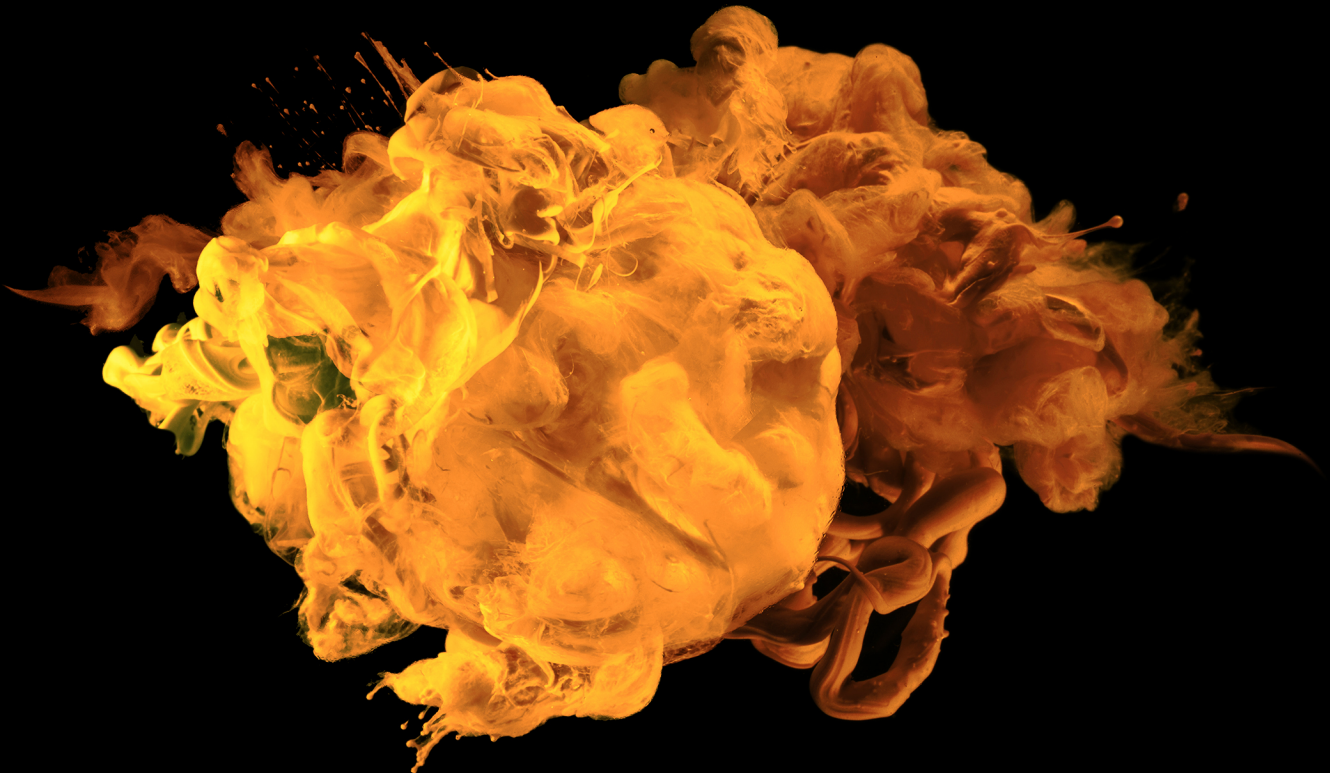


# Dyalog for UNIX UI Guide

Dyalog version **18.0**



# DYALOG

The tool of thought for software solutions

*Dyalog is a trademark of Dyalog Limited  
Copyright © 1982-2020 by Dyalog Limited  
All rights reserved.*

Dyalog for UNIX UI Guide

Dyalog version 18.0  
Document Revision: 20240625\_180

Unless stated otherwise, all examples in this document assume that `IO ML ← 1`

*No part of this publication may be reproduced in any form by any means without the prior written permission of Dyalog Limited.*

*Dyalog Limited makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Dyalog Limited reserves the right to revise this publication without notification.*

*email: [support@dyalog.com](mailto:support@dyalog.com)  
<https://www.dyalog.com>*

#### **TRADEMARKS:**

*SQAPL is copyright of Insight Systems ApS.*

*Array Editor is copyright of davidliebtag.com*

*Raspberry Pi is a trademark of the Raspberry Pi Foundation.*

*Oracle®, Javascript™ and Java™ are registered trademarks of Oracle and/or its affiliates.*

*UNIX® is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.*

*Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.*

*Windows® is a registered trademark of Microsoft Corporation in the United States and other countries.*

*macOS® and OS X® (operating system software) are trademarks of Apple Inc., registered in the U.S. and other countries.*

*All other trademarks and copyrights are acknowledged.*

---

# Contents

Overview .....	1
Entering Characters .....	3
Entering the Stencil and Nest Characters .....	3
Entering Commands .....	5
The Different Types of Input Windows .....	6
The Session window .....	6
Edit windows .....	6
Trace windows .....	6
SM (Screen manager) window .....	6
Driving the Dyalog APL tty version .....	7
Starting APL .....	14
Configuring the Editor .....	17
File permissions and FSTAC .....	19
Signals and TRAP, 4007 .....	20
SH, exit codes and stderr .....	22
SH and starting jobs in background .....	23
BuildID .....	24
Core and aplcore files .....	26
Appendix A: Table of keycodes and keystrokes using a terminal emulator under Linux GUIs .....	27
Appendix B: Table of keycodes and keystrokes for PuTTY .....	30
Appendix C: Unused keycodes .....	33
<b>Index .....</b>	<b>35</b>



---

# Overview

Dyalog APL was originally written for use with serially attached character based terminals, which had a fixed-sized viewing window, and a limited number of keystrokes.

This tty version is now usually run using either a terminal window in a GUI-based windows manager, or a terminal emulation application such as PuTTY. Although these allow for a greater range of keystrokes, and for the resizing of the terminal window in which Dyalog APL is running, they still emulate the original ASCII terminals, so the same techniques for controlling the display still apply.

**Note:** *From Version 14.1 onwards the Dyalog RIDE (Remote-IDE) can be used as the front-end for Dyalog running on any platform; the RIDE itself is currently supported on Windows, Linux and macOS/Mac OS X. For Mac OS RIDE is the default front-end for Dyalog, and is documented in the Mac OS-Specific documentation on <https://docs.dyalog.com>. RIDE can be downloaded from <https://my.dyalog.com>. **Be aware that RIDE 2.0 works only with version 14 interpreters, and that RIDE 3 works only with version 15.0 interpreters onwards.** Dyalog intends that this will be the last time that such an incompatibility will be introduced.*

It is possible to support most terminals or terminal emulators with the Dyalog APL tty version, and it is possible for any user to define their own input translate table so that the keystrokes to enter commands or characters can be unique to their environment (similarly the output translate table defines the colour scheme etc.). As such, this document does not in general refer to the actual keystrokes which are used to control Dyalog APL, but rather the keycodes to which keystrokes are mapped.

Indeed, much of the interface to Dyalog APL can be customised; this manual is written assuming that no changes have been made to the default configuration.

Appendix A lists the mapping between keystrokes and keycodes for all commands used when running under a terminal emulator/console under Linux; Appendix B lists the keystrokes and keycodes used when running PuTTY, a windows terminal emulator. Some keycodes are not relevant to the current tty versions of Dyalog APL; they may have been used in previous tty versions, or used in versions no longer supported, or are used in GUI-based versions of Dyalog APL. They are listed for completeness in Appendix C, but attempting to make use of them may lead to unexpected and/or undesirable results.

The keyboard is used for two purposes: to enter text and to enter commands. In classic editions text is limited to characters defined in `⎕AV`, in Unicode editions text can consist of any valid Unicode character. The main issue which has to be resolved is how to locate these characters and commands on the keyboard in such a way that they can be entered in a consistent manner, and without conflicts with other characters or functionality.

Given that the number of different characters and commands far exceeds the number of keys on a keyboard, different methods are supported for allowing one key to be used for more than one character or command. There are three methods that can be used, and that can be combined:

1. Use a **metakey** with the keystroke. The metakey is held down at the same time as the key to be pressed. Examples of metakeys are the Shift key, the Control (Ctrl) key and the Windows Key (WindowsKey).
2. Define multiple modes for the keyboard. Certain keystrokes are reserved for swapping between modes; the effect of hitting any other key differs depending on the current mode. This was extensively used for earlier versions of Dyalog APL, which used Ctrl-o and Ctrl-n to swap between ASCII and APL entry modes.
3. Define multiple temporary modes for the keyboard, those modes last for one keystroke only. This is used for entering many commands in the tty version.

## Entering Characters

It is necessary to select a metakey which is to be used to enter characters. In this document this metakey is represented by the string "APL". In a terminal window under a Linux GUI Dyalog recommends using the Windows key as the metakey to generate APL characters; with PuTTY and the Unicode IME the Control key is used (similarly to the Windows Unicode edition of Dyalog APL). So for example, in a terminal window `<WindowsKey><a>` generates an  $\alpha$ ; when using PuTTY the same APL character is entered by using `<Ctrl><a>`. [Note that under PuTTY, `Ctrl-xcv` are reserved for the operating system; we shall see later that `Ctrl-x` is used for another purpose. Rather than `<Ctrl>xcv` you must use `<Shift+Ctrl>xcv`.]

Linux Window managers are in generally in a state of flux, so it is best to look at the following article on the Dyalog Forum for the latest information about keyboard configuration:

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

## Entering the Stencil and Nest Characters

As of April 2017 it is not possible by default to enter the Stencil or Nest character as a single key-chord under windows managers under Linux; the updated keyboard mapping file is not yet included in Linux distributions.

Dyalog expects that future Linux distributions will have an updated mapping file, but until that time, and for existing versions of Linux distributions the methods available are:

- Update the mapping file. See below for more details
- Define the *Compose* key and enter *Stencil* by pressing *Compose Quad Diamond* and *Nest* by pressing *Compose Enclose Underbar*
- In APL, use *APL i* to swap into overstrike mode, and for *Stencil* enter *Quad <Cursor left> Diamond* for or *Diamond <Cursor Left> Quad* and use *APL i* to swap back to insert mode. For *Nest* use *Enclose <Cursor left> Underbar* or *Underbar <Cursor left> Enclose*

To update the mapping file, edit `/usr/share/X11/xkb/symbols/apl`:

- Search for the text `xkb_symbols "dyalog_base"`
- Look for the line
- `key <AB01> { [ U2282 ] }; // subset of`
- and replace with
- `key <AB01> { [ U2282, U2286 ] }; // subset of, enclose if simple`
- Look for the line
- `key <TLDE> { [ U22c4 ] }; // diamond`
- and replace with
- `key <TLDE> { [ U22c4, U233a ] }; // diamond, quad diamond`

Be aware that there are multiple occurrences of AB01 and TLDE; please ensure that you are editing the Dyalog APL section !

Logout and log back in again.

Be aware that these changes may be lost if you update the operating system.



# Entering Commands

Commands are either entered using the keys on the keyboard in conjunction with 0 or more metakeys, or when using the keyboard in different modes. A separate keystroke is used to move from one mode to the next; by default this is defined to be Ctrl-x. When Dyalog APL is started, you are in mode 0. With the exception of Move/Resize in the editor/tracer, all mode changes are effective for one keystroke only.

## Example:

- assume that you have just started APL
- assume that the WindowsKey is used to enter APL characters
- <> represents one keystroke, so <Ctrl+x><p> means: hit Ctrl+x then p

Keystrokes entered	How described in documentation	Outcome in Dyalog APL session
<p>	p	p appears in the session
<Shift+p>	P	P appears in the session
<WindowKey+p>	APL p	* appears in the session
<WindowsKey+Shift+p>	APL P	* appears in the session
<Ctrl+x><p>	Cmd-p	No noticeable effect. This is the command "Previous" (PV) used for search/replace. Note how Nrm in status line changes to Cmd when Ctrl-x is hit and then back to Nrm when the p is hit.
<Ctrl+x><Ctrl+x><p>	CMD-p	No noticeable effect. This is the command "Paste" (PT). Note how Nrm in status line changes to Cmd when Ctrl-x is hit, and then changes to CMD when Ctrl-x hit again, and then back to Nrm when the p is hit.
<Ctrl+x><g>	N/A	Nothing; this is an invalid character in Cmd mode. Note how Nrm in status line changes to Cmd when Ctrl-x is hit, and then back to Nrm when the g is hit.

**Notes:**

1. the words "Nrm", "Cmd" and "CMD" are configurable.
2. in this example each mode is temporary, lasting for only one subsequent keystroke.

## The Different Types of Input Windows

The tty version of Dyalog APL comprises of four different types of window:

### The Session window

There is one and only one session window. It is always present (although may be obscured by other windows. It cannot be resized from within APL (the terminal window or PuTTY session can be resized, and APL will respond to the resize event. Note that the contents of the window, including the status bar, may not correctly update until input is next received by the interpreter).

### Edit windows

Multiple edit windows can be open at any time, each on a separate object. The contents of edit windows can be altered, and these windows can be resized using the Move/Resize (MR) command.

### Trace windows

Multiple trace windows can be open at any time, one for each item on the stack. These windows are read-only, but these windows can be resized using the Move/Resize (MR) command.

### □SM (Screen manager) window

There can be only one □SM window; it exists only when □SM is not empty, and becomes visible either when waiting for user input (using □SR) or can be toggled to using the HotKey (HK) command.

---

## Driving the Dyalog APL tty version

The session window always occupies the whole of the APL "screen"; it may however be obscured by other windows. The session shows the expressions that have been entered, along with any output generated by those expressions. History cannot in general be altered or deleted; it is possible to alter lines in the history, but when Enter (ER) is hit, the altered line is added to the bottom of the history, and the altered line is reset to its original state.

The bottom line of the APL window is reserved for the status line. The status line is considered at all times to be 79 characters wide. It is divided into several fields, whose widths are fixed:

- The string "Search:"
- The current search string
- The string "Replace:"
- The current replace string
- The latest error message (is removed on next keystroke)
- The "name" field: this may contain the name of the workspace, or while in the editor or tracer, the name of the current object
- The name of the current keyboard input mode (see later)
- Whether input is in insert or overwrite (replace) mode

Some error conditions generate text that does not become part of the session, yet is written to the terminal. Additionally it is possible that other applications may write to the terminal. In such cases, and when the emulator window is resized, it may be necessary to perform a Screen Refresh (SR) which causes APL to rewrite the entire terminal emulator window according to what it believes should be present; this will effectively remove all extraneous text.

The session and the edit and trace windows form a loop; to cycle forwards between windows use the command Windows Tab (TB), to cycle backwards use the command Reverse Windows Tab (BT). At any time you can use the command Jump (JP) to toggle between the current edit/trace window and the session. Escape (EP) closes the current window, having saved any changes (where appropriate); QuiT (QT) closes the current window, but without saving any changes.

It is possible to move and to resize an edit or a trace window; hit Move/Resize (MR) to swap into this mode. In this mode the cursor keys move the current window around (note that when the window reaches the edge of the screen, its size will in many cases reduce as the opposite edge continues to move in the direction of movement. Up one Screen (US), Down one Screen (DS), Left one Screen (LS) and Right one Screen (RS) cause the right or bottom margin to extend or reduce as appropriate. Note that if the right or bottom edge is against the right or bottom edge of the session, then the window is made larger by "pushing" the left or top edge away as applicable.

---

Trace windows are read-only; however, it is possible to edit the currently traced object by hitting Edit (ED) while the cursor is on the first column of any line or by hitting ED while the cursor is on the name of the object. However, both in the Editor and Trace windows individual breakpoints (aka Stops) can be set and unset using the Toggle Breakpoint (BP) command. The Clear Breakpoints (CB) command will cause all breakpoints in the current object to be cleared. Note that by default there is no visible indication that either of these commands has been run; however, the output from `□STOP` will show whether either of these commands has been run. See "Configuring the Editor" for more details.

Edit windows and the session are read-write. By default input lines are in insert mode. It is possible to toggle to overwrite mode by using the Insert Toggle (IN) command. Note that this mode allows you to generate those overstrike APL characters which are supported by Dyalog APL; attempting to overwrite an existing character with one that does not form a valid APL character results in the original character being replaced with the newly-typed one. Destructive Backspace (DB) and Delete Item (DI) delete the character immediately before the cursor and the character under the cursor respectively. It is possible to define keycodes for Insert Item (II) and Non-destructive Backspace (NB) and Non-destructive Space (NS) but these are not in general use. Destructive Space (DP) is mapped to the Spacebar.

In an edit window Toggle Localisation (TL) will add the name currently under the cursor to the end of the header line so as to localise that name if it was not already present in the header; if the name is present in the header, it is removed from the header. Redraw (RD) causes the function to be reformatted, with indentations added etc.

It is possible to move or copy a line or a block of lines from one window to the other. It is also possible to Cut (CT) from the cursor position to the end of the line and to Paste (PT) the cut text; however, there is no other mechanism for selecting parts of a line although you can use the mouse and the facilities of the terminal window or emulator to move partial lines around. In this case you may find that it is best to have the editor or tracer windows maximised to avoid copying the line drawing characters that form the outline of the edit or trace windows too; Zoom (ZM) toggles windows between maximised and standard size.

Use the Tag (TG) command to select contiguous lines of text; identify the initial line with TG, move to the last line you wish to highlight and hit TG again. The next TG command only removes the tagging from the currently tagged block - it does not clear and initiate another selection. For Copy (CP) or Move (MV) move to the line immediately above where the text is to be placed, and hit CP or MV as appropriate. Use Delete Block (DK) to delete the highlighted lines. Note that it is possible to copy or move text between edit windows and the session.

Comments can be aligned to the column where the cursor is by hitting Align Comments (AC). Comments that appear in columns which precede the first tabstop are aligned to column 1.

Text searches can be made in all windows; the Search (SC) command defines the search string; hitting Enter (ER) to complete the definition also moves the cursor to the next instance of the search string in a forward direction. The Next (NX) and Previous (PV) commands moves the cursor to the next or previous instance of the search string; when there are no more instances in the specified direction the error field will contain either `No Match→` or `←No Match`.

Strings can be replaced in the Editor and Session windows; the cursor must be at the start of an instance of the search string. Replace (RP) command is used to specify the replacement string; if the cursor is at the start of an instance of the search string, that instance will be replaced with the replacement string. The Repeat (RP) command (also called Do) is used to make additional replacements. The Repeat All (RA) command will replace all instances of the search string with the replacement string in the current object, both forwards and backwards from the current position; in this case the cursor does not need to be at the start of an instance of the search string.

For both the Search and Replace commands EP is used to clear the definition of the appropriate string; the entire field will be removed from the status line.

Dyalog APL responds to weak and strong interrupts; the `kill` operating system command can be used to send a signal 2 (SIGINT) or 3 (SIGQUIT) respectively, or the user can hit the `intr` or `quit` keystrokes. The current mappings for these two keystrokes can be seen by running the operating system command `stty -a`. The most common keystrokes for `intr` and `quit` are `Ctrl-C` and `Ctrl-\` respectively. Note that when using PuTTY it will be necessary to swap out of the APL keyboard to generate these keystrokes.

The tables below show the keystrokes that can be used in the different windows.

**Commands Common to all Window Types**

Command	Code	Description
Cursor Move	LC RC UC DC	Left/Right/Up/Down one character
	LS RS US DS	Left/Right/Up/Down one screen
	LL RL UL DL	Left/Right/Up/Down to limit in that direction
	HO	Home Cursor .. to top left hand corner of object
Toggle line numbers	LN	Turn line numbers on or off in all trace and edit windows. This can be done from the session too
Screen Refresh	SR	Causes APL to redraw the session, removing all extraneous text that has come from external sources and resetting the session display

**Window Commands**

Command	Code	Description
Move between Windows	TB	Move to next window in loop
	BT	Move to previous window in loop
	JP	Jump - toggle between session and current window
Alter Windows	ZM	Zoom - toggle window to full size and back
	MR	Move/Resize: LC/RC/UC/DC: move window in that direction LS/RS/US/DS: move bottom right hand corner in selected direction relative to top left hand corner EP: exit move/resize mode

**Session Commands**

Command	Code	Description
Redo/Undo	FD	Show next line in input history
	BK	Show previous line in input history

**Editor Commands**

Command	Code	Description
Start/Stop	ED	Start Editor <sup>(1)</sup>
	EP	Fix and Close
	QT	Abort and Close
Fix function	FX	Causes the function to be fixed, without quitting the edit session
Redo/Undo	FD	Reapply last change
	BK	Undo last change (where possible)
Outlines	MO	When on the first or last line of a control structure, move to the opposite end <sup>(2)</sup>
	TO	Open/Close outlined blocks <sup>(2)</sup>
Toggle local	TL	For traditional functions, the name under the cursor is either added or removed from the list of localised names on the function's header line
Toggle Breakpoint	BP	Toggles a breakpoint on the current line <sup>(3)</sup>
Clear Breakpoints	CB	Clears all breakpoints in the current object <sup>(3)</sup>
Open Line	OP	Opens a line underneath the current line; in insert mode moving to the end of the line and hitting ER is equally effective
Reformat	RD <sup>(4)</sup>	Causes the function to be reformatted, with corrected indentation etc.
Comments	AC	Align comments to current column
	AO <sup>(4)</sup>	Add comment symbol at start of each tagged or current line
	DO <sup>(4)</sup>	Remove comment symbol which is first non-space character on each tagged or current line

**Notes:**

1. The editor can also be started using `)ED` or `[ED`. Hitting ED in the session with a suspended function on the stack will open the editor on that function; this is called Naked Edit.
2. By default outlines are not shown. See "Configuring the Editor" for further details.
3. By default there is no visual indication that a breakpoint has been set, although `[STOP` will show the breakpoints. However, it is possible to view breakpoints - see "Configuring the Editor" for further details.
4. AO, DO, RD only work in 13.1 onwards

**Tracer Commands**

Command	Code	Description
Start/Stop	TC	Start Tracer <sup>(1)</sup>
	EP	Cut stack back to calling function; close all windows to match new stack status
Execution	ER	Execute current line
	TC	Trace into any and all functions on current line
	FD	Skip over current line
	BK	Skip back one line
Toggle Breakpoint	BP	Toggles a breakpoint on the current line <sup>(2)</sup>
Clear Breakpoints	CB	Clears all breakpoints in the current object <sup>(2)</sup>
Continue	RM	Resume Execution - do not show trace windows on next error or stop
	BH	Run to Exit - but show trace windows on error or stop

**Notes:**

1. Hitting TC in the session with a suspended function on the stack will open one trace window for each function on the stack; this is called Naked Trace.
2. By default there is no visual indication that a breakpoint has been set, although `[STOP` will show the breakpoints. However, it is possible to view breakpoints - see "Configuring the Editor" for further details.



### Search and Replace Commands

Command	Code	Description
Define string	SC <sup>(1)</sup>	Search: having hit Search, type string to search for, and ER to find first occurrence. EP clears the field
	RP <sup>(2)</sup>	Replace: having hit Replace, type string to replace current search with; change will be effective once ER is hit. EP clears the field
Find and Replace	NX	Locate next match downwards
	PV	Locate previous match upwards
	RT	Repeat (Do) the same action again
	RA <sup>(3)</sup>	Repeat all - in both directions

### Notes:

1. Applies to session, editor and tracer
2. Applies to the session and editor only
3. Caution: the Repeat All replaces ALL matches in the current object

### Session-related Commands

Command	Code	Description
Selection	TG	Tag (highlight) blocks of text. Hit TG on initial line, move to last line to be tagged and hit TG again. Next TG clears the current tagging rather than initiating a new tag
Block commands	CP	Copy highlighted block to below current line
	DK	Delete highlighted block
	MV	Move the highlighted block to below the current line
Cut and Paste	CT	Cut from current cursor position to end of line
	PT	Paste last Cut text immediately after cursor

### Screen Manager Commands

Command	Code	Description
Move between <code>□SM</code> and session/trace/edit windows	HK	With non-empty <code>□SM</code> , toggle between <code>□SM</code> window and trace/edit/session window. HK is a valid exit key for <code>□SR</code> , but using it as such can be confusing !
Exit keys	EP QT ER	Default exit keys for <code>□SR</code>

## Starting APL

By default, to start the non-GUI versions of Dyalog APL, run the `mapl` script which is in the installation directory of Dyalog APL.

### Example:

```
$ /opt/mdyalog/18.0/64/unicode/mapl
```

The `mapl` script is supplied so that the user can start to use Dyalog APL immediately once the terminal environment has been setup. However, it should be treated more as a template for creating a startup script more appropriate for the environment and purposes that Dyalog will be used for.

The startup script usually sets a number of environment variables, and then calls the interpreter with one or more of its parameters. Although all the examples are written using the Korn shell, any shell can be used.

Note that under Microsoft Windows parameters appear *after* the name of the executable; this is not supported under UNIX, where values must be passed as environment variables.

The parameters are listed in the table below; the more frequently used environment variables are included in the following section.

**Table 1: Parameters for the mapl or dyalog script:**

Parameter	Purpose
-tty	Start APL using the terminal development environment. This is not necessary unless the wine (-wine) or MainWin (-mainwin) versions are installed too.
-c -rt -server	Causes dyalog.rt (the server version) to be started. This parameter is for backwards compatibility; the use of the -rt or -server parameter is recommended. See also the Note at the bottom of this table.
-*	Any other parameter that starts with a "-" will be passed to the interpreter; all parameters that start with a "." will be passed before any parameters that do not start with a "-".
*	This is usually the name of the workspace that is to be loaded when the interpreter is started. Unless the "-x" flag is passed to the interpreter, the latent expression in the workspace will be executed once the workspace has been loaded.

**Note:**

- the -c parameter has different uses depending on whether it is passed to the mapl script, or to the dyalog executable.

**Table 2: Parameters for the Dyalog interpreter:**

Parameter	Purpose
-a	Start in "User mode". If not present, then APL will start in "Programmer mode". See the section on I/O for further details.
-b	Suppress the banner in the session.
-c	Comment: the "-c" and anything following it will be treated as a comment, but will show up in a long process listing. By adding a suitable comment the user or system administrator can uniquely identify the individual APL processes.  See also the Note above this table.
-Dw	Check workspace integrity on return to session input.
-DW	Check workspace integrity after every line of APL (application will run slowly as a result)
-DK	Log session keystrokes in (binary) file <code>/aplllog</code> .
-q	Continue to run even if an error causes a return to the six-space prompt. Used when redirecting input to the session from a pipe or file.
+q	A return to the six-space prompt will result in the interpreter terminating.
-s	Turn off the session: APL acts similarly to a scrolling terminal.
+s	forces APL to enable the session.
-x	Do not execute the latent expression of any workspace that is <code>)LOADed</code> or <code>⎕load</code> d. This applies to every <code>)load</code> or <code>⎕load</code> during the life of the APL session.
ws	This is assumed to be a workspace which will be loaded once the interpreter has started. Unless the -x parameter is included on the command line, the latent expression will be run.
-cef/ -apl	See the <i>Dyalog Version 17.0 Release Notes</i> for more information

**Examples:**

```
mapl dfns
MAXWS=2G mapl dfns
MAXWS=2G DEFAULT_IO=0 mapl -x dfns
```

## 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]  |              r←2
[7] [6]  |          :EndIf
[8] [7]  |      :EndIf
[9] [8]  |      ∇
[10]
[11] [0]      dfn←{α+w}
[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+w}
[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	<pre>fn1 a :If a=1   b←2 :EndIf</pre>
22	<pre>[0]   fn1 a [1]   :If a=1 [2] o   b←2 [3]   :EndIf</pre>
26	<pre>fn1 a o   :If a=1     b←2     :EndIf</pre>
40	<pre>[0]   fn1 a [1]     :If a=1 [2] o   b←2 [3]     :EndIf</pre>

---

## File permissions and `⎕FSTAC`

Dyalog APL is a well behaved UNIX program and honours all standard UNIX file permissions. Commands such as `⎕FLIB` and `)LIB` read the magic number (the first few bytes) of each file in the directory in order to determine whether each file is a component file or workspace respectively; if the APL process cannot read those bytes, then it will assume that the file is not a component file or workspace.

Under UNIX, the first element of `⎕AI` is the user's effective uid, and `⎕AN` reports the user's real name, as it appears in `/etc/passwd`. When a component file is newly created, its UNIX file permissions will be defined by the umask for the effective user id. The APL file access matrix will be `(0 3ρ0)`, which means that even if the user's UNIX file permissions are such that anyone can read and write to the file, only the user in question will be able to access the file using Dyalog APL component file system functions. To allow any user to access the file (assuming that the UNIX file permissions are suitable) then run

```
(1 3ρ0 ~1 0)⎕fstac tieno
```

Any user with an effective uid 0 will be able to access any component file, irrespective of the file access matrix.

## Signals and `TRAP`, `14007`

### Signals and `TRAP`

Certain signals sent to a Dyalog APL process can be trapped and an event issued. These signals are:

1	SIGHUP
2	SIGINT
3	SIGQUIT
15	SIGTERM

No other signal is trapped by the interpreter; their default action will occur. For example when a Dyalog APL process receives a SIGSEGV (11) then it will terminate with a segmentation fault. Note that SIG\_USR1 is used by the interface between Dyalog APL and Auxiliary Processors: sending this signal to the interpreter may have "interesting" consequences.

The mapping between these signals and the event issued is non-trivial:

- If a SIGHUP is received, then the input stream is closed immediately, and an event 1002 will be issued at the end of the current line of code. Any subsequent attempt to read from the session will result in an EOF INTERRUPT being issued.
- If a SIGINT is received, then execution will end at the end of the current line of code. An event 1002 will be issued.
- If a SIGQUIT is received, then APL will terminate executing the current line of code as soon as possible - usually at the end of the current built-in command, and an event 1003 will be issued. However, if the end of the current line is reached, then an event 1002 will be signalled too.
- If a SIGTERM is received, then the input stream is closed immediately, and an event 1002 will be issued at the end of the current line of code. Any subsequent attempt to read from the session will result in an EOF INTERRUPT being issued.



## I4007

To aid the programmer in determining which signal was issued, the newly implemented system operator, **I** (I-Beam) has been extended to report this information.

WARNING: Although documentation is provided for I-Beam functions, any service provided using I-Beam should be considered as "experimental" and subject to change - without notice - from one release to the next. Any use of I-Beams in applications should therefore be carefully isolated in cover-functions that can be adjusted if necessary.

**4007I@** can be used to identify which signals have been received by the APL process and how many of them have been received. A side effect of calling **4007I@** is to reset all counters to 0.

**4007I@** returns a vector of integers; the length is dependent on the APL interpreter and the operating system, but is typically 63 or 255 elements long. Each element is a count number of each signal received and processed by the interpreter. Note that when a SIGQUIT is received by APL the count for both SIGINT and SIGQUIT will be incremented by one.

### Example:

```
      8↑4007I@
1 5 3 0 0 0 0 0
```

This means that since either the start of the current APL process, or since the last invocation of **4007I** APL has processed 1 SIGHUP, 2 SIGINTs and 3 SIGQUITs.

It is recommended that rather than trapping either event 1002 or 1003, the user traps event 1000, and queries the vector returned by **4007I@**. In particular if a SIGHUP or a SIGTERM has been received, then the user's code should terminate the application as soon as possible, and should be careful to avoid requiring input. SIGHUP has either been issued using the kill(1) command, or because either the device at the other end of the connection or the connection has terminated. This used to be common with serial or dialup terminals, but is now most frequently seen when terminal emulators or the PCs on which they run are terminated.

## □SH, exit codes and stderr

Note that □SH calls /bin/sh; this cannot be altered.

If the command, or command pipe issued using □SH exits with a non-zero exit code, then □SH will terminate with a **DOMAIN ERROR**, and all output from the command will be lost. To avoid this, add an exit 0 to the end of the command string, and the **DOMAIN ERROR** will be suppressed. However, this technique does require that some other method is used to determine that the command pipe failed.

Example:

```
ρ□SH 'grep no_such_user /etc/passwd'
DOMAIN ERROR
ρ□SH 'grep no_such_user /etc/passwd'
^
```

but

```
0 ρ□SH 'grep no_such_user /etc/passwd ; exit 0'
```

If you are interested in the exit code from the command pipe, rather than any partial output, then, in Version 18.0 onwards □DMX.Message has the exit code at the end of the text. Dyalog intends that this value will appear in a more user-friendly form in □DMX at some point.

Example:

```
z←□SH 'exit 17'
DOMAIN ERROR
□DMX.Message
Command interpreter returned failure code 17
```

□SH only captures stdout; unless redirected, any output on stderr will appear in the same terminal window as the session; hitting RD (default Ctrl-L) will force a screen redraw, thereby returning the session to its state before the error output appeared.

---

## SSH and starting jobs in background

It is possible to run tasks from within APL using SSH:

```
ssh 'myjob'
```

However, in this case, APL will wait until myjob has completed, and will return the output from myjob (assuming that is that myjob completes with a non-zero exit code). It is possible to start a job that will run in background, without APL waiting for that job to complete, with the job continuing even if APL is terminated:

Example:

```
ssh 'sleep 40000 </dev/null >/dev/null 2>&1 &'
```

More useful might be to save the stdout and stderr of the command, and pipe the input in from a file; it might also be useful to have the job continue to run even after the user has both quit APL and logs out from the server:

```
ssh 'nohup myjob <my.in >my.out 2>my.err &'
```

---

# 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
```

---

## Core and aplcore files

When Dyalog APL encounters an unexpected problem it is likely that the interpreter will terminate and generate either a core file or an aplcore file. Under Linux core files are not created by default; it is necessary to enable their creation.

An aplcore file contains the workspace at the point where the interpreter terminated, along with debug information that may enable Dyalog to identify and rectify the problem.

The Dyalog support department ([support@dyalog.com](mailto:support@dyalog.com), other means of contact on the Dyalog website) should be contacted if an aplcore file is generated. More immediately it may be possible to copy the contents of the aplcore into a new Dyalog process by running

```
)copy aplcore
```

Note however that it is possible that the `)COPY` itself will cause another aplcore; it is best to rename the original aplcore before attempting this course of action.

From Version 13.2 onwards in situations where a core file is generated, an aplcore file will be generated too; this is done by forking the failing APL process, so an additional APL process will appear in any process listing while the aplcore is being created. If the environment variable `APL_TEXTINAPLCORE` is set and has the value 1 then an "Interesting Information" section is appended to the aplcore which contains information such as the APL stack, the WSID of the originating workspace etc. This section can be extracted from an aplcore using

```
sed -n '/===== Interesting Information/,,$p' aplcore
```

# Appendix A: Table of keycodes and keystrokes using a terminal emulator under Linux GUIs

Keycodes, their common keystrokes, and the keystrokes specific to terminal emulators under Linux GUIs.

## Notes:

1. APL represents the metakey used as the APL character and command shift
2. Cmd represents the keystroke <Ctrl-x>
3. CMD represents the keystrokes <Ctrl-x><Ctrl-x>
4. The file \$DYALOG/aplkeys/xterm is certain to be up to date and should be treated as the definitive source of the keycode-keystroke translations

Keycode	Command	Common keystrokes	Terminal Emulator
AC	Align Comments	Cmd a	
AO	Comment Out	Cmd ,	
BH	Run to Exit	Cmd <	APL+Left
BK	Back	Cmd b	APL+Up
BP	Toggle Breakpoint	CMD b	APL+Backspace
BT	Back Tab Window	CMD Tab	Shift+APL+Tab
CB	Clear Breakpoints	CMD B	Shift+APL+Backspace
CP	Copy	Cmd c	APL+Insert
CT	Cut	CMD c	Shift+APL+Delete
DB	Backspace	Backspace	
DC	Down Cursor	Down	
DI	Delete Item	Delete	
DK	Delete Block	Cmd Delete	APL+Delete
DL	Down Limit	Ctrl+Down	Shift+APL+PgDn
DO	Uncomment	Cmd .	
DS	Down Screen	Shift+Down	APL+PgDn
ED	Edit	Cmd e	APL+Enter
EP	Escape	Esc	Esc

Keycode	Command	Common keystrokes	Terminal Emulator
ER	Enter	Enter	
FD	Forward	Cmd f	APL+Down
FX	Fix	Cmd x	
HK	Hot Key (□SM)	Cmd u	
HO	Home Cursor	Cmd h	
IN	Insert Mode	Cmd i	
JP	Jump	Cmd j	
LC	Left Cursor	Cursor Left	
LL	Left Limit	Ctrl+Left	Shift+APL+Home
LN	Line Numbers	Cmd l	APL+Numpad+-
LS	Left Screen	Shift+Left	APL+Home
MO	Move to Outline	CMD %	Shift+APL+Space
MR	Move/Resize	CMD m	
MV	Move block	Cmd m	
NX	Next	Cmd n	
OP	Open line	Cmd o	
PT	Paste	CMD p	Shift+APL+Insert
PV	Previous	Cmd p	
QT	Quit	Cmd q	APL+Esc
RA	Repeat All	CMD d	
RC	Cursor Right	Right	
RD	Redraw Function	CMD r	APL+Numpad-/
RL	Right Limit	Ctrl+Right	Shift+APL+End
RM	Resume All Threads	Cmd >	APL+Right
RP	Replace String	Cmd r	
RS	Right Screen	Shift+Right	APL+End
RT	Repeat (Do)	Cmd d	



<b>Keycode</b>	<b>Command</b>	<b>Common keystrokes</b>	<b>Terminal Emulator</b>
SC	Search	Cmd s	
SR	Redraw Screen	Ctrl+l (1)	
TB	Tab Window	Cmd Tab	APL+Tab
TC	Trace	Cmd Enter	Shift+APL+Enter
TG	Tag	Cmd t	APL+Numpad-*
TL	Toggle Localisation	CMD l	APL+Numpad+*
TO	Toggle Outline	CMD o	APL+Space
UC	Cursor Up	Cursor Up	
UL	Up Limit	Ctrl+Up	Shift+APL+PgUp
US	Up Screen	Shift+Up	APL+PgUp
ZM	Zoom	Cmd z	Shift+APL+F12

## Appendix B: Table of keycodes and keystrokes for PuTTY

Keycodes, their common keystrokes, and the keystrokes specific to the PuTTY terminal emulator.

### Notes:

1. APL represents the metakey used as the APL character and command shift
2. Cmd represents the keystroke <Ctrl-x>
3. CMD represents the keystrokes <Ctrl-x><Ctrl-x>
4. The file \$DYALOG\aplkeys\xterm is certain to be uptodate and should be treated as the definitive source of the keycode-keystroke translations

Keycode	Command	Common keystrokes	PuTTY
AC	Align Comments	Cmd a	
AO	Comment Out	Cmd ,	
BH	Run to Exit	Cmd <	
BK	Back	Cmd b	Shift+Ctrl+Backspace
BP	Toggle Breakpoint	CMD b	Shift+End
BT	Back Tab Window	CMD Tab	Shift+Ctrl+Tab
CB	Clear Breakpoints	CMD B	
CP	Copy	Cmd c	Ctrl+Insert
CT	Cut	CMD c	Shift+Delete
DB	Backspace	Backspace	Backspace
DC	Down Cursor	Down	
DI	Delete Item	Delete	
DK	Delete Block	Cmd Delete	Ctrl+Delete
DL	Down Limit	Ctrl+Down	Ctrl+End
DO	Uncomment	Cmd .	
DS	Down Screen	Shift+Down	PgDn
ED	Edit	Cmd e	Shift+Enter
EP	Escape	Esc	Esc
ER	Enter	Enter	Enter

Keycode	Command	Common keystrokes	PuTTY
FD	Forward	Cmd f	Shift+Ctrl+Enter
FX	Fix	Cmd x	
HK	Hot Key (⌘SM)	Cmd u	
HO	Home Cursor	Cmd h	
IN	Insert Mode	Cmd i	
JP	Jump	Cmd j	Shift+Ctrl+Home
LC	Left Cursor	Cursor Left	
LL	Left Limit	Ctrl+Left	
LN	Line Numbers	Cmd l	
LS	Left Screen	Shift+Left	Ctrl+Left
MO	Move to Outline	CMD %	Shift+Ctrl+Up
MR	Move/Resize	CMD m	
MV	Move block	Cmd m	Shift+Ctrl+Delete
NX	Next	Cmd n	Shift+Ctrl+Right
OP	Open line	Cmd o	Shift+Ctrl+Insert
PT	Paste	CMD p	Shift+Insert
PV	Previous	Cmd p	Shift+Ctrl+Left
QT	Quit	Cmd q	Shift+Esc
RA	Repeat All	CMD d	Ctrl+Down
RC	Cursor Right	Right	
RD	Redraw Function	CMD r	Shift+PgUp
RL	Right Limit	Ctrl+Right	
RM	Resume All Threads	Cmd >	
RP	Replace String	Cmd r	
RS	Right Screen	Shift+Right	Ctrl+PgDn
RT	Repeat (Do)	Cmd d	Shift+Ctrl+Down
SC	Search	Cmd s	

Keycode	Command	Common keystrokes	PuTTY
SR	Redraw Screen	Ctrl+l (1)	
TB	Tab Window	Cmd Tab	Ctrl+Tab
TC	Trace	Cmd Enter	Ctrl+Enter
TG	Tag	Cmd t	
TL	Toggle Localisation	CMD l	Ctrl+Up
TO	Toggle Outline	CMD o	Shift+Up
UC	Cursor Up	Cursor Up	
UL	Up Limit	Ctrl+Up	Ctrl+Home
US	Up Screen	Shift+Up	PgUp
ZM	Zoom	Cmd z	Shift+Ctrl+PgUp

**Notes:**

- If you are using PuTTY or another emulator that uses the Dyalog Unicode IME, it will be necessary to swap to a non-Dyalog APL keyboard before hitting Ctrl-l; hitting Ctrl-l while in a Dyalog APL keyboard will generate a Quad symbol.

## Appendix C: Unused keycodes

Keycodes defined for Dyalog APL, but not used or should not be used in the Dyalog APL tty version

Key Code	Command
AB	Abort Changes (effectively same as QT)
CB	Clear stop/trace/monitor
CH	Change Hint
Dc	Down with selection
DD	Drag and Drop
DH	Delete Highlighted section
DI	Down Limit with selection
Dn	Down Mouse key n, n ∈ {1 2 3 4 5}
Ds	Down Screen with selection
EN	End of Line
GL	Goto Line
HT	Horizontal Tab
IF	Insert Off
Lc	Left with selection
LI	Left Limit with selection
LW	Left Word
Lw	Left Word with selection
MC	Mode Change
PA	Paste ANSI
PR	Properties
PU	Paste Unicode
Rc	Right with selection
RI	Right Limit with selection
RW	Right Word
Rw	Right Word with selection

---

<b>Key Code</b>	<b>Command</b>
ST	Start of Line
TH	Reverse Horizontal Tab
UA	Undo All
Uc	Up with selection
Ul	Up Limit with selection
Un	Up Mouse key n, n ∈ {1 2 3 4 5}
Us	Up Screen with selection

# Index

## A

APLCore file 26

## B

BuildID in saved files 24

## C

### Commands

- common to all window types 10
- editor related 11
- overview and description 7
- screen manager related 14
- search and replace 13
- selection related 13
- session related 11
- tracer related 12
- window related 10

Configuring the editor 17

Core file 26

Count signals 20

## E

Edit Window 6

Entering characters 3

Entering commands 5

Entering Stencil and Nest 3

## F

File permissions

UNIX 19

FSTAC

UNIX 19

## I

Interpreter parameters 16

## K

Keystrokes for console/terminal windows 27

Keystrokes for PuTTY 30

## M

Metakey 2

## Q

quadSH

exit codes 22

stderr 22

## S

Screen Manager Window 6

Session Window 6

Starting APL under UNIX 14

Startup script parameters 15

Status line contents 7

## T

Trace Window 6

## U

Unused keystrokes 33

