



The tool of thought for expert programming

Dyalog™ for Windows

Installation & Configuration Guide

Version: 14.0

Dyalog Limited

email: support@dyalog.com

<http://www.dyalog.com>

Dyalog is a trademark of Dyalog Limited

Copyright © 1982-2014 by Dyalog Limited

All rights reserved.

Version: 14.0

Revision: 20150120

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.

TRADEMARKS:

SQAPL is copyright of Insight Systems ApS.

UNIX is a registered trademark of The Open Group.

Windows, Windows Vista, Visual Basic and Excel are trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

All other trademarks and copyrights are acknowledged.

Contents

Chapter 1: Installation and Configuration	1
Files and Directories	1
Unicode and Classic Editions	2
APL Fonts	3
Interoperability	4
The APL Command Line	9
APL Exit Codes	11
Configuration Parameters	12
Registry Sub-Folders	36
Window Captions	39
Workspace Management	41
Interface with Windows	42
Auxiliary Processors	42
Access Control for External Variables	44
Creating Executables	45
Run-Time Applications and Components	49
COM Objects and the Dyalog APL DLL	58
APL Application as a Service	61
APLService Logging Events	66
System Errors	71
 Chapter 2: Configuring the IDE	 77
The Configuration Dialog Box	77
Colour Selection Dialog	106
Print Configuration Dialog Box	109
 Index	 117

Chapter 1:

Installation and Configuration

Files and Directories

File Naming Conventions

The following file naming conventions have been adopted for the various files distributed with and used by Dyalog APL/W.

Extension	Description
.DWS	Dyalog APL Workspace
.DSE	Dyalog APL Session
.DCF ¹	Dyalog APL Component File
.DXV	Dyalog APL External Variable
.DIN	Dyalog APL Input Table
.DOT	Dyalog APL Output Table
.DFT	Dyalog APL Format File
.DXF	Dyalog APL Transfer File
.DLF	Dyalog APL Session Log File
.dyalog	Dyalog APL SALT file
.dyapp	Dyalog APL SALT application file

¹Note that DCF has become associated with a certain type of media file and is recognised as such by Windows.

Unicode and Classic Editions

Dyalog APL continues to be available in two separate editions; *Unicode* and *Classic*.

- The *Unicode* edition is intended for users who need to develop Unicode applications now, and are prepared to make the necessary (usually small) changes to existing applications in order to support new Unicode character types.
- The *Classic* edition is intended for customers who want to take advantage of other product enhancements, but do not wish to use Unicode at this time.

The two different editions are maintained from the same source code, and every effort will be made to ensure that they are identical except for the handling of character arrays, and the transfer of data into and out of the workspace.

APL Fonts

Unicode Edition

The default font for the Unicode Edition is APL385 Unicode¹ which is a TrueType font and is installed as part of Dyalog APL. APL385 Unicode is the font used to print APL characters in this manual. In principle, you may use any other Unicode font that includes the APL symbols, such as Arial Unicode MS (available from Microsoft).

Classic Edition

In the Classic Edition, there are two types of APL font provided; bitmap (screen) and TrueType. There are also two different layouts, which are referred to as *Std* and *Alt*.

The bitmap fonts are designed for the screen alone and are named *Dyalog Std* and *Dyalog Alt*. The TrueType fonts have a traditional 2741-style italic appearance and are named *Dyalog Std TT* and *Dyalog Alt TT*¹.

The *Std* layout, which was the standard layout for Versions of Dyalog APL up to Version 10.1 contains the APL underscored alphabet **A-Z**. **The underscored alphabet is a deprecated feature and is only supported in this Version of Dyalog APL for backwards compatibility.**

The *Alt* layout, which replaced the *Std* layout as the standard layout for Version 12.0 Classic Edition onwards, does not have the underscored alphabet, but contains additional National Language characters in their place. Note that the extra National Language symbols share the same **AV** positions with the underscored alphabet. If, for example, you switch from the *Std* font layout to the alternative one, you will see the symbol **Á** (A-acute) instead of the symbol **A** (A-underscore).

You may use either a bitmap font or a TrueType font in your APL session (see *User's Guide: Session Operations* for details). You **MUST** use a TrueType font for printing APL functions.

¹The Dyalog Std TT, Dyalog Alt TT, and APL385 Unicode fonts are the copyright of Adrian Smith.

Interoperability

Introduction

Workspaces and component files are stored on disk in a binary format (illegible to text editors). This format differs between machine architectures and among versions of Dyalog. For example a file component written by a PC may well have an internal format that is different from one written by a UNIX machine. Similarly, a workspace saved from Dyalog Version 14.0 will differ internally from one saved by a previous version of Dyalog APL.

It is convenient for versions of Dyalog APL running on different platforms to be able to *interoperate* by sharing workspaces and component files. From Version 11.0, component files and workspaces can generally be shared between Dyalog interpreters running on different platforms. However, this is not always possible, for example:

- Component files created by Version 10.1 can often not be shared across platforms, even when used by later versions.
- *Small-span* (32-bit) component files become read-only when opened on a different architecture from that on which they were created.

Note however that the system function `⎕FCOPY` can be used to make a logically identical copy of an old file, which is fully inter-operable.

The following sections describe other limitations in inter-operability:

Code

Code which is saved in workspaces, or embedded within `⎕OR`s stored in component files, can generally only be read by the version which saved them and later versions of the interpreter. In the case of workspaces, a load (or copy) into an older version would fail with the message:

```
this WS requires a later version of the interpreter.
```

Every time a `⎕OR` object is read by a Version later than that which created it, time may be spent in converting the internal representation into the latest form. Dyalog recommends that `⎕OR` should not be used as a mechanism for sharing code or objects between different versions of APL

"Ordinary" Arrays

With the exception of the Unicode restrictions described in the following paragraphs, Dyalog APL provides inter-operability for arrays which only contain (nested) character and numeric data. Such arrays can be stored in component files - or transmitted using `TCPSocket` objects and Conga connections, and shared between all versions and across all platforms.

As mentioned in the introduction, full cross-platform interoperability of component files is only available for large-span component files (see the following section), and for small-span component files created by Version 11.0 or later.

32 vs. 64-bit Component Files

Large-span (64-bit-addressing) component files are inaccessible to versions of the interpreter that pre-dated their introduction (versions earlier than 10.1).

From version 14.0 onwards it is no longer possible to create small-span (32-bit) files; Version 14.0 is still able to read and write to small span files. Setting the second item of the right argument of `⎕FCREATE` will generate a `DOMAIN ERROR`.

Note that *small-span* (32-bit-addressing) component files cannot contain Unicode data. Unicode editions of Dyalog APL can only write character data which would be readable by a Classic edition (consisting of elements of `⎕AV`).

External Variables

External variables are implemented as small-span (32-bit-addressing) component files, and subject to the same restrictions as these files. External variables are unlikely to be developed further; Dyalog recommends that applications which use them should switch to using mapped files or traditional component files. Please contact Dyalog if you need further advice on this topic.

32 vs. 64-bit Interpreters

From Dyalog APL Version 11.0 onwards, there are two separate versions of programs for 32-bit and 64-bit machine architectures (the 32-bit versions will also run on 64-bit machines running 64-bit operating systems). There is complete inter-operability between 32- and 64-bit interpreters, except that 32-bit interpreters are unable to work with arrays or workspaces greater than 2GB in size.

Unicode vs. Classic Editions

From Version 12.0 onwards, a Unicode edition is available, which is able to work with the entire Unicode character set. Classic editions (a term which includes versions prior to 12.0) are limited to the 256 characters defined in the atomic vector, `□AV`).

Component files have a Unicode property. When this is enabled, all characters will be written as Unicode data to the file. The Unicode property is always off for small-span (32-bit addressing) files, which may not contain Unicode data. For large-span (64-bit addressing) component files, the Unicode property is set *on* by Unicode Editions and *off* by Classic Editions, by default. The Unicode property can subsequently be toggled on and off using `□FPROPS`.

When a Unicode edition writes to a component file which may not contain Unicode data, character data is mapped using `□AVU`, and can therefore be read without problems by Classic editions.

A **TRANSLATION ERROR** will occur if a Unicode edition writes to a non-Unicode component (that is either a 32-bit file, or a 64-bit file when the Unicode property is currently off) if the data being written contains characters which are not in `□AVU`.

Likewise, a Classic edition (Version 12.0 or later) will issue a **TRANSLATION ERROR** if it attempts to read a component containing Unicode data not in `□AVU` from a component file. Version 11.0 cannot read components containing Unicode data and issues a **NONCE ERROR**.

A **TRANSLATION ERROR** will also be issued when a Classic edition `)LOADs` or `)COPYs` a workspace containing Unicode data which cannot be mapped to `□AV` using the `□AVU` in the recipient workspace.

`TCPSocket` objects have an `APL` property which corresponds to the Unicode property of a file, if this is set to `Classic` (the default) the data in the socket will be restricted to `□AV`, if Unicode it will contain Unicode character data. As a result, **TRANSLATION ERRORS** can occur on transmission or reception in the same way as when updating or reading a file component.

AVU changes

The implementation of the function `Right` in Version 13.0 led to the discovery that `AVU` incorrectly defined `AV[59+IO]` as `⌘` (`UCS 164`) rather than `⌞` (Right Tack, `UCS 8866`). This error has been corrected in the default `AVU` and in workspace `AVU.dws`. If you are operating in a mixed Unicode/Classic environment, this error will have caused earlier Classic editions to map `AV[59+IO]` to the wrong Unicode character (`⌘`). This may cause `TRANSLATION ERRORS` when a Version 13.0 Classic system attempts to read the data, as it will not be able to represent `⌘` in the Atomic Vector.

DECFs and Complex numbers

Version 13.0 introduced two new data types; DECFs and Complex numbers. Attempts to read components of these types in earlier interpreters will result in a `DOMAIN ERROR`.

Very large array components

The maximum size (in bytes) of a component written by Version 12.1 and prior is 2GB. This is the size of the component as held on disk which may be different than the size reported by `SIZE`. In Version 13.0 the maximum size of a component written by a 64-bit interpreter is 4GB. From Version 13.2 onwards, the limit on the size of arrays or components is so large that for most practical purposes, there is effectively no limit.

An attempt to read a component greater than 2GB in 32-bit interpreters will result in a `WS FULL`. An attempt to read such a component in 64-bit Versions 12.0 and 12.1 patched after 1st April 2011 will result in a `NONCE ERROR`; earlier patches generate a `FILE COMPONENT DAMAGED` error.

File Journaling

Version 12.0 introduced File Journaling (level 1), and 12.1 added journaling levels 2 and 3 and checksumming. Versions earlier than 12.0 cannot tie files which have any form of journaling or checksumming enabled. Version 12.0 cannot tie files with journaling levels greater than 1, or checksumming enabled. Attempting to tie such files will result in a `FILE NAME ERROR`. Files can be shared with earlier versions by using `FPROPS` to amend the journaling and checksumming levels.

File Component Compression

Version 14.0 introduced File Component Compression; earlier versions will be able to perform all file operation on such files with the exception of being able to `ⓘFREAD` any compressed component. In particular, it is possible for any earlier version to `ⓘFREPLACE` a compressed component with a non-compressed one.

Attempting to read a compressed component using earlier versions of Dyalog APL will generate an error:

- All 13.2 and 13.1.14842 and later:
`DOMAIN ERROR: Array is from a later version of APL`
- 13.1 before revision 14842:
`FILE COMPONENT DAMAGED: Incoming array is invalid`
- 13.0 and 12.1 after revision 11154:
`DOMAIN ERROR`
- 13.0 and 12.1 before revision 11154:
`FILE COMPONENT DAMAGED`

TCP.Sockets

TCP.Sockets used to communicate between differing versions of Dyalog APL are subject to similar limitations to those described above for component files. In particular TCP.Sockets with `'Style' 'APL'` will only be able to pass arrays that are supported by both versions.

Auxiliary Processors

A Dyalog APL process is restricted to starting an AP of exactly the same architecture. In other words, the AP must share the same word-width and byte-ordering as its interpreter process.

Session Files

Session (.dse) files may only be used on the platform on which they were created and saved.

The APL Command Line

The command line for Dyalog APL/W is described below; the command line for UNIX versions of Dyalog APL is documented in the *Dyalog APL for UNIX User Guide*.

Usually the command line is specified in the Target: field of the APL shortcut. The full pathname to the Dyalog executable is usually surrounded by double quotes as it contains spaces.

Command Line

dyalog [options] [debug] [ws] [param] [param] [param]...

where:

[dyalog]

Is the location of the Dyalog executable. Usually this is the full pathname, surrounded by double quotes.

[options]

- x No **DLX** execution on workspace loads.
- a Start in USER mode.
- b Suppress the banner in the Session..
- Fxx Default to creating xx-bit files (where xx is 32 or 64). This option is ignored in version 14.0 onwards.
- s Disable the Session. This option is ignored in Windows versions.
- q Don't quit APL on error (used when piping input into APL).
- c Signifies a command-line comment. All characters to the right are ignored.

[debug]

- Dc Check workspace integrity after every callback function.
- 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 **APLLOG**.

[ws]

The name of a Dyalog APL workspace to be loaded. Unless specified, on Windows the file extension .DWS is assumed.

[param]

A parameter name followed by an equals sign (=) and a value. Note that the parameter name may be one of the standard APL parameters described below, or a name and value of your own choosing (see *Object Reference: GetEnvironment Method*). Note that this differs from Dyalog APL under UNIX, where parameters **must** be passed as environment variables.

Examples:

```
"c:\program files\...\dyalog.exe" myapp maxws=64000
```

```
"c:\program files\...\dyalog.exe" session_file=special.dse
```

```
"c:\program files\...\dyalog.exe -x myapp aplt=mytrans.dot  
default_io=0 myparam=42"
```

APL Exit Codes

When APL or a bound .EXE terminates, it returns an exit code to the calling environment. If APL is started from a desktop icon, the return code is ignored. However, if APL is started from a script (UNIX) or a command processor, the exit code is available and may be used to determine whether or not to continue with other processing tasks. The return codes are:

0	successful <code>ⓘOFF</code> , <code>ⓂOFF</code> , <code>ⓂCONTINUE</code> , graphical exit from GUI
1	APL failed to start. This will occur if there was a failure to read a translate file, there is insufficient memory, or a critical parameter is incorrectly specified or missing.
2	APL was terminated by SIGHUP or SIGTERM (UNIX) or in response to a QUIT WINDOWS request. APL has done a clean exit.
3	APL issued a syserror.
4	Runtime violation. This occurs if a runtime application attempts to read input from the Session. Only a development version has a Session.

Notes:

Under UNIX exit codes greater than 127 indicates (127+signal number) of the untrapped signal which caused the process to terminate.

APL applications can generate a custom return code by specifying an integer value to the right of `ⓘOFF`. Dyalog recommends using values greater than 10 for this purpose.

Configuration Parameters

Introduction

Dyalog APL is customised using a set of configuration parameters which are defined in a registry folder.

In addition, parameters may be specified as environment variables or may be specified on the APL command line.

Furthermore, you are not limited to the set of parameters employed by APL itself as you may add parameters of your own choosing.

Although for clarity parameter names are given here in mixed case, they are case-independent under Windows. Under UNIX and Linux, Dyalog parameters must be specified as environment variables and must be named entirely in upper-case.

Setting Parameter Values

You can change the parameters in 4 ways:

- Using the Configuration dialog box that is obtained by selecting *Configure* from the *Options* menu on the Dyalog APL/W session. See [The Configuration Dialog Box on page 77](#) for details.
- By directly editing the Windows Registry using REGEDIT.EXE or REGEDIT32.EXE.
- By defining the parameters as environment variables.
- By defining the parameters on the APL command line.

This scheme provides a great deal of flexibility, and a system whereby you can override one setting with another. For example, you can define your normal workspace size (*maxws*) in the Registry, but override it with a new value specified on the APL command line. The way this is done is described in the following section.

How APL Obtains Parameter Values

When Dyalog APL/W requires the value of a parameter, it uses the following rules.

1. If the parameter is defined on the APL command line, this value is used.
2. Otherwise, APL looks for an environment variable of the same name and uses this value.
3. Otherwise, if the parameter in question is **infile**, the default value of `Software\Dyalog\Dyalog APL/W 13.0 Unicode (Unicode Edition)` or `Software\Dyalog\Dyalog APL/W 13.0 Classic (Classic Edition)` is assumed.
4. Otherwise, if the parameter in question is **dyalog**, the name of the directory from which the Dyalog APL program was loaded is assumed.
5. The value of any other parameter is obtained from the registry folder defined by the value of **infile**.

Note that the value of a parameter obtained by the `GetEnvironment` method (see *Object Reference: GetEnvironment Method*) uses exactly the same set of rules.

The following section details those parameters that are implemented by Registry Values in the top-level folder identified by **infile**. Values that are implemented in sub-folders are *mainly* internal and are not described in detail here. However, any Value that is maintained via a configuration dialog box will be named and described in the documentation for that dialog box in *The APL Environment*.

Specifying Size-related Parameters

Several of the configuration parameters define sizes.

The value of the parameter must consist of an integer value, optionally followed immediately by a single character which denotes the units to be used. If the value contains no character the units are assumed to be KiB.

Valid values for units are:

K(KiB), M(MiB), G(GiB), T(TiB), P(PiB) and E(EiB).

Specifying an invalid value will prevent Dyalog APL from starting.

AddClassHeaders

This parameter specifies what the Tracer displays when tracing the execution of a function in a script. If set to 1, the Tracer displays just the first line of the script and the function in question. If set to 0, the entire script is shown in the Tracer window.

APL_CODE_E_MAGNITUDE

Version 13.0 introduced decimal floating point numbers which have greater precision than IEEE floating point numbers. This increased the maximum allowable print precision from 17 to 34 and this had the side effect of changing the way numbers in function bodies are descanned¹. For example, the number one sextillion (10^{21}) in a function is descanned by Version 12.1 as **1E21** and by Version 13.0 as **1000000000000000000000**.

Note that only numbers X in the range $(10^{17}) \leq X < (10^{34})$ are affected.

Whilst this change has no other deleterious effect, it means that code that contains such numbers is harder to read, and the result of `⎕CR` (and other character representations) of the same function may have changed between Version 12.1 and Version 13.0 causing undesired affects in code management systems.

The **APL_CODE_E_MAGNITUDE** parameter allows the user to choose between current (Version 13.0 and onwards) and earlier behaviour.

If the **APL_CODE_E_MAGNITUDE** parameter is undefined or set to 0 (the default), numbers are descanned and displayed as normal.

If **APL_CODE_E_MAGNITUDE** is **-1**, numbers greater than or equal to 10^{17} will be displayed using exponential format, as in Version 12.1.

The effect of setting this parameter to any other value is undefined.

¹Descanning refers to the internal process used to convert the internal representation of APL code into a character array. For numbers in function statements, this process uses the maximum value of Print Precision.

APL_COMPLEX_AS_V12

Support for Complex Numbers means that some functions produce different results from older Versions of Dyalog APL. If **APL_COMPLEX_AS_V12** is set to 1 the behaviour of code developed using Version 12.1 or earlier will be unchanged; in particular:

- Power (*****) and logarithm (**⊗**) do not produce Complex Numbers as results from non-complex arguments.
- **⊔VFI** will not honour "J" or "j" as part of a number.
- **⌊4○Y** will be evaluated as **(⌊1+Y*2)*0.5**, which is positive for negative real arguments.

If **APL_COMPLEX_AS_V12** is set to any other value or is not set at all then code developed using version 12.1 or earlier may now generate Complex Numbers.

Note that this feature is provided to simplify the transition of older code to currently supported Versions of Dyalog APL. It does not prevent the generation and use of Complex Numbers using newer features (such as explicitly specifying a Complex Number literal), and the intention is that it will be removed in a future release of Dyalog APL.

APL_FCREATE_PROPS_C

This parameter specifies the default chksum level for newly-created component files. If unspecified, the default checksum level is 1.

APL_FCREATE_PROPS_J

This parameter specifies the default journaling level for newly-created component files. If unspecified, the default journaling level is 1.

APL_FAST_FCHK

This parameter specifies whether Dyalog APL should optimise `⎕FCHK` by allowing it to reliably determine whether a component file had been properly untied and therefore does not need to be checked (this is overridable using the `⎕FCHK` option 'force').

Optimising `⎕FCHK` in this way has a performance impact on `⎕FUNTIE` and it is recommended this optimisation is switched off if your application frequently ties and unties files.

Note: this only affects component files with journaling enabled.

The values of the parameter are:

0	Do not optimise <code>⎕FCHK</code> (optimise <code>⎕FUNTIE</code> instead)
1	Optimise <code>⎕FCHK</code>

The default values of the parameter are 0 on Windows and 1 on Linux/AIX. On Windows, setting the value 1 has no effect.

AplCoreName

This parameter may be a string or a Boolean value. If it is a string, it specifies the directory and name of the file in which the aplcore should be saved. The optional wildcard character (*) is replaced by a unique string when the file is written. For example:

```
APLCORENAME=C:\mycores\aplcore*.dat
```

If **AplCoreName** is set to Boolean 0, the generation of an aplcore file is suppressed and the application simply terminates with an exit code of 3.

aplk

(Classic Edition Only)

This parameter specifies the name of your Input Translate Table, which defines your keyboard layout. The keyboard combo in the *Configure* dialog box displays all the files with the .DIN extension in the APLKEYS sub-directory. You may choose any one of the supplied tables, and you may add your own to the directory. Note that the FILE.DIN table is intended for input from **file**, and should not normally be chosen as a keyboard table. Classic Edition only

aplkeys**(Classic Edition Only)**

This parameter specifies a search path for the Input Translate Table and is useful for configuring a run-time application. The directory paths are specified using Operating System specific conventions and separated by ";" (Windows) or ":" (UNIX). Its default value is the **aplkeys** sub-directory of the directory in which Dyalog APL/W is installed (defined by **dyalog**).

apluid

Under Windows, this parameter specifies the *user number* that is used by the component file system to control file sharing and security. If you wish to share component files and/or external variables in a network it is essential that each user has a unique **apluid** parameter. It may be any integer in the range 0 to 65535. Note that an **apluid** value of 0 causes the user to bypass APL's access control matrix mechanism.

Under UNIX, the *user number* is obtained from the Operating System (UID) and **apluid** is not used. If the user is "root", APL's access control mechanism is ignored.

When a user creates a component file, his *user number* is recorded in the file to identify him as its owner.

aplt

This parameter specifies the name of the Output Translate Table. On Windows the default is WIN.DOT and there is rarely a need to alter it.

apltrans

This parameter specifies a search path for the Output Translate Table and is useful for configuring a run-time application. The directory paths are specified using Operating System specific conventions and separated by ";" (Windows) or ":" (UNIX). Its default value is the sub-directory **apltrans** in the directory in which Dyalog APL/W is installed.

auto_pw

This parameter specifies whether or not the value of `□PW` is derived automatically from the current width of the Session Window. If **auto_pw** is 1, the value of `□PW` changes whenever the Session Window is resized and reflects the number of characters that can be displayed on a single line. If **auto_pw** is 0 (the default) `□PW` is independent of the Session Window size.

AutoFormat

This parameter specifies whether or not you want automatic formatting of Control Structures in functions. The default value is 1 which means that formatting is done automatically for you when a function is opened for editing or converted to text by `□CR`, `□NR` and `□VR`. Automatic formatting first discards all leading spaces in the function body. It then prefixes all lines with a single space except those beginning with a label or a comment symbol (this has the effect of making labels and comments stand out). The third step is to indent Control Structures. The size of the indent depends upon the **TabStops** parameter. To turn off automatic formatting, set **AutoFormat** to 0.

AutoIndent

This parameter specifies whether or not you want semi-automatic indenting during editing. The default value is 1. This means that when you enter a new line in a function, it is automatically indented by the same amount as the previous line. This option simplifies the entry of indented Control Structures.

ClassicMode

This parameter specifies whether or not the Session operates in *Dyalog Classic mode*. The default is 0. If this parameter is set to 1, the Editor and Tracer behave in a manner that is consistent with earlier versions of Dyalog APL.

ClassicModeSavePosition

This parameter specifies whether or not the current size and location of the first of the editor and tracer windows are remembered for next time. It applies only when Classic Mode is enabled.

The size and location of the windows are saved in the registry in the subfolder `WindowRects/EditWindow` and `TraceWindow`.

CMD_PREFIX and CMD_POSTFIX

These parameters defines strings within which operating system commands specified as the arguments to `[CMD` and `[SH`, and `)CMD` and `)SH`, are wrapped. Its purpose is to run the command arguments under a non-standard command shell. This applies to Windows only.

See *Language Reference: Windows Command* for implementation details.

confirm_abort

This parameter specifies whether or not you will be prompted for confirmation when you attempt to abort an edit session after making changes to the object being edited. Its value is either 1 (confirmation is required) or 0. The default is 0.

confirm_close

This parameter specifies whether or not you will be prompted for confirmation when you close an edit window after making changes to the object being edited. Its value is either 1 (confirmation is required) or 0. The default is 0.

confirm_fix

This parameter specifies whether or not you will be prompted for confirmation when you attempt to fix an object in the workspace after making changes in the editor. Its value is either 1 (confirmation is required) or 0. The default is 0.

confirm_session_delete

This parameter specifies whether or not you will be prompted for confirmation when you attempt to delete lines from the Session Log. Its value is either 1 (confirmation is required) or 0. The default is 1.

CreateAplCoreOnSyserror

This parameter specifies whether or not an aplcore file is generated when APL exits with a system error.

default_div

This parameter specifies the value of **□DIV** in a clear workspace. Its default value is 0.

DefaultHelpCollection

If **UseDefaultHelpCollection** is 1, Dyalog attempts to use the Microsoft Document Explorer and online help, for example from Visual Studio (if installed), to display help for external objects, such as .Net Types. In most cases the default settings of "ms-help://ms.mscv.v80" will be sufficient. On some configurations it may be necessary to change this. See [UseDefaultHelpCollection on page 32](#)

default_io

This parameter specifies the value of **□IO** in a clear workspace. Its default value is 1.

default_ml

This parameter specifies the value of **□ML** in a clear workspace. Its default value is 1.

default_pp

This parameter specifies the value of **□PP** in a clear workspace. Its default value is 10.

default_pw

This parameter specifies the value of **□PW** in a clear workspace. Note that **□PW** is a property of the Session and the value of default_pw is overridden when a Session file is loaded.

default_rl

This parameter specifies the value of **□RL** in a clear workspace. It must be a scalar and its default value is 16807.

default_rtl

This parameter specifies the value of **RTL** in a clear workspace. Its default value is 0.

default_wx

This parameter specifies the value of **WX** in a clear workspace. This in turn determines whether or not the names of properties, methods and events of GUI objects are exposed. If set (**WX** is 1), you may query/set properties and invoke methods directly as if they were variables and functions respectively. As a consequence, these names may not be used for global variables in GUI objects.

DMXOUTPUTONERROR

This parameter specifies in which windows DMX error messages are displayed. It is an integer whose value is the sum of the specified windows where 1 = Status Window and 2 = Session Window.

DockableEditWindows

This parameter specifies whether or not individual edit windows can be undocked from (and docked back into) the (MDI) Editor window. Its default value is 0. This parameter does not apply if **ClassicMode** is set to 1.

DoubleClickEdit

This parameter specifies whether or not double-clicking over a name invokes the editor. Its default is 1. If **DoubleClickEdit** is set to 0, double-clicking selects a word and triple-clicking selects the entire line.

dyalog

This parameter specifies the name of the directory in which Dyalog APL is installed.

DyalogEmailAddress

This parameter specifies the contact email address for Dyalog Limited.

DYALOG_EVENTLOGGINGLEVEL

This parameter applies under Windows only, and specifies whether a log entry is written to the Windows Event Log or not when Dyalog APL would pop up a message box due to an unexpected termination of Dyalog APL. See *User's Guide: Handling Unexpected Errors* for more information.

DYALOG_EVENTLOGNAME

This parameter applies under Windows only, and is either the name of the event log to which an event message will be written, or the source of the event message (depending on the registry entries which may or may not have been defined) when Dyalog APL would pop up a message box due to an unexpected termination of Dyalog APL. See *User's Guide: Handling Unexpected Errors* for more information.

DyalogHelpDir

This parameter specifies the full pathname of the directory that contains the Dyalog APL help file (dyalog.chm).

DyalogInstallDir

This parameter specifies the full pathname of the directory in which Dyalog APL is installed.

DYALOG_NOPOPUPS

This parameter specifies whether a MsgBox will appear (0, the default) or will not (1) when Dyalog APL terminates unexpectedly. This applies to APL on Windows only. See *User's Guide: Handling Unexpected Errors* for more information.

DyalogWebSite

This parameter specifies the URL for the Dyalog web site.

edit_cols, edit_rows

These parameters specify the initial size of an edit window in character units.

edit_first_x, edit_first_y

These parameters specify the initial position on the screen of the *first* edit window in character units. Subsequent edit windows will be staggered. These parameters only apply if **ClassicMode** is 1.

edit_offset_x, edit_offset_y

These parameters specify the amount by which an edit window is staggered from the previous one.

ErrorOnExternalException

This is a Boolean parameter that specifies the behaviour when a System Exception occurs in an external DLL. If this parameter is set to 1, and an exception occurs in a call on an external DLL, APL generates an **EXTERNAL DLL EXCEPTION** error (91), instead of terminating with a System Error. This error may be trapped.

EditorState

This is an internal parameter that remembers the state of the last edit window (normal or maximised). This is used to create the next edit window in the appropriate state.

greet_bitmap

This parameter specifies the filename of a bitmap to be displayed during initialisation of the Dyalog APL application. It is used typically to display a product logo from a runtime application. The bitmap will remain until either an error occurs, or it is removed using the GreetBitmap method of the Root object.

```
greet_bitmap=c:\myapp\logo.bmp
```

history_size

This parameter specifies the size of the buffer used to store previously entered (input) lines in the Session. See [Specifying Size-related Parameters on page 13](#) for further details about defining a valid value for this parameter.

infile

This parameter specifies the name of the Windows Registry folder that contains the configuration parameters described in this section. For example,

```
INIFILE=Software\Dyalog\mysettings
```

InitialKeyboardLayout**(Unicode Edition Only)**

This parameter specifies the name of the keyboard to be selected on startup. When you start an APL session, this layout will automatically be selected as the current keyboard layout if the value of **InitialKeyboardLayoutInUse** is 1.

InitialKeyboardLayoutInUse**(Unicode Edition Only)**

This Boolean parameter specifies whether or not the keyboard specified by **InitialKeyboardLayout** is selected as the current keyboard layout when you start an APL session.

InitialKeyboardLayoutShowAll**(Unicode Edition Only)**

This Boolean parameter specifies whether or not all installed keyboards are listed in the choice of keyboards in the Configuration dialog box (Unicode Input tab) .

input_size

This parameter specifies the size of the buffer used to store marked lines (lines awaiting execution) in the Session. See [Specifying Size-related Parameters on page 13](#) for further details about defining a valid value for this parameter.

KeyboardInputDelay

This parameter specifies the delay (in milliseconds) before the system reacts to a user keystroke by:

- updating the name of the Current Object in the Session statusbar. See *User's Guide: The Current Object*.
- offering a list of names for auto-completion. See [Auto Complete Tab on page 98](#)

lines_on_functions

This parameter specifies whether or not line numbers are displayed in edit and trace windows. It is either 0 (the default) or 1.

Note that this parameter determines your overall preference for line numbering, and this setting persists between APL sessions. You can however still toggle line numbering on and off dynamically as required by clicking *Line Numbers* in the *Options* menu on the Session Window. These temporary settings are not saved between APL sessions

localdyalogdir

This parameter specifies the name of the directory in which Dyalog APL/W is installed on the client, in a client/server installation

log_file

This parameter specifies the pathname to the Session log file; it can be absolute or relative to the working directory.

log_file_inuse

This parameter specifies whether or not the Session log is saved in a session log file.

log_size

This parameter specifies the size of the Session log buffer. See [Specifying Size-related Parameters on page 13](#) for further details about defining a valid value for this parameter.

mapchars**(Classic Edition Only)**

In previous versions of Dyalog APL, certain pairs of characters in **⎕AV** were mapped to a single font glyph through the output translate table. For example, the ASCII pipe `|` and the APL style `|` were both mapped to the APL style `|`. From Version 7.0 onwards, it has been a requirement that the mapping between **⎕AV** and the font is strictly one-to-one (this is a consequence of the new native file system). Originally, the mapping of the ASCII pipe and the APL style, the APL and ASCII quotes, and the ASCII `^` and the APL `^` were hard-coded. The mapping is defined by the **mapchars** parameter.

mapchars is a string containing pairs of hexadecimal values which refer to 0-origin indices in **⎕AV**. The first character in each pair is mapped to the second on output. The default value of **mapchars** is `DB0DEBA7EEC00BE0` which defines the following mappings.

From			To		
Hex	Decimal	Symbol	Hex	Decimal	Symbol
DB	219	‘	0D	13	
EB	235	^	A7	167	^
EE	238	⎕	C0	192	
0B	11	.	E0	224	.

To clear all mappings, set `MAPCHARS=0000`.

maxws

This parameter determines your workspace size and is the amount of memory allocated to the workspace at APL start-up. **MAXWS**. See [Specifying Size-related Parameters on page 13](#) for further details about defining a valid value for this parameter.

The default value is 64000(KiB - Kilobytes). Values less than 4MiB are ignored, and the maximum value is 15E. For example, to get a 4GB workspace, set:

```
MAXWS=4G
```

Dyalog APL places no implicit restriction on workspace size, and the virtual memory capability of the underlying operating system allows you to access more memory than you have physically installed. However if you use a workspace that **greatly** exceeds your physical memory you will encounter excessive *paging* and your APL programs will run slowly. You may also cause the system to crash.

Note that the memory used for the workspace must be *contiguous*.

32-bit versions of Dyalog APL are typically limited to about 1.3GiB under Windows, and 1.9GiB under UNIX. These are operating system limitations imposed on 32-bit processes rather than ones imposed by Dyalog APL.

64-bit versions of Dyalog APL have no such limitations; Dyalog has used workspaces of 96GiB on various platforms.

OverstrikesPopup**(Unicode Edition Only)**

This is a Boolean parameter that specifies whether or not the Overstrikes popup is enabled.

PassExceptionsToOpSys

This is a Boolean parameter that specifies the default state of the *Pass Exception* check box in the *System Error* dialog box. See *User's Guide: Handling Unexpected Errors* for more information.

pfkey_size

This parameter specifies the size of the buffer that is used to store programmable function key definitions. See *Language Reference: Program Function Key*. See [Specifying Size-related Parameters on page 13](#) for further details about defining a valid value for this parameter.

ProgramFolder

This parameter specifies the name of the folder in which the Dyalog APL program icons are installed.

PropertyExposeRoot

This Boolean parameter specifies whether or the names of Properties, Methods and Events of the Root object are exposed. If set, you may query/set the Properties of Root and invoke the Root Methods directly as if they were variables and functions respectively. As a consequence, these names may not be used for global variables in your workspace.

PropertyExposeSE

This Boolean parameter specifies whether or the names of Properties, Methods and Events of the Session object are exposed. If set, you may query/set the Properties of `⎕SE` and invoke `⎕SE` Methods directly as if they were variables and functions respectively. As a consequence, these names may not be used for global variables in the `⎕SE` namespace.

qcmd_timeout

This parameter specifies the length of time in milliseconds that APL will wait for the execution of a Windows command to start. Its default value is 5000 milliseconds.

ResolveOverstrikes**(Unicode Edition Only)**

Specifies whether or not the user may enter an APL composite symbol using overstrikes.

RunAsService

When RunAsService is set to 1 (the default is 0) Dyalog APL will not prompt for confirmation when the user logs off, and the interpreter will continue to run across the logoff/logon process.

SaveContinueOnExit

Specifies whether or not your current workspace is saved as `CONTINUE.DWS` before APL terminates.

SaveLogOnExit

Specifies whether or not your Session log is saved before APL terminates.

SaveSessionOnExit

Specifies whether or not your current Session is saved in your Session file before APL terminates.

Serial

Specifies your Dyalog APL/W Serial Number.

session_file

This parameter specifies the name of the file from which the APL session (`⎕SE`) is to be loaded when APL starts. If not specified, a `.dse` extension is assumed. This session file contains the `⎕SE` object that was last saved in it. This object defines the appearance and behaviour of the Session menu bar, tool bar(s) and status bar, together with any functions and variables stored in the `⎕SE` namespace.

SessionOnTop

Specifies whether or not the Session may appear on top of Edit and Trace Windows in Classic Dyalog mode. See [Classic Mode on page 18](#).

ShowStatusOnError

Specifies whether or not the Status window is automatically displayed (if required) when APL attempts to write output to it.

SingleTrace

Specifies whether there is a single Trace window, or one Trace window per function. This applies only if **ClassicMode** is 1.

SkipBlankLines

This parameter causes the Tracer to automatically skip lines that contain no executable statement (i.e. blank lines and comment lines), with the exception of the first line in the function, and in the case of a traditional function (not a dfn), the last line if it is a comment.

StatusOnEdit

Specifies whether or not a status bar is displayed at the bottom of an Edit window.

sm_cols, sm_rows

These parameters specify the size of the window used to display `□SM` when it is used *stand-alone*. They are **not** used if the window is specified using the SM object.

TabStops

This parameter specifies the number of spaces inserted by pressing the Tab key in the editor. Its default value is 4.

trace_cols, trace_rows

These parameters specify the initial size of a trace window in character units.

trace_first_x, trace_first_y

These parameters specify the initial position on the screen of the *first* trace window in character units. Subsequent trace windows will be staggered. This applies only if **ClassicMode** is 1.

trace_offset_x, trace_offset_y

These parameters specify the amount by which a trace window is staggered from the previous one. These apply only if **ClassicMode** is 1 and **SingleTrace** is 0.

Trace_level_warn

This parameter specifies the maximum number of Trace windows that will be displayed when an error occurs and **Trace_on_error** is set to 1. If there are a large number of functions in the state indicator, the display of their Trace windows may take several seconds. This parameter allows you to restrict the potential delay to a reasonable value and its default is 16. If the number of Trace windows would exceed this number, the system instead displays a warning message box. This parameter is ignored if you invoke the Tracer explicitly. This parameter applies only if **ClassicMode** is 1 and **SingleTrace** is 0.

Trace_on_error

This parameter is either 0 (the default) or 1. If set to 1, **Trace_on_error** specifies that the Tracer is automatically deployed when execution of a defined function halts with an error. A stack of Trace windows is immediately displayed, with the top Trace window receiving the input focus.

TraceStopMonitor

This parameter specifies which of the **TRACE** (1), **STOP** (2) and **MONITOR** (4) columns are displayed in Trace and Edit windows. Its value is the sum of the corresponding values.

UnicodeToClipboard**(Classic Edition only)**

This parameter specifies whether or not text that is transferred to and from the Windows clipboard is treated as Unicode text. If **UnicodeToClipboard** is 0 (the default), the symbols in **␣V** are mapped to ASCII text (0-255). In particular, the APL symbols are mapped to ASCII symbols according to their positions in the Dyalog APL font. If **UnicodeToClipboard** is 1, the symbols in **␣V** are mapped to Unicode text and the APL symbols are mapped to their genuine Unicode equivalent values.

UseDefaultHelpCollection

This parameter specifies whether or not Dyalog attempts to use the Microsoft Document Explorer and online help to display help for external objects, such as .Net Types. See [DefaultHelpCollection](#) on page 20.

WantsSpecialKeys**(Unicode Edition Only)**

This parameter specifies a list of applications (e.g. *putty.exe*) that use the command strings in the Input Translate Tables.

WrapSearch

This parameter specifies whether or not Search/Replace in the Editor stops at the bottom or top of the text (depending upon the direction of the search), or continues the search from the start or end as appropriate.

wspath

This parameter defines the workspace path. This is a list of directories that are searched in the order specified when you **)LOAD** or **)COPY** a workspace and when you start an Auxiliary Processor. The directory paths are specified using Operating System specific conventions and separated by ";" (Windows) or ":" (UNIX).

The following Windows example causes **)COPY**, **)LOAD** and **)LIB** to look first in the current directory, then in D:\MYWS, and then in the (normal) *installation workspace* directory.

```
wspath=.;D:\MYWS;C:\Program Files\Dyalog\Dyalog APL 13.2 Unicode\
ws
```

XPLookAndFeel

This parameter specifies whether or not *Native Look and Feel* is used. This affects the appearance of user-interface controls such as Buttons. The default is 1. See [The Configuration Dialog Box on page 77](#).

XPLookAndFeelDocker

This parameter specifies whether or not the title bars in docked windows honour *Native Look and Feel*, if this is enabled at the Windows level. If unspecified, the default is 0.

yy_window

This parameter defines how Dyalog APL is to interpret a 2-digit year number. If **yy_window** is not set (the default) then under Windows, Version 13.2 onwards will adhere to the rules specified in the Windows Region and Language 2-digit year settings.

Dyalog allows a choice of input date formats for **DSM** and GUI edit fields. If you have chosen a 2-digit year format such as MM/DD/YY, then an input of 02/01/00 will by default be interpreted as 1st February 1900 - not 1st February 2000.

If your application uses a 4-digit year format such as YYYY-MM-DD, the problem will not arise.

You can use the **yy_window** parameter to cause your application to interpret 2-digit dates in as required without changing any APL code.

Sliding versus Fixed Window

Two schemes are in common use within the industry: Sliding or Fixed date windows.

Use a Fixed window if there is a *specific year*, for example 1970, before which, dates are meaningless to your application. Note that with a fixed window, this date (say 1970) will still be the limit if your application is running in a hundred years time.

Use a Sliding window if there is a *time period*, for example 30 years, before which dates are considered too old for your application. With a sliding window, you will always be able to enter dates up to (say) 30 years old, but after a while, specific years in the past (for example 1970) will become inaccessible.

Setting a Fixed Window

To make a fixed window, set parameter **yy_window** to the 4-DIGIT year which is the earliest acceptable date. For example:

```
YY_WINDOW=1970
```

This will cause the interpreter to convert any 2-digit input date into a year in the range 1970, 1971 ... 2069

Setting a Sliding Window

To make a sliding window, set parameter **yy_window** to the 1- or 2-DIGIT year which determines the oldest acceptable date. This will typically be negative.

```
YY_WINDOW=-30
```

Conversion of dates now depends on the current year:

If the current year is 1999, the earliest accepted date is $1999-30 = 1969$.

This will cause the interpreter to convert any 2-digit input date into a year in the range 1969, 1970 ... 2068.

However if your application is still running in the year 2010, the earliest accepted date then will be $2010-30 = 1980$. So in the year 2010, a 2-digit year will be interpreted in the range 1980, 1981 ... 2079.

Advanced Settings

You can further restrict date windows by setting an upper as well as lower year limit.

```
YY_WINDOW=1970,1999
```

This causes 2-digit years to be converted only into the range 1970, 1971 ... 1999. Any 2-digit year (for example, 54) not convertible to a year in this range will cause a **DOMAIN ERROR**.

The sliding window equivalent is:

```
YY_WINDOW=-10,10
```

This would establish a valid date window, ten years either side of the current year. For example, if the current year is 1998, the valid range would be $(1998-10) - (1998+10)$, in other words: 1988, 1989, → 2008.

One way of looking at the **yy_window** variable is that it specifies a 2-element vector. If you supply only the first element, the second one defaults to the first element + 99.

Note that the system uses only the number of digits in the year specification to determine whether it refers to a fixed (4-digits) or sliding (1-, or 2-digits) window. In fact you can have a fixed lower limit and a sliding upper limit, or vice versa.

`YY_WINDOW=1990,10`

Allows dates as early as 1990, but not more than 10 years hence.

`YY_WINDOW=0,1999`

Allows dates from the current year to the end of the century.

If the second date is before, or more than 99 years after the first date, then any date conversion will result in a **DOMAIN ERROR**. This might be useful in an application where the end-user has control over the input date format and you want to disallow any 2-digit date input.

`YY_WINDOW=1,0`

Registry Sub-Folders

A large amount of configuration information is maintained in the Windows Registry in sub-folders of the main folder identified by **inifile**.

Many of these values are dynamic, for example the position of the various Session windows, is maintained in a Registry sub-folder so that their appearance is maintained from one invocation of APL to the next. These type of Registry values are considered to be internal and are therefore not described herein.

However, and Registry Value that is maintained via a configuration dialog box will be named and described in the documentation for that dialog box in Chapter 2.

AutoComplete

This contains registry entries that describe your personal AutoComplete options. See [Auto Complete Tab on page 98](#).

Captions

This contains registry entries to customise the Captions used in the various windows of the Dyalog APL IDE. See [Window Captions on page 39](#).

Colours

This contains entries that describe the colour schemes you have and your personal preferences. See [Colour Selection Dialog on page 106](#).

Event Viewer

This contains entries that describe your settings for the Event Viewer. See *User's Guide: The Event Viewer*.

Explorer

This contains entries that describe your settings for the Workspace Explorer. See *User's Guide: The Workspace Explorer Tool*.

files

This contains the size of your recently used file list (see [General Tab on page 77](#)) and the list of your most recently loaded workspaces.

KeyboardShortcuts

This contains the definitions of your Keyboard Shortcuts (Unicode Edition only). See [Keyboard Shortcuts Tab on page 84](#).

LanguageBar

This contains the definitions of the symbols, tips, and help for the symbols in the LanguageBar.

Printing

This contains the entries for your Printer Setup options. See [Print Configuration Dialog Box on page 109](#).

SALT

This contains entries for SALT. See [SALT on page 100](#).

Search

This contains dynamic entries for the Find Objects Tool. See *User's Guide: Find Objects Tool*.

Threads

This contains entries to remember your preferences for Threads. See *User's Guide: The Threads Menu*.

UnicodeIME

This contains entries for the Dyalog Unicode IME.

ValueTips

This contains entries for your Value Tips preferences. See *User's Guide: Value Tips*.

WindowRects

This contains entries to maintain the position of various Session tool windows so that they remain consistent between successive invocations of APL.

Array Editor

The Array Editor stores its settings in the following registry sub-folder:

```
HKEY_CURRENT_USER\Software\DavidLiebtag.com\Array Editor\1.1\  
Options
```

Window Captions

The captions of the various windows that comprise the Dyalog Integrated Development Environment (IDE) are user-configurable and defined by entries in the Windows registry in the *Captions* subkey of the main Dyalog key.

Note that this only applies when the windows are floating (un-docked). When a window is docked Dyalog displays a fixed non-configurable caption.

Note also that the *Captions* subkey is not created by the interpreter; the user must create the subkey and the values.

Each entry is a string value whose name identifies the window as follows:

Window Name	Description
Session	The main Dyalog APL session window
Editor	The Editor window
MessageBox	The notification Message Box that is displayed in various circumstances; for example, when an object cannot be fixed by the Editor
Explorer	The Workspace Explorer tool
Rebuild Errors	The dialog box that is displayed if one or more objects cannot be re-instantiated when a workspace is loaded
Status	The Status window
Refactor	The Refactor as Method/Field/Property window that is displayed by the Editor
Event Viewer	The Event Viewer
FindReplace	The Find/Replace dialog box used by the Editor
ExitDialog	The Exit dialog box that is displayed when the user closes the Session window
WSSearch	The Find Objects tool

Each string value should contain a mixture of your own text and keywords which are enclosed in braces, e.g. {TITLE}. Keywords act like variables and are replaced at display time by corresponding values as described in the table below.

Keyword	Value
{TITLE}	The window name shown in the first column of the previous table
{WSID}	Workspace ID (WSID)
{NSID}	Current Namespace
{SNSID}	Current Namespace (short version)
{PRODUCT}	The name of the Dyalog product, e.g. "Dyalog APL/W - 64"
{VER_A}	The main version number, e.g. "14"
{VER_B}	The secondary version number, e.g. "0"
{VER_C}	The tertiary version number (currently the internal revision number)
{PID}	The process ID
{CHARS}	"Classic" or "Unicode"
{BITS}	"32" or "64"
{XLOC}	The namespace currently being explored (Explorer only)

For example, if the Registry contains `.\Captions\Session` whose value is:

```
My APL ({WSID}) Version {VER_A}.{VER_B}[{VER_C}] - {PID}
```

then the caption displayed in a new Dyalog APL Session window might be:

```
My APL (CLEAR WS) Version 14.0[20105] - 4616
```

Workspace Management

Workspace Size and Compaction

The *maximum* amount of memory allocated to a Dyalog APL workspace is defined by the **maxws** parameter.

Upon **)LOAD** and **)CLEAR**, APL allocates an amount of memory corresponding to the size of the workspace being loaded (which is zero for a clear ws) plus the *workspace delta*.

The workspace delta is $1/16^{\text{th}}$ of maxws, except if there is less than $1/16^{\text{th}}$ of **maxws** in use, delta is $1/64^{\text{th}}$ of **maxws**. This may also be expressed as follows:

```
delta←maxws{⌈α÷÷(ω>α÷16)⌋64 16}ws
```

where **maxws** is the value of the **maxws** parameter and **ws** is the currently allocated amount of workspace. If **maxws** is 16384KB, the workspace delta is either 256KB or 1024 KB, and when you start with a **clear ws** the workspace occupies 256KB.

When you erase objects or release symbols, areas of memory become free. APL manages these free areas, and tries to reuse them for new objects. If an operation requires a contiguous amount of workspace larger than any of the available free areas, APL reorganises the workspace and amalgamates all the free areas into one contiguous block as follows:

1. Any un-referenced memory is discarded. This process, known as *garbage collection*, is required because whole cycles of refs can become un-referenced.
2. Numeric arrays are *demoted* to their tightest form. For example, a simple numeric array that happens to contain only values 0 or 1, is demoted or *squeezed* to have a **DDR** type of 11 (Boolean).
3. All remaining used memory blocks are copied to the low-address end of the workspace, leaving a single free block at the high-address end. This process is known as *compaction*.
4. In addition to any extra memory required to satisfy the original request, an additional amount of memory, equal to the workspace delta, is allocated. This will always cause the process size to increase (up to the **maxws** limit) but means that an application will typically achieve its working process size with at most 4+15 memory reorganisations.
5. However, if after compaction, the amount of used workspace is less than $1/16$ of the Maximum workspace size (**maxws**), the amount reserved for working memory is reduced to $1/64^{\text{th}}$ **maxws**. This means that workspaces that are operating within $1/16^{\text{th}}$ of **maxws** will be more frugal with memory

Note that if you try to create an object which is larger than free space, APL reports **WS FULL**.

The following system function and commands force a workspace reorganisation as described above:

⎕WA,)RESET,)SAVE,)LOAD,)CLEAR

However, in contrast to the above, *any spare workspace above the workspace delta is returned to the Operating System*. On a Windows system, you can see the process size changing by using Task Manager.

The system function **⎕WA** may therefore be used judiciously (workspace reorganisation takes time) to reduce the process size after a particularly memory-hungry operation.

Note that in Dyalog APL, the SYMBOL TABLE is entirely dynamic and grows and shrinks in size automatically. There is no **SYMBOL TABLE FULL** condition.

Additional control of managing the memory used for the workspace can be found in *Language Reference: Memory Management Statistics* and *Language Reference: Specify Workspace Available*.

Interface with Windows

Windows Command Processor commands may be executed directly from APL using the system command **)CMD** or the system function **⎕CMD**. This system function is also used to start other Windows programs. For further details, see the appropriate sections in Language Reference.

Auxiliary Processors

Introduction

Auxiliary Processors (APs) are non-APL programs which provide Dyalog APL users with additional facilities. They run under the control of Dyalog APL.

Typically, APs are used where speed of execution is critical, for utility libraries, or as interfaces to other products. APs may be written in any compiled language, although C is preferred and is directly supported.

Dyalog would recommend that rather than creating APs, customers should now create DLLs (Dynamic Shared Libraries)/shared libraries. If very high performance is required, customers should consider DWA (Direct Workspace Access); contact support@dyalog.com for more information about DWA, including pre-requisite training courses.

Starting an AP

An Auxiliary Processor is invoked using the dyadic form of `⌈CMD`. The left argument to `⌈CMD` is the name of the program to be executed; the value of the `wspath` parameter is used to find the named file. In Dyalog APL/W, the right argument to `⌈CMD` is ignored.

```
'xutils' ⌈CMD ''
```

On locating the specified program, Dyalog APL starts the AP and initialises a memory segment for communication between the workspace and the AP. This communication segment allows data to be passed from the workspace to the other process, and for results to be passed back. The AP then sends APL some information about its external functions (names, code numbers and calling syntax), which APL enters in the symbol table. APL then continues processing while the AP waits for instructions.

Using the AP

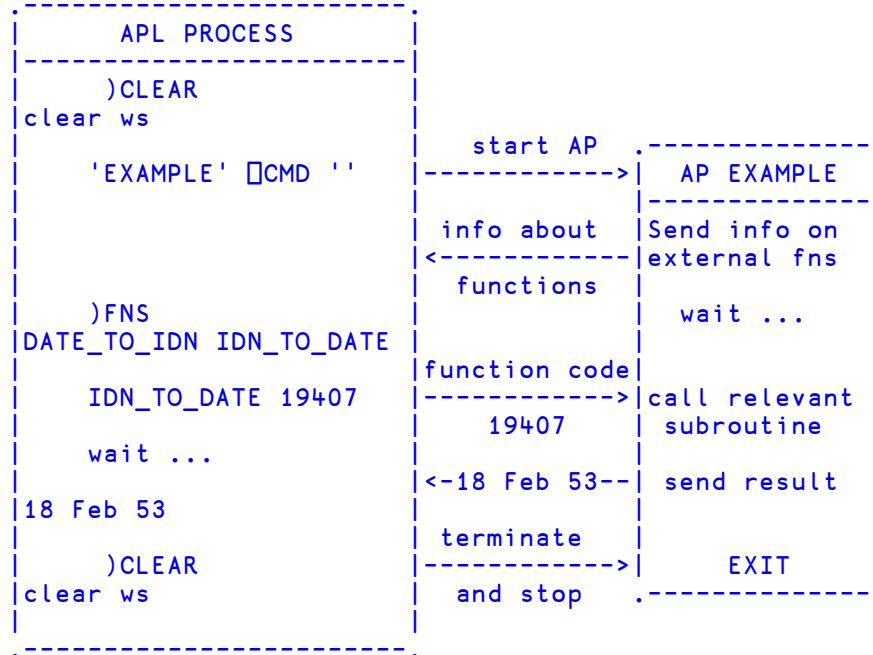
Once established, an AP is used by making a reference to one of its external functions. An external function behaves as if it was a locked defined function, but it is in effect an entry point to the AP. When an external function is referenced, APL transmits a code number to the AP, followed by any arguments. The AP then takes over and performs the desired processing before posting the result back.

Terminating the AP

An AP is terminated when all the last of its external functions is expunged from the active workspace. This could occur with the use of `⌋CLEAR`, `⌋LOAD`, `⌋ERASE`, `⌈EX`, `⌋OFF`, `⌋CONTINUE` or `⌈OFF`.

Example:

Start an Auxiliary Processor called **EXAMPLE**. This fixes two external functions called **DATE_TO_IDN** and **IDN_TO_DATE** which deal with the conversion of International Day Numbers to Julian Dates.



Access Control for External Variables

External variables may be **EXCLUSIVE** or **SHARED**. An exclusive variable can only be accessed by the owner of the file. If you are on a Local Area Network (LAN) a shared external variable may be accessed (concurrently) by other users. The exclusive or shared status of an external variable is set by the **XVAR** function in the UTIL workspace.

Access to an external variable is faster if it has exclusive status than if it is shared. This is because if several users are accessing the file data must always be read and written directly to disk. If it has exclusive status, the system uses buffering and avoids disk accesses where possible.

Creating Executables

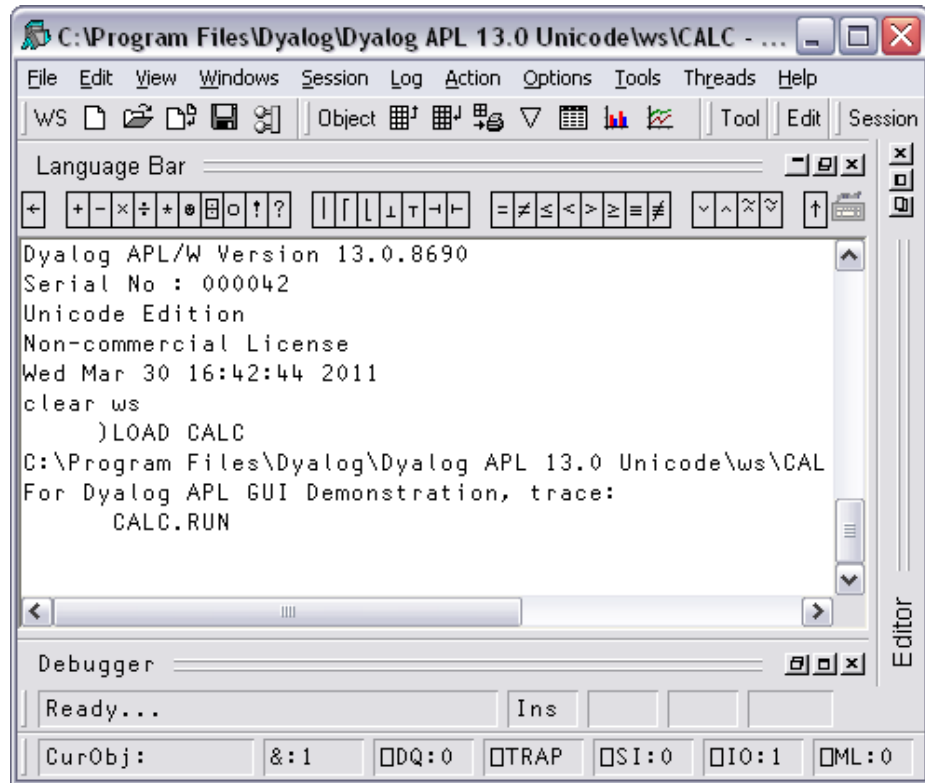
Dyalog APL provides the facility to package an APL workspace as a Windows executable (EXE). This may be done by selecting *Export ...* from the *File* menu of the APL Session window.

The system provides the following options:

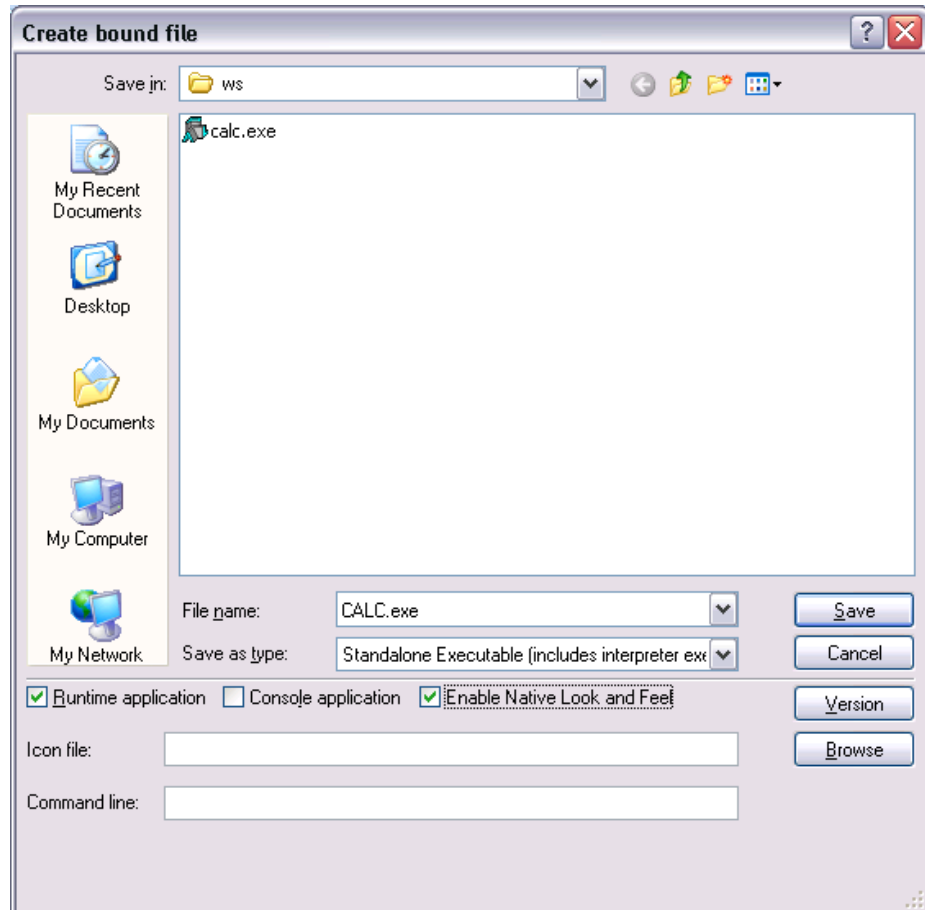
- You may bind your EXE as a Dyalog APL run-time application, or as a Dyalog APL developer application. The second option will allow you to debug the application should it encounter an APL error.
- You may bind your EXE as a console-mode application. A console application does not have a graphical user interface, but runs as a background task using files or TCP/IP to perform input and output.
- You may specify whether or not your .EXE will honour *Native Look and Feel* if this is enabled at the Windows level.

You can package the workspace as a stand-alone executable or as a .EXE file that must be accompanied by the Dyalog APL Dynamic Link Library (dyalog132.dll or dyalog132rt.dll), in which case the DLL should be installed in the same directory (as the EXE) or in the Windows System directory.

The following example illustrates how you can package the supplied workspace `calc.dws` as an executable. Before making the executable, it is essential to set up the latent expression to run the program using `⎕LX` as shown. Notice that in this case it is not necessary to execute `⎕OFF`; the `calc.exe` program will terminate normally when the user closes the calculator window and the system returns to Session input.



Then, when you select *Export...* from the *File* menu, the following dialog box is displayed.



In the example shown, the program is to be saved in `ws`, the (supplied workspaces) directory from which the workspace was loaded (the default).

The *Save as Type* option has been set to *Standalone Executable (includes interpreter exe)* which means that a single `.EXE` will be created containing the Dyalog APL executable and the `CALC` workspace.

The *Runtime application* checkbox is checked, indicating that `calc.exe` is to incorporate the runtime version of Dyalog APL..

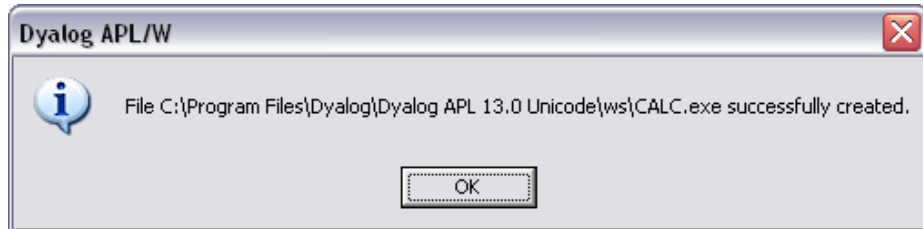
As this is a GUI application, the *Console application* checkbox is left unset.

The *Enable Native Look and Feel* checkbox has been set so that `calc.exe` will honour *Native Look and Feel* if it is enabled at the Windows level.

Note that if you enter the name of a file containing an icon (use the *Browse* button to browse for it) that icon will be bound with your executable and be used instead of the standard Dyalog APL icon.

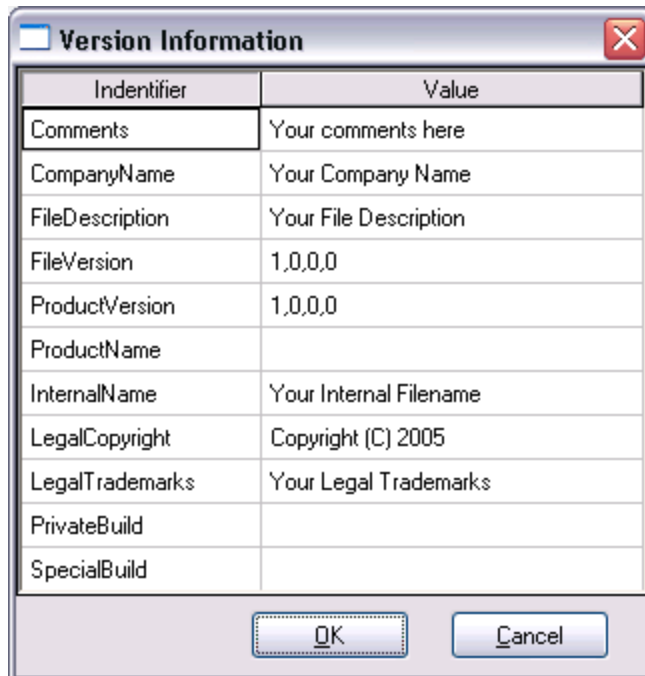
The *Command Line* box allows you to enter parameters and values that are to be passed to your executable when it is invoked.

On clicking *Save*, the following message box is displayed to confirm success.



Version Information

You may embed version information into your .exe by clicking the *Version* button and then completing the *Version Information* dialog box that is illustrated below.



Run-Time Applications and Components

Using Dyalog APL you may create different types of run-time applications and components. Note that the distribution of run-time applications and components requires a Dyalog APL Run-Time Agreement. Please contact Dyalog or your distributor, or see the Dyalog web page for more information.

The following table shows a list of distributable components for the two Editions. These are referred to in the remainder of this Chapter by the name shown in the first column of the table. It is essential that you distribute the components that are appropriate for the Edition you are using.

Name	File
32-bit Unicode	Dyalog APL 14.0 Unicode\
Run-Time EXE	dyalogrt.exe
Run-Time DLL	dyalog140rt_unicode.dll
Bridge DLL	bridge140_unicode.dll
Dyalog DLL	dyalog32.dll
DyalogProvider DLL	dyalogprovider.dll
DyalogNet DLL	dyalognet.dll
APLScript Compiler	dyalogc_unicode.exe
32-bit Classic	Dyalog APL 14.0 Classic\
Run-Time EXE	dyalogrt.exe
Run-Time DLL	dyalog140rt.dll
Bridge DLL	bridge140.dll
Dyalog DLL	dyalog32.dll
DyalogProvider DLL	dyalogprovider.dll
DyalogNet DLL	dyalognet.dll
APLScript Compiler	dyalogc.exe

Name	File
64-bit Unicode	Dyalog APL-64 14.0 Unicode\
Run-Time EXE	dyalogrt.exe
Run-Time DLL	dyalog140_64rt_unicode.dll
Bridge DLL	bridge140-64_unicode.dll
Dyalog DLL	dyalog64.dll
DyalogProvider DLL	dyalogprovider.dll
DyalogNet DLL	dyalognet.dll
APLScript Compiler	dyalogc64_unicode.exe
64-bit Classic	Dyalog APL-64 14.0 Classic\
Run-Time EXE	dyalogrt.exe
Run-Time DLL	dyalog140_64rt.dll
Bridge DLL	bridge140-64.dll
Dyalog DLL	dyalog64.dll
DyalogProvider DLL	dyalogprovider.dll
DyalogNet DLL	dyalognet.dll
APLScript Compiler	dyalogc64.exe

Stand-alone run-time

This is the simplest type of run-time to install. Using the *File/Export* menu item on the Session window, you can create a standard Windows executable program file (EXE) which contains your workspace and the Run-Time version of the Dyalog APL interpreter. To distribute your application, you need to supply and install:

1. your bound executable (EXE)
2. whatever additional files that may be required by your application

The command-line for your application should simply invoke your EXE, with whatever start-up parameters it may require. Note that your application icon and any start-up parameters for the Run-Time Interpreter are specified and bound with the EXE when you make it.

If your application uses any component of the Microsoft .Net Framework, you must distribute the Bridge DLL and DyalogNet DLLs. These DLLs must either be on the system path or placed in the same directory as your EXE. If you are going to use your application with ASP.NET, the DLLs must also be installed in the global assembly cache (GAC) using the `gacutil.exe` utility program.

Bound run-time

This option requires the separate installation of the Run-Time DLL, but compared with the stand-alone executable option, may save disk space and memory if your customer installs and runs several different Dyalog applications. Using the File/Export menu item on the Session window, you can create a standard Windows executable program file (EXE) which contains your workspace bound to the Run-Time DLL. To distribute your application, you need to supply and install:

1. your bound executable (EXE)
2. The Run-Time DLL

whatever additional files that may be required by your application

The command-line for your application should simply invoke your EXE, with whatever start-up parameters it may require. Note that your application icon and any start-up parameters for the Run-Time DLL are specified and bound with the EXE when you make it.

If your application uses any component of the Microsoft .Net Framework, you must distribute the Bridge DLL and DyalogNet DLLs. These DLLs must either be on the system path or placed in the same directory as your EXE. If you are going to use your application with ASP.NET, the DLLs must also be installed in the global assembly cache (GAC) using the `gacutil.exe` utility program.

Workspace based run-time

A workspace based run-time application consists of the Dyalog APL Run-Time Program (Run-Time EXE) and a separate workspace. To distribute your application, you need to supply and install:

1. your workspace
2. the Run-Time EXE
3. whatever additional files that may be required by your application

The command-line for your application invokes the Run-Time EXE, passing it start-up parameters required for the Run-Time EXE itself (such as MAXWS) and any start-up parameters that may be required by your application. You will need to associate your own icon with your application during its installation.

If your application uses any component of the Microsoft .Net Framework, you must distribute the Bridge DLL and DyalogNet DLLs. These DLLs must either be on the system path or placed in the same directory as your EXE. If you are going to use your application with ASP.NET, the DLLs must also be installed in the global assembly cache (GAC) using the `gacutil.exe` utility program.

Out-of-process COM Server

To make an out-of-process COM Server, you must:

1. establish one or more OLEServer namespaces in your workspace, populated with functions and variables that you wish to export as methods, properties and events.
2. use the *File/Export ...* menu item on the Session window to register the COM Server on your computer so that it is ready for use.

The command-line for your COM Server invokes the Run-Time EXE, passing it the start-up parameters required for the Run-Time EXE itself (such as MAXWS) and any start-up parameters that may be required by your application.

To distribute an out-of-process COM Server, you need to supply and install the following files:

1. your workspace
2. the associated Type Library (.tlb) file (created by *File/Export*)
3. the Run-Time EXE
4. whatever additional files that may be required by your application

To install an out-of-process COM Server you must set up the appropriate Windows registry entries. See Interface Guide for details.

In-process COM Server

To make an in-process COM Server, you must:

1. establish one or more OLEServer namespaces in your workspace, populated with functions and variables that you wish to export as methods, properties and events.
2. use the *File/Export ...* menu item on the Session window to create an in-process COM Server (DLL) which contains your workspace bound to the Run-Time DLL. This operation also registers the COM Server on your computer so that it is ready for use.

To distribute your component, you need to supply and install

Your COM Server file (DLL)

1. the Run-Time DLL
2. whatever additional files that may be required by your COM Server.

Note that you must register your COM Server on the target computer using the `regsvr32.exe` utility.

ActiveX Control

To make an ActiveX Control, you must:

1. establish an ActiveXControl namespace in your workspace, populated with functions and variables that you wish to export as methods, properties and events.
2. use the *File/Export ...* menu item on the Session window to create an ActiveX Control file (OCX) which contains your workspace bound to the Run-Time DLL. This operation also registers the ActiveX Control on your computer so that it is ready for use.

To distribute your component, you need to supply and install

Your ActiveX Control file (OCX)

1. the Run-Time DLL
2. whatever additional files that may be required by your ActiveX Control.

Note that you must register your ActiveX Control on the target computer using the `regsvr32.exe` utility.

Microsoft .Net Assembly

A Microsoft .Net Assembly contains one or more .Net Classes. To make a Microsoft .Net Assembly, you must:

1. establish one or more NetType namespaces in your workspace, populated with functions and variables that you wish to export as methods, properties and events.
2. use the *File/Export ...* menu item on the Session window to create a Microsoft .Net Assembly (DLL) which contains your workspace bound to the Run-Time DLL.

To distribute your .Net Classes, you need to supply and install

1. your Assembly file (DLL)
2. the Run-Time DLL
3. the Bridge DLL
4. the DyalogNet DLL
5. whatever additional files that may be required by your .Net Assembly.
6. the Bridge DLL and DyalogNet DLLs must either be on the system path or placed in the same directory as your EXE. If you are going to use your Assembly with ASP.NET, the DLLs must also be installed in the global assembly cache (GAC) using the `gacutil.exe` utility program.

Data Binding

If your runtime application makes use of Data Binding, you will also need to include `dyalogdata4.5.dll`.

The `dyalogdata4.5.dll` provides advanced support for Data Binding and Syncfusion. In particular it provides the `INotifyCollectionChanged` interface which is required to support data binding of collections and lists. This DLL requires .NET Version 4.5 and is not used unless .NET 4.5 is enabled.

Additional Files for Syncfusion

Under a licensing agreement with Syncfusion, Dyalog includes the Syncfusion library of WPF controls. These may be used by Dyalog APL users to develop applications, and may be distributed with Dyalog APL run-time applications.

The Syncfusion libraries comprise a set of .NET assemblies which are supplied in the *Syncfusion/4.5* sub-directory of the main Dyalog APL installation directory (for example: *c:\Program Files\Dyalog\Dyalog APL-64 14.0 Unicode\Syncfusion\4.5*).

If you use any of the Syncfusion controls in your runtime application, you must include the SyncFusion library.

Additional Files for SQAPL

If your application uses the *SQAPL/EL ODBC* interface, you must distribute and install four additional files, according to the Edition you are using, as shown in the tables below.

Name	File
32-bit Unicode	Dyalog APL 14.0 Unicode\
SQAPL INI	sqapl.ini
SQAPL ERR	sqapl.err
SQAPL DLL	cwdya62u32w.dll
APLUNICD INI	aplunlcd.ini
32-bit Classic	Dyalog APL 14.0 Classic\
SQAPL INI	sqapl.ini
SQAPL ERR	sqapl.err
SQAPL DLL	cwdya62c32w.dll
APLUNICD INI	aplunlcd.ini
Name	File
64-bit Unicode	Dyalog APL-64 14.0 Unicode\
SQAPL INI	sqapl.ini
SQAPL ERR	sqapl.err
SQAPL DLL	cwdya62u64w.dll
APLUNICD INI	aplunlcd.ini
64-bit Classic	Dyalog APL-64 14.0 Classic\
SQAPL INI	sqapl.ini
SQAPL ERR	sqapl.err
SQAPL DLL	cwdya62c64w.dll
APLUNICD INI	aplunlcd.ini

The SQAPL DLL must be installed in the user's Windows directory or be on the user's path.

Miscellaneous Other Files

AUXILIARY PROCESSORS

If you use any of the Auxiliary Processors (APs) included in the sub-directory `xutils`, you must include these with your application. Note that, like workspaces, Dyalog APL searches for APs using the **wspath** parameter. If your application uses APs, you must ensure that you specify **wspath** or that the default **wspath** is adequate for your application..

DYALOG32 and/or DYALOG64

This DLL is used by some of the functions provided in the `QUADNA.DWS` workspace. If you include any of these in your application this DLL must be installed in the user's Windows directory or be on the user's path.

Registry Entries for Run-Time Applications

The Run-Time DLL does not obtain any parameter values from the Windows registry. If you need to specify any Dyalog APL parameter values, they must be defined in the command line when you create an EXE.

The Run-Time EXE *does* obtain parameter values for the Windows registry, but does not require them to be present. If the default values of certain parameters are inappropriate, you may specify their values on the command line. There is normally no requirement to install registry entries for a run-time application that uses the Run-Time EXE.

For example, your application may require a greater or lesser **maxws** parameter (workspace size) than the default value. This may be done by adding the phrase `MAXWS=nnnn` (where `nnnn` is the required workspace size) after the name of your application workspace on the command line, for example:

```
dyalogrt.exe MYAPP.DWS MAXWS=200M
```

Note that the default value of the **DYALOG** parameter (which specifies where it looks for various other files and sub-directories) is the directory from which the application (`dyalogrt.exe`) is loaded.

Nevertheless, registry entries will be required in the following circumstances.

1. If your Classic Edition run-time application requires that the user inputs APL characters, you will need to specify input/output tables (parameters **APLK**, **APLT**, **APLKEYS** and **APLTRANS**).

Installing Registry Entries

To specify parameters using the Registry, you must install a suitable registry folder for each user of your application. By default, Version 14.0 will use registry folders similar to:

```
HKEY_CURRENT_USER\Software\Dyalog\  
Dyalog APL/W 14.0 Unicode
```

or

```
HKEY_LOCAL_MACHINE\SOFTWARE\Dyalog\  
Dyalog APL/W-64 14.0
```

You may choose a different name for your registry folder if you wish. If so, you must tell Dyalog APL the name of this folder by specifying the **INIFILE** parameter on the command line. For example:

```
dyalogrt.exe myapp.dws INIFILE=Software\MyCo\MyApplication
```

You may install entries into the registry folder in one of two ways:

1. Using a proprietary installation program such as *InstallShield*
2. Using the REGEDIT utility. This utility program installs registry entries defined in a text file that is specified as the argument to the program. For example, if your file is called `APLAPP.REG`, you would install it on your user's system by executing the command:

```
REGEDIT APLAPP.REG
```

An example 5-line file that specifies the **APLNID** and **MAXWS** parameters might be as follows:

```
Windows Registry Editor Version 5.00
```

```
[HKEY_CURRENT_USER\Software\Dyalog\Dyalog APL/W 14.0]  
"aplnid"="42"  
"maxws"="8096"
```

COM Objects and the Dyalog APL DLL

Introduction

In each Edition, there are two versions of the Dyalog APL Dynamic Link Library, named `dyalog140_unicode.dll` and `dyalog140rt_unicode.dll` (Unicode Edition) and `dyalog140.dll` and `dyalog140rt.dll` (Classic Edition).

`dyalog140_unicode.dll` and `dyalog140.dll` are complete Dyalog APL development systems packaged as Dynamic Link Libraries.

`dyalog140_unicode.dll` and `dyalog140rt_unicode.dll` and `dyalog140rt.dll` are the run-time versions of `dyalog140.dll`.

In the remainder of this section, the term *the Dyalog APL DLL* is used to refer to any one of these DLLs. The term *COM object* is used to refer to a Dyalog APL in-process OLE Server (OLEServer object) or a Dyalog APL ActiveX Control (ActiveXControl object).

The Dyalog APL DLL is used to host COM objects and .Net objects written in Dyalog APL. Although this section describes how it operates with COM objects, much of this also applies when it hosts .Net objects. Further information is provided in the *.Net Interface Guide*.

Classes, Instances and Namespace Cloning

A COM object, whether written in Dyalog APL or not, represents a class. When a host application loads a COM object, it actually creates an instance of that class.

When a host application creates an instance of a Dyalog APL COM object, the corresponding OLEServer or ActiveXControl namespace is cloned. If the host creates a second instance, the original namespace is cloned a second time.

Cloned OLEServer and ActiveXControl namespaces are created in almost exactly the same way as those that you can make yourself using `⎕OR` and `⎕WC` except that they do not have separate names. In fact, each clone believes itself to be the one and only original OLEServer or ActiveXControl namespace, with the same name, and is completely unaware of the existence of other clones.

Notice that cloning does not initially replicate all the objects within the OLEServer or ActiveXControl namespace. Instead, the objects inside the cloned namespaces are actually represented by pointers to the original objects in the original namespace. Only when an object is changed does any information get replicated. Typically, the only objects likely to differ from one instance to another are variables, so only one copy of the functions will exist in the workspace. This design enables many instances of a Dyalog APL COM object to exist without overloading the workspace.

Workspace Management

By default, the Dyalog APL DLL does not use a fixed maximum workspace size, but automatically increases the size of its active workspace as required. If you write a run-away COM object, or if there is insufficient computer memory available to load a new control, it is left to the host application or to Windows itself to deal with the situation.

Nevertheless, it is possible to specify a value for **MAXWS** for the application in which the Dyalog APL DLL is embedded. This is achieved by defining a Registry key named:

```
HKLM\Software\Dyalog\Embedded\<appname>
```

where <appname> is the name of the application, containing a String Value named `maxws` set to the desired size. If you were running an APL in-process server from Microsoft Excel, the application name would be `excel.exe`.

When an application loads its first Dyalog APL COM object, it starts the Dyalog APL DLL which initialises a **CLEAR WS**. It then copies the namespace tree for the appropriate OLEServer or ActiveXControl object into its active workspace.

This namespace tree comprises the OLEServer or ActiveXControl namespace itself, together with all its parent namespaces *with the exception of* the root workspace itself. Note that for an ActiveXControl, there is at least one parent namespace that represents a Form.

For example, if an ActiveXControl namespace is called `#.F.Dual`, the Dyalog APL DLL will copy the contents of `#.F` into its active workspace when the first instance of the control is loaded by the host application.

If the same host application creates a *second instance* of the *same* OLEServer or ActiveXControl, the original namespace is cloned as described above and there is no further impact on the workspace.

If the same host application creates an instance of a *different* Dyalog APL COM object, the namespace tree for this second object is copied from its DLL or OCX file into the active workspace. For example, if the second control was named `X.Y.MyControl`, the entire namespace `X` would be copied.

This design raises a number of points:

1. Unless you are in total control of the user environment, you should design a Dyalog APL COM object so that it can operate in the same workspace as another Dyalog APL COM object supplied by another author. You cannot make any assumptions about file ties or other resources that are properties of the workspace itself.
2. If you write an ActiveXControl whose ultimate parent namespace is called **F**, a host application could not use your control at the same time as another ActiveXControl (perhaps supplied by a different author) whose ultimate parent namespace is also called **F**.
3. Dyalog APL COM objects must not rely on variables or utility functions that were present in the root workspace when they were saved. These functions and variables will *not* be there when the object is run by the Dyalog APL DLL.
4. A Dyalog APL COM object may *create* and subsequently *use* functions and variables in the root workspace, but if two different COM objects were to adopt the same policy, there is a danger that they would interfere with one another. The same is true for **SE**.

Multiple COM Objects in a Single Workspace

If your workspace contains several OLEServer or ActiveXControl objects which have the same ultimate parent namespace, the Dyalog APL DLL will copy them all into the active workspace at the time when the first one is instantiated. If the host application requests a second COM object that is already in the workspace, the namespace tree is not copied again.

If the workspace contains several OLEServer or ActiveXControl objects which have different ultimate parents, their namespace trees will be copied in separately.

Parameters

With the exception of **maxws** (see above) the Dyalog APL DLL does not read parameters from the registry, command-line or environment variables. This means that all such parameters will have their default values.

APL Application as a Service

Introduction

Dyalog APL provides a mechanism for users to register and manage an application workspace as a Windows service. The application workspace must implement an interface to handle messages from the Windows Service Control Manager (SCM) in addition to the code required to drive the application.

Windows Services run as background tasks controlled by the SCM. When the computer is started, Windows Services are run before a user logs on to the system and do not normally interact with the desktop. A Dyalog service is run under the auspices of *Local System*.

Installing and Uninstalling a Dyalog Service

To install a Dyalog service it is necessary to run `dyalog.exe` from the command line with administrator privileges, specifying the application workspace and the following parameters, where *service_name* is a name of your choice.

- **APL_ServiceInstall=service_name**

The command must specify the full pathname to `dyalog.exe` and to the application workspace. A slightly modified version of this command line will be stored by the SCM and re-executed whenever the service is started.

Dyalog installs the service with a *Startup Type* of *Automatic*. This means that it will be started automatically whenever the computer is restarted. However, it is necessary to start it manually (using the SCM) the first time after it is installed.

The same command must be used to uninstall the service, but with:

- **APL_ServiceUninstall=service_name**

The following table summarises the parameters that can be specified by the user. Other parameters will appear on the command line in the SCM, but should not be specified by the user.

Parameter	Description
APL_ServiceInstall	Causes Dyalog to register the named service, using the current command line, but with APL_ServiceRun replacing APL_ServiceInstall in the SCM.
APL_ServiceUninstall	Causes Dyalog to uninstall the named service.

The Application Workspace

The application workspace must be designed to handle and respond (in a timely manner) to notification messages from the SCM as well as to provide the application logic. SCM notifications include instructions to start, stop, pause and resume.

SCM notification messages generate a ServiceNotification event on the Root object. To handle these messages, it is necessary to attach a callback function to this event, and to invoke the Wait method or `□DQ'.'` to process them. This must be executed in thread 0.

If the application is designed to be driven from events such as Timer or TCPSocket or user-defined events, it too may be implemented via callbacks in thread 0 under the control of the same Wait method or `□DQ'.'`. If the application uses Conga it is recommended that it runs in a separate thread.

The workspace `samples\aplservice\aplservice.dws` is included in the APL release. Its start-up function is as follows:

```

□IX←'Start'

▽ Start;ServiceState;ServiceControl
[1]   :If 'W'≠3>#.□WG'APLVersion'
[2]   □←'This workspace only works using Dyalog APL for
      Windows version 14.0 or later'
[3]   :Return
[4]   :EndIf
[5]   :If 0εp2 □NQ'.' 'GetEnvironment' 'RunAsService'
[6]   Describe
[7]   :Return
[8]   :EndIf
[9]   A Define SCM constants
[10]  HashDefine
[11]  A Set up callback to handle SCM notifications
[12]  '.'□WS'Event' 'ServiceNotification' 'ServiceHandler'
[13]  A Global variable defines current state of the service
[14]  ServiceState←SERVICE_RUNNING
[15]  A Global variable defines last SCM notification to the
      service
[16]  ServiceControl←0
[17]  A Application code runs in a separate thread
[18]  Main&0
[19]  □DQ'.'
[20]  □OFF
▽

```

Handling ServiceNotification Events

To give the workspace (which may be busy) time to respond to SCM notifications, Dyalog responds immediately to confirm that the service has entered the appropriate pending state. For example, if the notification is `SERVICE_CONTROL_STOP`, Dyalog informs the SCM that the service state is `SERVICE_STOP_PENDING`. It is then up to the callback function to confirm that the state has reached `SERVICE_STOPPED`.

The following sample function is provided in `APLService.dws`.

ServiceHandler Callback Function

```

▽ r←ServiceHandler(obj event action state);sink
[1]  A Callback to handle notifications from the SCM
[2]
[3]  A Note that the interpreter has already responded
[4]  A automatically to the SCM with the corresponding
[5]  A "_PENDING" message prior to this callback being reached
[6]
[7]  A This callback uses the SetServiceState Method to confirm
[8]  A to the SCM that the requested state has been reached
[9]
[10] r←0  A so returns a 0 result (the event has been handled,
[11]       A no further action required)
[12]
[13]  A It stores the desired state in global ServiceState to
[14]  A notify the application code which must take appropriate
[15]  A action. In particular, it must respond to a "STOP" or
[16]  A "SHUTDOWN" by terminating the APL session
[17]
[18]  :Select ServiceControl←action
[19]  :CaseList SERVICE_CONTROL_STOP SERVICE_CONTROL_SHUTDOWN
[20]      ServiceState←SERVICE_STOPPED
[21]      state[4 5 6 7]←0
[22]
[23]  :Case SERVICE_CONTROL_PAUSE
[24]      ServiceState←SERVICE_PAUSED
[25]
[26]  :Case SERVICE_CONTROL_CONTINUE
[27]      ServiceState←SERVICE_RUNNING
[28]  :Else
[29]      :If state[2]=SERVICE_START_PENDING
[30]          ServiceState←SERVICE_RUNNING
[31]      :EndIf
[32]  :EndSelect
[33]  state[2]←ServiceState
[34]  sink←2 □NQ'.' 'SetServiceState' state
▽

```

The Application Code

The following function illustrates how the application code for the service might be structured. It is merely an illustration, but however it is done, it is important that the code handles the instructions to pause, continue and stop in an appropriate manner. In this example, the function **Main** creates a log file and writes to it when the state of the service changes.

```

▽ Main arg;nid;log;LogFile
[1]  □NUNTIE □NNUMS
[2]  log←{((⌘TS),' ',ω,□UCS 13 10)□NAPPEND α}
[3]  LogFile←'c:\ProgramData\TEMP\APLServiceLog.txt'
[4]  :Trap 22
[5]      nid←LogFile □NCREATE 0
[6]  :Else
[7]      :Trap 22
[8]          nid←LogFile □NTIE 0
[9]          0 □NRESIZE nid
[10]  :Else
[11]      □←'Unable to tie or create logfile'
[12]  :EndTrap
[13] :EndTrap
[14] nid log'Starting'
[15] :While ServiceState≠SERVICE_STOPPED
[16]     :If ServiceControl≠0 ♦
[17]         nid log'ServiceControl=',⌘ServiceControl ♦ :EndIf
[18]     :If ServiceState=SERVICE_RUNNING
[19]         nid log'Running'
[20]     :ElseIf ServiceState=SERVICE_PAUSED
[21]         α Pause application
[22]     :EndIf
[23]     ServiceControl←0 α Reset (we only want to log changes)
[24]     □DL 10 α Just to prevent busy loop
[25] :EndWhile
[26] □NUNTIE nid
    □OFF 0
▽

```

Debugging Dyalog Services

Services are run in the background under the auspices of *Local System*, and not associated with an interactive user. Neither the APL Session nor any GUI components that it creates will be visible on the desktop. This prevents the normal editing and debugging tools from being available.

However, the Dyalog APL Remote Integrated Development environment (RIDE) may be connected to any APL session, including one running as a Windows Service, and provide a debugging environment. For more information, see the *Dyalog RIDE User Guide*.

Event Logging

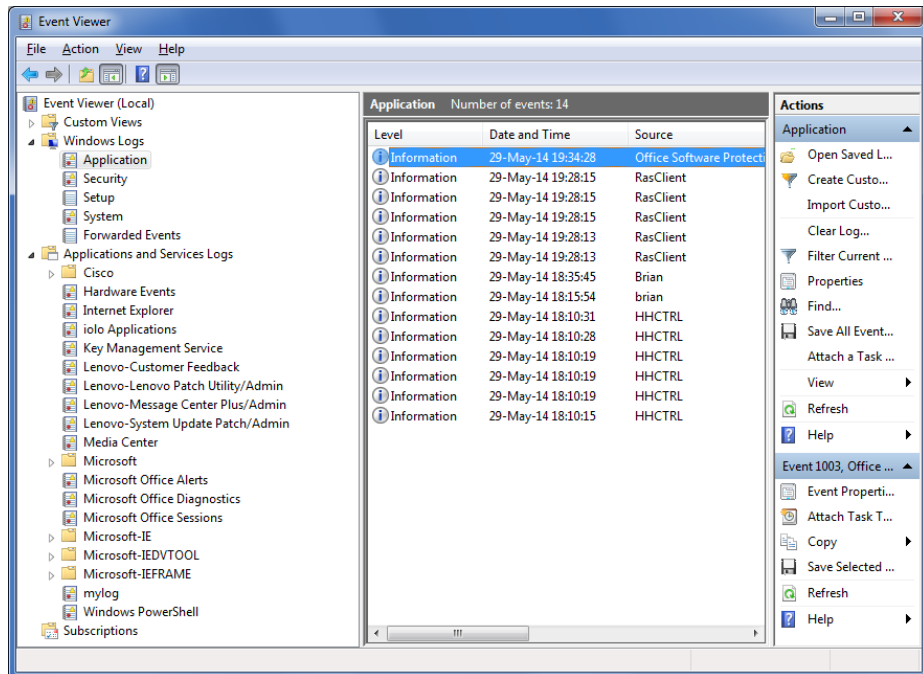
When a service is installed or removed, Dyalog APL records events in the Dyalog APL section of the *Applications and Services Logs* which can be viewed using the Windows system *Event Viewer*.

APLService Logging Events

The `APLService.dws` workspace contains the class `SysLog` which can be used to log events to the Windows Event Log. These events can be accessed programmatically or viewed using the Windows Event Viewer found in the Windows Administrative Tools.

Windows Event Log Concepts

Every message logged in the Windows Event Log has a named source. Frequently this source will be the name of the application which generates the message. Windows has multiple event log files. By default, messages will be logged in the Application log file found in the Windows Logs section of the Windows Event Viewer. Alternatively, you can create a custom log located in the Applications and Services Logs section in the Windows Event Viewer as shown by the "mylog" entry in the screenshot below. Multiple applications can use the same source and multiple sources can write to the same log file, but a given source may only write to a single log file.



Using SysLog in Your Application

Before deploying your Dyalog APL application as a service, you should:

1. Consider what events or messages the application should log and their severity level. **SysLog** allows you to specify severity levels of Error, Warning, and Informational.
2. Create the log source and optionally its custom log using **SysLog.CreateEventSource**. This must be done when running Dyalog as an administrator and prior to running your Dyalog service. Once the event source is created, it is not necessary to run your application as an administrator in order to write to the Windows Event Log.
3. Within your application, you have two options for writing to the Windows Event Log.
 - a. You may use the **SysLog.WriteLog** method. **SysLog.WriteLog** will verify that the log source exists and then write your message. This has the advantage of being standalone and can be called whenever you desire.
 - b. You may create an instance of the **SysLog** class and use the **Write** method. This has the advantage of not incurring the overhead of verifying the existence of the log source each time a log message is written

SysLog Usage

SysLog implements an interface to a subset of the functionality of Microsoft's `System.Diagnostics.EventLog` class. Some of **SysLog**'s methods, namely **CreateEventSource**, **DeleteEventSource** and **DeleteLog**, require you to run Dyalog **as an administrator** to be fully functional.

All of the methods in **SysLog** with the exception of **Write** are shared methods meaning you do not have to create an instance of **SysLog** in order to execute them.

SysLog.CreateEventSource sourcename {logname}**Purpose:**

Creates a new Windows Event Log source and optionally specifies or creates a Windows Event Log for the source.

Argument	Description
<code>sourcename</code>	character vector source name that does not already exist
<code>{logname}</code>	optional character vector log name with which to associate the source name. If not supplied, the source will be associated with the Windows Logs/Application log. If there is no log named <code>logname</code> , it will be created.

{level} SysLog.WriteLog sourcename message**Purpose:**

Writes a message to the Windows Event Log associated with `sourcename`, optionally specifying a severity level.

Argument	Description
<code>sourcename</code>	character vector source name of an existing source
<code>message</code>	character vector message to write to the log
<code>{level}</code>	optional singleton indicating the severity level of the message; defaults to informational if <code>level</code> is not specified: 1, 'E' or 'e' may be used for error messages 2, 'W' or 'w' may be used for warning messages 3, 'I' or 'i' may be used for informational messages

{level} instance.Write message**Purpose:**

Writes a message to the Windows Event Log associated with source name specified for the **SysLog** instance, optionally specifying a severity level.

Argument	Description
sourcename	character vector source name of an existing source
message	character vector message to write to the log
{level}	optional singleton indicating the severity level of the message; defaults to informational if level is not specified: 1, 'E' or 'e' may be used for error messages 2, 'W' or 'w' may be used for warning messages 3, 'I' or 'i' may be used for informational messages

Example:

```
logger←NEW SysLog 'mysource'  
1 logger.Write 'The sky is falling!'
```

Boolean←SysLog.LogExists logname**Purpose:**

Returns 1 if a Windows Event Log named **logname** exists, 0 otherwise.

Argument	Description
logname	character vector Windows Event Log log name

Boolean←SysLog.EventSourceExists sourcename

Purpose:

Returns 1 if a Windows Event Log source named **sourcename** exists, 0 otherwise.

Argument	Description
sourcename	character vector Windows Event Log source name

logname←LogNameFromSourceName sourcename

Purpose:

Returns the Windows Event Log log name associated with the source named **sourcename**.

Argument	Description
sourcename	character vector Windows Event Log source name
logname	character vector Windows Event Log log name

DeleteEventSource sourcename

Purpose:

Deletes the Windows Event Log source named **sourcename**.

Argument	Description
sourcename	character vector Windows Event Log source name

DeleteLog logname

Purpose:

Deletes the Windows Event Log log named **logname**.

Argument	Description
logname	character vector Windows Event Log log name

System Errors

Introduction

Dyalog APL will display a System Error Dialog and (normally) terminate in one of two circumstances:

- As a result of the failure of a workspace integrity check
- As a result of a System Exception

aplcore file

In these circumstances, APL saves an *aplcore* file which may be sent to Dyalog for diagnosis. The default name and location of the *aplcore* file may be specified by the **AplCoreName** parameter. This parameter may also be used to suppress the generation of the *aplcore* file.

From version 13.2 onwards useful information, including (where possible) the SI stack at the point where the *aplcore* was generated is written to the end of *aplcore* files; the section begins with the string

```
'===== Interesting Information'.
```

Workspace Integrity

When you **)SAVE** your workspace, Dyalog APL first performs a workspace integrity check. If it detects any discrepancy or violation in the internal structure of your workspace, APL does not overwrite your existing workspace on disk. Instead, it displays the System Error dialog box and saves the workspace, together with diagnostic information, in an *aplcore* file before terminating.

A System Error code is displayed in the dialog box and should be reported to Dyalog for diagnosis.

Note that the internal error that caused the discrepancy could have occurred at any time prior to the execution of **)SAVE** and it may not be possible for Dyalog to identify the cause from this *aplcore* file.

If APL is started in debug mode with the **-Dc**, **-Dw** or **-DW** flags, the Workspace Integrity check is performed more frequently, and it is more likely that the resulting *aplcore* file will contain information that will allow the problem to be identified and corrected. It is also possible to enable or alter the debugging level from within APL using the **SetDFlags** methods; Dyalog support will direct the use of this feature when necessary.

System Exceptions

Non-specific System Errors are the result of Operating System exceptions that can occur due to a fault in Dyalog APL itself, an error in a Windows or other DLL, or even as a result of a hardware fault. The following system exceptions are separately identified.

Code	Description	Suggested Action
900	A Paging Fault has occurred	As the most likely cause is a temporary network fault, recommended course of action is to restart your program.
990 & 991	An exception has occurred in the Development or Run-Time DLL.	
995	An exception has occurred in a DLL function called via <code>⌈NA</code>	Carefully check your <code>⌈NA</code> statement and the arguments that you have passed to the DLL function
996	An exception has occurred in a DLL function called via a threaded <code>⌈NA</code> call	As above
997	An exception has occurred while processing an incoming OLE call	
999	An exception has been caused by Dyalog APL or by the Operating System	

Recovering Data from aplcore files

Objects may often (but not always) be recovered from aplcore using `)COPY`. Note that because (by default) the aplcore file has no extension, it is necessary to explicitly add a dot, or APL will attempt to find the non-existent file `aplcore.DWS`, i.e.

```
)COPY aplcore.
```

Reporting Errors to Dyalog

If APL crashes and saves an aplcore file, please email the following information to support@dyalog.com:

1. a brief description of the circumstances surrounding the error
2. details of your version of Dyalog APL: the full version number, whether it is Unicode or Classic Edition, and the BuildID. This information appears in the *Help->About* box; the *Copy* button copies this information into the clipboard, from where it can be pasted into an email etc.
3. the aplcore file itself

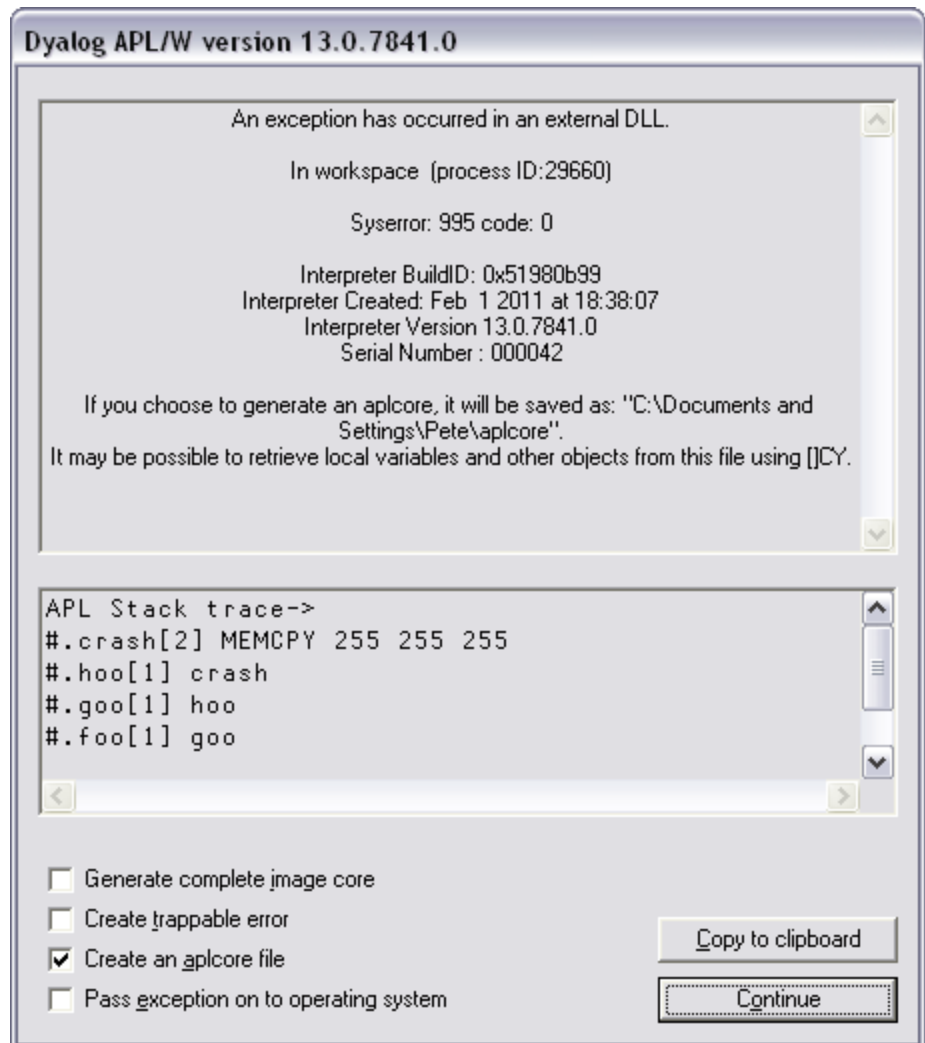
If the problem is reproducible, i.e. can be easily repeated, please also send the appropriate description, workspace, and other files required to do so.

System Error Dialog Box

The System Error Dialog illustrated below was produced by deliberately inducing a system exception in the Windows DLL function `memcpy()`. The functions used were:

```
[1] ▽ foo
    ▽ goo
    ▽ goo
[1] ▽ hoo
    ▽ hoo
[1] ▽ crash
    ▽

    ▽ crash
[1] ▽NA'dyalog32|MEMCPY u u u'
[2] MEMCPY 255 255 255
    ▽
```



Options

Item	Parameter	Description
Generate complete image core		Dumps a complete core image with the <i>User Mode Process Dumper</i> (a Microsoft tool) - see below.
Create Trappable Error		If you check this box (only enabled on System Error codes 995 and 996), APL will not terminate but will instead generate an error 91 (EXTERNAL DLL EXCEPTION) when you press <i>Dismiss</i> .
Create an aplcore file	CreateAplCoreonSyserror	If this box is checked, an aplcore file will be created.
Pass exception on to operating system	PassExceptionsToOpSys	If this box is checked, the exception will be passed on to your current debugging tool (e.g. <i>Visual Studio</i>).
Copy to clipboard		Copies the contents of the APL stack trace window to the Clipboard.

Generate complete image core

The *Generate complete image core* option attempts to execute [SYSDIR] \userdump.exe, where [SYSDIR] is the windows system directory (typically c:\windows\system32, and userdump.exe is the User Mode Process Dumper, a Microsoft tool that can be downloaded from the following url (which you may copy from Winhelp and paste into a browser):

```
http://www.microsoft.com/downloads/details.aspx?
FamilyID=e23cd741-d222-48df-9cd8-28796f414256&DisplayLang=en
```

The process creates a file called dyalog.core in the current directory. This file contains much more debug information than a normal aplcore (and is much larger than an aplcore) and can be sent to Dyalog Limited (zip it first please). Alternatively the file can be loaded into Visual Studio .Net to do your own debugging.

Debugging your own DLLs

If you are using Visual Studio, the following procedure should be used to debug your own DLLs when an appropriate Dyalog APL System Error occurs.

Ensure that the *Pass Exception* box is checked, then click on *Dismiss* to close the System Error dialog box.

The system exception dialog box appears. Click on *Debug* to start the process in the Visual Studio debugger.

After debugging, the system exception dialog box appears again. Click on *Don't send* to terminate Windows' exception handling.

ErrorOnExternalException Parameter

This parameter allows you to prevent APL from displaying the System Error dialog box (and terminating) when an exception caused by an external DLL occurs. The following example illustrates what happens when the functions above are run, but with `ErrorOnExternalException` set to 1.

```

      ⍵←2 ⍵NQ'.' 'GetEnvironment' 'ErrorOnExternalException
n'
1
      foo
EXTERNAL DLL EXCEPTION
crash[2] MEMCPY 255 255 255
      ^
      ⍵EN
91
      )SI
crash[2]*
hoo[1]
goo[1]
foo[1]

```


Chapter 2:

Configuring the IDE

The Configuration Dialog Box

General Tab

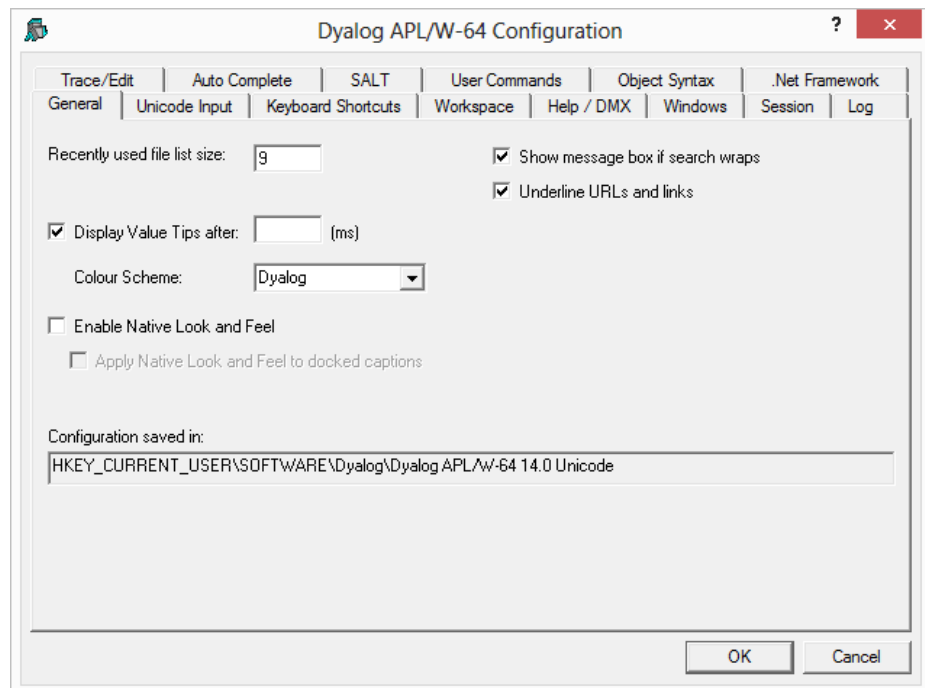


Table 1: Configuration dialog: General

Label	Parameter	Description
Show line numbers	lines_on_functions	Determines whether or not line numbers are shown in edit/trace windows
Recently used file list size	file_stack_size	Specifies the number of the most recently used workspaces displayed in the File menu.
Display Value Tips after	ValueTips/Delay	Specifies the delay before APL will display the value of a variable or the code for a function when the user hovers the mouse over its name.
Colour Scheme	ValueTips/ColourScheme	Specifies the colour scheme used to display the value of a variable or the code for a function when the user hovers the mouse over its name.
Enable Native Look and Feel	XPLookAndFeel	Specifies whether or not <i>Native Look and Feel</i> is enabled. This changes the appearance of user-interface controls such as Buttons in both the Session and the Dyalog GUI.
Apply Native Look and Feel to docked captions	XPLookAndFeelDocker	Specifies whether or not <i>Native Look and Feel</i> is honoured when drawing the title bars of docked windows, including docked Session windows.
Underline URLs and links	URLHighlight	Specifies whether or not URLs and links are highlighted in Session and Edit windows.
Configuration saved in	inifile	Specifies the full pathname of the registry folder used by APL

Underline URLs and links

If this option is selected, valid URLs are identified when the cursor is in the Session or in an Edit or Trace window. When the mouse pointer is over a URL, the URL is underscored and the appropriate items in the Session Popup menu are activated. These allow you to open the link or copy it to the clipboard.

You may also open a URL using Ctrl+Click (Left Mouse button).

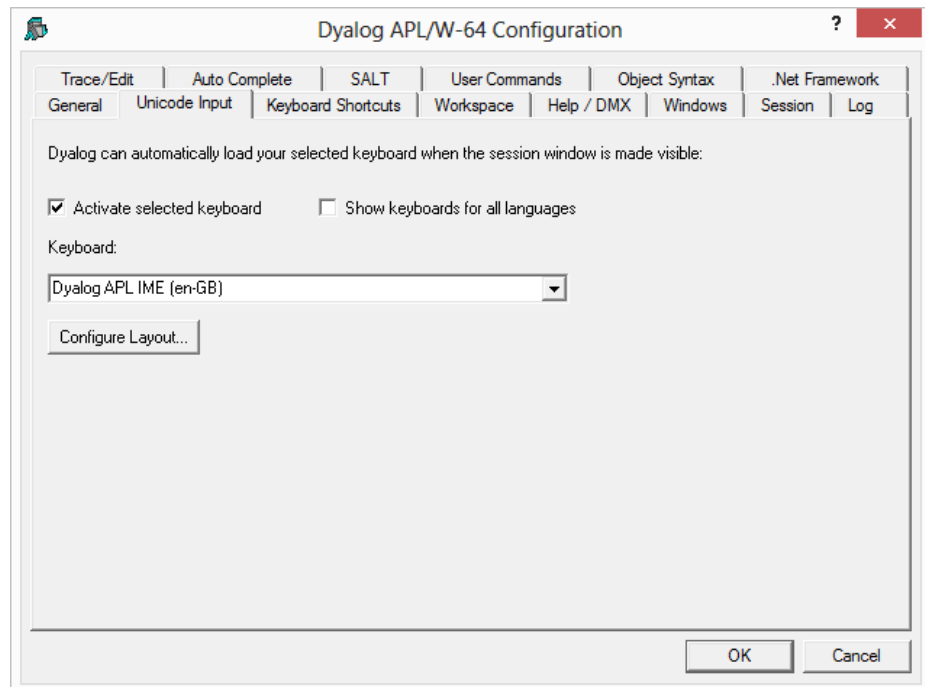
Currently a URL string is defined to be a string starting with any of the following strings:

- http://
- https://
- www.
- mailto:

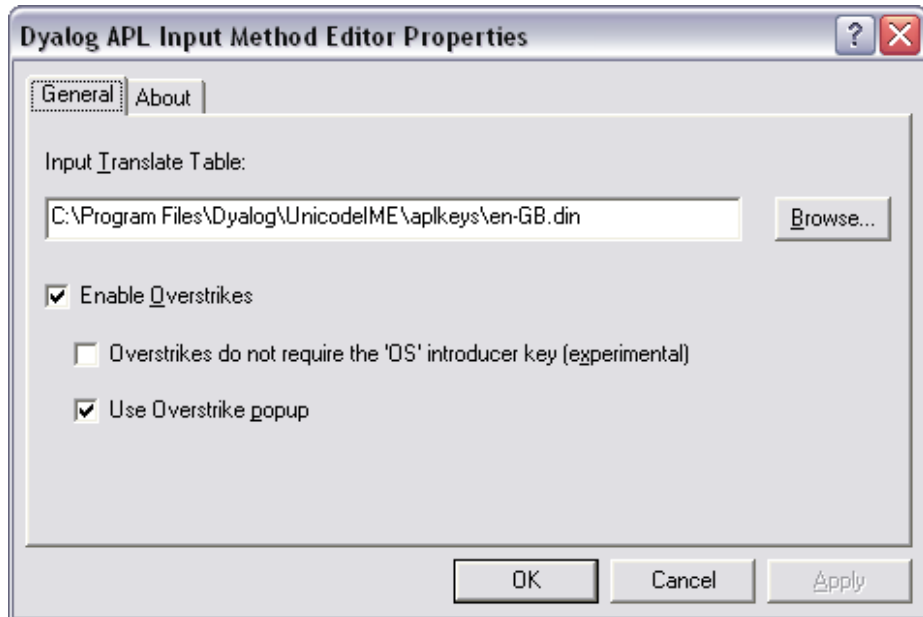
Unicode Input Tab (Unicode Edition Only)

Unicode Edition can optionally select your APL keyboard each time you start APL.

To choose this option, select one of your installed APL keyboards, enable the *Activate selected keyboard* checkbox, then click *OK*



Label	Parameter	Description
Activate selected keyboard	InitialKeyboardLayoutInUse	1 = automatically select the specified APL keyboard on start-up. 0 = no action
Show keyboards for all Languages	InitialKeyboardLayoutShowAll	1 = show list of all installed keyboards 0 = show only the Dyalog keyboards
Keyboard	InitialKeyboardLayout	the name of the APL keyboard to be selected.

**Table 2: Configuration dialog: Unicode Input/Configure Layout**

Label	Parameter	Description
Enable Overstrikes	ResolveOverstrikes	1 = enable overstrikes. 0 = disable overstrikes
Overstrikes do not require the <OS> key		1 = IME identifies overstrike operation automatically 0 = IME requires the <OS> key to signal an overstrike operation
Use Overstrike popup	OverstrikesPopup	1 = enable the overstrike popup. 0 = disable the overstrike popup

Input Tab (Classic Edition Only)

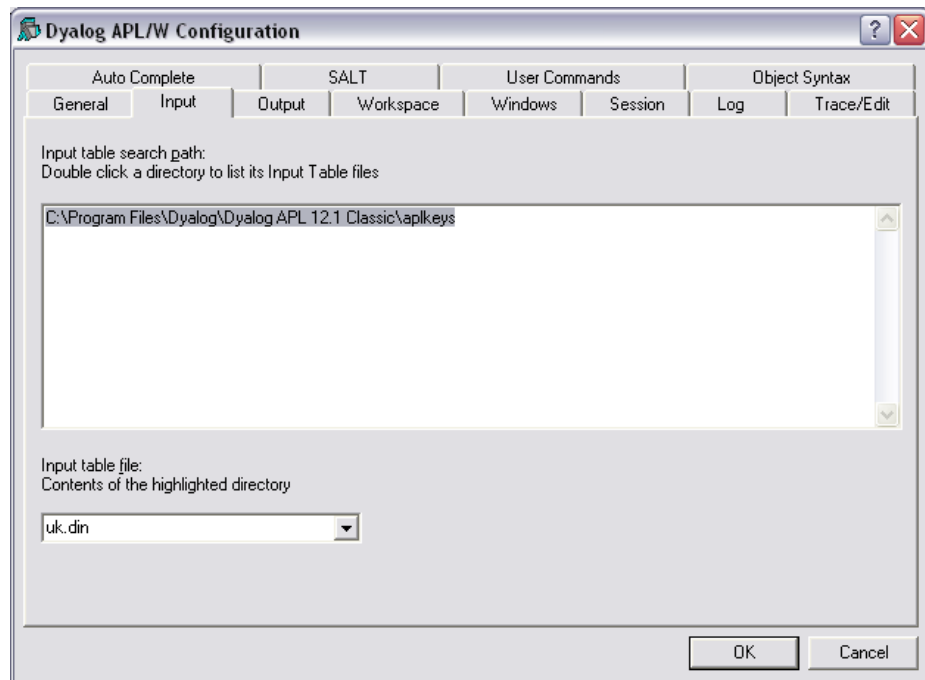


Table 3: Configuration dialog: Keyboard

Label	Parameter	Description
Input table search path	aplkeys	A list of directories to be searched for the specified input table
Input table file	aplk	The name of the input table file (.DIN)

Output Tab (Classic Edition Only)

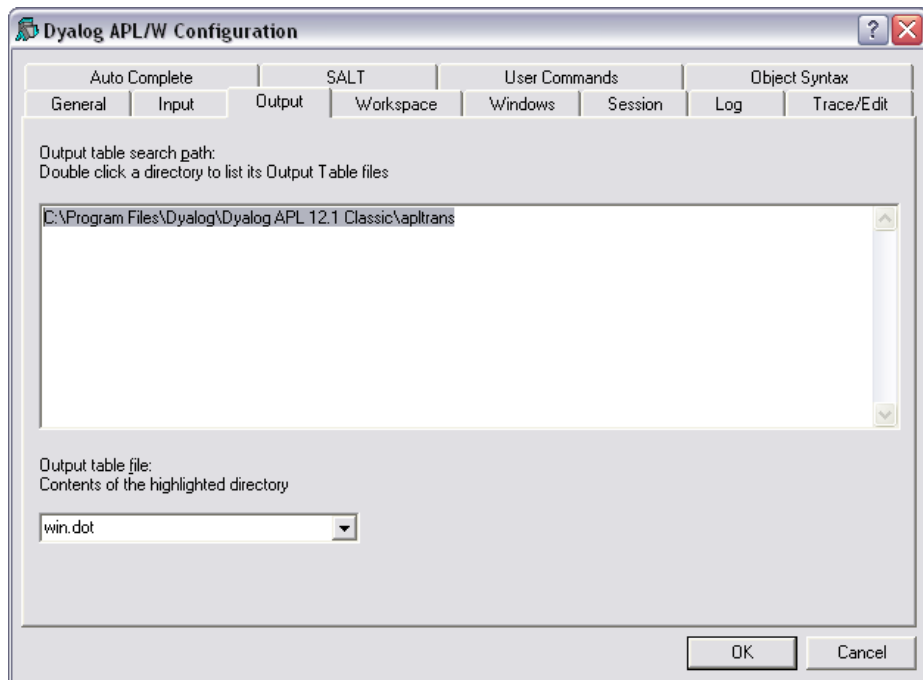
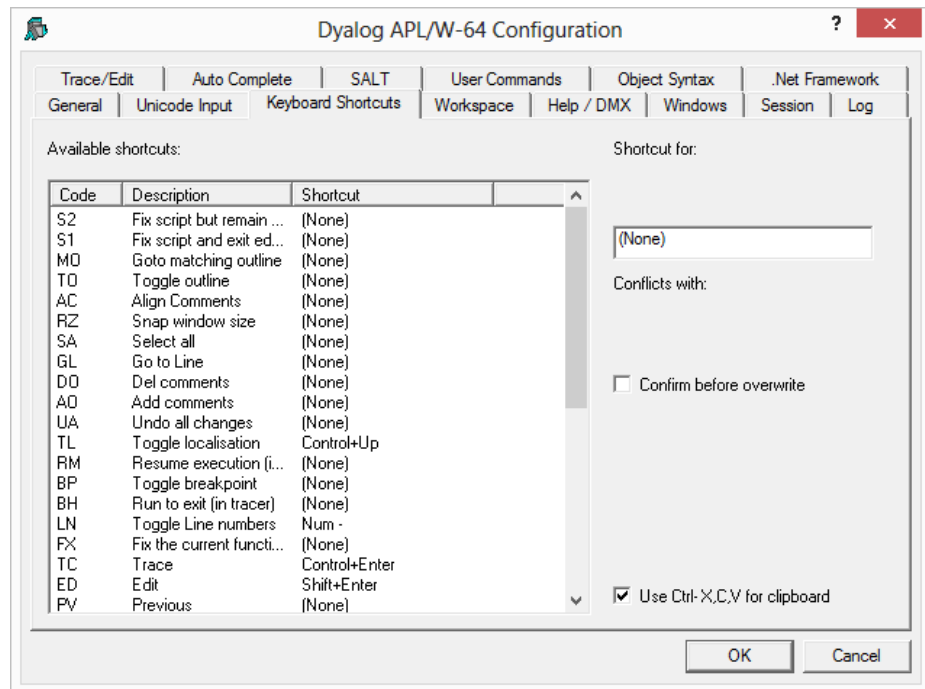


Table 4: Configuration dialog: Output

Label	Parameter	Description
Output table search path	apltrans	A list of directories to be searched for the specified output table
Output table file	aplt	The name of the output table file (.DOT)

Keyboard Shortcuts Tab



To alter the keystroke associated with a particular action, simply select the action required and press the keystroke. For example, to change the keystroke associated with the action <UA> (undo all changes) from (None) to Ctrl+Shift+u, simply select the corresponding row in the list and press Ctrl+Shift+u. If *Confirm before Overwrite* is checked, you will be prompted to confirm or cancel before each and every change is written back to the registry.

Note that clicking on the column headings will sort on that column; shift and mouse click will sort in reverse order.

Workspace Tab

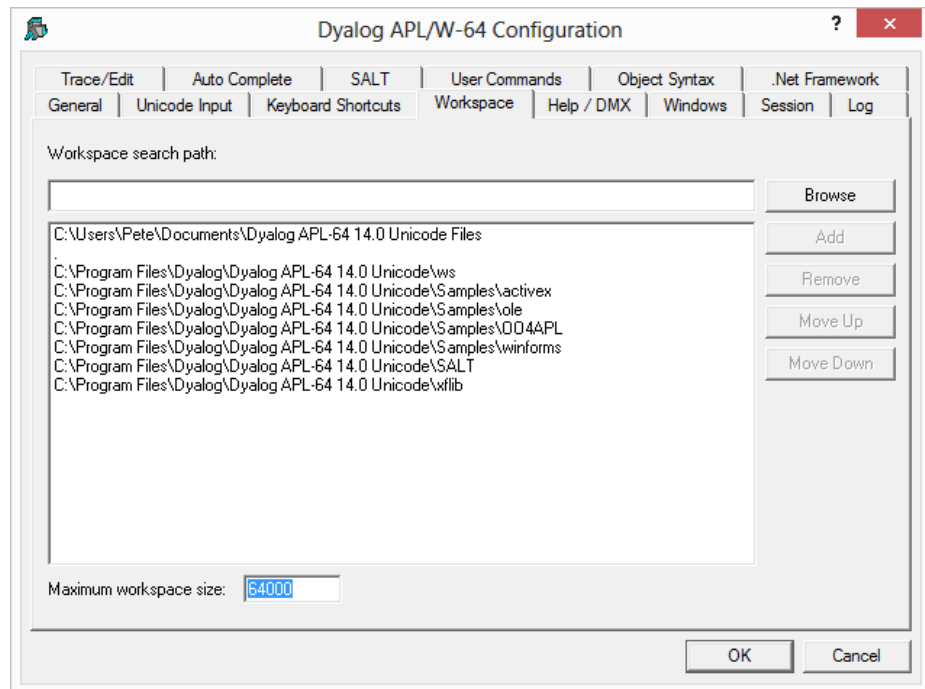


Table 5: Configuration dialog: Workspace

Label	Parameter	Description
Workspace search path	wspath	A list of directories to be searched for the specified workspace when the user executes <code>)LOAD wsname. "."</code> must be included in order to load workspaces from the current directory
Maximum workspace size	maxws	The maximum size of the workspace. Default is 64000 (KiB). Size is defined as an integer value followed by one of K, M, G, T, P or E. If no character is included, units default to K (Kilobytes)

Help/DMX Tab

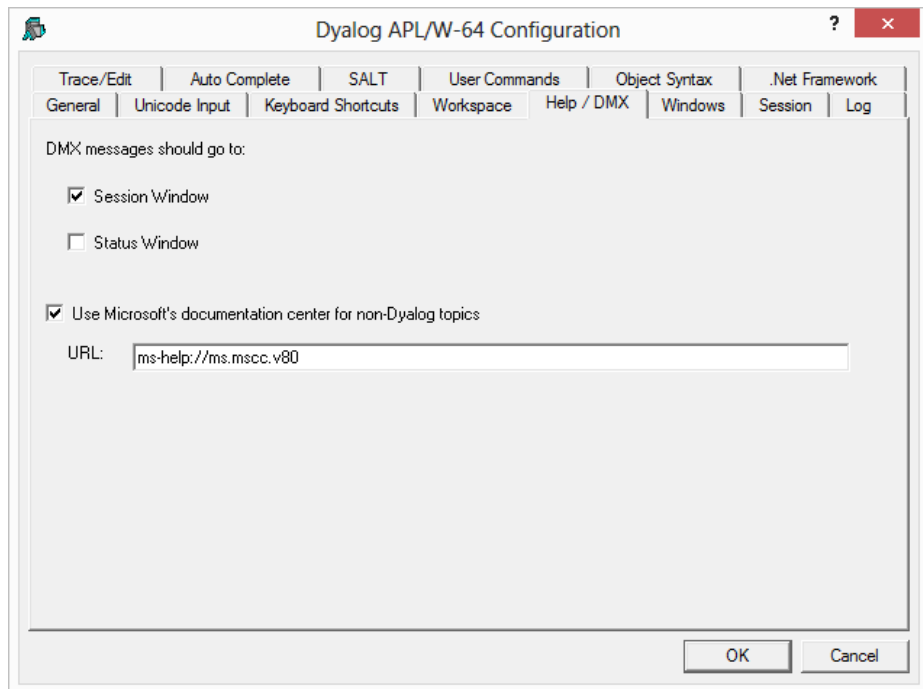


Table 6: Configuration dialog: Help/DMX

Label	Parameter	Description
DMX messages should go to	DMXOUTPUTONERROR	If checked, these boxes cause APL to display □DMX messages in the corresponding window (s).
Use Microsoft's documentation centre for non-Dyalog topics	UseDefaultHelpCollection	If this option is checked, APL will look for help at Microsoft's documentaion center for the the current string under the cursor.
URL	DefaultHelpCollection	The url for the documentation center.

Windows Tab

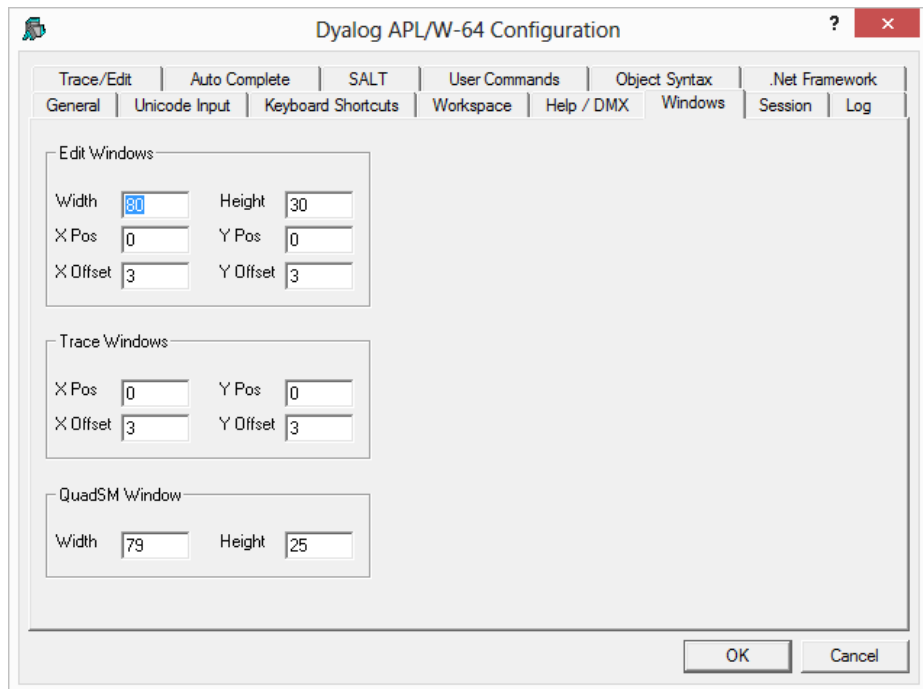


Table 7: Configuration dialog: Windows (Edit Windows)

Label	Parameter	Description
Width	edit_cols	The maximum number of rows displayed in a new edit window.
Height	edit_rows	The maximum number of columns displayed in a new edit window.
X Pos	edit_first_x	The initial horizontal position in characters of the first edit window.
Y Pos	edit_first_y	The initial vertical position in characters of the first edit window.
X Offset	edit_offset_x	The initial horizontal position in characters of the second and subsequent edit windows relative to the previous one.
Y Offset	edit_offset_y	The initial vertical position in characters of the second and subsequent edit windows relative to the previous one.

Table 8: Configuration dialog: Windows (Trace Windows)

Label	Parameter	Description
X Pos	trace_first_x	The initial horizontal position in characters of the first trace window.
Y Pos	trace_first_y	The initial vertical position in characters of the first trace window.
X Offset	trace_offset_x	The initial horizontal position in characters of the second and subsequent trace windows relative to the previous one.
Y Offset	trace_offset_y	The initial vertical position in characters of the second and subsequent trace windows relative to the previous one.

Table 9: Configuration dialog: Windows (QuadSM Window)

Label	Parameter	Description
Width	sm_cols	The width of the QuadSM and prefect windows.
Height	sm_rows	The height of the QuadSM and prefect windows.

Session Tab

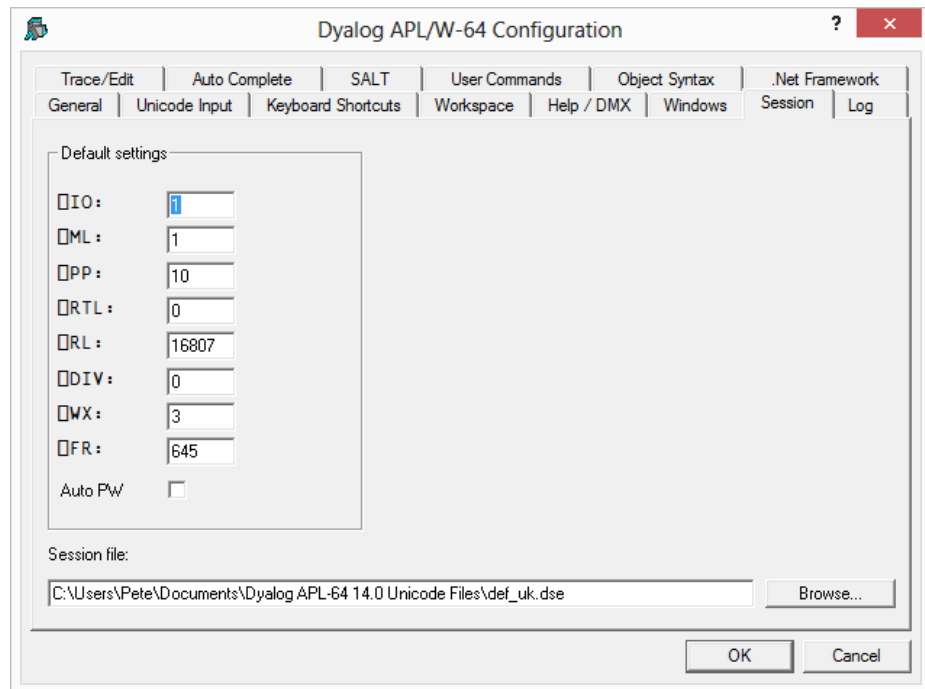


Table 10: Configuration dialog: Session

Label	Parameter	Description
<code>IO</code>	default_io	The default value of <code>IO</code> in a <code>clear ws</code> .
<code>ML</code>	default_ml	The default value of <code>ML</code> in a <code>clear ws</code> .
<code>PP</code>	default_pp	The default value of <code>PP</code> in a <code>clear ws</code> .
<code>RTL</code>	default_rtl	The default value of <code>RTL</code> in a <code>clear ws</code> .
<code>RL</code>	default_rl	The default value of <code>RL</code> in a <code>clear ws</code> .
<code>DIV</code>	default_div	The default value of <code>DIV</code> in a <code>clear ws</code> .
<code>WX</code>	default_wx	The default value of <code>WX</code> in a <code>clear ws</code> .
Auto PW	auto_pw	If checked, the value of <code>PW</code> is dynamic and depends on the width of the Session Window.
Session file	session_file	The name of the Session file in which the definition of your session (<code>SE</code>) is stored.

Log Tab

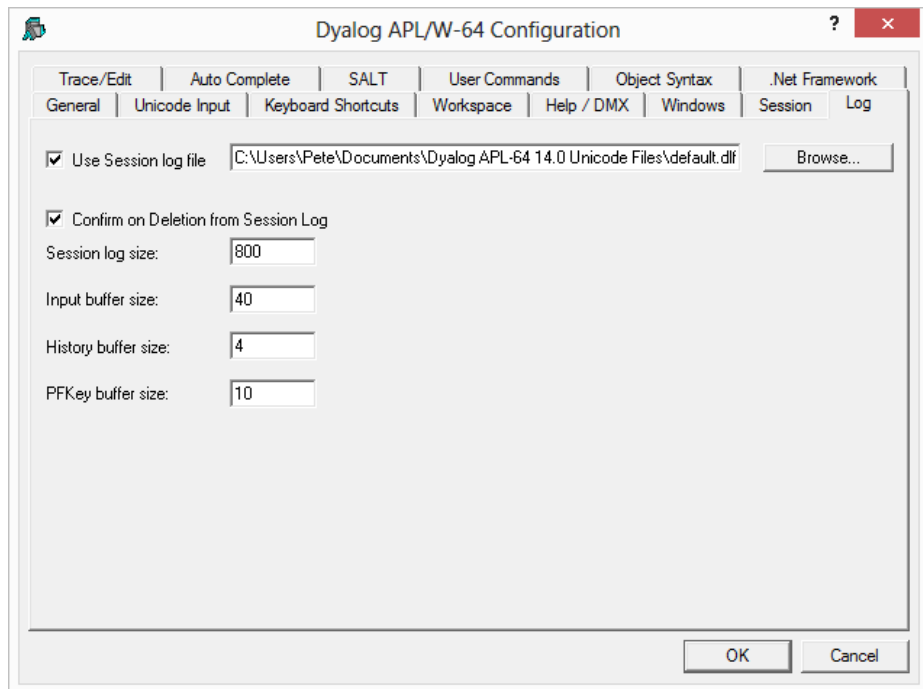


Table 11: Configuration dialog: Log

Label	Parameter	Description
Use Session log file	log_file_inuse	Specifies whether or not the Session log is saved in a session log file
Use Session log file	log_file	The full pathname of the Session log file
Confirm on Deletion from Session log	confirm_session_delete	Specifies whether or not you are prompted to confirm the deletion of a line from the Session (and Session log).
Session log size	log_size	The size of the Session log buffer.
Input buffer size	input_size	The size of the buffer used to store marked lines (lines awaiting execution) in the Session.
History size	history_size	The size of the buffer used to store previously entered (input) lines in the Session
PFKey buffer size	pfkey_size	The size of the buffer used to store PFKey definitions (PFKEY)

Note: The value of size-related values defined in the above table is specified as an integer value followed by one of K, M, G, T, P or E. The default, where no character is included, is K (Kilobytes).

Trace/Edit Tab

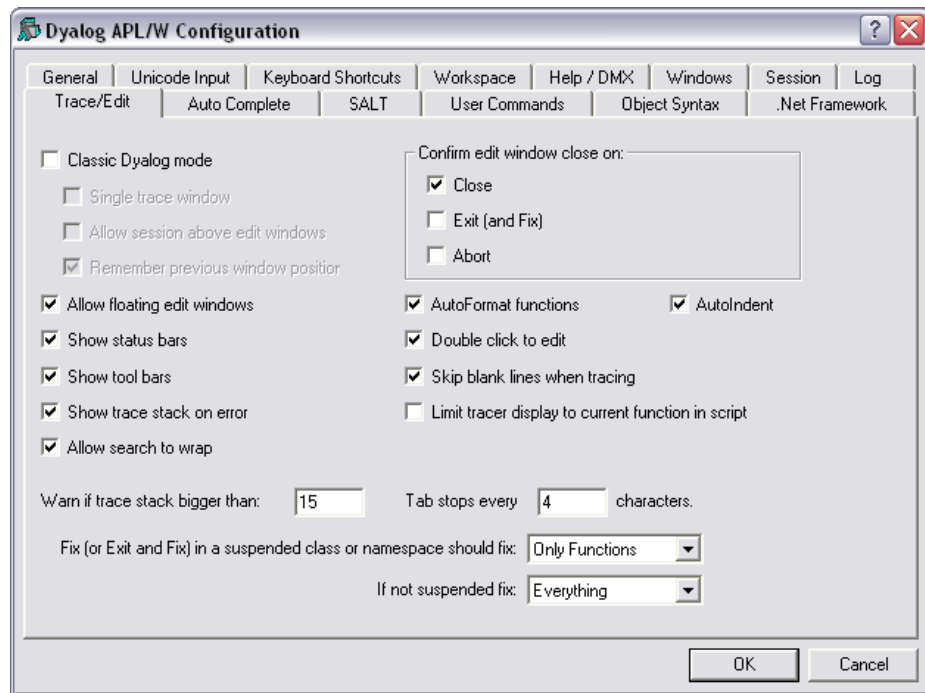


Table 12: Configuration dialog: Trace/Edit

Label	Parameter	Description
Classic Dyalog mode	ClassicMode	Selects pre-Version 9 behaviour for Edit and Trace windows
Allow session above edit windows	SessionOnTop	Specifies whether or not the Session may appear on top of Edit and Trace Windows
Single trace window	SingleTrace	Specifies whether or not there is a single Trace window
Remember previous window position	ClassicModeSavePosition	Specifies whether or not the current size and location of the first of the editor and tracer windows are remembered in the registry for next time.
Allow floating edit windows	DockableEditWindows	Allows individual Edit windows to be undocked from (and redocked in) the main Edit window
Show status bars	StatusOnEdit	Specifies whether or not status bars are displayed along the bottom of individual Edit windows
Show tool bars	ToolBarsOnEdit	Specifies whether or not tool bars are displayed along the top of individual Edit windows
Show trace stack on error	Trace_on_error	Specifies whether or not the Tracer is automatically invoked when an error or stop occurs in a defined function
Search Wraps around	WrapSearch	Specifies whether or not Search/Replace in the Editor stops at the top or bottom of the text, or continues from the start or end as appropriate.
Warn if trace stack bigger than	Trace_level_warn	Specifies the maximum stack size for automatic deployment of the Tracer.

Label	Parameter	Description
Confirm edit window close on Close	confirm_close	Specifies whether or not a confirmation dialog is displayed if the user alters the contents of an edit window, then closes it without saving
Confirm edit window close on Edit (and Fix)	confirm_fix	Specifies whether or not a confirmation dialog is displayed if the user alters the contents of an edit window, then saves it using <i>Fix</i> or <i>Exit</i>
Confirm edit window close on Abort	confirm_abort	Specifies whether or not a confirmation dialog is displayed if the user alters the contents of an edit window, then aborts using
Autoformat functions	AutoFormat	Selects automatic indentation for Control Structures when function is opened for editing
Autoindent	AutoIndent	Selects semi-automatic indentation for Control Structures while editing
Double-click to Edit	DoubleClickEdit	Specifies whether or not double-clicking over a name invokes the editor
Skip blank lines when tracing	SkipBlankLines	If enabled, this causes the Tracer to automatically skip lines that contain no executable statement.
Limit tracer display to current function in script	AddClassHeaders	When Tracing the execution of a function in a script, the Tracer displays either just the first line of the script and the function in question (option enabled), or the entire script (option disabled).

Label	Parameter	Description
Paste text as Unicode (Classic Edition only)	UnicodeToClipboard	Specifies whether or not text transferred to and from the Windows clipboard is to be treated as Unicode
Tab stops every	TabStops	The number of spaces inserted by pressing Tab in an edit window
Exit and fix ...		See Fixing Scripts below
If not ...		See Fixing Scripts below

Fixing Scripts

When using the Editor to edit a script such as a Class or Namespace you can specify whether, when you Fix the script and Exit the Editor, just the functions in the script are re-fixed, or whether the whole script is re-executed, thereby re-initialising any Fields or variables defined within.

These two actions always appear in the Editor File menu, but you can specify which is associated with the <EP> (Esc) key by selecting the appropriate option in the drop-downs labelled:

- *Exit and save changes (EP) in a suspended class or namespace should fix:*
- *If not suspended fix:*

In both cases, you may select either *Only Functions* or *Everything*.

The label for the corresponding items on the Editor File menu (see *User's Guide: Editor (The File Menu, editing a script)*) will change according to which behaviour applies. Note that if you specify a keystroke for <S1> in the *Keyboard Shortcuts* tab, this will be associated with the unselected action.

Auto Complete Tab

Dyalog APL/W-64 Configuration

General | Unicode Input | Keyboard Shortcuts | Workspace | Help / DMX | Windows | Session | Log
Trace/Edit | **Auto Complete** | SALT | User Commands | Object Syntax | .Net Framework

☒ Use Auto complete

Make suggestions after characters. Delay completion for milliseconds.

Suggest up to items at a time.

Show up to columns at a time.

☒ Keep history

History Length entries.

☒ Include filenames

OK Key: or:

Cancel Key: or:

Common Key:

OK Cancel

Note: To enter values in the *OK Key* and *Cancel Key* fields, click on the field with the mouse and then press the desired keystroke.

Table 13: Configuration dialog: Auto Complete

Label	Parameter	Description
Use Auto Complete	Enabled	Specifies whether or not Auto Completion is enabled.
Make suggestions after	PrefixSize	Specifies the number of characters you must enter before Auto Completion begins to make suggestions
Delay completion for	KeyboardInputDelay	Specifies the delay in milliseconds before Auto Completion begins to make suggestions

Label	Parameter	Description
Suggest up to	Rows	Specifies the maximum number of rows (height) in the AutoComplete pop-up suggestions box.
Show up to	Cols	Specifies the maximum number of columns (width) in the AutoComplete pop-up suggestion box
Keep History	History	Specifies whether or not AutoComplete maintains a list of previous AutoCompletions.
History Length	HistorySize	Specifies the number of previous AutoCompletions that are maintained
Include filenames	ShowFiles	Specifies whether or not AutoCompletion suggests directory and file names for) LOAD ,) COPY and) DROP system commands.
OK Key	CompleteKey1 CompleteKey2	Specifies two possible keys that may be used to select the current option from the Auto Complete suggestion box.
Cancel Key	CancelKey1 CancelKey2	Specifies two possible keys that may be used to cancel (hide) the Auto Complete suggestion box.
Common Key	CommonKey1	Specifies the key that will auto-complete the <i>common prefix</i> . This is defined to be the longest string of leading characters in the currently selected name that is shared by at least one other name in the Auto Complete suggestion box.

SALT

SALT is the Simple APL Library Toolkit, a simple source code management system for Classes and script-based Namespaces. SPICE uses SALT to manage development tools which “plug in” to the Dyalog session

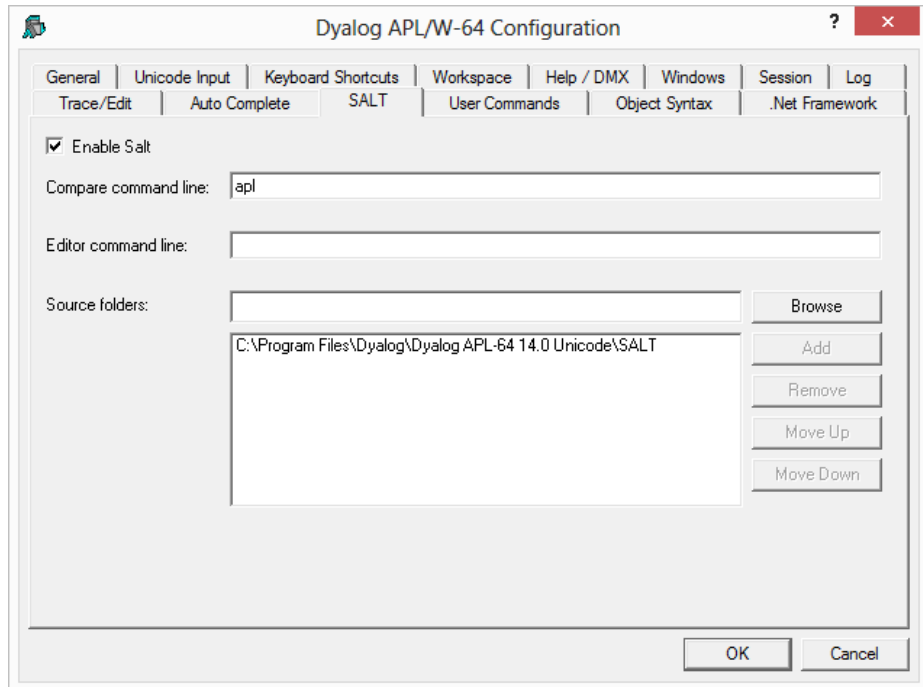
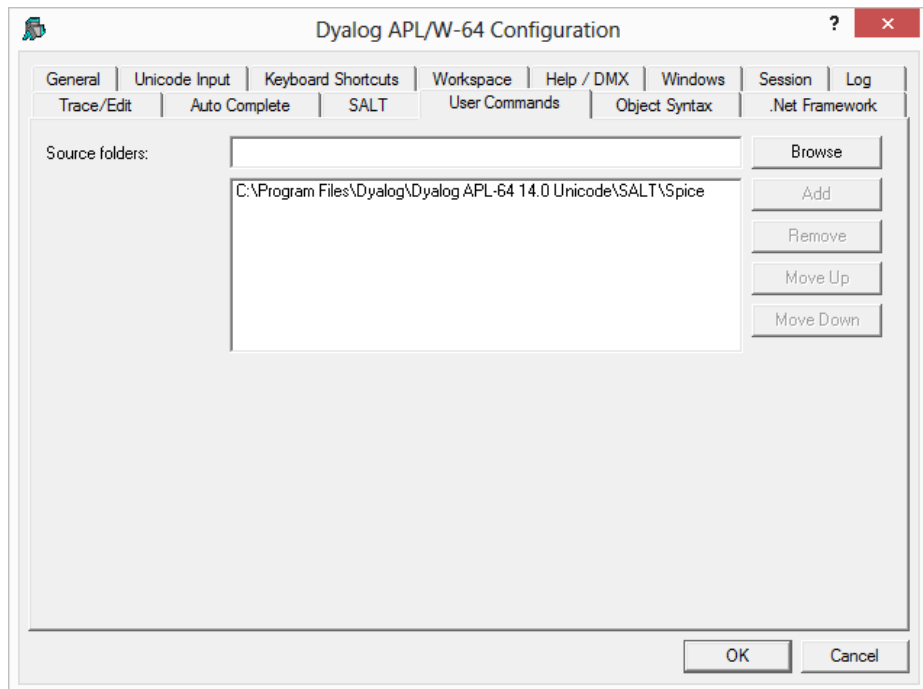


Table 14: Configuration dialog: SALT

Label	Parameter	Description
Enable Salt	AddSALT	Specifies whether or not SALT is enabled
Compare command line	CompareCMD	The command line for a 3 rd party file comparison tool to be used to compare two versions of a file. See note.
Editor	Editor	Name of the program to be used to edit script files (default "Notepad").
Class source folders	SourceFolder	Sets the SALT working directory; a list of folders to be searched for source code. Include "." on a separate line to include source files from the current working directory

User Commands Tab

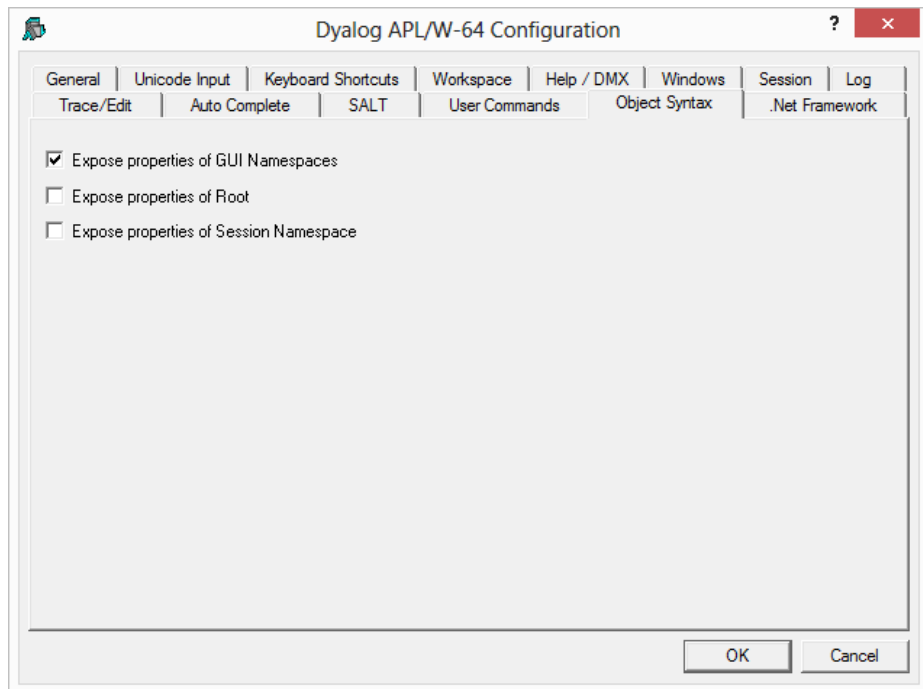


This page is used to specify and organise a list of folders that contain User-Command files. When you issue a User Command, these folders will be searched for the source of the command in the order in which they appear in this list.

Table 15: Configuration dialog: User Commands

Label	Parameter	Description
Source Folders	SALT\CommandFolder	Use this field to add folders to the list of folders that will be searched for User Commands.

Object Syntax Tab



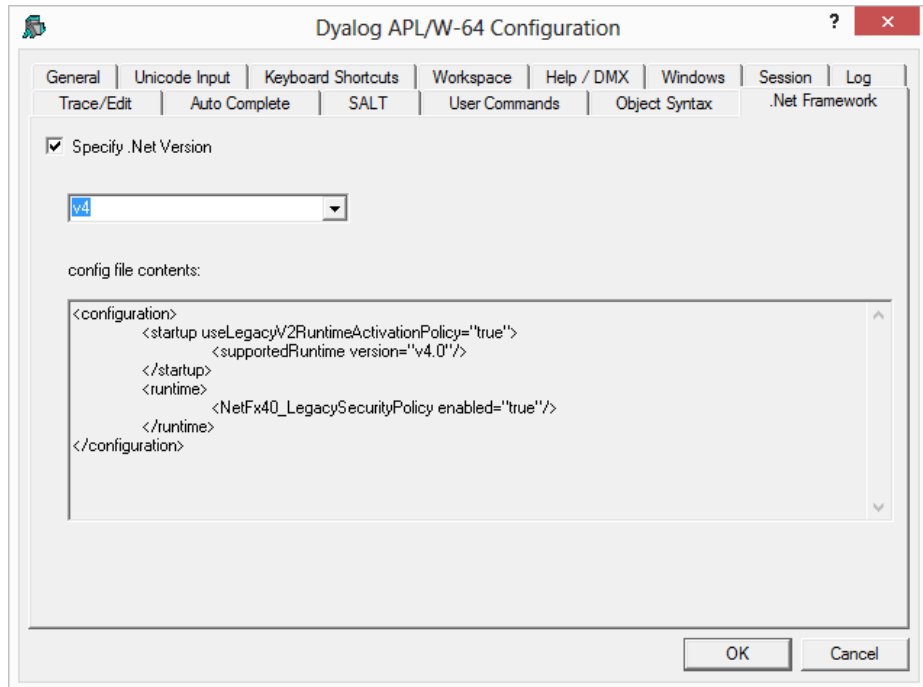
The *Object Syntax* tab of the *Configuration* dialog is used to set your *default preferences* for Object Syntax.

The Object Syntax settings for the current workspace are reflected by the *Object Syntax* submenu of the *Options* menu. Use *Options/Object Syntax* to change them. These settings are saved in the workspace.

Table 16: Configuration dialog: Object Syntax

Label	Parameter	Description
Expose properties of GUI Namespaces	default_wx	Specifies the value of <code>default_wx</code> in a clear workspace. This in turn determines whether or not the names of properties, methods and events of GUI objects are exposed. If set (<code>default_wx</code> is 1), you may query/set properties and invoke methods directly as if they were variables and functions respectively. As a consequence, these names may not be used for global variables in GUI objects.
Expose properties of Root	PropertyExposeRoot	Specifies whether or not the names of properties, methods and events of the Root object are exposed. If set, you may query/set the properties of Root and invoke the Root methods directly as if they were variables and functions respectively. As a consequence, these names may not be used for global variables in your workspace.
Expose properties of Session Namespace	PropertyExposeSE	Specifies whether or not the names of properties, methods and events of the Session object are exposed. If set, you may query/set the properties of <code>SE</code> and invoke <code>SE</code> methods directly as if they were variables and functions respectively. As a consequence, these names may not be used for global variables in the <code>SE</code> namespace.

.Net FrameworkTab



This dialog box allows you to specify which version of the .NET Framework you want to use with Dyalog APL.

If *Specify .Net Version* is selected, APL will display the versions of the .NET Framework that are installed in the combo box below.

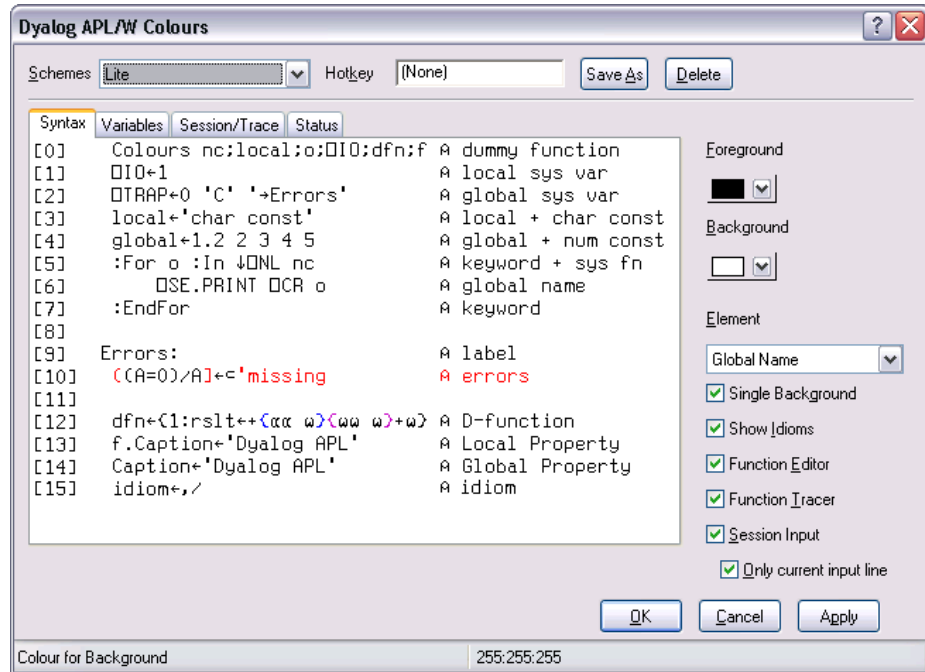
Choose the version you require and click *OK*.

Unlike the other configuration dialogs, which typically set values in the Registry, this dialog creates a configuration file `dyalog.exe.config` in the same directory as the Dyalog APL program. Note the following:

- Any existing `dyalog.exe.config` file will simply be overwritten, losing any changes that the file might contain.
- For .NET Version 2 (the default for Dyalog APL), no configuration file is required; if you select Version 2 having previously selected Version 4, the file will be deleted.
- With default privileges, the `dyalog.exe.config` file will be put in the Windows VirtualStore; Dyalog recommends changing this option in an APL which has elevated privileges (right click on the Dyalog APL shortcut and select "Run as Administrator").

The dialog box shows the contents of this file as illustrated above.

Colour Selection Dialog



The Colour Selection dialog box allows you to select colours for:

- Syntax colouring
- Edit, Trace and Session windows
- Status window

The colour selection dialog box is selected by the **[ChooseColor]** system action which by default is attached to the *Options/Colours* menu item on the Session menubar and to the *Colours* menu item in the Session pop-up menu.

Syntax Colouring

Syntax colouring allows you to visually identify various components in the function edit and session windows by assigning different colours to them, such as:

- Global references (functions and variables)
- Local references (functions and variables)
- Primitive functions
- System functions
- Localised System Variables
- Comments
- Character constants
- Numeric constants
- Labels
- Control Structures
- Unmatched parentheses, quotes, and braces

Schemes

You may define a number of different syntax colouring schemes which are suitable for different purposes and a selection of schemes is provided. Choose the scheme you wish to use from the Combo box provided. If you change a colour allocation, you may overwrite an existing Colour Scheme or define a new one by clicking *Save As* and then entering the name of the Scheme. You may delete a Colour Scheme using the *Delete* button.

Changing Colours

To allocate a colour to a syntax element, you must first select the syntax element. You may select a syntax element from the Combo box provided, or by clicking on an example in the sample function provided. Having selected a syntax element, choose a colour using the *Foreground* or *Background* selectors as appropriate.

Show Idioms

The *Show Idioms* checkbox allows you to choose whether or not idioms are to be identified by syntax colouring.

Single Background

The *Single Background* checkbox allows you to choose whether to impose a single background colour, or to allow the use of different background colours for different syntax elements.

Function Editor

Check this box if you want to enable syntax colouring in Edit windows.

Function Tracer

Check this box if you want to enable syntax colouring in Trace windows.

Session Input

Check this box if you want to enable syntax colouring in the Session window. Note that the colour scheme used for the Session may differ from the colour scheme selected for Edit windows and is specified by the *Session Colour Scheme* box on the *Session/Trace* tab.

Only current input line

This option only applies if Session syntax colouring is enabled. Check this box if you want syntax colouring to apply only to the current input line. Clear this box, if you want to apply syntax colouring to all the input lines in the current Session window. Note that syntax colouring of input lines is not remembered in the Session log, so input lines from previous sessions do not have syntax colouring.

HotKeys

You may associate a different *hot key* with any or all of your colour schemes.

When you depress a hot key over a function in an Edit window, the function is displayed using the scheme associated with the hot key. Releasing the hot key causes it to be displayed in the normal scheme.

This feature is intended to allow you to quickly check for certain syntax elements. For example, you may define a special scheme that only highlights global names and associate a hot key with it. Pressing the hot key will temporarily highlight the globals for you.

To associate a hot key with a colour scheme, click on the *Hotkey* field, and then make the desired keystroke. To disassociate a hot key, use <backspace>.

Print Configuration Dialog Box

The Print Configuration dialog box is displayed by the system operation `[PrintSetup]` that is associated with the *File/Print Setup* menu item. It is also available from Edit windows and from the *Workspace Explorer* and *Find Objects* tools.

There are four separate tabs namely *Setup*, *Margins*, *Header/Footer* and *Printer*.

Note that the printing parameters are stored in the Registry in the Printing sub-folder

Setup Tab

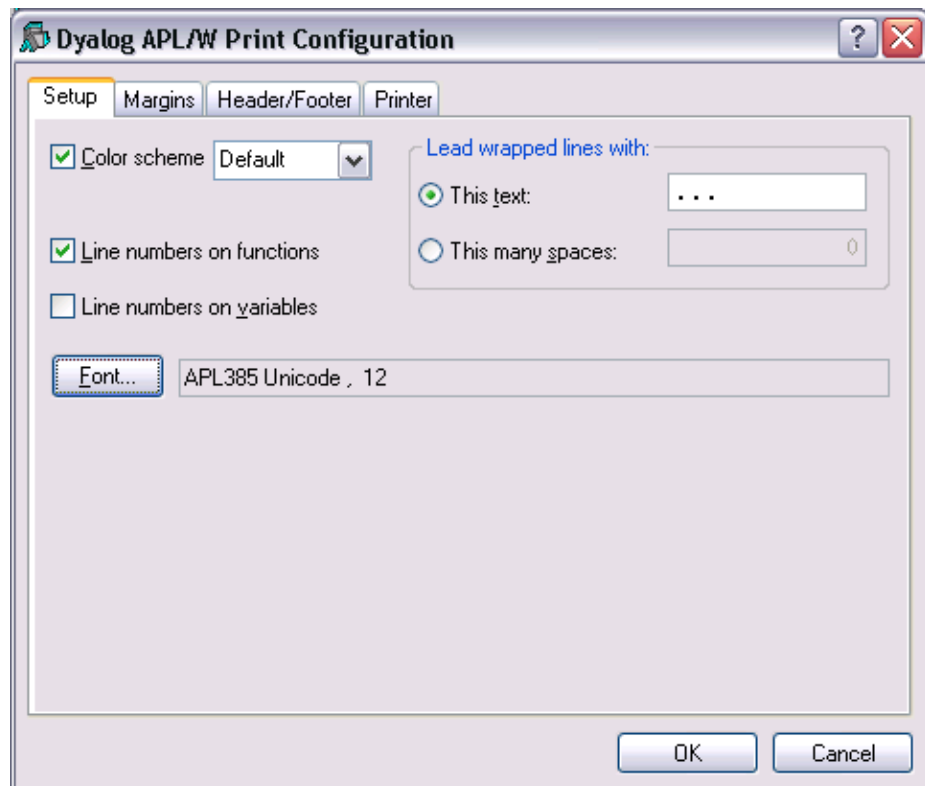


Table 17: Print Configuration dialog: Setup

Label	Parameter	Description
Color scheme	InColour	Check this box if you want to print functions with syntax colouring. Note that that printing in colour is slower than printing without colour.
Color scheme	SchemeName	Select the colour scheme to be used for printing.
This text	WrapWithText	Check this option button if you wish to prefix wrapped lines (lines that exceed the width of the paper) with a particular text string
This text	WrapLeadText	Specifies the text for prefixing wrapped lines
This many spaces	WrapWithSpaces	Check this option button if you wish to prefix wrapped lines with spaces.
This many spaces	WrapLeadSpaces	Specifies the number of spaces to be inserted at the beginning of wrapped lines.
Line numbers on functions	LineNumsFns	Check this box if you want line numbers to be printed in defined functions.
Line numbers on variables	LineNumsVars	Check this box if you want line numbers to be printed in variables. If you choose this option, line numbering starts at 10 .
Font	Font	Click to select the font to be used for printing. Note that only fixed-pitch fonts are supported.

Margins Tab

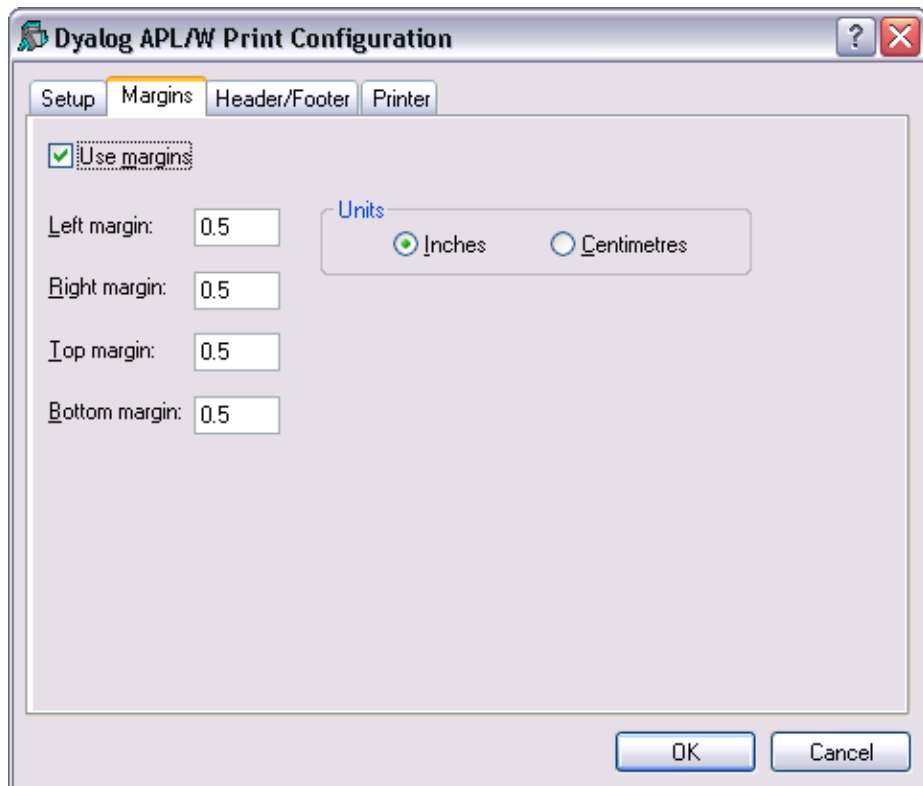


Table 18: Print Configuration dialog: Margins

Label	Parameter	Description
Use margins	UseMargins	Check this box if you want margins to apply
Left margin	MarginLeft	Specifies the width of the left margin
Right margin	MarginRight	Specifies the width of the right margin
Top margin	MarginTop	Specifies the height of the top margin
Bottom margin	MarginBottom	Specifies the height of the bottom margin

Label	Parameter	Description
Inches	MarginInch	Specifies that the margin units are inches
Centimetres	MarginCM	Specifies that the margin units are centimetres

Header/Footer Tab

The screenshot shows the 'Dyalog APL/W Print Configuration' dialog box with the 'Header/Footer' tab selected. The dialog has four tabs: 'Setup', 'Margins', 'Header/Footer', and 'Printer'. The 'Header/Footer' tab contains the following options:

- ☒ **Header**: The text field contains 'Workspace {wsname}{cr}'.
- ☒ **Footer**: The text field contains '{cr}{hl}Printed at {printdate} {printtime}{rj}Page {cu}'.
- ☒ **Prefix functions with:**: The text field contains '{hl}{objname}{rj} Author:{author} Last Fixed:{fixdate}'.
- ☒ **Prefix variables with:**: The text field contains '{hl}{objname}{cr}'.
- ☒ **Prefix other objects with**: The text field contains '{hl}{objname}{cr}'.

At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

Table 19: Print Configuration dialog: Header/Footer

Label	Parameter	Description
Header	DoHeader	Specifies whether or not a header is printed at the top of each page
Header	HeaderText	The header text
Footer	DoFooter	Specifies whether or not a footer is printed at the bottom of each page
Footer	FooterText	The footer text
Prefix functions with	DoSepFn	Specifies whether or not text is printed before each defined function
Prefix functions with	SepFnText	The text to be printed before each defined function. This can include its name, timestamp and author
Prefix variables with	DoSepVar	Specifies whether or not text is printed before each variable.
Prefix variables with	SepVarText	The text to be printed before each variable. This can include its name.
Prefix other objects with	DoSepOther	Specifies whether or not text is printed before other objects. These include locked functions, external functions, UNA functions, derived functions and namespaces.
Prefix other objects with	SepOtherText	The text to be printed before other objects. This can include its name.

The specification for headers and footers may include a mixture of your own text, and keywords which are enclosed in braces, e.g. {objname}. Keywords act like variables and are replaced at print time by corresponding values.

Any of the following fields may be included in headers, footers and separators.

{WSName}	{WS}	Workspace name
{NSName}	{NS}	Namespace name
{ObjName}	{OB}	Object name
{Author}	{AU}	Author
{FixDate}	{FD}	Date function was last fixed
{FixTime}	{FT}	Time function was fixed
{PrintDate}	{PD}	Today's date
{PrintTime}	{PT}	Current time
{CurrentPage}	{CP}	Current page number
{TotalPages}	{TP}	Total number of pages
{RightJustify}	{RJ}	Right-justifies subsequent text/fields
{HorizontalLine}	{HL}	Inserts a horizontal line
{CarriageReturn}	{CR}	Inserts a new-line

For example, the specification:

Workspace: {wsname} {objname} {rj} Printed {PrintTime} {PrintDate}

would cause the following header, footer or separator to be printed at the appropriate position in each page of output:

Workspace: U:\WS\WDESIGN WIZ_change_toolbar Printed 14:40:11 02 March 1998

Printer Tab

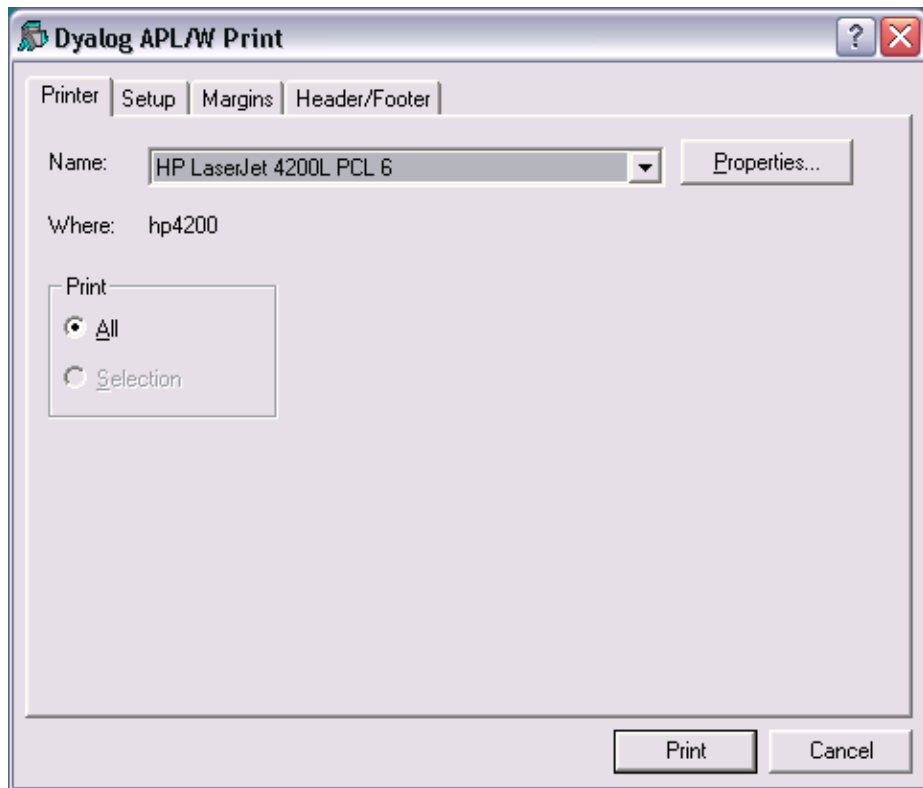


Table 20: Print Configuration dialog: Print

Label	Parameter	Description
Name	PrinterField	The name of the printer to be used for printing from Dyalog APL.
Properties		Click this to set Printer options.
Where		Reports the printer device
Print		Allows you to choose between printing all of the current object or just the selection. Note that this option is present only when the dialog box is displayed in response to selecting Print.

Index

A

ActiveX control 53
 AddClassHeaders 14
 AddClassHeaders parameter 96
 APL_CODE_E_MAGNITUDE 14
 APL_COMPLEX_AS_V12 15
 APL_FAST_FCHK 16
 APL_FCREATE_PROPS_C 15
 APL_FCREATE_PROPS_J 15
 aplcore 16, 19, 71
 aplcorename parameter 16, 71
 aplk parameter 16, 82
 aplkeys parameter 17, 82
 aplnid parameter 17
 APLScript compiler 49-50
 APLService
 Logging Events 66
 aplt parameter 17, 83
 apltrans parameter 17, 83
 aplunibd.ini 55
 Array Editor 38
 auto_pw parameter 18, 91
 AutoComplete
 registry entries 36
 AutoFormat parameter 18, 96
 AutoIndent parameter 18, 96
 auxiliary processors 42

B

bridge dll 49-52, 54
 Build runtime application 47

C

CancelKey (AutoComplete) parameter 99
 captions
 registry entries 36

Classic Edition 13, 16-17, 26, 32, 97
 ClassicMode parameter 18, 21, 23, 30-31, 95
 ClassicModeSavePosition 95
 CMD_POSTFIX parameter 19
 CMD_PREFIX parameter 19
 colour selection dialog 106
 colours
 registry entries 36
 colourscheme parameter 78
 Cols (AutoComplete) parameter 99
 COM server
 in-process 53
 out-of-process 52
 command line 9
 command processor 42-43
 CommandFolder parameter 102
 CommonKey (AutoComplete) parameter 99
 CompleteKey (AutoComplete) parameter 99
 configuration dialog 77
 .net framework tab 105
 autocomplete tab 98
 general tab 77
 help/dmx tab 86
 input tab 82
 keyboard shortcuts tab 84
 log tab 92
 object syntax tab 103
 output tab 83
 session tab 90
 trace/edit tab 94
 unicode input tab 79
 user commands tab 102
 windows tab 87
 workspace tab 85
 configuration parameters 12
 confirm_abort parameter 19, 96
 confirm_close parameter 19, 96
 confirm_fix parameter 19, 96
 confirm_session_delete parameter 19
 COPY system command 72
 CreateAplcoreOnSyserror parameter 19
 CreateAplCoreonSyserror parameter 75
 creating executables 45

D

default_div parameter 20, 91

default_io parameter 20, 91
default_ml parameter 20, 91
default_pp parameter 20, 91
default_pw parameter 20
default_rl parameter 20, 91
default_rtl parameter 21, 91
default_wx parameter 21, 91, 104
DefaultHelpCollection parameter 20, 86
delay parameter 78
division method 20
DMXOUTPUTONERROR parameter 21, 86
DockableEditWindows parameter 21, 95
DoubleClickEdit parameter 21, 96
Dyalog APL DLL 58
 classes, instances and cloning 58
 workspace management 59
dyalog.dll 49-50
dyalog parameter 13, 17, 21
dyalog.chm 22
dyalog.exe.config 105
DYALOG_EVENTLOGGINGLEVEL
parameter 22
DYALOG_EVENTLOGNAME parameter 22
DYALOG_NOPOPUPS parameter 22
dyalog32.dll 56
DyalogEmailAddress parameter 21
DyalogHelpDir parameter 22
DyalogInstallDir parameter 22
dyalognet.dll 49-52, 54
dyalogprovider.dll 49-50
DyalogWebSite parameter 22

E

edit window geometry 23
edit_cols parameter 23, 88
edit_first_x parameter 23, 88
edit_first_y parameter 23, 88
edit_offset_x parameter 23, 88
edit_offset_y parameter 23, 88
edit_rows parameter 23, 88
EditorState parameter 23
Enabled (AutoComplete) parameter 98
environment variables 12-13
ErrorOnExternalException parameter 23
event viewer
 registry entries 36

exit codes 11, 16
Export menu item 45
external variables
 sharing 44

F

file extensions 1
file_stack_size parameter 78
files
 registry entries 36

G

GetEnvironment method 13
global assembly cache 51-52, 54
greet_bitmap parameter 23

H

History (AutoComplete) parameter 99
history_size parameter 24, 93
HistorySize (AutoComplete) parameter 99
hot keys
 syntax colouring 108

I

index origin 20
infile parameter 13, 24, 78
InitialKeyboardLayout 24
InitialKeyboardLayout parameter 80
InitialKeyboardLayoutInUse parameter 24, 80
InitialKeyboardLayoutShowAll parameter 24, 80
INotifyCollectionChanged interface 54
input translate table 16
input_size parameter 24, 93
interface with Windows 42
Interoperability 4

K

keyboard shortcuts
 registry entries 37

KeyboardInputDelay parameter 98

L

language bar

registry entries 37

lines_on_functions parameter 25, 78

localdialogdir parameter 25

log_file parameter 25, 93

log_file_inuse parameter 93

log_size parameter 25, 93

logfileinuse parameter 25

M

mapchars parameter 26

maxws parameter 27, 41, 59-60, 85

Microsoft Document Explorer 20

migration level 20

N

Net assembly 54

O

output translate table 17

OverstrikesPopup parameter 27, 81

P

page width 20

PassExceptionsToOpSys parameter 27, 75

pfkey_size parameter 28, 93

PrefixSize (AutoComplete) parameter 98

print configuration dialog 109

header/footer Tab 112

margins tab 111

printer tab 115

setup tab 109

print precision 20

printing

registry entries 37

programfolder parameter 28

PropertyExposeRoot parameter 28, 104

PropertyExposeSE parameter 28, 104

Q

qcmd_timeout parameter 28

QUADNA workspace 56

R

random link 20

registry entries

run-time installation 56

ResolveOverstrikes parameter 28, 81

response time limit 21

return code 11

Rows (AutoComplete) parameter 99

run-time

applications 49

bound 51

stand-alone 50

workspace based 51

run-time applications 47

run-time dll 49-51, 53-54

run-time exe 49-52, 56

RunAsService parameter 29

S

SALT 100

registry entries 37

SaveContinueOnExit parameter 29

SaveLogOnExit parameter 29

SaveSessionOnExit parameter 29

Serial parameter 29

session object 29

session_file parameter 29, 91

SessionOnTop parameter 29, 95

ShowFiles (AutoComplete) parameter 99

ShowStatusOnError parameter 30

SingleTrace parameter 30-31, 95

SkipBlankLines parameter 30, 96

sm_cols parameter 30, 89

sm_rows parameter 30, 89

SPICE 100

sqapl.dll 55

sqapl.err 55

sqapl.ini 55
StatusOnEdit parameter 30, 95
Syncfusion 54
syntax colouring 107
system error codes 72
system error dialog 27, 71, 73
system exceptions 72

T

TabStops parameter 18, 30, 97
ToolBarsOnEdit 95
trace window geometry 30
trace_cols parameter 30
trace_first_x parameter 31, 89
trace_first_y parameter 31, 89
Trace_level_warn parameter 31, 95
trace_offset_x parameter 31, 89
trace_offset_y parameter 31, 89
Trace_on_error parameter 31, 95
trace_rows parameter 30
TraceStopMonitor parameter 31

U

Unicode and Classic Editions 2
Unicode Edition 13, 24, 27-28, 32
UnicodeToClipboard parameter 32, 97
URLHighlight parameter 78
UseDefaultHelpCollection parameter 32, 86
UTIL workspace 44

V

value tips
 colourscheme parameter 78
 delay parameter 78
 registry entries 37
valuetips
 registry entries 37
Version information
 for a bound executable 48

W

WantsSpecialKeys parameter 32

Window Captions 39
window expose 21, 104
windowrects
 registryentries 37
workspace explorer
 registry entries 36
workspace integrity check 71
workspace size 27, 41, 59
WrapSearch 95
WrapSearch parameter 32
wspath parameter 32, 43, 56, 85

X

XPLookAndFeel parameter 33, 78
XPLookAndFeelDocker parameter 33, 78
XVAR function 44

Y

year 2000 compliance 33
yy_window parameter 33