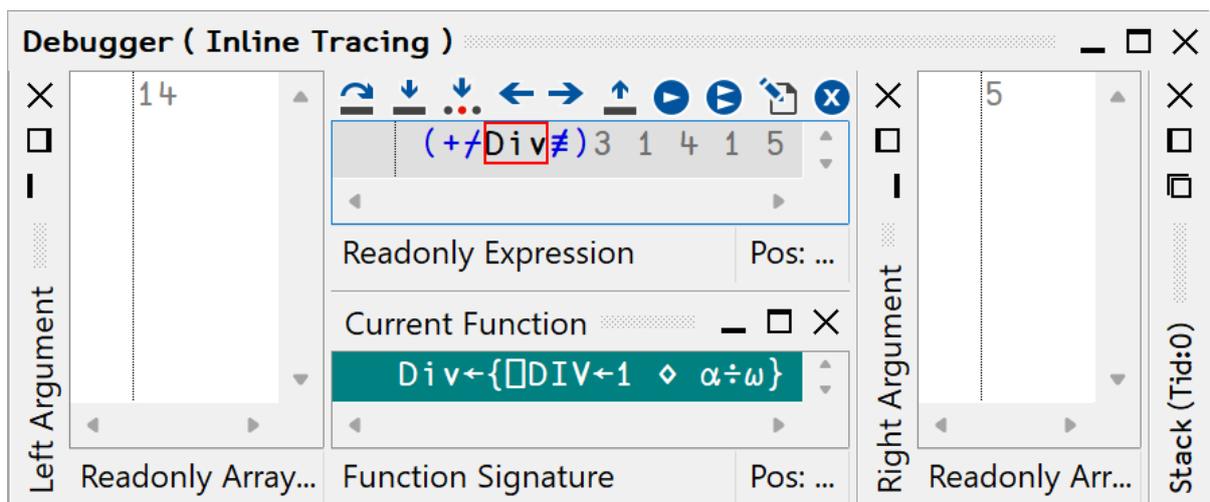
- calling a function with an argument and some aspect, version, or modification of that argument. For example, to check whether a matrix is symmetric over its diagonal: `⊖⊖≡ myMatrix`.
- pre-processing a left argument. For example, to check whether a set of given words case-insensitively are members of an already case-folded word list: `givenWords ⊖⊖ε wordList`.

## Executing External Programs

The system functions `⎕CMD` and `⎕SH` offer limited control over the execution of external programs. They cannot be interrupted, dealing with input and output streams is not without pain, and output is discarded if the called process did not exit cleanly. They have now been superseded by a new system function, `⎕SHELL`, which eliminates these issues and adds significant functionality, including the ability to run external processes in the background, with custom environments, and in specific start-directories. `⎕SHELL` also allows the inspection of output that has been produced before the process terminates.

## Inline Tracing

Inline tracing is an extension to tracing that allows you to step through the execution of individual primitives within expressions, examining intermediate results and arguments of sub-expressions. It enables an in-depth inspection of complex expressions typed directly into the session, and can be used in conjunction with the traditional tracing mode to skip lines you're not interested in and step through primitive-by-primitive in complex expressions where required.



This will be particularly valuable both for those learning APL, and for anyone who needs to understand and debug complicated APL expressions and functions, for example, those that involve intricate tacit functions.

## .NET Generics

In .NET, generic classes and methods are classes and methods that have type parameters which must be specified to create concrete versions of the class or method in question. `⋄3⊖632` allows you to specify the required types.

Note: The I-Beam is a temporary solution that we expect to deprecate soon in favour of generic support integrated with the core language.

## Component File Enhancements

Writing to a component file is not always permitted; restrictions could be imposed by the operating system, the filesystem, or individual component file properties.

The system functions `⊞FSTIE` and `⊞FTIE` now have a **Mode** variant option that lets you specify that you intend to write to the file that you are attempting to tie. This means that you can be notified immediately if the file cannot be written to, rather than tying the file and then generating an error when attempting to write to it.

## Native File Enhancements

Dyalog v20.0 introduces several smaller quality-of-life improvements to system functions that interact with native files:

- `⊞MKDIR` can create uniquely named directories, obviating the process of using `⊞NCREATE` 'Unique' followed by `⊞DELETE` and `⊞MKDIR`.
- `⊞NINFO` can now set file properties as well as report them; it can also provide progress reports while processing time-consuming operations.
- `⊞NPUT` can now read and write matrices directly, removing the need to mix and split vectors of character vectors.

## Text Tools

Dyalog's text manipulation capabilities have been increased with the addition of Unicode normalisation (using `5581⊞`).

Our support for regular expressions in `⊞R`, `⊞S`, and tools that rely on them, as well as in the Microsoft Windows IDE's Search, Find, and Replace functionalities, has been updated to the latest PCRE2. This version is not 100% compatible with the previous version, but it is unlikely that typical uses will be affected.

UUID versions 4 and 7 can now be generated using `120⊞`. This is simpler to use and more performant than the previous technique of writing APL code or using .NET (or another library). To illustrate this, we can define functions for generating a UUIDv4 in different ways – using .NET, writing APL code, and using the new I-beam:

```
DotNet_UUIDv4←{⊞System.Guid.NewGuid}
APL_UUIDv4←{'-'@(4+5×ι4)⊞C(⊞D,⊞A)[4(9+|)@20⊖5@15?36ρ16]}
IBeam_UUIDv4←{120⊞4}
```

Comparing the performance of these shows the performance improvement when using the new I-beam:

```
DotNet_UUIDv4⊞ → 1.1E-5 | 0% ████████████████████████████████████████
* APL_UUIDv4⊞ → 7.0E-6 | -35% ████████████████████████
* IBeam_UUIDv4⊞ → 2.0E-6 | -82% ██████
```

## IDE Improvements

In addition to the major features already mentioned, both the Microsoft Windows IDE and Ride have seen a variety of smaller, but significant, functionality improvements. Many of these are primarily to support array notation, but some of them are also very useful in general. For example, the Home and End keys in the TTY interface and the Microsoft Windows IDE are more versatile, the *Reformat* command (<RD>) now works in the Session, and the *Undo All* command (<UA>) (and, in Ride and the Windows IDE, a button in the gutter) exits multi-line input.

The Microsoft Windows IDE has also been enhanced by adding the keyboard function keys (programmable using the `PFKEY` system function) to the keyboard shortcuts, allowing you to bind this macro-like functionality to any keystroke. For example, you can make Ctrl+Backspace delete a word to the left of the cursor by assigning F42 to this action ('Lw' 'DI' `PFKEY 42`) and then binding Ctrl+Back to F42 in the **Keyboard Shortcuts** tab of the **Configuration** dialog box.

## Handling Language Evolution

Dyalog Ltd takes backwards-compatibility seriously, and you can generally rely on existing code continuing to work in the future. One exception to this is I-beams, as we reserve the right to change, remove, or replace them by other functionality.

To ease you through such transitions, we have introduced a new logging facility with an option (using `13I`) to detect any use of specific deprecated features within your code. You can choose to have such instances written to a file (using `109I`). You can also use `3535I` to search for files of a deprecated type. These logging facilities will be expanded in a future release.

## New APL Font

Our default font, *APL385 Unicode*, originated as cross-vendor work in the late 1980s, and was subsequently maintained as a Unicode font by Adrian Smith. However, APL has developed significantly since then, and new glyphs have been added. In 2019, Adám Brudzewsky published *APL386 Unicode*, gradually adding glyphs and fixing issues, helped by community feedback. Inspired by this work, we hired an external contractor in the summer of 2024 to redraw the font. After delivery, we opened up the process to the community as *APL387 Unicode*; Madeline "RubenVerg" Vergani volunteered to fix remaining issues, and continues to enhance the font.

The new font is installed with Dyalog v20.0, but the default font for APL within Dyalog remains *APL385 Unicode*. If the new font is well received it might become the default for future versions of Dyalog. The Dyalog v20.0 Release Notes include information on how to [explore the new font and provide feedback](#).

Visit the [Documentation Centre](#) for the documentation for Dyalog v20.0.