*Version: 19.0*

*Revision: 4025 dated 20240422*

**Please note that unless otherwise stated, all the examples in this document assume that ⎕IO is 1, and ⎕ML is 1.**



*email: support@dyalog.com*
*https://www.dyalog.com*

# Contents

# Introduction

This manual is designed to assist users of Dyalog APL on platforms other than Microsoft Windows. For further information, see the Dyalog UNIX and Linux forum.

The *Dyalog for UNIX UI Guide* and the *Dyalog for Raspberry Pi User Guide* are also non-Windows specific. Users should also review the *Dyalog Version 19.0 Release Notes* and the file *dyalog_readme.htm*. All of these files and the other Dyalog-supplied documentation can be found in the directory $DYALOG/help, and are available online at https://docs.dyalog.com/19.0. https://help.dyalog.com/19.0 contains an online help system for the Dyalog APL documentation. These websites are updated from time to time, and have the latest revisions of the documentation.

Version 19.0 supports RIDE, the Dyalog Remote IDE, versions 3 or 4. RIDE 3 is available for Windows and macOS (it is the default interface on macOS), and RIDE D4 is available on Windows, Linux, macOS and Raspberry Pi; over time Dyalog intends to make RIDE the default interface under windows managers on all platforms. For more information about RIDE, see the *RIDE User Guide* for more information.

Throughout the document the directory in which Dyalog APL has been installed is referred to as **$DYALOG**; this is because it is the name of an environment variable, whose value can most easily found by running the following expression in Dyalog:

```
+2 ⎕nq '.' 'GetEnvironment' 'DYALOG'
```

Two versions of the interpreter are shipped with each Dyalog APL release: the development version and the server version.

The server version has the same functionality as the development version, other than that any attempt to read from the session, or use `⎕SM` or use `⎕ARBIN` will result in an `EOF INTERRUPT`. It is mainly intended for using Dyalog APL as a server process, where all I/O is processed using TCPSockets, or possibly via an auxiliary processor written by the user. Dyalog recommends using Conga in preference to native TCPSockets.

There are different licences associated with the development and server versions, which affects how each might be distributed. For more information, please contact sales@dyalog.com.

All examples are written assuming that the Korn shell is being used.

# Installation

This manual covers the installation of the non-GUI version of Dyalog APL on AIX, and on Linux distributions which use either .rpm or .deb files for installing software. If you are using a Linux distribution which uses some other method, or you wish to have a non-default installation, then there are some suggestions about how such an installation might be completed.

Dyalog APL version 19.0 is supplied in either 32 or 64 bit versions, and in either Classic or Unicode editions. The installation procedure for Dyalog APL is the same in each case. Note that the 64-bit versions of Dyalog APL will only run on a 64-bit operating systems; the 32-bit versions of Dyalog APL will run on both 32 and 64 bit operating systems.

It is assumed that in all cases the installation image has been downloaded into /tmp on the local machine.

The default installation subdirectory will be formed as:

```
/opt/mdyalog/19.0/<APLWidth>/<APLEdition>
```

or, in the case of AIX:

```
/opt/mdyalog/19.0/<APLWidth>/<APLEdition>/<platform>
```

So for example, Dyalog APL Version 19.0 32 bit Unicode for POWER6 hardware on AIX will by default be installed into

```
/opt/mdyalog/19.0/32/unicode/p6
```

whereas on a Linux distribution the equivalent version would be installed in

```
/opt/mdyalog/19.0/32/Unicode
```

This naming convention began with Version 12.0, and is planned to continue into the future. This ensures that all versions and releases of Dyalog APL can be installed in parallel.

As part of installing Dyalog on Linux (including Pi) the script */usr/bin/dyalog* is created; this is a copy of the $DYALOG/mapl script and can be used to start Dyalog APL. Note that this script will start the most recently installed version of Dyalog APL. This script is used in the target of the Dyalog APL icon on Linux desktops. If preferable, Dyalog can be started by calling the script mapl in the appropriate Dyalog installation directory.

When supplying updates or fixes, Dyalog issues a full installation image; this means that any file under the installation subdirectory may be overwritten. It is therefore strongly recommended that users do not alter issued files, as those changes could be lost if an update is installed.

Dyalog APL version 19.0 for Linux is supplied as a zip file which contains both a .deb- and a .rpm-based installation image.

# Installing under AIX

For each version of Dyalog APL on AIX three separate hardware-specific builds are created for each of the four combinations of 32 or 64 bit versions, Classic or Unicode editions. For version 19.0 specific builds for p5, p6 and p7 are created.

```
$ su -
# cd /opt
# cpio -icdvum </tmp/dyalog-20090901-64-unicode-p6.cpi
# /opt/mdyalog/19.0/64/unicode/p6/make_scripts
# exit
```

Dyalog APL is now installed. To run as any user, type

```
$ /opt/mdyalog/19.0/64/unicode/p6/mapl
```

Notes:

- Version 19.0 is compiled on AIX6.1.

# Installing on an RPM-based Linux Distribution

```
$ unzip linux_64_15.0.26964_unicode.zip
$ sudo rpm --install linux_64_15.0.26964_unicode.x86_64.rpm
```

Dyalog APL is now installed. To run as any user, type

```
$ dyalog
```

or

```
$ /opt/mdyalog/15.0/64/unicode/mapl
```

### Notes:

- It may be necessary to use the --force flag or equivalent if an earlier version of Dyalog APL is to be installed on the same server as a later version. This is safe since the versions have no files in common.
- It has been noticed that in some circumstances the 32-bit installs fail on 64-bit operating systems due to a missing ncurses package. However, it appears that that package is indeed installed. What is required however is the 32-bit version: once installed, Dyalog APL will then install.

# Installing on a DEB-based Linux Distribution

```
$ unzip linux_64_15.0.26964_unicode.zip
$ sudo dpkg --install linux_64_15.0.26964_unicode.x86_64.deb
```

Dyalog APL is now installed. To run as any user, type

```
$ dyalog
```

or

```
$ /opt/mdyalog/15.0/64/unicode/mapl
```

### Notes:

- It may be necessary to use the --force flag or equivalent if an earlier version of Dyalog APL is to be installed on the same server as a later version. This is safe since the versions have no files in common.
- If dpkg generates dependency errors, run `apt-get install -f` (as root)
- It has been noticed that in some circumstances the 32-bit installs fail on 64-bit operating systems due to a missing ncurses package. However, it appears that that package is indeed installed. What is required however is the 32-bit version: once installed, Dyalog APL will then install.

# Installing in a non-default location

It is possible to install Dyalog APL for UNIX in non-default locations, without the need for root privileges.

For all UNIXes,

```
cd <directory under which I wish to install Dyalog APL>
```

For AIX:

```
cpio -icvdum <installation_image.cpi
```

For .deb based Linux distributions:

```
/usr/bin/dpkg --extract installation_image.deb .
```

For .rpm based Linux distributions

```
rpm2cpio installation_image.rpm | cpio -icdvum
```

For all UNIXes:

```
find opt/mdyalog -name make_scripts -exec {} \;
```

This last step generates the mapl script; should you chose to move the installation directory, it will be necessary to re-run the make_scripts script so that the environment variable $DYALOG is set correctly.

# Deinstalling Dyalog APL

In the following examples, it is assumed that only Dyalog APL 14.0 64-bit Unicode is installed on the server; the commands to delete directories will need to be more specific if multiple versions of Dyalog APL are installed.

Should it be necessary to deinstall Dyalog APL, then the process is:

## Deinstalling under AIX

```
$ su -
# cd /opt
# rm -rf mdyalog/14.0
```

## Deinstalling on an RPM-based Linux Distribution

```
$ su -
# rpm -e dyalog.32.classic-14.0-20090901
# cd /opt
# rm -rf mdyalog/14.0
# exit
```

## Deinstalling on a DEB-based Linux Distribution

```
$ sudo su -
# apt-get purge dyalog-unicode-140
# cd /opt
# rm -rf mdyalog/14.0
# exit
```

# Upgrading APL

## Applying a later release of the same version

In general Dyalog will issue a new installation image if a problem is discovered which requires a new version of the interpreter. Dyalog recommends that the entire installation image is installed over the existing installation, but that is not essential. Particularly in a live environment it may be preferable to install only a revised interpreter. This can be done by extracting the individual files from the installation image, and copying them into the correct place in the installation directory tree. To apply a fix image, run the appropriate installation command with the -force option if appropriate. Be aware: the process of installing a later installation image over an already installed version of Dyalog APL WILL result in all files being overwritten. If you have changed any, it will be necessary to take copies of them, and then to reapply local alterations to the new files. Please contact support@dyalog.com for further advice.

For rpm-based installation, run

```
$ sudo --Uvh <new installation image>
```

For deb-based installation, run

```
$ sudo dpkg -i <new installation image>
```

See https://packages.dyalog.com/ for details of updating on the Pi.

## Upgrading from an earlier version

Newer versions of Dyalog APL will be placed in new subdirectories, rather than in the same location as the currently installed versions. This means that both old and new versions can be run in parallel, but extra disk space in /opt will be required to cater for the multiple releases. Note however that once a workspace has been saved in a later version of Dyalog APL, it is most likely that it will not be possible to `)LOAD` or `)COPY` the workspace by an earlier version. Once happy with the new version, then de-install the earlier version.

# Dyalog Serial Number

If you have registered your copy of Dyalog or have a commercial licence then you will have been sent a Dyalog serial number; this serial number is individual to you and corresponds to the type of licence that you are entitled to use.

Dyalog Ltd recommends setting the serial number either by editing a file containing the serial number directly or by running a function in a Dyalog Session to update the file containing the serial number. The next time Dyalog is started after setting the serial number, the **DYALOG_SERIAL** environment variable is set to the contents of this file. However, if the **DYALOG_SERIAL** environment variable already exists and has a non-empty value, then its value is not updated.

In a multi-user environment it might be desirable to set the **DYALOG_SERIAL** environment variable in a system configuration file so that the serial number is held in a single location.

To set your Dyalog serial number by editing the serial number file directly, edit the `$HOME/.dyalog/serial`[1] text file so that it contains just the string `serialnumber`, where `serialnumber` is your Dyalog serial number.

To set your Dyalog serial number from within a Session:

```
]SE.Dyalog.Serial serialnumber
```

where **`serialnumber`** is your Dyalog serial number. This updates the value stored in the serial number file `$HOME/.dyalog/serial`. To complete the process you must exit and restart the Session.

When you start a Session, your serial number is displayed in the banner . To see your serial number at any time, enter:

```
      +2⎕NQ'.' 'GetEnvironment' 'DYALOG_SERIAL'
```

or

```
]SE.Dyalog.Serial ''
```

### NOTE:

Using or entering a serial number other than the one issued to you is not permitted.
Transferring the serial number to anyone else is not permitted.
For the full licence terms and conditions, see:
https://www.dyalog.com/uploads/documents/Terms_and_Conditions.pdf

[1] `$HOME/.dyalog/serial` is the default location for your serial number file but you can set the **DYALOG_SERIALFILE** environment variable to point to any other valid location.

# Configuring a Console/terminal Window to support Dyalog APL for UNIX

In order to support Dyalog APL for UNIX in a console/terminal window under a Linux window manager, it is necessary to install and configure the Dyalog APL keyboard support. Additionally it is possible to install the APL385 Unicode font, to be used instead of the built in fonts which include APL characters.

## Keyboard support

Dyalog submitted APL Language keyboard support to Xorg at the end of 2011; most Linux distributions released after mid-2012 have the Dyalog APL keyboard support included with the distribution. Such distributions include openSUSE 12.2, Ubuntu 12.10 and Fedora 17.

Support for the Key character was submitted to Xorg in mid-2014; if your distribution does not support this character, contact Dyalog support for assistance.

Details of how to configure the keyboard under KDE4 appear below; keyboard support for other window managers (such as Gnome and Unity) is in a state of flux. The latest information about the process of installing and configuring Dyalog APL keyboard support for such environments can be found at:

https://www.dyalog.com/forum/viewtopic.php?f=20&t=210

or by contacting Dyalog support. The same resources can be used to obtain information and guidance on installing keyboard support for earlier Linux distributions.

## Configuring the APL keyboard under KDE4

(These instructions were drawn up using openSUSE 12.2; other KDE4 environments may vary slightly)

- Select Configure Desktop
- Select Input Devices
- Select Keyboard
- Select Layouts
- Select the "Configure layouts" tickbox
- Select Add
- In the Add Layout dialog box, select the Layout "APL Keyboard Symbols", and then the "dyalog" option
- Close the Add Layout dialog box
- The list of layouts should now include APL Keyboard Symbols, with one of the dyalog variants.

- Click on "Main shortcuts" in the "Shortcuts for Switching Layout" group; where possible, Dyalog recommends selecting "Any Win key (while pressed)" so that either Windows key causes APL characters to be generated.

### APL font support

APL characters are available under Linux window managers. However some of the characters may appear inelegant; most noticeable are very small "⋄" and overly large "⌶". To resolve this, it is possible to use the Freemono fonts (these are installed by default on some distributions (such as openSUSE)), or to download and install the APL385 Unicode font. This font is freely downloadable from:

https://www.dyalog.com/apl-font-keyboard.htm

Details of how to install the font will appear in the documentation for your window manager.

## Dyalog APL, RDP and VNC

Due to the different ways that Microsoft Windows and Linux/UNIX handle keyboards, it is not possible to use RDP or VNC or X-Windows from a Windows client to control a Dyalog APL session running under a UNIX window manager. In particular, all of the X-Window clients that Dyalog is aware of do not fully support xkb key mappings.

It is possible to use VNC from a Linux client to connect to a remote Linux desktop and control an APL session running there; the keyboard support will however need to be added to the local machine.

# Using PuTTY under Windows

Dyalog APL for UNIX comes with support for the PuTTY terminal emulator. PuTTY is freely downloadable, supports ssh and telnet protocols, and supports Unicode keystrokes and fonts. To be able to generate and see APL characters it is also necessary to install the Dyalog UnicodeIME and the APL385 Unicode font.

## Downloading and installing the Dyalog UnicodeIME

The UnicodeIME can be freely downloaded from https://www.dyalog.com/apl-font-keyboard.htm. It is also included with all Unicode Windows versions of Dyalog from 13.0 onwards. There are two versions of the UnicodeIME; one for 32 bit Windows, and one for 64 bit; please ensure that the correct version is downloaded.

Details of how to install the UnicodeIME are on the download webpage.

## Downloading and installing the APL385 font

The APL385 can be freely downloaded from https://www.dyalog.com/apl-font-keyboard.htm. Details of how to install the font appear on the download webpage.

## Downloading and Installing PuTTY

PuTTY is available from https://www.chiark.greenend.org.uk/~sgtatham/putty. Full details of how to download and install PuTTY, along with the licence terms and conditions are available from the above URL.

## Configuring PuTTY to support Dyalog APL for UNIX

Firstly ensure that you are able to login to the UNIX server which has Dyalog APL installed on it. If you are using an AIX server, it is recommended that in the Keyboard category you set the backspace key to Control-H.

For APL support the follow settings are required:

*Window/Appearance Font settings/Font:* set to *APL385 Unicode*

*Window/Translation/Character set translation on received data:* set *Received data assumed to be in which character set* to *UTF-8*

You should ensure that *Terminal/Keyboard/The Backspace key* is set appropriately for the remote operating system. AIX defaults to Ctrl-h whereas most other operating systems default to Ctrl-?

Having set these values, it is recommended that you save the settings; if you will need to connect to multiple servers, it is recommended that you save the above settings as the default options (Highlight the "*Default Settings*" in *Saved Sessions* and click on *Save*).

# Configuration Parameters

Dyalog can be customised using configuration parameters. These can be set in various ways; if a configuration parameter is set in multiple places the following descending order of precedence applies:

1. command line settings
2. application configuration file settings
3. environment variable settings
4. user configuration file settings
5. built-in defaults

This provides a great deal of flexibility, enabling a user to override one setting with another. For example, a "usual" workspace size (**MAXWS**) can be defined in the user configuration file, but be temporarily superseded by entering a different value when starting a Dyalog Session from the command line.

For more information on configuration files, see *Configuration Files* on page 11. For more information on environment variables, see *Environment Variables* on page 14.

# Configuration Files

A configuration file is a text file containing configuration parameters and values. It can cascade, that is, it can extend (inherit) configuration values from other configuration files, and supplement and/or override them. Configuration files use JSON5 (a superset of standard JSON) syntax and are portable across all systems supported by Dyalog.

The key benefits of defining configuration parameters using configuration files include:

- Configuration files are text-based. They are, therefore, easily managed along with the source code for an application, using industry standard tools for source code management and continuous integration.
- Application configuration files can be placed in application folders and define the configuration settings for a specific application.
- User configuration files provide settings that are the same for all applications. Typically, these files are used to configure the development environment.
- Interpreter configuration can be performed in the same way across all supported platforms.
- Dyalog can be launched from a text file that defines a function, namespace or class. If a configuration file exists with the same name as this file (but with a *.dcfg* extension), then Dyalog will detect this on launching and use the configuration parameter settings it defines.
- Configuration files are easy to read, and can be written directly or by using `⎕JSON` (which supports JSON5).

- Both application and user configuration files can cascade, overriding settings defined in a more generic configuration file; this simplifies the configuration of components which share some configuration.

Dyalog Ltd recommends that configuration files are used for all run-time applications, and that the use of environment variables for this purpose is eliminated.

There are two different types of configuration file:

- A *user configuration file* – this defines configuration values for the current (possibly only) user of the system. The first time a new version of Dyalog is launched it creates and initialises a user configuration file called *$HOME/.dyalog/dyalog.<version-specific>.dcfg*, where the version-specific information comprises the version number, edition and width. For example, a 64-bit Unicode edition of Dyalog version 18.0 will be identified as *180U64*. The name of this file should not be changed.
- An *application configuration file* – this contains configuration values associated with a specific application. This is created by the user and should be saved at the same level as the application. It can either be given the same name as the workspace/script that is loaded when the application starts (but with the extension *.dcfg*) or the name should be stored in the CONFIGFILE parameter.

An additional configuration file called *$HOME/.dyalog/dyalog.dcfg* is also created the first time any version of Dyalog is run. This can be edited to include configuration parameter values that should always be applied irrespective of Dyalog version so that they do not have to be redefined in multiple version-specific user configuration files.

Prior to Dyalog version 18.0, configuration parameters could be specified as environment variables and set in the *$HOME/.dyalog/dyalog.config* script. This is no longer referenced, and any settings that should be retained must be re-entered in the appropriate *$HOME/.dyalog/dyalog.<version-specific>.dcfg* configuration file.

## Configuration File Structure

Configuration files define configuration parameters using JSON5. A JSON object contains data in the form of key/value pairs and other JSON objects. The keys are strings and the values are the JSON types. A key and its value are seperated by a colon (`:`) character. Entries (key/value pairs) are separated by comma (`,`) characters.

The top-level object defines an optional key called `Extend` and an optional object called `Settings`:

- `Extend` is a string value containing the name of a configuration file to import. The extended (imported) file can extend another configuration file. Configuration values from the imported file(s) can be overridden by redefining them. The file name is implicitly relative to the name of the file that imports it (any file name extension must be explicitly specified).
- `Settings` is an object containing the names of configuration parameters and their values. The values can be a string, a number or an array of strings.

The names and values correspond to configuration parameters, and names are not case sensitive. Any named values can be defined; an APL application could query the values using `+2⎕NQ'.' 'GetEnvironment' <name>` or using the `]Config` user command.

If the same name is defined multiple times within a configuration file then the first definition will be used and a warning will be generated.

## Arrays

An array can be used to define file paths, for example, `WSPATH: ["/dir1", "/dir2"]`. The only parameters which can be defined as arrays are **WSPATH**, **WSEXT** and **CFEXT**.

## References to other Configuration Parameters

Configuration parameters that are string values can include references to other configuration parameters (irrespective of where they are defined) using square bracket delimiters. For example, `MySetting: "[DYALOG]/MyFile"` will replace `[DYALOG]` with the value of the **DYALOG** configuration parameter.

If the referenced configuration parameter is not defined then no substitution will take place; the reference, including the square bracket delimiters, will remain in place.

To include literal square brackets in a string, prefix them with a `\` character.

## Nested Structures

Configuration files support nested parameter structures by defining an object that corresponds to the structure. For example:

```
Captions: {
    Session: "My Dyalog Session"
    Status: "My Status window"
}
```

```
      +2 ⎕NQ '.' 'GetEnvironment' 'Captions\Session'
My Dyalog Session
```

### Example Configuration File Content

```
{
   Extend: "my_default_configuration.dcfg",
   Settings: {
   // maximum workspace
   MAXWS: "2GB",
   WSPATH: ["/dir1", "/dir2", ""],
   UserOption: 123,
   ROOTDIR: "/my/root/directory",
   // references to other configuration parameters
   FNAME: "[rootdir]/filename",
   }
}
```

# Environment Variables

Environment variables are used to configure various aspects of Dyalog APL. The complete list appears in the *Dyalog for Microsoft Windows Installation and Configuration Guide: Configuration Parameters*; this section discusses those variables which are of particular importance to the Non-GUI versions of Dyalog APL, and lists those that have meaning to the UNIX versions. Additionally there some non-GUI-specific variables which are described below and some which either do not apply, or may not work as the user might at first expect.

Under UNIX, all environment variables should appear in UPPER CASE. For example, to set the default value of `⎕ml` to 3, then

```
$ export DEFAULT_ML=3
```

If a configuration parameter described in the *Dyalog for Microsoft Windows Installation and Configuration Guide* has a backslash "\" in its name (strictly speaking, appears in a subkey of the Dyalog key in the Windows Registry), this should be replaced with an underscore in the equivalent environment variable. This applies for example to SALT\CommandFolder.

Many of these environment variables are set in the mapl script; their values are either appropriate for the installation location of Dyalog APL, or are set to define reasonable default values.

The environment variables are broken down into several tables:

- Table E1: The most commonly defined and used for non-GUI versions of Dyalog APL under UNIX. Most of these variables are essential for a usable APL session
- Table E2: Variables used to control default values in the workspace
- Table E3: Variables used to configure the Session
- Table E4: Miscellaneous Variables used by non-GUI Dyalog APL
- Table E5: Editor-related environment variables
- Table E6: Tracer-related environment variables
- Table E7: RIDE-related environment variables
- Table E8: SALT and User Command-related environment variables

**Table E1: Commonly used Variables**

| Variable | Notes |
|---|---|
| TERM<br>APLK<br>APLK0<br>APLT<br>APLTn | Define the input and output translate tables used by Dyalog APL. The values of APLK0 and APLTn override the values of APLK and APLT if set, and they in turn override the value of (Unicode) *default*, or (Classic) TERM if set.<br><br>APLK is for input translation, APLT for output translation.<br><br>These are used in conjunction with .. |
| APLKEYS<br>APLTTRANS | Define the search path for the input and output translate tables respectively. If unset, the interpreter will default to $DYALOG; if $DYALOG too is not set, will default to /usr/dyalog. |
| APLNID | This variable is ignored by the UNIX versions of Dyalog APL: `⎕ai` and `⎕an` pick up their values from the user's uid and /etc/passwd. |

| Variable | Notes |
|---|---|
| APLSTATUSFD | If set, this defines the stream number on which all messages for the Status Window appear. It is then possible to redirect this output when APL is started.<br>If unset, the output will appear in the same terminal window as the APL session, although it is not part of the session; such output can be removed by hitting SR (Screen Redraw - often defined to be Ctrl-L). |
| DYALOG_NETCORE | This parameter is a Boolean value with a default value of 1. If set to 0, it disables the .NET interface. |
| DYALOG_SERIAL | This parameter contains your Dyalog serial number. This must be set to the serial number issued to you. If not set, then the software is unregistered. For the full licence terms and conditions, see https://www.dyalog.com/uploads/documents/Terms_and_Conditions.pdf. |
| DYALOG_SERIALFILE | This parameter specifies the full path to the text file containing your Dyalog serial number. |
| ENABLE_CEF | This parameter is a Boolean value with a default value of 1. If set to 0, it disables the Chromium Embedded Framework (CEF)[1]. and at attempt to create an HTMLRenderer object will fail with an error meessage. See Note (below). |
| ERRORONEXTERNALEXCEPTION | By default, any error when calling ⎕NA will result in APL terminating; if `ERRORONEXTERNALEXCEPTION` is set to 1, then APL will instead generate an event 91: `EXTERNAL DLL EXCEPTION`. Be aware however that the workspace may become corrupted. This is best used when developing ⎕NA code rather than in production. |

[1] https://en.wikipedia.org/wiki/Chromium_Embedded_Framework

| Variable | Notes |
|---|---|
| LIBPATH | A suitable entry for the Conga libraries needs to be added to the LIBPATH variable if Conga is to be used. For more information see the Conga Guide. |
| MAXWS | Defines the size of the workspace that will be presented to the user when Dyalog APL is started. A simple integer value will be treated as being in KB. K, M and G can be appended to the value to indicate KiB, MiB and GiB (binary) respectively. If unset, the default value is 256M. |
| WSPATH | Defines the search path for both workspaces and Auxiliary processors.<br><br>If unset, there is no default value. Workspaces and APs that are not on the WSPATH can be accessed using absolute or relative pathnames. |

## Note

Currently the value of the **Enable_CEF** parameter defined in the Windows Registry or in a Configuration file is ignored. Only the value set in the command line or as an environment variable is honoured. If not defined in this way, the default value is used.

Under macOS and Linux, if the configuration parameter **ENABLE_CEF** is 1, Auxiliary Processors cannot be used (they hang on error). The default value is 1 unless you are not running under a desktop (for example, you are running Dyalog in a PuTTY session when the default is 0).

**Table E2: Default workspace values**

| Variable | Notes |
|---|---|
| DEFAULT_DIV | Default value for `⎕div` in a clear workspace. |
| DEFAULT_IO | Default value for `⎕io` in a clear workspace. |
| DEFAULT_ML | Default value for `⎕ml` in a clear workspace. |
| DEFAULT_PP | Default value for `⎕pp` in a clear workspace. |
| AUTO_PW<br>DEFAULT_PW | `⎕pw` is set by the interpreter when it starts, or when the session window is resized. Under UNIX if the terminal window is resized, the session will be resized when the interpreter next checks for input. |
| DEFAULT_RTL | Default value for `⎕rtl` in a clear workspace. |
| DEFAULT_WX | Default value for `⎕wx` in a clear workspace.<br><br>Note that although the UNIX versions of Dyalog APL do not have GUI objects, `⎕se` is present, and the value of `⎕wx` will affect the programmer's ability to run expressions such as `⎕se.PropList`. |

For numeric values, the interpreter takes the value of the environment variable, and prepends a "0" to that string. It then parses the string, accepting characters until the first non-digit character is reached.

This string, now of digits only, is converted into an integer. If the resulting value is valid, then that is the value that will be used in the workspace. If the resulting value is invalid, then the default value will be used instead.

**Table E3: Variables used to configure the Session.**

| Variable | Notes |
|---|---|
| DYALOGLINK | Specifies the directory for Link |
| DYALOGSTARTUPSE | Specifies one or more *Session initialisation* directories that contain APL code to be installed in `⎕SE` |
| DYALOGSTART_X | Specifies whether the `Run` function is executed during Session startup |
| DYALOG_GUTTER_ ENABLE | Enable or disable Session Gutter |
| HISTORY_SIZE | The size of the prior line buffer |
| INPUT_SIZE | The size of the buffer used to store lines marked for execution |
| LOG_FILE<br>LOG_FILE_INUSE<br>LOG_SIZE | These three variables determine the name of the session log file (default ~/.dyalog/session_log_ <DyalogMajor><DyalogMinor><U\|C><bits>_*.dlf, for example, ~/.dyalog/session_log_190U64_*.dlf), whether a log file is created or not, and the size of the log file in KB. Be aware: the session log file is not interchangeable between the different editions and widths of APL; in a mixed environment it is strongly recommended to use a different log file for each version. |
| PFKEY_SIZE | The size of the buffer used to hold `⎕pfkey` definitions: if this is too small, an attempt to add a new definition will result in a LIMIT ERROR. |
| SESSION_FILE | Defines the location of your session file; session file support was added in Dyalog 13.1. The default value is $DYALOG/default.dse |

To set values, use K to indicate KB. Note that the buffers will contain other information, so the buffer size will not be exact. Note also that multibyte Unicode characters will take up more space than single byte characters, and that 32 and 64 bit versions of Dyalog APL can require different amounts of space for holding the same information.

Example:

```
$ HISTORY_SIZE=4K my_apl_startup_script
```

**Table E4: Miscellaneous Variables used by non-GUI Dyalog APL**

| Variable | Notes |
|---|---|
| APL_ TEXTINAPLCORE | If set with the value 1 the "Interesting Information" section is included in an aplcore file. Otherwise this section is omitted. By default the interpreter has this set to 0; it is the default APL script which sets it to 1. |
| AUTOFORMAT TABSTOPS | If AUTOFORMAT is 1, then control structures will be shown with indents, set at TABSTOPS spaces; the changes are reflected in the editor window when the RD (ReDraw) command key is hit. |
| AUTOINDENT | If AUTOINDENT is set to 1, then if a line is added it is indented the same as the previous line. |
| AUTO_PW | Introduced in 13.0. With `AUTO_PW=0`, `⎕pw` remains fixed at the size of the terminal window when APL was started. When set to 1, or unset, `⎕pw` alters each time the terminal window is resized. |
| DYALOG | This variable is defined in the supplied mapl startup script, and is used to form the default values for APLKEYS, APLTRANS, WSPATH etc.<br><br>If it is necessary to identify the location of the Dyalog executable, then a more reliable method is to determine the full path name from the appropriate file in the /proc/<process_id_of_APL_session>/ subdirectory or from the output of ps. |

These are the remaining variables listed in the *Dyalog for Microsoft Windows Installation and Configuration Guide* which are effective in the non-GUI UNIX versions of Dyalog APL

**Table E5: Editor-related environment variables**

| Variable | Notes |
|---|---|
| EDITOR_ COLUMNS_* | See *Configuring the Editor* on page 22. Can be one of<br>EDITOR_COLUMNS_CHARACTER_ARRAY<br>EDITOR_COLUMNS_CLASS<br>EDITOR_COLUMNS_FUNCTION<br>EDITOR_COLUMNS_NAMESPACE<br>EDITOR_COLUMNS_NUMERIC_ARRAY |
| DYALOG_ DISCARD_FN_ SOURCE | Specifies whether source code is retained in the workspace |

**Table E6:Tracer-related environment variables**

| Variable | Notes |
|---|---|
| TRACE_ON_ ERROR | With this is set to 1 (the default) the tracer is opened if an untrapped error occurs. |

**Table E7:RIDE-related environment variables**

| Variable | Notes |
|---|---|
| RIDE_INIT | Enables and configures RIDE; see the *RIDE User Guide* for more information. |

**Table E8: SALT and user commands related environment variables**

| Variable | Notes |
|---|---|
| SESSION_FILE | Specifies the location of the file containing ⎕SE. The default value is $DYALOG/default.dse |
| UCMDCACHEFILE | Specifies the location of the user command cache file. Defaults to "`UserCommand{UcmdMajor}{UcmdMinor}.{DyalogMajor}{DyalogMinor}{U\|C}{bits}.cache`" e.g. `UserCommand25.182U64.cache` in the `dyalog` directory. |

Further information about SALT and user commands appear in the *User Commands User Guide* and the *SALT User Guide*.

# Configuring the Editor

The editor in non-GUI versions of Dyalog APL can be considered to have 5 separate functional columns. Below is the contents of the editor window, which shows the namespace ns, which has two traditional-style functions and one dfn. The statement `5 ⎕STOP 'ns.fn1'` has been run too:

```
[0]          :Namespace ns
[1]  [0]    ├    ∇ r←fn1 a
[2]  [1]    ├      :If a=1
[3]  [2]    │          r←1
[4]  [3]    │      :Else
[5]  [4]    ├          :If today≡'Friday'
[6]  [5]  o │              r←2
[7]  [6]    ├          :EndIf
[8]  [7]    ├      :EndIf
[9]  [8]    ├    ∇
[10]
[11] [0]           dfn←{α+ω}
[12]
[13] [0]    ├    ∇ r←a fn2 w
[14] [1]    │      r←a+w
[15] [2]    ├    ∇
[16]         :EndNamespace
```

This is formed of 5 separate columns:

| C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|
| [0] | | | | :Namespace ns |
| [1] | [0] | | ├ | ∇ r←fn1 a |
| [2] | [1] | | ├ | :If a=1 |
| [3] | [2] | | │ | r←1 |
| [4] | [3] | | │ | :Else |
| [5] | [4] | | ├ | :If today≡'Friday' |
| [6] | [5] | o | │ | r←2 |
| [7] | [6] | | ├ | :EndIf |
| [8] | [7] | | ├ | :EndIf |
| [9] | [8] | | ├ | ∇ |
| [10] | | | | |
| [11] | [0] | | | dfn←{α+ω} |
| [12] | | | | |
| [13] | [0] | | ├ | ∇ r←a fn2 w |
| [14] | [1] | | │ | r←a+w |
| [15] | [2] | | ├ | ∇ |
| [16] | | | | :EndNamespace |

| Functional Column | Value (see below) | Purpose |
|---|---|---|
| C1 | 4 | Line numbers for entire object |
| C2 | 64 | Line numbers for functions etc. within scripted namespaces |
| C3 | 2 | Trace/Stop points |
| C4 | 8 | Control Structure Outlining |
| C5 | 16 | Text (or content)This value is ignored; this column is always present |

It is possible to control at startup time which of these columns are visible. By default, for all types of object, only the text column is visible; this can be overridden on a per-object basis by setting one or more of the EDITOR_COLUMNS_ variables listed in Table E5. The value of these variables is the sum of the values for each of the columns which are desired.

## Examples:

EDITOR_COLUMNS_NAMESPACE=94 shows all columns (the first example in this section)

Various values for EDITOR_COLUMNS_FUNCTION

| Value | Editor window appearance |
|---|---|
| 0 | `fn1 a`<br>`:If a=1`<br>`  b←2`<br>`:EndIf` |
| 22 | `[0]  fn1 a`<br>`[1]  :If a=1`<br>`[2] o  b←2`<br>`[3]  :EndIf` |
| 26 | `  fn1 a`<br>`  ├ :If a=1`<br>`o│  b←2`<br>`  ├ :EndIf` |
| 40 | `[0]  fn1 a`<br>`[1]  ├ :If a=1`<br>`[2] o│  b←2`<br>`[3]  ├ :EndIf` |

# Miscellaneous

## Running from scripts

Dyalog APL can be run with input being directed from a script file, and output being redirected as well.

The script file needs to be built in such a way that it contains valid input according to the input translate table that is defined in the APLK variable.

The classic edition of Dyalog APL expects that the input script by default uses Ctrl-O and Ctrl-N to swap between APL and ASCII characters, and Ctrl-H is used to create overstrikes. Be aware that when editing such an input file, cut and paste of ^H, ^N or ^O may well result in the two character sequences being copied, rather than the single character Ctrl-H, Ctrl-N and Ctrl-O.

The Unicode edition by default expects that the input file has Unicode characters in it; a Unicode-aware editor is therefore required. Note however that applications such as Notepad will add BOMs (Byte Order Markers) to the Unicode text; these must be removed as the Dyalog APL input translate table does not have BOMs defined in it.

The example below shows the same set of APL expressions as they would appear in a script file for classic and Unicode editions: it is rather easier to read the Unicode edition's input !

### Classic example:

```
^O(2^NLnqK.K K^OGetBuildID^NK^O),
(^NK.KLwgK^OAPLVersion^NK^O)
^Ovar^N[1+1 J^HC^O Check input from file: Classic
)si
^N"si
^Nloff
```

### Unicode example:

```
(+2⎕nq'.'  'GetBuildID'),('.'⎕wg'APLVersion')
var←1÷1 ⍝ Check input from file: Unicode
)si
)si
⎕off
```

# The file command and magic

All Dyalog APL binary files have a unique magic number: the first byte is always 0xAA (decimal 170), and the second identifies the type of Dyalog file. Additional bytes may in some cases be used to further identify the type, version and state of the file. UNIX systems include the `file` command which use the information in the magic file to describe the contents of files.

## magic and AIX

AIX still uses a very early version of `magic`, so it is not possible to give as much information about Dyalog APL files as on Linux.

Dyalog provides a file, `magic`, which is located in the top level installation directory of Dyalog APL. To use this file to extend the capabilities of the `file` command either run

```
file -m /opt/mdyalog/19.0/32/classic/p5/magic *
```

or catenate the contents of /opt/mdyalog/19.0/32/classic/p5/magic onto /etc/magic, and then run

```
file *
```

## Example:

```
$ file -m /opt/mdyalog/19.0/32/classic/p6/magic *
1_apl_j1: Dyalog APL component file 64-bit level 1 journaled
non-checksummed
1_apl_j2: Dyalog APL component file 64-bit level 2 journaled
checksummed
1_apl_qfile: Dyalog APL component file 64-bit non-journaled non-
checksummed
1_big1: Dyalog APL component file 64-bit level 2 journaled
checksummed
1_big2: Dyalog APL component file 64-bit level 1 journaled
checksummed
apl64u: Dyalog APL workspace type 12 subtype 4 64-bit unicode
big-endian
aplout: Dyalog APL workspace type 12 subtype 0 32-bit classic
little-endian
aplcore: Dyalog APL workspace type 12 subtype 4 32-bit classic
little-endian
colours: Dyalog APL workspace type 12 subtype 4 32-bit classic
little-endian
core: data or International Language text
signals: Dyalog APL workspace type 12 subtype 4 32-bit classic
little-endian
utf8: Dyalog APL workspace type 12 subtype 4 32-bit unicode
little-endian
```

## magic and Linux

Most Linux distributions include details about Dyalog-related files in their magic files; Dyalog has submitted two versions of the magic file for inclusion in distributions. To check whether your Linux distribution has the more recent version, create a journaled component file and then run the file command against that component file. The two examples below show the output with the earlier and later versions of magic in use.

# Example, using the older default magic file:

```
$ file *
1_apl_j1: data
1_apl_j2: data
1_apl_qfile: data
1_big1: data
1_big2: data
apl64u: \012- Dyalog APL\012- workspace\012- version 12\012- .4
aplout: \012- Dyalog APL\012- workspace\012- version 12\012- .0
aplcore: \012- Dyalog APL\012- workspace\012- version 12\012- .4
colours: \012- Dyalog APL\012- workspace\012- version 12\012- .4
core: ELF 32-bit LSB core file Intel 80386, version 1 (SYSV),
SVR4-style
signals: \012- Dyalog APL\012- workspace\012- version 12\012- .4
utf8: \012- Dyalog APL\012- workspace\012- version 12\012- .4
```

# Example, with more recent magic file:

```
$ file *
1_apl_j1: Dyalog APL component file 64-bit level 1 journaled
non-checksummed
1_apl_j2: Dyalog APL component file 64-bit level 2 journaled
checksummed
1_apl_qfile: Dyalog APL component file 64-bit non-journaled non-
checksummed
1_big1: Dyalog APL component file 64-bit level 2 journaled
checksummed
1_big2: Dyalog APL component file 64-bit level 1 journaled
checksummed
apl64u: Dyalog APL workspace type 12 subtype 4 64-bit unicode
big-endian
aplout: Dyalog APL workspace type 12 subtype 0 32-bit classic
little-endian
aplcore: Dyalog APL workspace type 12 subtype 4 32-bit classic
little-endian
colours: Dyalog APL workspace type 12 subtype 4 32-bit classic
little-endian
core: ELF 32-bit LSB core file Intel 80386, version 1 (SYSV),
SVR4-style, from '/opt/mdyalog/14.0/32/classic/dyalog'
```

```
signals: Dyalog APL workspace type 12 subtype 4 32-bit classic
little-endian
utf8: Dyalog APL workspace type 12 subtype 4 32-bit unicode
little-endian
```

The most recent version of the magic file can be found in the top level of the installation directory; see the man page for the file command for details of how to update the system magic file, or use the syntax described in the /etc/magic and AIX section above to override the default magic file with the one supplied in the installation directory.

# ⎕SE, User Commands and SALT

## Summary

Support for user commands is included in non-Windows versions of Dyalog APL. Many of the user commands which were originally written for running under Microsoft Windows will run under the various flavours of UNIX.

Under UNIX there is no autocompletion of user command names.

The SALT code resides in ⎕SE, which is saved in a session file. The location of the session file is controlled by the environment variable SESSION_FILE; by default this file is $DYALOG/default.dse. Setting SESSION_FILE=/dev/null results in an empty ⎕SE and SALT being disabled.

See the *User Commands User Guide* and the *SALT User Guide* for more information.

## Caching User Command information

When a Dyalog APL session is started, SALT is loaded, and checks the details of all of the files which contain user commands with a previously cached version of this information. If Dyalog APL has never been run before, or the cache file does not exist, SALT rebuilds the cache file. This can take a few seconds, especially on the Raspberry Pi.

By default the cache file is called $HOME/.dyalog/UserCommand20.cache.

This can be overridden by specifying the environment variable UCMDCACHEFILE.

It is expected that the structure of files in ~/.dyalog will change in future versions of Dyalog APL.

## Assigning Contents of Session Log

It is possible to assign the contents of the Session Log to a variable:

```
z←'⎕se'⎕wg'Log'
```

# The Directory ~/.dyalog

In Version 19.0 Dyalog APL by default creates a directory to hold various configuration and log files; in previous versions these files were located in differing directories. The contents of this directory are expected to be extended in future versions of Dyalog APL, and allow for multiple versions and editions of Dyalog APL to be run concurrently.

On Linux (including Raspberry Pi), macOS and AIX this directory is called .dyalog and is located in the user's home directory.

The default Dyalog startup script checks for the existence of the directory, and if it does not exist, creates it.

This directory now contains:

- the user configuration files *dyalog.dcfg* and *dyalog.<version-specific>.dcfg*, where the version-specific information comprises the version number, edition and width. For example, a 64-bit Unicode edition of Dyalog version 18.0 will be identified as *180U64*. The names/locations of these files should not be changed.
- the session log, which by default is called *default.dlf*
- the user command cache file, which by default is called *UserCommand20.cache*
- the file containing the SALT settings which is called *SALT.settings*

Note that many of the default names and locations can be altered. Remember that earlier versions of Dyalog will generate/use copies of these files in other locations: you may need to move or delete earlier versions of these files, or change the default values of their names and/or locations in earlier versions of Dyalog APL.

# ⎕NA under UNIX

## Introduction

⎕NA is fully supported under UNIX; the Conga communications package for example is a shared library on all platforms.

⎕NA supports user-written shared libraries and also system-supplied shared libraries. Dyalog APL under UNIX is supplied with a shared library, dyalog32.so or dyalog64.so which contains the same functions as the DLLs which are described in the ⎕NA documentation in the *Dyalog Language Reference Guide*. Additionally, the function `getlasterror` is included; this returns the error code at the point when the called function failed (which may be different from its value at the point where a previous error occurred).

It is necessary to specify the complete name of the file containing the shared library, no extension is added by Dyalog APL.

When developing code using ⎕NA it may be useful to set the environment variable ERRORONEXTERNALEXCEPTION= 1. When this is set, Dyalog APL will generate an event 91, `EXTERNAL DLL EXCEPTION` rather than a syserror should a call on a functions defined by ⎕NA be ill-specified. It should be noted however that the workspace may become corrupt, so it is not recommended to run in production with this variable set.

## System Shared Libraries

On AIX many system library functions appear in libc.a.

When calling system shared libraries under AIX, you must refer to them as:

64-bit: libc.a(shr_64.o)

32-bit: libc.a(shr.o)

It is not always possible to access all library functions - on AIX for example it is not possible to access memcpy() or strncpy(). it is for this reason that dyalog*.so includes MEMCPY and STRNCPY.

On Linux, it is a little more difficult to location the libc.so file; the function `libc` in the supplied workspace quadna (which contains two namespaces, `Windows` and `NonWindows`)can be used to locate this file.

### Definitions

In the remainder of this section references are made to the APL variables `sharedlib` and `dyalib`; the definitions for both vary between AIX and Linux, and between 32 and 64 bit interpreters.

Under AIX, `sharedlib` is defined as:

```
sharedlib←'libc.a(shr_64.o)' ⍝ 64 bit
sharedlib←'libc.a(shr.o)'    ⍝ 32 bit
```

Under Linux, it is necessary to identify the shared library:

```
)copy quadna NonWindows.libc
sharedlib←libc ⍬
```

For all UNIX platforms, the dyalog shared library is identified as

```
dyalib←'dyalog64.so'         ⍝ 64 bit
dyalib←'dyalog32.so'         ⍝ 32 bit
```

## Example 1

getpid() is common to all UNIX platforms; it returns an int which is the process ID of the current process. It is defined to be

pid_t getpid(void)

where pid_t is a 4-byte integer.

The APL code to instantiate this function is

```
⎕na 'I4 ',sharedlib,'|getpid'
```

## Example 2

This is a slightly more complex example, which uses the STRNCPY function in the Dyalog-supplied shared library to retrieve the value of a variable which is referenced by a pointer, returned from the system library function:

getenv()returns a pointer to the value of the environment variable which is the argument of the function. It is defined to be

char *getenv(const char *name)

```
∇r←GetEnv envvar;getenv;P;get
 r←''
 ⎕NA'P ',sharedlib,'|getenv <0T1[]'
 'get'⎕NA dyalib,'|STRNCPY >0U1[] P U4'
 P←getenv⊂'UTF-8'⎕UCS ⎕UCS envvar
 →0⍴⍨P=0
 r←'UTF-8'⎕ucs get 4096 P 4096
∇

      GetEnv'MAXWS'
4G
```

Note: the call to STRNCPY has been defined to return a vector of integers so that the result can be passed directly to ⎕UCS.

## geterrno

The dyalog shared libary under UNIX includes the function `geterrno`. This returns the current value of errno; be aware that it may not have the same value as at the point when the error was raised. To use this function:

```
      ⎕na 'I ',dyalib,'|geterrno'
      geterrno
5
```

## Shared libraries and APL threads

Any shared library function must mask out all signals for new threads which it creates. Failure to do so will result in a catastrophic failure of APL's signal handling.

# Session logfile

By default the session logfile is called default.dlf. By default this file is created as ~/.dyalog/default.dlf on Linux, AIX and macOS, and in ~/.config/dyalog/default.dlf on the Pi. This can be overridden by setting the environment variable **LOGFILE**.

# Status window output

By default under UNIX what would appear in the status window in the GUI versions appears in the same terminal window as the APL session, but the text is not part of the session. If such text appears, the APL session can be redrawn using the SR command, thus removing the status window text.

It is possible to redirect the status window output; to do so select an unused stream number as the stream have the status window output appear on, and then redirect that stream. Note that it will be necessary to associate a valid output translate table (usually apltrans/file) with that stream.

Example:

```
$ export APLSTATUSFD=9
$ export APLT9=file
$ mapl 9>/dev/null
```

More useful may be to redirect the status window output into a file, and in another terminal window run `tail -f` on that file.

# BuildID

Each interpreter has its own unique BuildID. This is a 32-bit checksum of the program file which is the Dyalog APL interpreter. This checksum allows Dyalog Ltd. support staff to uniquely identify the interpreter and from that determine the version, edition, platform etc. of the interpreter.

For that reason, Dyalog Ltd. support staff ask that whenever an issue is raised with them that the BuildID is included in all communications.

The BuildID is included in binary form in any aplcore that is generated; if a core file is created, then is it possible to identify the BuildID using the following command:

```
$ strings -a -n 14 core | grep "BuildID="
```

Additionally, the BuildID is included in the "Interesting Information" section of aplcore files provided that the environment variable APL_TEXTINAPLCORE is set to 1.

The BuildID can be identified both from within the interpreter (using the GetBuildID method), and also from the BuildID executable which is supplied with the product on UNIX.

Both of these methods can be used for any file; they are useful and very fast ways of keeping track of workspaces versions etc. although md5sum and others may be more appropriate.

### Examples:

At the command line:

```
$ cd /opt/mdyalog/12.1/32/classic/p6
$ ./BuildID dyalog
70a3446e
$ ./BuildID magic
0a744663
```

In APL:

```
      +2 ⎕nq '.' 'GetbuildID'
70a3446e
      magicfile←'/opt/mdyalog/12.1/32/classic/p6/magic'
      +2 ⎕nq '.' 'GetBuildID' magicfile
0a744663
      )sh
$ echo $PPID
$ kill -11 $PPID
/opt/mdyalog/12.1/32/classic/p6/mapl[58]: 274434
Segmentation fault(coredump)
$ strings -a -n14 core | grep BuildID=
BuildID=70a3446e
```

# Index