UNIX Installation and User Guide

Version 13.2

DYALOG APL

The tool of thought for expert programming

Version: 13.2

Revision: 22186

Contents

Introduction

This manual is designed to assist users of the non-GUI versions of Dyalog APL on UNIX platforms. It applies to Versions 12.1 onwards.

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.

Changes in 13.2 specific to UNIX

SQAPL is now bundled with the Unicode Edition of Dyalog APL for Linux. It was compiled against unixODBC 2.2.11.

The .deb files now define the pre-requisite filesets needed to run 32 bit interpreters on 64 bit kernels. They also recommend the unixODBC driver filesets.

Links from xterm to screen now make it easier to run APL under screen. Simply run **screen -U**.

If a core file is generated by a Dyalog APL process, both a core and an aplcore file will be generated. If `APL_TEXTINAPLCORE=1` additionally the useful information section is included at the end of the aplcore file.

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.

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 edtion of Dyalog APL). So for example, in a terminal window <WindowsKey><a>generates an ⍺; 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.]

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).

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.

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 (1)
	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 (2)
	TO	Open/Close outlined blocks(2)
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(3)
Clear Breakpoints	CB	Clears all breakpoints in the current object(3)
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(4)	Causes the function to be reformatted, with corrected indentation etc
Comments	AO(4)	Add comment symbol at start of each tagged or current line
	DO(4)	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(1)
	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(2)
Clear Breakpoints	CB	Clears all breakpoints in the current object(2)
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(1)	Search: having hit Search, type string to search for, and ER to find first occurrence. EP clears the field
	RP(2)	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(3)	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 ⎕SM and session/trace/edit windows	HK	With non-empty ⎕SM, toggle between ⎕SM window and trace/edit/session window. HK is a valid exit key for ⎕SR, but using it as such can be confusing !
Exit keys	EP QT ER	Default exit keys for ⎕SR

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 V12 onwards 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/<APLVersion>/<APLWidth>/<APLEdition>
```

or, in the case of AIX:

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

So for example, Dyalog APL 12.1 32bit Unicode for POWER6 hardware on AIX will by default be installed into

```
/opt/mdyalog/12.1/32/unicode/p6
```

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

```
/opt/mdyalog/12.1/32/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.

Other than the Dyalog APL application shortcut (.desktop), Icon and the rpm/deb related database information no files are created outside the Dyalog installation directory; in particular, unlike earlier versions of Dyalog APL, no files are placed in /usr/bin.

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.

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 13.0 and prior these are p3, p5 and p6. For 13.1 onwards (as of September 2012) specific builds for p5, p6 and p7 are created.

```
$ su -
# cd /opt
# cpio -icdvum </tmp/dyalog-20090901-64-unicode-p6.cpi
# /opt/mdyalog/12.1/64/unicode/p6/make_scripts
# exit
```

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

```
$ /opt/mdyalog/12.1/64/unicode/p6/mapl
```

Notes:

- Version 12.1 onwards are compiled on AIX6.1: the 32 bit version of 12.1 has also been tested on AIX5.3 Technical Maintenance Level 9

Installing on an RPM-based Linux Distribution

```
$ su -
# rpm -install /tmp/dyalog-121C32r14707-20120921.linux.i386.rpm
# /opt/mdyalog/12.1/32/classic/make_scripts
# exit
```

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

```
$ /opt/mdyalog/12.1/32/classic/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.
- Version 12.1 has been successfully installed on RHEL4/Centos4, and SUSE10.3, and later versions of these distributions.

Installing on a DEB-based Linux Distribution

```
$ sudo su -
# dpkg --install dyalog-unicode_13.2.16658_i386.deb
# exit
```

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

```
$ /opt/mdyalog/13.2/32/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.

Minimal Installation

An absolutely minimal installation of the development version of Dyalog APL consists of something similar to one of the two following examples. The maximum workspace size would be 4MB, and all workspaces and auxiliary processors would have to be accessed using either absolute or relative pathnames. It is not necessary to have superuser permissions to perform this installation, since no files are being written to directories for which the user does not have write permission.

In this example it is assumed that a full install has been completed of the 32-bit Unicode version on Linux. Once the files have been extracted, then the installation could be removed. It is also assumed that suitable preparation of the terminal environment has been completed (see Terminals and Terminal Emulators).

To set up:

```
$ mkdir dyalog_min
$ cd dyalog_min
$ mkdir aplkeys
$ mkdir apltrans
$ cp /opt/mdyalog/12.1/32/unicode/dyalog .
$ cp /opt/mdyalog/12.1/32/unicode/apltrans/xterm aplkeys
$ cp /opt/mdyalog/12.1/32/unicode/apltrans/xterm apltrans
$ cp /opt/mdyalog/12.1/32/unicode/apltrans/file apltrans
```

To run:

```
$ cd dyalog_min
$ export APLKEYS=`pwd`/aplkeys
$ export APLTRANS=`pwd`/apltrans
$ ./dyalog
```

```
Dyalog APL/S Version 12.1.0
Unicode Edition
Sun Oct 18 13:45:21 2009
clear ws

      16 16⍴⎕av
...
```

or

To set up:

```
$ mkdir dyalog_min
$ cd dyalog_min
$ cp /opt/mdyalog/12.1/32/unicode/dyalog .
$ cp /opt/mdyalog/12.1/32/unicode/aplkeys/xterm xterm.in
$ cp /opt/mdyalog/12.1/32/unicode/apltrans/xterm xterm.out
$ cp /opt/mdyalog/12.1/32/unicode/apltrans/file .
```

To run:

```
$ cd dyalog_min
$ export APLKEYS=`pwd`
$ export APLTRANS=`pwd`
$ export APLK=xterm.in
$ export APLT=xterm.out
$ export APLT10=file
$ export APLT11=file
$ ./dyalog
```

```
Dyalog APL/S Version 12.1.0
Unicode Edition
Sun Oct 18 13:45:21 2009
clear ws

      16 16⍴⎕av
...
```

In the first example the input and output translate tables are in separate sub-directories, so there is no need to override the TERM variable with values for APLK and APLT.

In the second example the two translate tables both are named differently from the value of $TERM, so each has to be defined. Note also that it may be necessary to have the file output translate table present: like other UNIX processes, Dyalog APL inherits the open file descriptors of its parent process, and each of them will need to have a translate table associated with it. So for example, under KDE4 on openSUSE, a Konsole terminal window has file descriptors 0 1 2 10 11 open; Dyalog APL will need therefore to have APLT10 and APLT11 defined too.

Further details regarding configuring the Dyalog APL environment can be found in the Starting APL section.

Deinstalling Dyalog APL

In the following examples, it is assumed that only Dyalog APL 12.1 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/12.1
```

Deinstalling on an RPM-based Linux Distribution

```
$ su -
# rpm -e dyalog.32.classic-12.1-20090901
# cd /opt
# rm -rf mdyalog/12.1
# exit
```

Deinstalling on a DEB-based Linux Distribution

```
$ sudo su -
# apt-get purge dyalog-unicode-132
# cd /opt
# rm -rf mdyalog/13.2
# 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.

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.

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/12.1/32/classic/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.

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 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 "Prog (rammer) mode". This refers to input translate tables, but is primarily meant for backwards compatibility. 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.
-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. If not used, then a return to the six-space prompt will result in a CONTINUE being generated, and the interpreter terminating.
+q	suppress this behaviour.
-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 `)LOAD`ed or `⎕load`ed. This applies to every `)load` or `⎕load` during the life of the APL session.
filename	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 immediately after the load has finished.

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.

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:

http://forums.dyalog.com/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:

http://www.dyalog.com/resources

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

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 http://www.dyalog.com/resources. 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 http://www.dyalog.com/resources. Details of how to install the font appear on the download webpage.

Downloading and Installing PuTTY

PuTTY is available from http://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*

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*).

Environment Variables

Environment variables are used to configure various aspects of Dyalog APL. The complete list appears in the Dyalog APL Users Guide; 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
```

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 buffers and logfiles etc
- Table E4: Miscellaneous Variables used by non-GUI Dyalog APL
- Table E5: Editor-related environment variables

Table E1: Commonly used Variables

Variable	Notes
TERM APLK APLK0 APLT 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 is set, and they in turn override the value of TERM if set. APLK is for input translation, APLT for output translation. These are used in conjunction with ..
APLKEYS APLTTRANS	Define the search path for the input and output translate tables respectively. If unset, the interpreter will default to /usr/dyalog.
APLFSCB FILE_CONTROL	Defines the method used for multi-user access of Dyalog component files, and the location of the APL FileSystem Control Block if FILE_CONTROL=1. In earlier versions of Dyalog APL the APLFSCB method was the default for ensuring coherent multi-user access of Dyalog APL component files. It is now recommended that this is replaced with the default FILE_CONTROL=2 – using standard operating system facilities. Note: it is essential that all users sharing the same component files use the same FILE_CONTROL mechanism, and if using APLFSCB, the same FSCB file. Without these precautions component files WILL become corrupted.
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.
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. 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).

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 4096KB.
WSPATH	Defines the search path for both workspaces and Auxiliary processors. If unset, there is no default value. Workspaces and APs that are not on the WSPATH can be accessed using absolute or relative pathnames.

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 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_RL	Default value for `⎕rl` in a clear workspace.
DEFAULT_RTL	Default value for `⎕rtl` in a clear workspace.
DEFAULT_WX	Default value for `⎕wx` in a clear workspace. 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 buffers and logfiles etc

Variable	Notes
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 LOG_FILE_INUSE LOG_SIZE	These three variables determine the name of the session log file (default ./default.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.

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
APLFORMATBIAS	See the Language Reference for more details.
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	
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. 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 User Guide which are effective in the non-GUI UNIX versions of Dyalog APL

Table E5: Editor-related environment variables

Variable	Notes
EDITOR_ COLUMNS_*	More details below. Can be one of EDITOR_COLUMNS_CHARACTER_ARRAY EDITOR_COLUMNS_CLASS EDITOR_COLUMNS_FUNCTION EDITOR_COLUMNS_NAMESPACE EDITOR_COLUMNS_NUMERIC_ARRAY
LINES_ON_ FUNCTIONS	Whether line numbers are on or off. This is used in conjunction with the variables which determine which features of the editor are enabled.

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←{⍺+⍵}
[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]	○	│	r←2
[7]	[6]		├	:EndIf
[8]	[7]		├	:EndIf
[9]	[8]		├	∇
[10]				
[11]	[0]			dfn←{⍺+⍵}
[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`
	`:If a=1`
	`    b←2`
	`:EndIf`

22	`[0]   fn1 a`
	`[1]   :If a=1`
	`[2] ○     b←2`
	`[3]   :EndIf`

26	`   fn1 a`
	` ├ :If a=1`
	`○│     b←2`
	` ├ :EndIf`

40	`[0]    fn1 a`
	`[1]   ├ :If a=1`
	`[2] ○│     b←2`
	`[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/12.1/32/classic/p5/magic *
```

or catenate the contents of /opt/mdyalog/12.1/32/classic/p5/magic onto /etc/magic, and then run

```
file *
```

Example:

```
$ file -m /opt/mdyalog/12.1/32/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/12.1/32/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.

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 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 that user. The APL file access matrix will be `(0 3⍴0)`, 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 3⍴0 ¯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.

Session logfile

By default the session logfile is called default.dlf, and if necessary will be created in the current working directory. (The mapl script supplied by Dyalog overrides this).

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.

In V12.1 onwards 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.

Signals and ⎕TRAP, ⌶4007

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 Auxilliary 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.
- 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.

⌶4007

To aid the programmer in determining which signal was issued, the newly implemented system operator, ⌶ (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.

4007⌶⍬ 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 4007⌶⍬ is to reset all counters to 0.

4007⌶⍬ 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↑4007⌶⍬
1 4 2 0 0 0 0 0
```

This means that since either the start of the current APL process, or since the last invocation of 4007⌶ APL has processed 1 SIGHUP, 2 SIGINT and 2 SIGQUIT.

It is recommended that rather than trapping either event 1002 or 1003, the user traps event 1000, and queries the vector returned by 4007⌶⍬. 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 ⎕SHcalls /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:

```
      ⍴⎕SH 'grep no_such_user /etc/passwd'
DOMAIN ERROR
      ⍴⎕SH 'grep no_such_user /etc/passwd'
     ^
```

but

```
      ⍴⎕SH 'grep no_such_user /etc/passwd ; exit 0'
0
```

⎕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.

⎕SH and starting jobs in background

It is possible to run tasks from within APL using ⎕SH:

```
      ⎕sh'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:

```
      ⎕sh '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:

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

BuildID

In Version 12.1 of Dyalog APL for UNIX, it is now possible to identify the BuildID of each interpreter. This is a checksum of the interpreter (or indeed any other file) which can be used to identify the particular build. It is not unique, but is a 32-bit checksum, so clashes should be rare.

Dyalog Ltd request that the BuildID of the interpreter is included when asking for assistance; Dyalog keeps a record of the BuildID of every interpreter that is compiled, so that it can quickly identify the exact details (date, version, build platform etc) which assist in resolving questions and problems. Dyalog recommends that especially in multi-server environments or environments where there are multiple versions of Dyalog APL that a record of the BuildID of every interpreter is kept.

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=”
```

This information can be used by the user to identify which interpreter had the problem which caused the core file.

BuildID can be used for any file; it is a useful method of keeping track of workspaces versions etc., although md5sum and others may be more appropriate.

The BuildID can be identified both from within the interpreter, and also from the BuildID executable which is supplied with the product.

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, 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 uptodate and should be treated as the definitive source of the keycode-keystroke translations

Keycode	Command	Common keystrokes	Terminal Emulator
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

Keycode	Command	Common keystrokes	Terminal Emulator
EP	Escape	Esc	Esc
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

Keycode	Command	Common keystrokes	Terminal Emulator
RP	Replace String	Cmd r	
RS	Right Screen	Shift+Right	APL+End
RT	Repeat (Do)	Cmd d	
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**
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

Keycode	**Command**	**Common keystrokes**	**PuTTY**
EP	Escape	Esc	Esc
ER	Enter	Enter	Enter
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 >	

Keycode	Command	Common keystrokes	PuTTY
RP	Replace String	Cmd r	
RS	Right Screen	Shift+Right	Ctrl+PgDn
RT	Repeat (Do)	Cmd d	Shift+Ctrl+Down
SC	Search	Cmd s	
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

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
Dl	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
Ll	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
Rl	Right Limit with selection
RW	Right Word
Rw	Right Word with selection

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