Dyalog for Microsoft Windows Installation and Configuration Guide

Dyalog version 20.0

DYALOG

Dyalog for Microsoft Windows Installation and Configuration Guide

Dyalog version: 20.0
Document Revision: 2025-10-30 main:e0843eae32

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

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

Contents

1 Installation and Configuration

1.1 Documentation

The documentation set for Dyalog is installed in the `help` sub-directory of the main Dyalog installation directory.

The latter is given by the expression:

```
      ⎕←2 ⎕NQ'.' 'GetEnvironment' 'DYALOG'
C:\Program Files\Dyalog\Dyalog APL-64 15.0 Unicode
```

EXAMPLE

```
      dyalog←2⎕NQ'.' 'GetEnvironment' 'DYALOG'
      ⎕CMD 'dir "',dyalog,'/help"'
 Volume in drive C is OS
 Volume Serial Number is 3013-866E

 Directory of C:\Program Files\Dyalog\Dyalog APL-64 15.0 Unicode\help

18/01/2016  11:53    <DIR>          .
18/01/2016  11:53    <DIR>          ..
11/01/2016  17:20           182,965 APL Workspace Transfer Guide.pdf
11/01/2016  17:20           467,005 Application Tuning Guide.pdf
11/01/2016  17:20           587,605 Code Libraries Reference Guide.pdf
11/01/2016  17:20           249,461 Compiler User Guide.pdf
11/01/2016  17:20           451,949 Conga User Guide.pdf
...
```

1.2 Files and Directories

Unicode and Classic Editions

Dyalog is available in two separate editions:

- The *Unicode* edition is Dyalog's strategic edition; it is the edition that is generally available, and the only one available to new users. It is compatible with many

third-party applications and tools, is capable of handling data from third-party tools and websites, and can be used with source code management systems (such as Git) to store APL code.
- The *Classic* edition is only available for existing customers who have been using Dyalog for a long time and for whom moving to the Unicode edition would involve considerable effort.

32-Bit and 64-Bit Widths

Two separate widths of Dyalog for Microsoft Windows are available. The 32-bit width will run on both 32-bit and 64-bit operating systems; the 64-bit width will only run on a 64-bit operating system.

Files

The names of the files that are included in a Dyalog installation can vary slightly between the different editions and widths.

For information about licences, see https://www.dyalog.com/prices-and-licences.htm or contact sales@dyalog.com.

Distributable Development Components

This section lists the files that are included with Dyalog for Microsoft Windows that can be distributed as part of end-user applications, under the terms and conditions of your Dyalog Run-Time Licence or Royalty Licence.

The following files have names that are consistent between editions; differences in names for the different widths are indicated by **<width>**, which can be either **32** or **64**):

- **dyascript.exe**
- **dyalogrt.exe**
- **dyalog<width>.dll**
- **dyares200_<width>.dll**
- **dyalogprovider.dll** (.NET Framework Interface)
- **dyalognet.dll** (.NET Framework Interface)
- **conga36ssl<width>.dll** (Conga and Ride)
- **conga36_<width>.dll** (Conga and Ride)
- **exestub.dll**
- **dllstub.dll**
- **sqapl.ini**
- **sqapl.err**

- **aplunicd.ini**
- **sharpplot.dll**
- **sharpplot.xml**

The following files relate to the .NET Interface, and are only available in Unicode editions:

- **Dyalog.Net.Bridge.Host.Windows.dll**
- **Dyalog.Net.Bridge.dll**
- **Dyalog.Net.Bridge.deps.json**
- **Dyalog.Net.Bridge.runtimeconfig.json**

The following files have names that change between editions and widths:

- 64-bit Unicode:
  - **dyalog200_64rt_unicode.dll**
  - **bridge200-64_unicode.dll**
  - **dyalogc64_unicode.exe**
  - **cwdya64u64w.dll**
- 32-bit Unicode:
  - **dyalog200rt_unicode.dll**
  - **bridge200_unicode.dll**
  - **dyalogc_unicode.exe**
  - **cwdya64u32w.dll**
- 64-bit Classic:
  - **dyalog200_64rt.dll**
  - **bridge200-64.dll**
  - **dyalogc64.exe**
  - **cwdya64c64w.dll**
- 32-bit Classic:
  - **dyalog200rt.dll**
  - **bridge200.dll**
  - **dyalogc.exe**
  - **cwdya64c32w.dll**

Non-Distributable Development Components

The following files are included with Dyalog for Microsoft Windows. These files must not be distibuted without an appropriate licence:

- **dyalog.exe**
- **dyalog200_<width>.dll** or **dyalog200_<width>_unicode.dll**

File Extension Conventions

The following file extension conventions have been adopted for the various files distributed with, and used by, Dyalog.

Extension	Description
.dws	Dyalog workspace
.dse	Dyalog Session
.dcf	Dyalog component file
.DXV	Dyalog external variable
.din	Dyalog input table
.dot	Dyalog output table
.dft	Dyalog format file
.DXF	Dyalog transfer file
.dlf	Dyalog Session log file
.dyalog	Dyalog SALT file
.dyapp	Dyalog SALT application file

> **Information**
>
> Some of these extensions (notably **.dcf**, **.dlf**, **.dot**, and **.DXF**) are not unique to Dyalog, and conflict with the same extensions used by other software applications. Although all the above file extensions are associated with Dyalog during its installation, these associations could subsequently be changed by the installation of other software or by a Microsoft Windows System restore.

1.3 File Associations

During installation, `setup.exe` associates a number of file extensions with Dyalog applications.

Workspace files with extension `.dws` and files with extension `.dyapp`, which are used to bootstrap SALT-based applications, are associated with `dyalog.exe`.

The following file types are associated with the Dyalog APL Editor `dyaedit.exe`. They are used by various source code management tools, including Link and SALT and 3rd party tools like Acre Desktop.

`.aplf`	`Functions`
`.aplo`	`Operators`
`.apln`	`Namespaces`
`.aplc`	`Classes`
`.apli`	`Interfaces`
`.dyalog`	`Generic`

Additionally, Link uses `.apla` files to store serialised arrays. These are likely to become associated with `dyaedit.exe` in a future release.

1.4 APL Fonts

Unicode Edition

The default font for the Unicode Edition is APL385 Unicode[1] 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.

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 A̲ (A-underscore).

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

1.5 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 20.0 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 and the following sections describe limitations in interoperability:

Code and ⎕ORs

Code that is saved in workspaces, or embedded within ⎕ORs stored in component files, can only be read by the Dyalog 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.
```

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

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 ⎕ORs 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 interoperability for arrays that 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.

Full cross-platform interoperability of component files is only available for large-span component files.

Null Items (⎕NULL) and Compressed Components

⎕NULLs and components from compressed component files that were created in Version 18.0 and later can be brought into Versions 16.0, 17.0 and 17.1 provided that the interpreters have been patched to revision 38151 or higher. Attempts to bring ⎕NULL or compressed component into earlier versions of Dyalog APL or lower revisions of the aforementioned versions will fail with:

```
DOMAIN ERROR: Array is from a later version of APL.
```

Object Representations (⎕OR)

An attempt to ⎕FREAD a component containing a ⎕OR that was created by a later version of Dyalog APL will generate `DOMAIN ERROR: Array is from a later version of APL`. This also applies to APL objects passed via Conga or TCPSockets, or objects that have been serialised using `220⌶`.

32 vs. 64-bit Component Files

It is no longer possible to *create* or write to small-span (32-bit) files; however it is still currently possible to *read* from small span files. Setting the second item of the right argument of ⎕FCREATE to anything other than 64 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 subject to the same restrictions as small-span component files regarding Unicode data. 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

There is complete interoperability between 32- and 64-bit interpreters, except that 32-bit interpreters are unable to work with arrays or workspaces greater than 2GB in size.

Note however that under Windows a 32-bit version of Dyalog APL may only access 32-bit DLLs, and a 64-bit version of Dyalog APL may only access 64-bit DLLs. This is a Windows restriction.

Unicode vs. Classic Editions

Two editions are available on some platforms. Unicode editions work with the entire Unicode character set. Classic editions (which are only available to commercial and enterprise users for legacy applications) 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, as these cannot 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 that cannot contain Unicode data, character data is mapped using `⎕AVU`; it can therefore be read without problems by Classic editions.

A `TRANSLATION ERROR` will occur if a Unicode edition writes to a non-Unicode component file (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 that are not in `⎕AVU`.

Likewise, a Classic edition will issue a `TRANSLATION ERROR` if it attempts to read a component containing Unicode data that is not in `⎕AVU` from a component file.

A `TRANSLATION ERROR` will also be issued when a Classic edition attempts to `)LOAD` or `)COPY` a workspace containing Unicode data that cannot be mapped to `⎕AV` using the `⎕AVU` in the recipient workspace.

`TCPSocket` objects have an `APL` property that 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 ERROR`s can occur on transmission or reception in the same way as when updating or reading a file component.

Some APL glyphs are only available in the Unicode edition, and need to be replaced with Unicode hex values in the Classic edition. These are:

Glyph	Classic replacement	Description
⍧	`⎕U2286`	*nest/partition* function
⍸	`⎕U2378`	*where/interval index* function
⍤	`⎕U2364`	*atop/rank* operator
⍠	`⎕U2360`	*variant* operator
⍈	`⎕U2338`	*key* operator
⍣	`⎕U233A`	*stencil* operator
⍥	`⎕U2365`	*over* operator
⍢	`⎕U235B`	*behind* operator

In both Unicode and Classic editions, the *variant* operator can also be represented by `⎕OPT`.

Very large array components

An attempt to read a component greater than 2GB in 32-bit interpreters will result in a `WS FULL`.

TCPSockets and Conga

TCPSockets and Conga can be used to communicate between differing versions of Dyalog APL and are subject to similar limitations to those described above for component files.

Auxiliary Processors

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

Session Files

Session (.dse) files can only be used on the platform on which they were created and saved. Under Microsoft Windows, Session files may only be used by the architecture (32-bit-or 64-bit) of the Version of Dyalog that saved them.

1.6 The APL Command Line

The command line for Dyalog APL is described below; the command line for non-Windows versions of Dyalog APL is very similar and is also documented in *Dyalog for UNIX UI Guide: Starting APL*.

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	Disables the execution of the ⎕LX expression and the derived expression when code is loaded from a source code file or directory. This applies only at start-up and does not apply to workspaces or source files that are loaded subsequently. See *Dyalog APL Language: Lx* and *Load*
-a	Start in USER mode.
-b	Suppress the banner in the Session..
-s	Disable the Session. This option is ignored in Windows versions.
+s	Force the display of the Session when it would otherwise not be shown.
-q	Don't quit APL on error (used when piping input into APL).
+q	Quit APL on error. In earlier versions of Dyalog, quitting on error saved a workspace with the reserved name CONTINUE; this behaviour can be re-enabled using 2704⌶. See *Dyalog APL Language: Continue Autosave* .
-c	Signifies a command-line comment. All characters to the right are ignored.
-cef -cef_all	Instructs Dyalog to ignore the parameter that immediately follows or all the parameters that follow. These options are intended to isolate parameters intended for the built-in Chromium Embedded Framework (CEF). See *Object Reference: Htmlrenderer* .

[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. The parameter name may be one of the standard APL parameters (see *Section 1.9.1*) or a name and value of your own choosing (see *Object Reference: Getenvironment*) . If the parameter is in a

registry sub-folder (see *Section 1.10*), its name must be preceded by the name of the sub-folder, followed by a backslash (\) or underscore (_).

> **Note**
>
> Instead of a loading a workspace specified by the **ws** option, APL can be instructed to load a program from a script file. For further information, see *Section 1.9.81*.

Examples

Start APL using the configuration file `myconfig.dcfg`:

```
"c:\program files\…\dyalog.exe" ConfigFile="myconfig.dcfg"
```

Load the workspace `myapp`, setting **MaxWS** parameter:

```
"c:\program files\…\dyalog.exe" myapp maxws=2G
```

Load the workspace `myapp`, set an application specific parameter, but do not execute the latent expression:

```
"c:\program files\…\dyalog.exe" -x myapp myparam=8080
```

Run the function defined in `myfn.aplf`:

```
"c:\program files\…\dyalog.exe" load=myfn.aplf
```

Start APL and output "Hello World":

```
"c:\program files\…\dyalog.exe" lx="⎕←'Hello World'"
```

1.7 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 `⎕OFF`, `)OFF`, `)CONTINUE`, 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.
5	APL was unable to load the Conga libraries (14.1.25383 onwards). In 16.0 the Ride libraries have been included in the Conga libraries.
6	RIDE_INIT or one of its components was ill-defined, or APL was unable to use the port, and/or unable to resolve the hostname (14.1.25383 onwards)
7	Reserved
8	Windows rejected APL's request to create a session window (in earlier versions this generated a syserror 126)
9	Dyalog has encountered a Microsoft Windows-related error when starting and is unable to continue. For example it cannot register clipboard formats.
10	CEF sub-process crash - something has gone unexpectedly wrong with either the HTMLRenderer or CEF sub-processes and cannot continue
11	Cannot create c-stack (macOS only)

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 12 for this purpose.

1.8 Dyalog Serial Number

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

The serial number should be entered during the installation process (if you already have a version of Dyalog installed then the installer should pre-populate this field with your serial number). This is recommended because if you enter it as part of the installation process then all users will automatically detect the same serial number.

If the serial number is not entered during the installation process, then it can be set by running `⎕SE.Dyalog.Serial` from within a Dyalog session. However, each individual user of that installation will have to perform this task.

To set your Dyalog serial number from within a Session:

```
      ⎕SE.Dyalog.Serial serialnumber
```

where `serialnumber` is your Dyalog serial number. This updates the registry string value **DYALOG_SERIAL** in `HKEY_CURRENT_USER\Software\Dyalog\Dyalog <version>`[2]. To complete the process you must exit and restart the Session.

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

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

or

```
      ⎕SE.Dyalog.Serial ''
```

> **Note**
>
> Using or entering a serial number other than the one issued to you is not permitted. Transferring the serial number to anyone else is not permitted.For the full licence terms and conditions, see: Terms and Conditions

[2] This string can also be set using regedit but Dyalog Ltd does not recommend this approach.

1.9 Configuration Parameters

1.9.1 Configuration Parameters

Introduction

Dyalog APL is customised using a set of **configuration parameters**. These may be defined in a number of ways, which take precedence as follows:

- Command-line settings
- Application configuration file settings
- Environment variable settings
- User configuration file settings
- Settings in the registry section defined by the **IniFile** parameter (Windows only)
- Built-in defaults

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.

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, if Dyalog parameters are specified as environment variables they must be named entirely in upper-case.

Note that the value of a parameter obtained by the GetEnvironment method (see *Object Reference: Getenvironment*) 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 **IniFile**. 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.

Changing parameter values in the Registry

You can change parameters in the Registry in one of two ways:

- Using the Configuration dialog box that is obtained by selecting *Configure* from the *Options* menu on the Dyalog APL/W session. See *Section 2.1.1* for details.
- By directly editing the Windows Registry using `REGEDIT.EXE` or `REGEDIT32.EXE`. This is necessary for parameters that are not editable via the Configuration dialog box.

1.9.2 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.

See also *Section 2.1.10*

1.9.3 APLAN_FOR_EDITOR

This parameter specifies whether (`1`) or not (`0`, the default) new **Edit** windows containing arrays should open using *Programming: Array Notation* when possible.

The setting can be toggled with the `]APLAN.Editor` user command.

In Ride and the Microsoft Windows IDE, once an **Edit** window is open, its mode can be toggled by clicking the

icon in the Editor's toolbar.

See also *APLAN_FOR_OUTPUT*.

1.9.4 APLAN_FOR_OUTPUT

This parameter specifies whether (`1`) or not (`0`, the default) to use *Programming: Array Notation* for session output when possible.

The setting can be toggled with the `]APLAN.Output` user command

In the Microsoft Windows IDE, the setting can also be toggled by clicking the

icon in the session toolbar.

See also *APLAN_FOR_EDITOR*.

1.9.5 AplCoreName

This parameter specifies the directory and name of the file in which *aplcore* should be saved. The optional wild-card character (`*`) is replaced by a number when the file is written. If there is more than one `*` in **AplCoreName**, the string is used as is; no substitution is made. For more details, including how to prevent aplcore files from being generated, see *Section 1.9.88*.

Note that APL terminates with an exit code of 3 when an aplcore file is generated.

See also *Dyalog APL Language: Aplcore Parameters*.

1.9.6 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 directory specified by the **APLKEYS** parameter. 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.

See also *Section 2.1.3*.

1.9.7 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**).

See also *Section 2.1.3*.

1.9.8 aplnid

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 **aplnid** parameter. It may be any integer in the range 0 to 65535. Note that an **aplnid** 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 **aplnid** 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.

1.9.9 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.

See also *Section 2.1.4*.

1.9.10 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.

See also *Section 2.1.4*.

1.9.11 APL_CODE_E_MAGNITUDE

The introduction of decimal floating point numbers lead to the maximum allowable print precision being increased from 17 to 34, which resulted in a change in the way numbers in the range `(10*17) to (10*34)` in function bodies are descanned[3] . 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`.

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 later versions of Dyalog causing undesired affects in code management systems.

The **APL_CODE_E_MAGNITUDE** parameter allows the user to choose between the behaviour seen in Version 12.1 and earlier and in more recent behaviour. It also allows the user to specify the size of numbers above which those numbers are display in exponential format.

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

---

3 Descanning 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.

If **APL_CODE_E_MAGNITUDE** has a value between 2 and 34, numbers greater than or equal to 10^{value} will be displayed using exponential format.

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

1.9.12 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.

In addition, if **APL_COMPLEX_AS_V12** is set to 1, objects containing complex numbers cannot be transferred to or from component files, TCP/IP (CONGA), or auxiliary processors and may not be used as an argument to Serialise/Deserialise Array (`220⌶`). Instead, a `DOMAIN ERROR` will be issued.

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.

1.9.13 APL_FCREATE_PROPS_C

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

1.9.14 APL_FCREATE_PROPS_J

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

1.9.15 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 `⎕FCHK` (optimise `⎕FUNTIE` instead)
1	Optimise `⎕FCHK`

The default value of the parameter is 0 on all platforms. On Windows, setting the value 1 has no effect.

1.9.16 APL_MAX_THREADS

Specifies the maximum number of system threads that are to be used for parallel execution. The default is 1 and the maximum value is 64.

1.9.17 APL_TextInAplCore

This Boolean parameter specifies whether or not certain information is written to an *aplcore* file when a *Programming: System Errors* occurs. The default is 1.

1.9.18 AutoDPI

This parameter determines whether or not the Dyalog program registers the application as DPI-Aware when it initialises. If 1, (the default), Dyalog performs the auto-scaling; if 0, scaling is the responsibility of the programmer or operating system. See also *Section 2.1.1*.

1.9.19 AutoComplete

AutoComplete/CancelKey1

Specifies the first of two possible keys that may be used to cancel (hide) the Auto Cancel suggestion box.

See also *Section 2.1.11*.

AutoComplete/CancelKey2

Specifies the second of two possible keys that may be used to cancel (hide) the Auto Cancel suggestion box.

See also *Section 2.1.11*.

AutoComplete/Cols

This parameter specifies the maximum number of columns (width) in the Auto Complete pop-up suggestions box.

See also *Section 2.1.11*.

AutoComplete/CommonKey1

Specifies the key that will auto-complete the *common prefix*. 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.

See also *Section 2.1.11*.

AutoComplete/CompleteKey1

Specifies the first of two possible keys that may be used to select the current option from the Auto Complete suggestion box.

See also *Section 2.1.11*.

AutoComplete/CompleteKey2

Specifies the second of two possible keys that may be used to select the current option from the Auto Complete suggestion box.

See also *Section 2.1.11*.

AutoComplete/Enabled

This parameter specifies whether or not Auto Completion is enabled

See also *Section 2.1.11*.

AutoComplete/History

Specifies whether or not Auto Complete maintains a list of previous Auto Completions.

See also *Section 2.1.11*.

AutoComplete/HistorySize

Specifies the number of previous Auto Completions that are maintained when History is 1. See *Section 1.9.19.8*.

See also *Section 2.1.11*.

AutoComplete/PrefixSize

This parameter specifies the threshold (number of characters) before Auto Completeion displays suggestions.

See also *Section 2.1.11*.

AutoComplete/Rows

This parameter specifies the maximum number of rows (height) in the Auto Complete pop-up suggestions box.

See also *Section 2.1.11*.

AutoComplete/ShowFiles

Specifies whether or not Auto Completion suggests directory and file names for `)LOAD`, `)COPY` and `)DROP` system commands.

See also *Section 2.1.11*.

1.9.20 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.

See also *Section 2.1.10*.

1.9.21 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.

See also *Section 2.1.10*.

1.9.22 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 under Windows) □PW is independent of the Session Window size.

See also *Section 2.1.9*.

1.9.23 CFEXT

This parameter specifies component file filename extensions.

CFEXT is a string that specifies a colon-separated list of one or more extensions, including any period (".") which separates the extension from its basename.

If undefined, CFEXT defaults to `.dcf:` on Windows and macOS, and `.dcf:.DCF:` on all other platforms.

In the Windows case, this means that `'myfile'⎕FTIE 0` will search first for a file named `myfile.dcf` , and then for a file named `myfile` (with no extension). As file names are not case-sensitive under Windows, this will find `myfile.DCF` or `MyFile.Dcf` and so forth. If none are found with this extension, it will load `myfile` , `MyFile` , `MYFILE` etc.

In the second (non-Windows) case note that `'myfile'⎕FTIE 0` will search first for a file named `myfile` , then `myfile.dcf` , then `myfile.DCF` .

1.9.24 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.

Note that in this mode, a maximum of 50 Trace windows may be displayed.

See also *Section 2.1.10*.

1.9.25 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. This applies only if **ClassicMode** is 1. See *Section 1.9.24*.

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

See also *Section 2.1.10*.

1.9.26 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 *Dyalog APL Language: Cmd* for implementation details.

1.9.27 ConfigFile

This parameter specifies the name of the Application Configuration file. See *Section 1.11*.

1.9.28 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.

See also *Section 2.1.10*.

1.9.29 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.

See also *Section 2.1.10*.

1.9.30 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.

See also *Section 2.1.10*.

1.9.31 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.

See also *Section 2.1.9*.

1.9.32 Default_DIV

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

See also *Section 2.1.9*.

1.9.33 Default_IO

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

See also *Section 2.1.9*.

1.9.34 Default_ML

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

See also *Section 2.1.9*.

1.9.35 Default_PP

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

See also *Section 2.1.9*.

1.9.36 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.

1.9.37 Default_RTL

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

See also *Section 2.1.9*.

1.9.38 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.

See also *Section 2.1.14* and *Section 2.1.9*.

1.9.39 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.

See also *Section 2.1.7*.

1.9.40 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 applies only if **ClassicMode** is 1. See *Section 1.9.24*.

See also *Section 2.1.10*.

1.9.41 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.

See also *Section 2.1.10*.

1.9.42 Dyalog

This parameter specifies the name of the directory in which Dyalog APL is installed. If undefined, the name of the directory from which the Dyalog APL program was loaded is assumed.

1.9.43 DyalogEmailAddress

This parameter specifies the contact email address for Dyalog Limited.

1.9.44 DyalogHelpDir

This parameter specifies identifies the location of the HTML-based help that is used for a request for help (from the Help menu or pressing `F1`) from the Session. It may be:

- The full pathname of the directory that contains the Dyalog APL help file (`dyalog.chm`). This is the default.

- The index page of the on-line HTML-based help located on `help.dyalog.com` for example, `https://help.dyalog.com/18.0`.

1.9.45 DyalogInstallDir

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

1.9.46 DyalogLink

This parameter specifies the name of the directory containing the code for Link. The default is `[DYALOG]/StartupSession/Link`.

Note that Link is required for Session initialisation.

For further information, see https://dyalog.github.io/link/4.0/Usage/Installation.

1.9.47 DyalogStartup

This parameter specifies the name of a file that contains APL code to be run each time Dyalog starts. If this is undefined, the default file is named `SessionStartup` with the file extension `.aplf`, `.apln` or `.aplc`, in the `Dyalog` directory.

See also *User Interface: Session Initialisation*.

1.9.48 DyalogStartupSE

This parameter specifies one or more *Session initialisation* directories that contain APL code to be installed in `⎕SE`. If this parameter is not specified, the default is a directory named `StartupSession` located in three standard locations.

Under Windows these might be:

1. `C:\Program Files\Dyalog\Dyalog APL-64 19.0 Unicode`
2. `C:\Users\Pete\Documents\Dyalog APL Files`
3. `C:\Users\Pete\Documents\Dyalog APL-64 19.0 Unicode Files`

The version-specific name is :

```
Dyalog APL{bit} {version} {edition}
```

where:

- {bit} is "`-64`" if 64-bit version, otherwise nothing
- {version} is the main and secondary version numbers of dyalog.exe separated by ".".
- {edition} is "`Unicode`" for the Unicode Edition, otherwise nothing

The parameter is a string containing the list of directory names separated by ";" on Windows, ":" elsewhere.

If DyalogStartupSE begins with the specified separator, the default list is *extended* rather than *replaced*.

Note that the effective sequence of directories specified by this parameter is converted to a vector of character vectors and stored in `⎕SE.Dyalog.StartupSession.AllPaths`.

If unset or extended (that is, starts with a : separator):

- the effective StartupSession directory in [DYALOG] is available as `⎕SE.Dyalog.StartupSession.Dyalog`.
- the StartupSession directory in the version-agnostic directory is available as `⎕SE.Dyalog.StartupSession.VerAgno`.
- the StartupSession directory in the version-specific directory is available as `⎕SE.Dyalog.StartupSession.VerSpec`.

See also *User Interface: Session Initialisation*.

1.9.49 DyalogStartup_X

During Session initialisation, code is loaded from the directories specified by the **DyalogStartupSE** parameter into a corresponding namespace tree in the Session namespace `⎕SE`. Optionally, the code is then executed.

If **DyalogStartup_X** is 0 (the default if not defined), the `Run` function (if it exists) in each *top-level* namespace loaded during Session start-up is executed. The namespaces are processed in alphabetical order.

If **DyalogStartup_X** is 1, the `Run` function is not executed.

Other values are reserved for future extension.

See also: *Section 1.9.48*.

1.9.50 DyalogWebSite

This parameter specifies the URL for the Dyalog web site.

1.9.51 DYALOG_DISCARD_FN_SOURCE

This Boolean parameter specifies whether (1) or not (0) source code is discarded from the workspace when an object is fixed. The default value is 0 which means that source code is retained in the workspace and will subsequently be presented for editing as it had been saved previously.

For further information, see *Dyalog APL Language: Discard Source Information* and *Earlier Release Notes: Source As Typed*.

1.9.52 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 *Programming: Handling Unexpected Errors* for more information.

1.9.53 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 *Programming: Handling Unexpected Errors* for more information.

1.9.54 DYALOG_GUTTER_ENABLE

This Boolean parameter specifies whether (1) or not (0) a Gutter is displayed in the left-most column of the Session window. This gutter is used to display:

- A small red circle. This indicator is used on every line that is modified in the session, including old ones (for example, if you move up the session and modify them, without pressing `<ER>`) . The indicators show which session lines will be re-executed when you subsequently press `<ER>`.
- A left bracket `[` to identify groups of default output. Note that other forms of output are not identified in this way.

The default value is 0 for the TTY interface, and 1 otherwise.

1.9.55 Dyalog_LineEditor_Mode

This Boolean parameter specifies whether or not multi-line input is enabled in the Session.

See also *Section 2.1.9* and *User Interface: Multiline Session Input*.

1.9.56 Dyalog_NETCore

This Boolean parameter specifies whether the .NET interface is enabled. On Windows the default is 0 which disables the .NET interface in favour of the .NET Framework interface. If it is set to 1, Dyalog uses .NET instead of the .NET Framework.

On other platforms which support .NET, the default is 1.

1.9.57 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 *Programming: Handling Unexpected Errors* for more information.

1.9.58 Dyalog_Pixel_Type

When the Coord property is set to `'Pixel'` , this parameter specifies how it is interpreted. If the value of **Dyalog_Pixel_Type** is RealPixel or if Dyalog_Pixel_Type is undefined, the object behaves as if Coord was `'RealPixel'` . If the value of **Dyalog_Pixel_Type** is ScaledPixel, the object behaves as if Coord were `'ScaledPixel'`. See *Object Reference: Coord*.

See also *Section 2.1.1*.

1.9.59 DYALOG_SERIAL

This parameter contains your Dyalog serial number. This must be set to the serial number issued to you. If not set, then the software is unregistered.

For the full licence terms and conditions, see https://www.dyalog.com/uploads/documents/Terms_and_Conditions.pdf.

1.9.60 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.

1.9.61 Edit_Cols

This parameter specifies the initial width of an edit window in character units.

See also *Section 2.1.8*.

1.9.62 Edit_First_X

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

See also *Section 2.1.8*.

1.9.63 Edit_First_Y

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

See also *Section 2.1.8*.

1.9.64 Edit_Offset_X

This parameter specify the number of characters by which an edit window is staggered horizontally from the previous one.

See also *Section 2.1.8*.

1.9.65 Edit_Offset_Y

This parameter specify the number of characters by which an edit window is staggered vertically from the previous one.

See also *Section 2.1.8*.

1.9.66 Edit_Rows

This parameter specifies the initial height of an edit window in character units.

See also *Section 2.1.8*.

1.9.67 Enable_CEF

This parameter is a Boolean value with a default value of 1. If set to 0, it disables the Chromium Embedded Framework (CEF). and an attempt to create an *Object Reference: Htmlrenderer* object will fail with an error message.

Note

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

1.9.68 ErrorOnExternalException

This parameter 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.

1.9.69 ExternalHelpURL

If **UseExternalHelpURL** 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. This parameter specifies the URL to be used. In most cases the default setting will be sufficient. On some configurations it may be necessary to change this. See *Section 1.9.125*.

See also *Section 2.1.7*.

1.9.70 File_Stack_Size

This parameter specifies the number of the most recently used workspaces displayed in the Session File menu. See *Section 2.1.1*.

1.9.71 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 *Object Reference: Greetbitmap* method of the Root object.

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

1.9.72 History_Size

This parameter specifies the size of the buffer used to store previously entered (input) lines in the Session. See *Section 1.9.1* for further details about defining a valid value for this parameter. The maximum value is 2Gb.

See also *Section 2.1.9*.

1.9.73 IniFile

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
```

The default values for **IniFile**, for the 64-bit and 32-bit versions respectively, are:

Unicode Edition

```
    Software\Dyalog\Dyalog APL/W-64 20.0 Unicode
    Software\Dyalog\Dyalog APL/W 20.0 Unicode
```

Classic Edition

```
    Software\Dyalog\Dyalog APL/W-64 20.0
    Software\Dyalog\Dyalog APL/W 20.0
```

See also *Section 2.1.1*.

1.9.74 InitFullScriptNormal

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 (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 Editor (The File Menu, editing a script)) will change according to which behaviour applies. Note that if you specify a keystroke for in the *Keyboard Shortcuts* tab, this will be associated with the unselected action.

See also *Section 2.1.10*.

1.9.75 InitFullScriptSusp

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 (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 Editor (The File Menu, editing a script)) will change according to which behaviour applies. Note that if you specify a keystroke for in the *Keyboard Shortcuts* tab, this will be associated with the unselected action.

See also *Section 2.1.10*.

1.9.76 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.

See also *Section 2.1.2*.

1.9.77 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.

See also *Section 2.1.2*.

1.9.78 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).

See also *Section 2.1.2*.

1.9.79 Input_Size

This parameter specifies the size of the buffer used to store marked lines (lines awaiting execution) in the Session. See *Section 1.9.1* for further details about defining a valid value for this parameter.

See also *Section 2.1.9*.

1.9.80 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 Interface: Session Manager*.
- offering a list of names for auto-completion. See *Section 2.1.11*

1.9.81 Load

This parameter is a character string that specifies the name of a workspace, or a directory or text file containing APL source code, to be loaded when Dyalog starts.

If Load specifies a text file, `2 ⎕FIX` is used to import the file contents and associate that file with each of the objects that have been fixed in the workspace.

If Load specifies a directory, Link is used to associate the directory with the active workspace and to import the code. For more information about Link, see https://dyalog.github.io/link.

The **Load** parameter will normally be specified on the command line or in a Configuration file.

Having loaded the workspace, or fixed the code from the named file or directory, Dyalog executes the expression specified by the `LX` parameter if it is set. See *Section 1.9.86*.

If `LX` is not set, Dyalog checks whether or not the **-x** command line option was specified. If so, no further action is taken. See *Section 1.6*.

Otherwise, Dyalog executes an expression which is derived as follows.

If the value of Load is a directory, Dyalog will execute the expression:

```
Run ,⊂<Load>
```

where `<Load>` is the value of the **Load** parameter.

If the value of Load is the name of a file, Dyalog determines whether or not the file is a workspace by its internal signature.

If the file is a workspace the expression to be executed is specified by its `⎕LX`. See *Dyalog APL Language: Lx*.

Otherwise, if the file extension is `.aplf` `.aplc` or `.apln` the expression is shown in the table below, where `filename` is the file name specified by the **Load** parameter without its extension.

File Extension	Type	Expression
`.aplf`	Function source code	`filename 0⍴⊂''`
`.aplc`	Class source code	`filename.Run 0⍴⊂''`
`.apln`	Namespace source code	`filename.Run 0⍴⊂''`

Notes

- The **Load** parameter overrides a workspace name specified as the last item on the command line.
- The argument `0⍴⊂''` may change in a future version of Dyalog.
- Nothing is executed when code is loaded from source files that define operators (`.aplo`) or Interfaces (`.apli`).

1.9.82 localdyalogdir

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

1.9.83 Log_File

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

The Session log file is not interchangeable between different versions/editions/widths of Dyalog – this means that opening a new instance of Dyalog will overwrite any contents of the Session log file populated by an already-running instance. However, if the LOG_FILE parameter contains a '`*`' (for example, `JD.*.dlf`) then at start-up Dyalog will attempt to open, and then **lock**, a file where the '`*`' has been replaced with an increasing integer value (starting with 000, so `JD.000.dlf`, `JD.001.dlf` etc). If said file cannot be opened and locked, the value will be incremented. The process will fail, and no log will be used if the extension number would exceed 999.

The default is `Users\<username>\Documents\Dyalog APL-<bits> <DyalogMajor>.<DyalogMinor> <Unicode|Classic> Files\default_*.dlf`, for

example, `Users\Bob\Documents\Dyalog APL-64 19.0 Unicode Files\default_*.dlf`

Note that the LogFile property of `⎕SE` reports the name of the log file that is being used.

See also *Section 2.1.9*.

1.9.84 Log_File_InUse

This Boolean parameter specifies whether or not the Session log is saved. The default is 1 meaning that the Session log is saved in a Session log file and loaded the next time a Session is started. If set to 0, the Session log is not saved

See also *Section 2.1.9*.

1.9.85 Log_Size

This parameter specifies the size of the Session log buffer. See *Section 1.9.1* for further details about defining a valid value for this parameter. The maximum value is 2Gb.

See also *Section 2.1.9*.

1.9.86 LX

This parameter specifies an expression to be executed after Dyalog has started and loaded a workspace or a text file containing APL source code. Also see *Section 1.9.81*. This expression is run only on Dyalog start-up and overrides the workspace latent expression `⎕LX`.

The LX parameter applies only to the development version of Dyalog and is ignored in run-time applications.

The LX parameter is ignored when a workspace is loaded other than at start-up of the Dyalog program.

The LX parameter applies only to the Unicode edition of Dyalog and is ignored in Classic edition.

1.9.87 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 .

1.9.88 MaxAplCores

This parameter is used in conjunction with the **AplCoreName** parameter to control the maximum number of*aplcore* files that are saved. It applies when the string specified by **AplCoreName** ends with an asterisk (). *If so, when saving an* aplcore* file, Dyalog performs the following steps:

1. Identifies the highest number ending of those files that match the directory/ name pattern specified by **AplCoreName** . If none, assume 0.
2. Increments that number, then saves the *aplcore* in a new file ending with the new number.
3. If necessary, deletes lower-numbered files to retain only the maximum number of files specified by **MaxAplCores** .

See also: *Section 1.9.5*.

See also *Dyalog APL Language: Aplcore Parameters*.

1.9.89 MaxWS

This parameter determines your workspace size and is the amount of memory allocated to the workspace at APL start-up. See *Section 1.9.1* for further details about defining a valid value for this parameter.

The default value is 256M (256MiB), with the exception of the Raspberry Pi where the default is 64M. Values less than 4M are ignored, and the maximum value is 15E.

For example, to get a 4GiB 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 between 1.3GiB to 1.9GiB under Windows, and 1.9GiB under UNIX. These are operating system limitations imposed on 32-bit processes rather than ones imposed by Dyalog APL, and are affected by the number and size of DLLs/shared libraries that are loaded into the process space.

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

See also *Section 2.1.6*.

1.9.90 OverstrikesPopup

Unicode Edition only.

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

1.9.91 PassExceptionsToOpSys

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

1.9.92 PFKey_Size

This parameter specifies the size of the buffer that is used to store programmable function key definitions. See *Dyalog APL Language: Pfkey*.

For further details about defining a valid value for this parameter, see *Section 1.9.1*.

See also *Section 2.1.9*.

1.9.93 ProgramFolder

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

1.9.94 PropertyExposeRoot

Each workspace contains a flag that 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. This parameter determines the default value of the flag in a `CLEAR WS`.

See also *Section 2.1.14*.

1.9.95 PropertyExposeSE

Each workspace contains a flag that 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. This parameter determines the default value of the flag in a `CLEAR WS`.

See also *Section 2.1.14*.

1.9.96 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.

1.9.97 ResolveOverstrikes

Unicode Edition only.

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

1.9.98 RIDE_Init

This parameter determines how the interpreter should behave with respect to the Ride protocol. Setting this configuration parameter on the machine that hosts the interpreter enables the interpreter-Ride connection.

RIDE_Init can only be used to specify a limited number of Ride configuration options; the rest must be specified in a Ride ini file. Full details describing how to configure Ride, including using certificates to authenticate connections can be found in the Ride User Guide.

The format of the value is:

```
<setting> : <address> : <port>
```

setting is the action the interpreter should take. Valid values, which are case-insensitive, are as follows:

- serve – listen for incoming connections
- http - listen for an incoming request for Zero Footprint Ride
- connect – connect to the specified Ride and end the session if this fails
- poll – try to connect to the specified Ride at regular intervals and reconnect if the connection is lost
- config - specifies the name of the Ride ini file to be used

For serve and http,

is a list of IPv4 or IPv6 addresses and/or DNS names of interfaces in the machine where the APL process is running, and specifies the interfaces through which incoming requests to connect are accepted. If
is empty, incoming requests are accepted only from the machine itself (the interpreter will listen on the loopback addresses only). If
is set to "" then the interpreter will listen for requests through all the available interfaces in the local machine.*

If *setting* is `serve` or `http` then *address* is a list of IPv4 or IPv6 addresses and/or DNS names of interfaces in the machine where the APL process is running, and specifies the interfaces through which incoming requests to connect are accepted. If *address* is empty, incoming requests are accepted only from the machine itself (the interpreter will listen on the loopback addresses only). If *address* is set to "*" then the interpreter will listen for requests through all the available interfaces in the local machine.

If *setting* is `connect` or `poll` then *address* is an IP address or DNS name of an interface in a remote machine to which the interpreter should attempt to connect. Valid address values are:

- a resolvable name
- an IPv4 or IPv6 address
- empty – the local machine only
- ◦ ▪ (valid only when setting is serve or http) the interpreter listens on all local network interfaces

port is the TCP port to listen on

Settings specified by the **RIDE_Init** configuration parameter take precedence over the same setting specified in the Ride ini file. Note that the **RIDE_Init** configuration parameter can specify both *config* and one of *serve*, *http*, *connect* or *poll*. For example,

```
RIDE_INIT=serve:*:4502,config=/home/andys/.dyalog/secureride.ini
```

This is most useful when multiple interpreters need to be run, each with its own Ride connection as each must have a separate port number.

Note that the **RIDE_Init** configuration parameter is set automatically when launching a new Dyalog Session from Ride.

Examples

To allow an incoming connection through any interface in the machine running the interpreter:

```
RIDE_INIT=serve:*:4052
```

To allow incoming Zero Footprint Ride connection through just one interface of the machine running the interpreter:

```
RIDE_INIT=http:192.168.0.10:8080
```

To attempt to connect to Ride running on my colleague's machine:

```
RIDE_INIT=connect:pete.dyalog.com:4052
```

1.9.99 Ride_Spawned

If non-zero, this parameter disables `⎕SR` and `)SH` which instead generate `DOMAIN ERROR`. This parameter is used to prevent certain user-interfaces from being executed from a Ride session which does not support them, and which would otherwise cause the Ride session to become unresponsive. See Ride User Guide.

1.9.100 RunAsService

When RunAsService is set to 1 or 2 (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. The value 2 reduces the resources used by a Dyalog service by disabling the graphical user-interface features. In this mode, `⎕WC object` will fail with a `LIMIT ERROR` unles the object is Timer, which is the only one that remains enabled.

1.9.101 SaveContinueOnExit

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

1.9.102 SaveLogOnExit

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

1.9.103 SaveSessionOnExit

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

1.9.104 Serial

Legacy: Specifies your Dyalog APL/W Serial Number. See*Section 1.9.59* which supercedes it..

1.9.105 SessionOnTop

Specifies whether or not the Session may appear on top of Edit and Trace Windows in Classic Dyalog mode. This applies only if **ClassicMode** is 1. See *Section 1.9.24*.

See also *Section 2.1.10*.

1.9.106 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.

See also *Section 2.1.9*.

1.9.107 ShowStatusOnError

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

1.9.108 SingleTrace

Specifies whether there is a single Trace window, or one Trace window per function. This applies only if **ClassicMode** is 1. See *Section 1.9.24*.

See also *Section 2.1.10*.

1.9.109 SkipLines

This parameter causes the Tracer to automatically skip lines that contain no executable statement, 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. SkipLines is an integer made up of the sum of the following values:

1	Skip blank lines. See also *Skip blank lines when tracing* .
2	Skip comment lines. See also *Skip comment lines when tracing* .
4	Skip locals lines. See also *Skip locals lines when tracing* .

1.9.110 SM_Cols

This parameter specifies the width in characters of the window used to display ⎕SM when it is used *stand-alone* . It is **not** used if the window is specified using the SM object.

1.9.111 SM_Rows

This parameter specifies the height in characters of the window used to display SM when it is used *stand-alone* . It is **not** used if the window is specified using the SM object.

1.9.112 StatusOnEdit

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

See also *Section 2.1.10*.

1.9.113 TabStops

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

See also *Section 2.1.10*

1.9.114 ToolBarsOnEdit

Specifies whether or not tool bars are displayed along the top of individual Edit windows.

See also *Section 2.1.10*.

1.9.115 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.

1.9.116 Trace_First_X

This parameters specifies the initial horizontal 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.

See also *Section 2.1.8*.

1.9.117 Trace_First_Y

This parameters specifies the initial vertical 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.

See also *Section 2.1.8*.

1.9.118 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.

See also *Section 2.1.10*.

1.9.119 Trace_Offset_X

This parameter specifies the number of characters by which a trace window is staggered horizontally from the previous one. This applies only if **ClassicMode** is 1 and **SingleTrace** is 0.

See also *Section 2.1.8*.

1.9.120 Trace_Offset_Y

This parameter specifies the number of characters by which a trace window is staggered vertically from the previous one. This applies only if **ClassicMode** is 1 and **SingleTrace** is 0.

See also *Section 2.1.8*.

1.9.121 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.

See also *Section 2.1.10*.

1.9.122 UCMDCacheFile

This parameter specifies the name of the User Command cache file.

The default value is:

```
UserCommand{UcmdMajor}{UcmdMinor}.{DyalogMajor}{DyalogMinor}{U|C}
{bits}.cache
```

For example, `UserCommand25.182U64.cache`

1.9.123 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 ⎕AV 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 ⎕AV are mapped to Unicode text and the APL symbols are mapped to their genuine Unicode equivalent values.

See also *Section 2.1.10*

1.9.124 URLHighlight

Specifies whether or not URLs and links are highlighted in Session and Edit windows. Its value is either 1 (highlight) or 0. The default is 0.

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:

See also *Section 2.1.1*.

1.9.125 UseExternalHelpURL

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 *Section 1.9.69*.

See also *Section 2.1.7*.

1.9.126 UserConfigFile

This parameter specifies the name of the User Configuration file. See *Section 1.11*.

1.9.127 UseXCV

This Boolean parameter specifies how the commonly used keystrokes for copy (ctrl+c), cut Ctrl+x) and paste (ctrl+v) are processed.

0 = process normally (via the appropriate .DIN file)1 = pass untranslated to the host application

The **UseXCV** parameter is defined for the IME in the Registry section `HKEY_CURRENT_USER\Software\Dyalog\UnicodeIME\`

When **UseXCV** is 1, the keystrokes Ctrl+X, Ctrl+C and Ctrl+V are passed untranslated to `dyalog.exe` which treats them as CT, CP and PT respectively. This is likely to be true for other host applications using the Dyalog keyboard.

The standard Dyalog keyboard (*.din) files map Shift+Del to CT, Ctrl+Ins to CP, and Shift+Ins to PT. These will therefore work independently of the UseXCV** option.

The standard Dyalog keyboard (*.din) files map BOTH Ctrl+X and Ctrl+Shift+Xto ⊃ So if UseXCV** is set to 1, you must use Ctrl+Shift+X to obtain ⊃. Likewise for C and V.

1.9.128 ValueTips

ValueTips/ColourScheme

This parameter specifies the colour scheme used to display a Value Tip when the user hovers the mouse over a name.

See also *Section 2.1.1*.

ValueTips/Delay

This parameter specifies the delay before a Value Tip is displayed when the user hovers the mouse over a name.

See also *Section 2.1.1*.

ValueTips/Enabled

This parameter specifies whether or not Value Tips are enabled. When enabled, Dyalog displays the value of a variable or the code for a function when the user hovers the mouse over its name.

See also *Section 2.1.1*.

1.9.129 WantsSpecialKeys

Unicode Edition only.

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

1.9.130 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.

See also: *Section 2.1.10*.

1.9.131 WrapSearchMsgBox

Specifies whether or not a message box is displayed to inform the user when the search wraps.

See also *Section 2.1.10*.

1.9.132 WSEXT

This parameter specifies workspace filename extensions. It complements the **WSPATH** parameter in that together they determine the file search order to satisfy `)LOAD` or `)COPY`; it also specifies the filename extension to add on `)SAVE` or `)CONTINUE` if none is explicitly provided.

WSEXT is a string that specifies a colon-separated list of one or more extensions, including any period (".") which separates the extension from its basename.

If undefined, WSEXT defaults to `.dws:` on Windows and macOS, and `:.dws:.DWS` on all other platforms.

In the Windows case, this means that `)LOAD myws` will search first for a file named `myws.dws` , and then for a file named `myws` (with no extension). As file names are not case-sensitive under Windows, this will find `myws.DWS` or `MyWs.Dws` and so forth. If none are found with this extension, it will load `myws` , `MyWs` , `MYWS` etc.

In the second (non-Windows) case note that `)LOAD myws` will search first for a file named `myws` , then `myws.dws` , then `myws.DWS`.

When `)SAVE` and `⎕SAVE` is used without specifying a file extension, the first extension defined by WSEXT is applied to complete the file name. The default is therefore `.dws` in all cases.

1.9.133 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 without explicitly specifying a path in the name. The directory paths are specified using Operating System specific conventions and separated by ";" (Windows) or ":" (UNIX).

Note that to load workspaces from the current directory, "." must be included in the list defined by **WSPath**..

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

```
WSPath=.;D:\MYWS
```

See also *Section 2.1.6*.

1.9.134 XPLookAndFeel

This Boolean 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.

1.9.135 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 `⎕SM` 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

1.10 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 types of Registry values are considered to be internal and are therefore not described herein.

However, any 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 *Section 2.1.11*

Captions

This contains registry entries to customise the Captions used in the various windows of the Dyalog APL IDE. See *Section 1.12*.

Colours

This contains entries that describe the colour schemes you have and your personal preferences. See *Section 2.2*.

Editor

This contains certain entries for the Editor.

Event Viewer

This contains entries that describe your settings for the Event Viewer. See UI Guide:

The Event Viewer.

Explorer

This contains entries that describe your settings for the *User Interface: Workspace Explorer*.

files

This contains the size of your recently used file list (see *Section 2.1.1*) and the list of your most recently loaded workspaces.

KeyboardShortcuts/keys

This contains the definitions of your Keyboard Shortcuts (Unicode Edition only). See *Section 2.1.5*.

KeyboardShortcuts/chars

This contains the Registry Keyboard mappings between keystrokes and APL characters (Unicode Edition only). See *User Interface: Apl Keyboards*.

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 *Section 2.3*.

SALT

This contains entries for SALT. See *Section 2.1.12*.

Search

This contains dynamic entries for the Find Objects Tool. See UI Guide:

Find Objects Tool.

Threads

This contains entries to remember your preferences for Threads. See UI Guide:

The Threads Menu.

UnicodeIME

This contains entries for the Dyalog Unicode IME.

ValueTips

This contains entries for your Value Tips preferences. See UI 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.

1.11 Configuration Files

Introduction

A configuration file is an optional text file containing configuration parameters and their values. It may cascade, that is, it can extend (inherit) configuration values from other configuration files, and supplement and/or override them.

Configuration files use JSON5 (a superset of standard JSON) syntax, as described below. These files are portable across all systems supported by Dyalog.

Although it is possible to include user credentials such as login details or passwords in configuration files, Dyalog very strongly recommends against doing this even if the credentials are encrypted as this should be considered to be a very significant security risk.

Names of configuration parameters defined in Configuration files may be specified in any combination of alphabetic case.

Dyalog processes up to two kinds of configuration file (each of which may cascade):

1. An application configuration file which contains configuration values associated with a specific application
2. A user configuration file which defines configuration values for the current, and possibly only, user of the system.

Application Configuration File

When Dyalog starts, it derives the name of the application configuration file as follows:

- The name in the configuration parameter **ConfigFile** if it is set, otherwise
- The name of the workspace or script loaded at start-up using the **Load** parameter, with the extension replaced by `.dcfg`, if that file exists, otherwise
- Nothing.

User Configuration File

The name of the user configuration file is specified by the **UserConfigFile** parameter. Under Windows, this parameter is not set by default but may be defined by the user.

Precedence

Configuration files supplement existing methods of defining parameters. The following precedence table shows the order of precedence when a setting is defined in multiple places:

- Command-line settings override
- Application configuration file settings, which override
- Environment variable settings, which override
- User configuration file settings, which override
- Settings in the registry (Windows only), which override
- Built-in defaults

Configuration Files and The Configuration Dialog

The Configuration Dialog reflects the values of parameters stored in the Windows Registry and ignores overriding values defined on the command-line, in configuration files or in environment variables. If the user changes parameters using the

Configuration Dialog, the new values are recorded in the Registry, but remain overridden by those that take precedence.

Configuration File Structure

Configuration files define configuration parameters using JSON5. A JSON object contains data in the form of key/value pairs and other JSON objects. The keys are strings and the values are the JSON types. Keys and values are separated by colon. Each entry (key/value pair) is separated by comma.

The top-level object defines an optional key named **Extend** and an optional object named **Settings**.

Extend is a string value containing the name of a configuration file to import. The extended (imported) file may in turn extend another configuration file. Configuration values from the imported file(s) may be overridden by redefining them. The file name is implicitly relative to the name of the file which imports it. Any file name extension must be explicitly specified.

Settings is an object containing the names of configuration parameters and their values. The values may be:

- A string
- A number
- An array of strings

The names and values correspond to configuration parameters, but names are not case sensitive. Any named values may be defined; an APL application may query the values using `+2⎕NQ'.' 'GetEnvironment' name`, or using the `]config` user command. Note that `GetEnvironment` returns the value in use as defined by the precedence rules (see **Precedence above**).

Example

```
      +2 ⎕NQ '.' 'GetEnvironment' ('MaxWS' 'Captions\Session')
┌────┬──────────────────────┐
│256M│My Dyalog V18.0 Session│
└────┴──────────────────────┘
      ]config MaxWS Captions\Session
┌──────────────┬──────────────────────┐
│MaxWS         │256M                  │
├──────────────┼──────────────────────┤
│Captions\Session│My Dyalog V18.0 Session│
└──────────────┴──────────────────────┘
```

A warning will be given if names are redefined in the same configuration file; the second and subsequent definitions will be discarded.

File Names

Pathnames specified in configuration files should be specified using portable forward slashes "/" rather than back-slashes "\" as the latter are used as escape characters by JSON.

`WSPATH: ["c:/Dyalog18.0"]` or `WSPATH: ["c:\\Dyalog18.0"]` specifies the file `c:\Dyalog18.0`.

whereas,

`WSPATH: ["c:\Dyalog18.0"]` means `c:Dyalog18.0`.

EXAMPLE

```
{
  Extend: "my_default_configuration.dcfg",

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

Arrays

An array may be used to define file paths etc. For example,

```
WSPATH: ["/dir1", "/dir2"]
```

The only parameters which may be defined as arrays are **WSPATH**, **WSEXT** and **CFEXT**.

References to other Configuration Parameters

Configuration parameters which are string values may include references to other configuration parameters (regardless of where they are defined) using square bracket delimiters. For example:

```
MySetting: "[DYALOG]/MyFile"
```

will replace `[DYALOG]` with the value of the **DYALOG** configuration value.

If the string inside the `[]` delimiters is `"."`, the `"."` is replaced is replaced with the path of the directory containing the configuration file itself. Therefore,

```
FILENAME: "[.]/x.txt"
```

will set the parameter **FILENAME** to a value which is a reference to a file called `x.txt` in the same directory as the configuration file defining it.

Note that:

- If the referenced configuration parameter is not defined then no substitution will take place; the reference, including square bracket delimiters, will remain in place.
- To include square brackets in a string, prefix the '[' with a '\' character.

Nested Structures

Some parameters are stored in sub-folders in the Windows Registry. Currently, all such parameters used by Dyalog APL itself relate to the Windows IDE, but you can create your own application-specific structures..

The Configuration file supports this structure by defining an object that corresponds to a Registry sub-folder. For example:

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

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

1.12 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
SysTray	The hint on Dyalog icons in the System Tray
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
Event Viewer	The Event Viewer
FindReplace	The Find/Replace dialog box
ExitDialog	The Exit dialog box that is displayed when the user closes the Session window
WSSearch	The Find Objects tool
Syserror	The Syserror Message Box

Each string value should contain a mixture of your own text and keywords which are enclosed in braces, for example, {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, for example, "Dyalog APL/W - 64"
{VER_A}	The main version number, for example, "14"
{VER_B}	The secondary version number, for example, "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
```

1.13 Workspace Management

Workspace Size and Compaction

The *maximum* amount of memory allocated to a Dyalog APL workspace is defined by the **maxws** parameter (on non-Windows platforms this is defined by the environment variable MAXWS).

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 1/16th of maxws, except if there is less than 1/16th of **maxws** in use, delta is 1/64th of **maxws**. This may also be expressed as follows:

```
      delta←maxws{⌈⍺÷⍨(⍵>⍺÷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 `⎕DR` 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 1/64th **maxws**. This means that workspaces that are operating within 1/16th 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 functions for managing the memory used by the workspace are described in *Dyalog APL Language: Memory Manager Statistics* and *Dyalog APL Language: Specify Workspace Available*.

1.14 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.

1.15 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

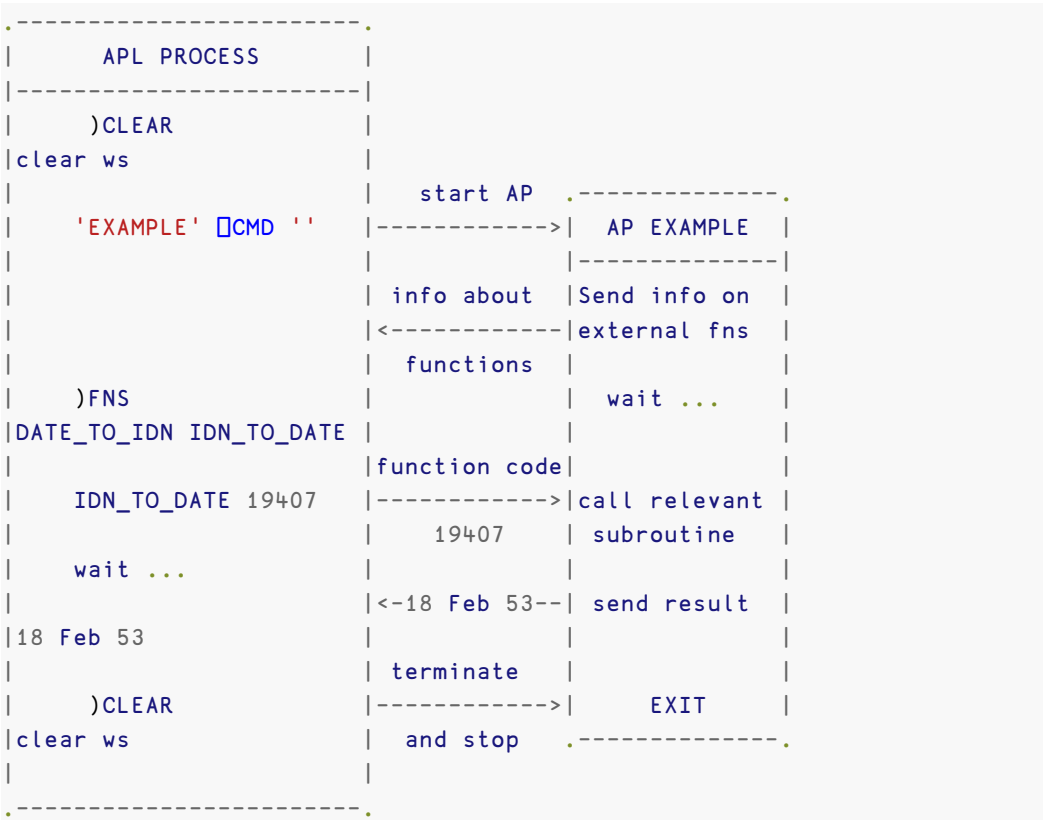
Once established, an AP is used by making a reference to one of its external functions. An external function behaves as if it was a locked defined function, but it is in effect an entry point to the AP. When an external function is referenced, APL transmits a code number to the AP, followed by any arguments. 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.



1.16 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.

> **Information**
>
> Support for external variables has been deprecated, and is scheduled for removal in a future release. For information on how to identify calls to external variables in your existing codebase, see the *Release Notes*.

1.17 Shell Scripts

Shell scripts are typically executed from a terminal (or *shell*).

A script is executed by typing its name. User input is entered from the same terminal or shell and output is displayed on the terminal or shell.

UNIX

On UNIX (and related) systems a Dyalog APL *shell script* is a text file with the following as the first line:

```
#!/usr/local/bin/dyalogscript
```

The script file must be executable. There are three execute bits relating to the user, the group and everyone else.

Windows

On Windows systems a Dyalog APL shell script is a text file with a `.apls` file extension. An initial line beginning with `#!` is only required to include configuration parameters (see below), but if included it must include a file name even though that will be ignored. For portability it is recommended that you include the `#!` line.

Note

Shell scripts are Unicode only and are not supported by the Classic Edition.

Any content that follows the `#!` line (if present) is used as input into a Dyalog session (as if the *Extended Multiline Input* feature has been enabled).

Input and Output

⍞ and ⎕ input are taken from characters typed by the user into the terminal or shell (Standard input or *stdin* for short). Anything assigned to ⍞ and ⎕ will be displayed in the terminal window using streams Standard output (*stdout*) and Standard error (*stderr*) respectively. Note that default output, that is, output to the session without assignment to ⍞ or ⎕ is NOT displayed. Redirections of *stdin*, *stdout*, and *stderr* are supported.

Examples

The following then are all valid APL shell scripts:

```
#!/usr/local/bin/dyalogscript
'this text will not be seen'
⍞←2+2
```

```
#!/usr/local/bin/dyalogscript
∇r←l plus r
r←l+r
∇
⍞←2 plus 2
```

```
#!/usr/local/bin/dyalogscript
plus←{
⍺+⍵
}
⍞←2 plus 2
```

Errors

Untrapped errors in a script will cause the termination of the process, further lines in the script will NOT be processed.

```
#!/usr/local/bin/dyalogscript
⍞←'this will be seen'
⍞←÷0
⍞←'this will NOT be seen'
```

However, the multiline input mechanism allows for `:Trap` statements, so the following will run to completion:

```
#!/usr/local/bin/dyalogscript
⎕←'this will be seen'
:Trap 0
    ⎕←÷0
:EndTrap
⎕←'this will ALSO be seen'
```

Configuration Parameters

Configuration parameters may be specified in a Configuration file located in the same directory as the script, or may be specified on the first line of the script. The name of the configuration file is derived from the name of the script file by replacing its file extension (if any) by the extension `.dcfg`. Configuration parameters specified in the Windows Registry or by environment variables are not honoured in Dyalog Shell Scripts.

Example (first line of script)

```
#!/usr/local/bin/dyalogscript MAXWS=3GB
⎕←⎕WA
⎕←2 ⎕NQ '#' 'GetEnvironment' ('MAXWS' 'WSPATH')
```

Example (configuration file)

```
{ settings: {
        /* Maximum workspace size */
        MAXWS: "256M",
        /* wspath */
        WSPATH: ["c:/tmp","f:/devt/tmp"]
}}
```

Note that the interpreter reads both of these locations, the command line in the script file overrides any setting in the .dcfg file.

Debugging

It is not currently possible to use Ride to debug APL shell scripts. However there is an I-beam function, which can be used to provide some simple debugging/diagnostic information.

1.18 Creating Executables and COM Servers

Dyalog APL provides the facility to package an APL workspace as a Windows executable (EXE), an OLE Server (in-process or out-of-process), an ActiveX Control or a .NET Assembly. This may be done by selecting *Export ...* from the *File* menu of the APL Session window which brings up the *Create bound file* dialog box as illustrated later in this section.

The *Create bound file* dialog box offers selective options according to the type of file you are making. The system detects which of these types is most appropriate from the objects in your workspace. For example, if your workspace contains an ActiveXControl namespace, it will automatically select the *ActiveX Control* option.

If you are creating an executable (EXE) 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*.

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 (`dyalog150.dll` or `dyalog150rt.dll`), in which case the DLL should be installed in the same directory (as the EXE) or in the Windows System directory.

Various Dyalog-supplied files are required (such as the runtime DLL for creating a bound runtime executable); all such files are assumed to reside in the **Dyalog** directory, as specified by default in the registry. The location of this directory is most easily reported by calling

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

The creation of both in-process and out-of-process COM servers produces a .TLB (Type Library) file. This file is created in the same directory as the workspace - so write access must be allowed to this directory. In the case of an in-process server, the content of this file is then embedded into the DLL, and the file is deleted. For an out-of-process server the file persists and may be needed at runtime. This requirement means that even if you do not `)SAVE` the workspace, you should set the workspace name so that `)SAVE` would work - that is the directory where the workspace would be saved has write access.

In addition, a temporary copy of your workspace is created, the location of which is determined by the Windows function `GetTempPath()`.

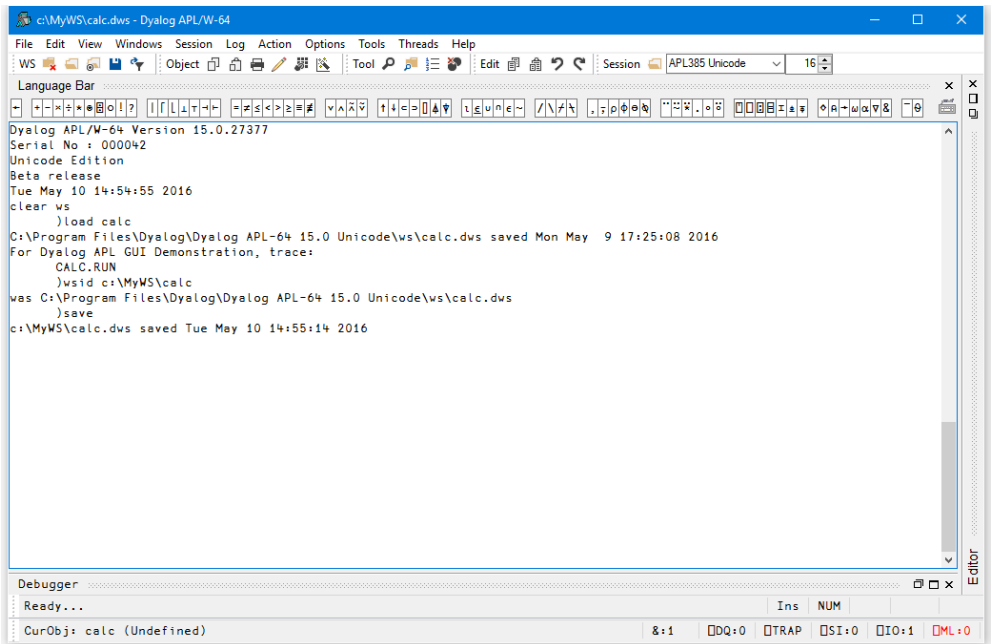
All registration information is written to HKEY_LOCAL_MACHINE in the registry which will require enhanced permissions (aka "run as administrator") for the Dyalog interpreter. Later versions of the interpreter may provide an option to write to HKEY_CURRENT_USER.

The *Create bound file* dialog box contains the following fields. These will only be present if applicable to the type of bound file you are making.

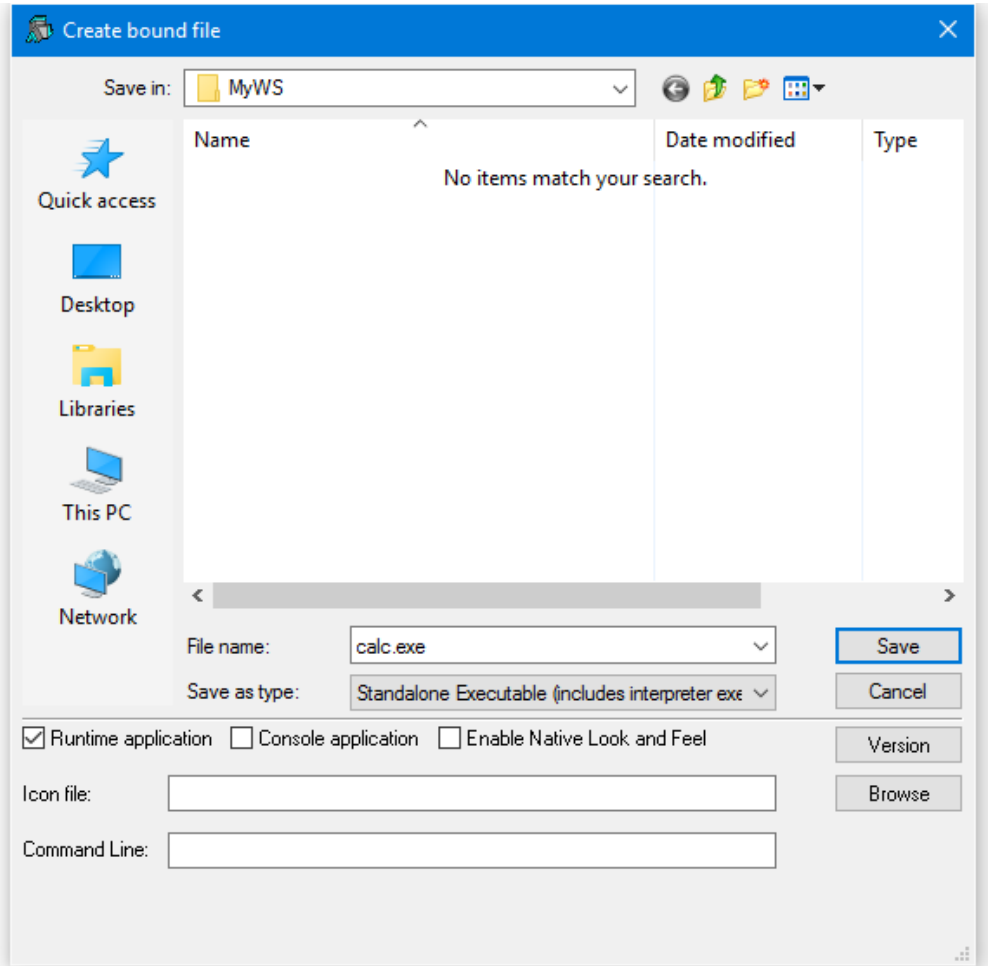
Item	Description
File name	Allows you to choose the name for your bound file. The name defaults to the name of your workspace with the appropriate extension.
Save as type	Allows you to choose the type of file you wish to create
Runtime application	If this is checked, your application file will be bound with the Run-Time DLL. If not, it will be bound with the Development DLL. The latter should normally only be used to permit debugging.
Console application	Check this box if you want your executable to run as a console application. This is appropriate only if the application has no graphical user interface.
Enable Native Look and Feel	If checked, *Native Look and Feel* will be enabled for your bound file; otherwise it will be disabled.
Icon file	Allows you to associate an icon with your executable. Type in the pathname, or use the *Browse* button to navigate to an icon file.
Command line	For an out-of-process COM Server, this allows you to specify the command line for the process. For a bound executable, this allows you to specify command-line parameters for the corresponding Dyalog APL DLL.

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.

In this example, the supplied workspace `calc.dws` is first saved to a directory to which the user has write access and, just to make certain, the Dyalog program is run as Administrator.



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



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.

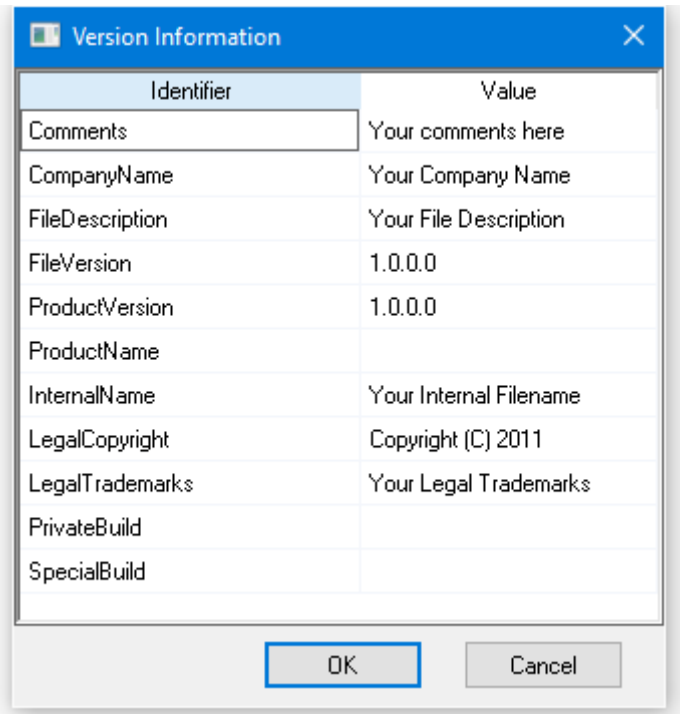
As this is a GUI application, the *Console application* checkbox is left unset.

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.

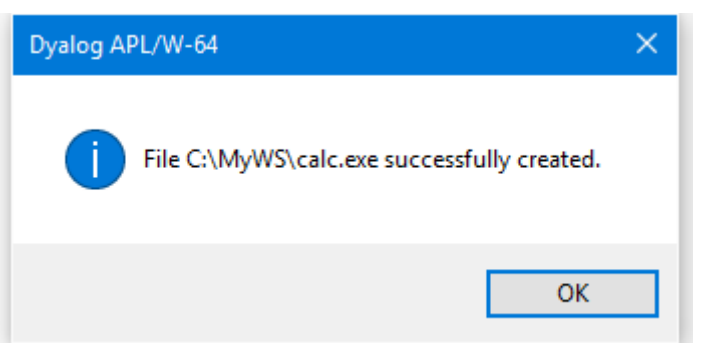
The *Command Line* box allows you to enter parameters and values that are to be passed to your executable when it is invoked.

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.



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



1.19 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.

For a list of the distributable components and their corresponding file names, for the different versions of Dyalog, see *Section 1.2*. These components are referred to in hereafter 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.

The various types of run-time applications and components are as follows:

1. Workspace or source code run-time
2. Stand-alone run-time
3. Bound run-time
4. Out-of-Process COM Server
5. In-Process COM Server
6. ActiveX Control
7. Microsoft .NET Assembly

All but the first of these are made using the *Export* dialog box accessed from the *File/ Export* menu item of the Session window.

Configuration Parameters

Configuration parameters for these run-time applications, both for the Dyalog engine and for your own application settings, may be specified in a number of ways. See *Section 1.9.1*.

Nevertheless, it is strongly recommended that you use Configuration files. In this section we will discuss only Application Configuration files, although User Configuration files may be used as well.

Workspace or source code based run-time

A workspace or source code based run-time application consists of the Dyalog APL Run-Time Program (Run-Time EXE), a separate workspace or text file containing APL source

code, and an optional configuration file. To distribute your application, you need to supply and install:

1. your workspace or source code
2. the Run-Time EXE
3. a configuration file (optional)
4. whatever additional files that may be required by your application
5. a command-line to start the application

The command-line for your application invokes the Run-Time EXE and directly or indirectly specifies the name of the workspace or source code file and the optional configuration file. You will need to associate your own icon with your application during its installation.

The name of the workspace or source code file may be specified by the **Load** parameter on the command line. If the application uses a workspace, the name of the workspace may instead be supplied as the last item on the command-line.

The name of the configuration file may be specified on the application command-line, using the **ConfigFile** parameter. Alternatively, the name of the configuration file is derived from the name of the workspace or source code file.

The action to start the application when a workspace or source code file is loaded is specified by the **LX** parameter or, for a workspace, by its latent expression (`⎕LX`).

In the command-line examples that follow, the name of the Run-Time EXE has been shortened to `dyalogrt.exe` for brevity.

Using a workspace

```
dyalogrt.exe myapp.dws
```

The application starts by running `⎕LX` in `myapp.dws`. If a configuration file named `myapp.dcfg` in the same directory, it is loaded and applied.

Using a source code file

```
dyalogrt.exe Load=myfn.aplf
```

The application loads the file named `myfn.aplf` which contains the source code for a function, and executes the expression `(myfn 0⍴c'')` (see *Section 1.9.81*). If a configuration file named `myfn.dcfg` in the same directory, it is loaded and applied.

If your application uses any component of the Microsoft .NET Framework, you must distribute the Bridge DLL and DyalogNet DLLs. These DLLs must be placed in the same directory as your EXE.

Stand-alone and Bound run-times

A stand-alone run-time is a single .EXE that contains a workspace and a copy of the Run-Time version of the Dyalog APL interpreter. It is the simplest type of run-time to install because it has the fewest number of dependencies.

A bound run-time is a workspace packaged as a .EXE that relies upon and requires the separate installation of the Run-Time DLL. Compared with the stand-alone executable option, bound run-times may save disk space and memory if your customer installs and runs several different Dyalog applications.

Both these run-times are created using the *File/Export* menu item on the Session window.

To distribute your application, you need to supply and install:

1. your stand-alone or bound .EXE
2. the Run-Time DLL (bound .EXE only)
3. a configuration file (optional)
4. whatever additional files that may be required by your application
5. a command-line to start the application

When you build your .EXE using the Export dialog, you may specify the name(s) of the configuration file(s) using the **ConfigFile** and/or **UserConfigFile** parameters in the field labelled *Command Line*.

An alternative is to specify these parameters in the command-line that you use to run your .EXE (note that this is not the same as the *Command Line* in the *Export* dialog box). If so, the Dyalog parameter(s) must be preceded by the **-apl** option.

If your application uses any component of the Microsoft .NET Framework, you must distribute the Bridge DLL and DyalogNet DLLs. These DLLs must be placed in the same directory as your EXE.

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 must be specified in the field labelled *Command Line* in the *Export* dialog box. The field is initialised to invoke the Run-Time EXE with the name of your workspace in the same fashion as the workspace-based run-time discussed above. This command-line is recorded in the Windows Registry to be invoked when a client application requests it.

You may change the contents of the *Command Line* field to use a configuration file, in the same way as for a workspace-based runtime. The following example uses the Loan COM Server. See *Interfaces: The Loan Workspace*.

Example

```
dyalog.exe C:\Dyalog18.0\myloan.dws
```

The command-line above will, on invocation, cause Dyalog to load the `myloan.dws` workspace together with the configuration file `myloan.dcfg` if it exists in that directory.

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. a configuration file (optional)
5. 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.

As there is no command-line available, to specify a configuration file for an in-process COM server, it is necessary to define the **ConfigFile** parameter and/or the **UserConfigFile** parameter as an environment variable or in the registry.

To distribute your component, you need to supply and install

1. Your COM Server file (DLL)
2. the Run-Time DLL
3. a configuration file (optional) and the means to define **ConfigFile** and/or **UserConfigFile**
4. 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.

As there is no command-line available, to specify a configuration file for an in-process COM server, it is necessary to define the **ConfigFile** parameter and/or the **UserConfigFile** parameter as an environment variable or in the registry.

To distribute your component, you need to supply and install

1. Your ActiveX Control file (OCX)
2. the Run-Time DLL
3. a configuration file (optional) and the means to define **ConfigFile** and/or **UserConfigFile**
4. 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.

If the option selected in the *Isolation Mode* field of the *Export* dialog is either:

- Each assembly has its own workspace, or
- Each assembly attempts to use local bridge and interpreter libraries

you may enter configuration parameters or specify a Configuration file for your Dyalog assembly in the field labelled *Command Line*.

For the other isolation modes, this is not appropriate because only the command line from the first assembly loaded into the interpreter could be honoured, and the order in which assemblies are loaded is unpredictable. However, configuration files may be specified using the **ConfigFile** parameter and/or the **UserConfigFile** parameter specified as an environment variable or in the registry.

For more information, see *.NET Framework Interface: Isolation Mode*.

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. a configuration file (optional) and, depending upon the isolation mode, the means to define **ConfigFile** and/or **UserConfigFile**
6. whatever additional files that may be required by your .NET Assembly.

All the DLLs and subsidiary files must be installed in the same directory as the .NET Assembly.

1.20 Run-Time Applications Additional Considerations

Accessing your Application using Ride

If you wish to access your run-time application remotely using Ride, you must put a copy of the appropriate Conga DLLs (see *Section 1.2*) in the same directory as your .EXE or workspace.

Additional Files for SQAPL

If your application uses the *SQAPL/EL ODBC* interface, you must distribute and install four additional components.

- SQAPL INI
- SQAPL ERR
- SQAPL DLL
- APLUNICD INI

For the names of the files corresponding to these components, see *Section 1.2*.

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

Miscellaneous Other Files

DyaRes DLL

If your run-time application uses any of the bitmaps or other GUI resources that are built into the Dyalog Session, you must include the DyaRes DLL with your application.

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.

Universal C Runtime DLLs

Under Windows, many of the Dyalog APL run-time components (.EXE and .DLL) are linked dynamically with the Microsoft Universal C Runtime library (the UCRT) which is supplied and installed as part of the normal Dyalog development installation.

At execution time it is important that the Dyalog runtime components bind with a version of the UCRT that is compatible with (that is, the same as or newer than) the one with which they were built.

Windows 10

If the end-user of the Dyalog application is known to be running Windows 10, the Dyalog application will pick up the system-wide UCRT which is part of Windows 10. There is therefore no need to include the UCRT with a Dyalog run-time application.

Other Versions of Windows

The UCRT is not supplied with versions of Windows prior to Windows 10. On these platforms, it is therefore necessary to install the UCRT as part of the installation of the Dyalog run-time application. There are two ways to achieve this which are referred to herein as the VCRedist installation and App-local installation. Dyalog recommends the former.

VCRedist Installation (Recommended)

The VCRedist package, which includes the UCRT, is supplied with the Dyalog development package.

Simply copy the `vc_redistx86.exe` (32-bit version) or `vc_redistx64.exe` (64-bit version) program from the Dyalog development package into your own installation package and execute it as part of the installation of your Dyalog run-time application. This installs the UCRT into a shared Windows location; in effect the UCRT becomes part of the Windows system. The installation therefore requires Administrator privileges.

App-local Installation

An alternative is to install the UCRT components into the same directory as your Dyalog run-time application. There are two ways to obtain these files.

Either

Install the Dyalog development package (ideally onto a separate system just for this purpose) without administrator rights. This will perform an App-local installation of Dyalog itself. Then copy the UCRT files into your installation package. These files are:

- those beginning with `api-ms*`
- `ucrtbase.dll`
- `vcruntime140.dll`

Or

Download and install the Windows 10 SDK from: https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk, and follow the instructions in the link below. https://blogs.msdn.microsoft.com/vcblog/2015/03/03/introducing-the-universal-crt

Finally, modify your installer to add these files to the same folder as your Dyalog run-time application.

1.21 COM Objects and the Dyalog APL DLL

Introduction

Each different implementation of Dyalog contains two versions of the Dyalog APL Dynamic Link Library, a development version (Development DLL) and a run-time version (Run-Time DLL). For further details, see *Section 1.2*.

In the remainder of this section, the term *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>
```

or on 64-bit Windows:

```
HKLM\Software\Wow6432Node\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 instanced. 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.

1.22 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 `ws\aplservice.dws` is included in the APL release. Its start-up function is as follows:

```
     ⎕LX←'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 0∈⍴2 ⎕NQ'.' 'GetEnvironment' 'RunAsService'
[6]        Describe
[7]        :Return
[8]    :EndIf
[9]    ⍝ Define SCM constants
[10]   HashDefine
[11]   ⍝ Set up callback to handle SCM notifications
[12]   '.'⎕WS'Event' 'ServiceNotification' 'ServiceHandler'
[13]   ⍝ Global variable defines current state of the service
[14]   ServiceState←SERVICE_RUNNING
[15]   ⍝ Global variable defines last SCM notification to the
         service
[16]   ServiceControl←0
[17]   ⍝ 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]   ⍝ Callback to handle notifications from the SCM
[2]
[3]   ⍝ Note that the interpreter has already responded
[4]   ⍝ automatically to the SCM with the corresponding
[5]   ⍝ "_PENDING" message prior to this callback being reached
[6]
[7]   ⍝ This callback uses the SetServiceState Method to confirm
[8]   ⍝ to the SCM that the requested state has been reached
[9]
[10]   r←0  ⍝  so returns a 0 result (the event has been handled,
[11]        ⍝ no further action required)
[12]
[13]  ⍝ It stores the desired state in global ServiceState to
[14]  ⍝ notify the application code which must take appropriate
[15]  ⍝ action. In particular, it must respond to a "STOP or
[16]  ⍝ "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),' ',⍵,⎕UCS 13 10)⎕NAPPEND ⍺}
[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 ⋄
              nid log'ServiceControl=',⍕ServiceControl ⋄ :EndIf
[17]      :If ServiceState=SERVICE_RUNNING
[18]          nid log'Running'
[19]      :ElseIf ServiceState=SERVICE_PAUSED
[20]          ⍝  Pause application
[21]      :EndIf
[22]      ServiceControl←0 ⍝ Reset (we only want to log changes)
[23]      ⎕DL 10 ⍝ Just to prevent busy loop
[24]  :EndWhile
[25]  ⎕NUNTIE nid
[26]  ⎕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 Ride User Guide. Note however that the Conga DLLs/shared libraries must be available - usually they should reside in the same directory as the interpreter. In previous versions of Dyalog separate Ride DLLs/shared libraries were supplied; these have been subsumed into the Conga libraries in 16.0.

Event Logging

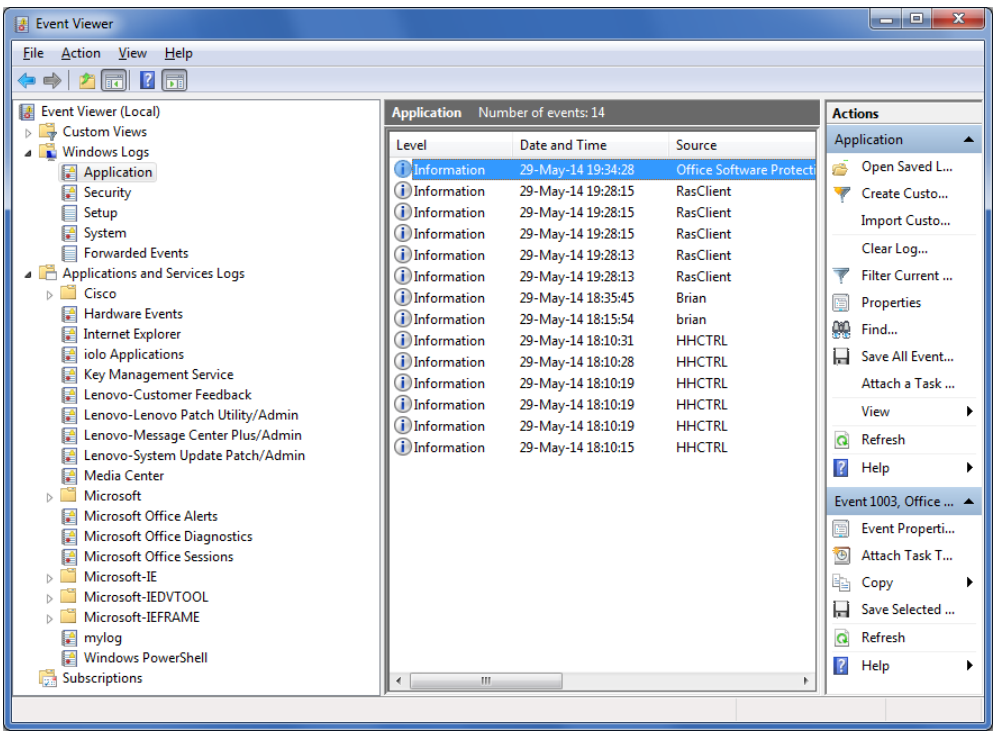
When a service is installed or removed, Dyalog APL records events in the Dyalog APL section of the *Applications and Services Logs* which can be viewed using the Windows system *Event Viewer*.

1.23 APLService Logging Events

The `aplservice` 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: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 desireYou 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

4. 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
5. 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
`sourcename`	character vector source name that does not already exist
`{logname}`	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 `logname` , 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						
`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	`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
`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	`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
`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

2 Configuring the IDE

2.1 Configuration Dialog

2.1.1 General Tab

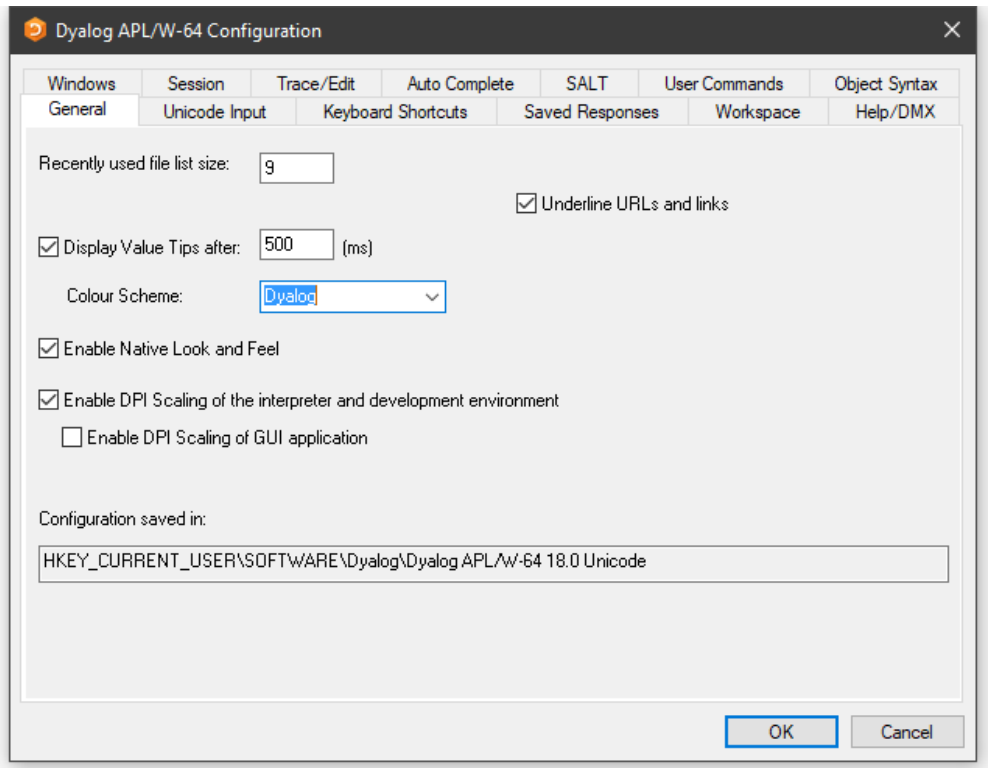


Table: Configuration dialog: General

Label	Parameter	Description
Recently used file list size	*File_Stack_Size*	Specifies the number of the most recently used workspaces displayed in the File menu.
Underline URLs and links	*URLHighlight*	Specifies whether or not URLs and links are highlighted in Session and Edit windows.
Display Value Tips	*Enabled*	Specifies whether or not Value Tips are enabled.
Display Value Tips after	*Delay*	Specifies the delay before APL displays a Value Tip.
Colour Scheme	*ColourScheme*	Specifies the colour scheme used to display Value Tips.
Enable DPI Scaling of the interpreter and development environment	*AutoDPI*	Enables or disables DPI scaling for the APL Session
Enable DPI scaling of GUI application	*Dyalog_Pixel_Type*	Determines whether Coord `'Pixel'` is treated as ScaledPixel or RealPixel.
Configuration saved in	*IniFile*	Specifies the full pathname of the registry folder used by APL

2.1.2 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*

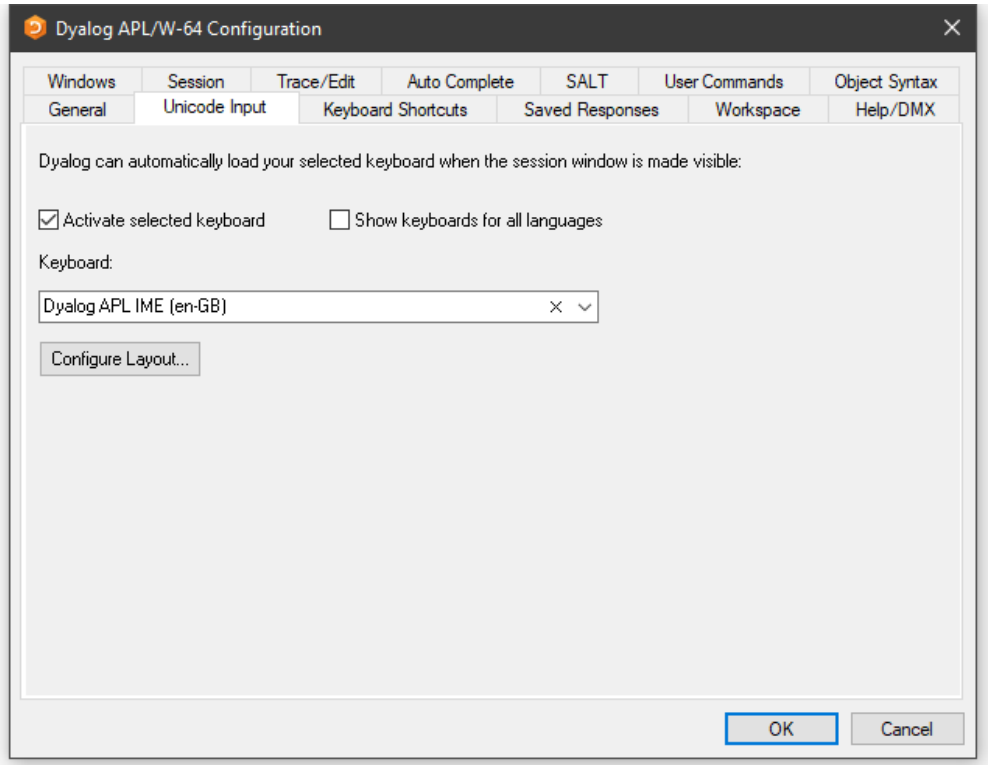


Table: Configuration dialog: Unicode Input

Label	Parameter	Description
Activate selected keyboard	*InitialKeyboardLayoutInUse*	If checked, the specified APL keyboard is activated on start-up.
Show keyboards for all Languages	*InitialKeyboardLayoutShowAll*	If checked, all installed keyboards are displayed. Otherwise, only Dyalog keyboards are shown
Keyboard	*InitialKeyboardLayout*	the APL keyboard to be selected.
Configure Layout		Displays thefollowng dialog box.

Input Method Editor Properties

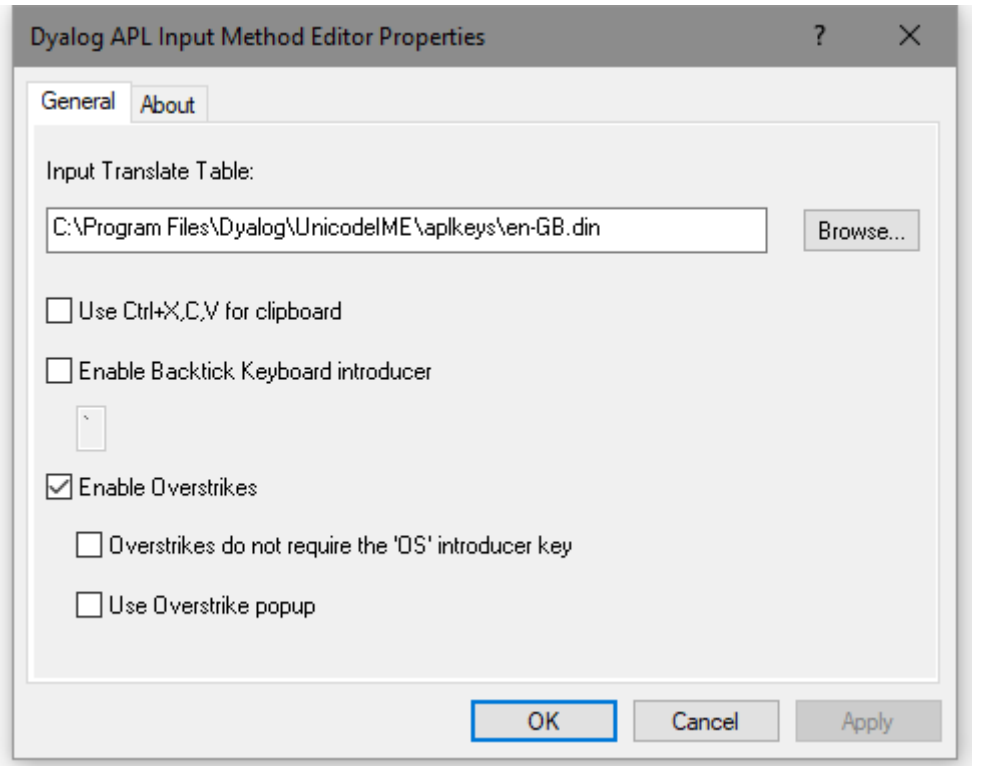


Table: Dyalog Input Method Editor Properties

Label	Parameter	Description
Use Ctrl-X,C,V for clipboard	*UseXCV*	specifies whether or not the commonly used keystrokes for copy, cut and paste are recognised as such.
Enable Backtick Keyboard introducer		
Enable Overstrikes	*ResolveOverstrikes*	1 = enable overstrikes. 0 = disable overstrikes
Overstrikes do not require the OS introducer key		1 = IME identifies overstrike operation automatically 0 = IME requires the key (default Ctrl+Bksp) to signal an overstrike operation
Use Overstrike popup	*OverstrikesPopup*	1 = enable the overstrike popup. 0 = disable the overstrike popup

2.1.3 Input Tab (Classic Edition Only)

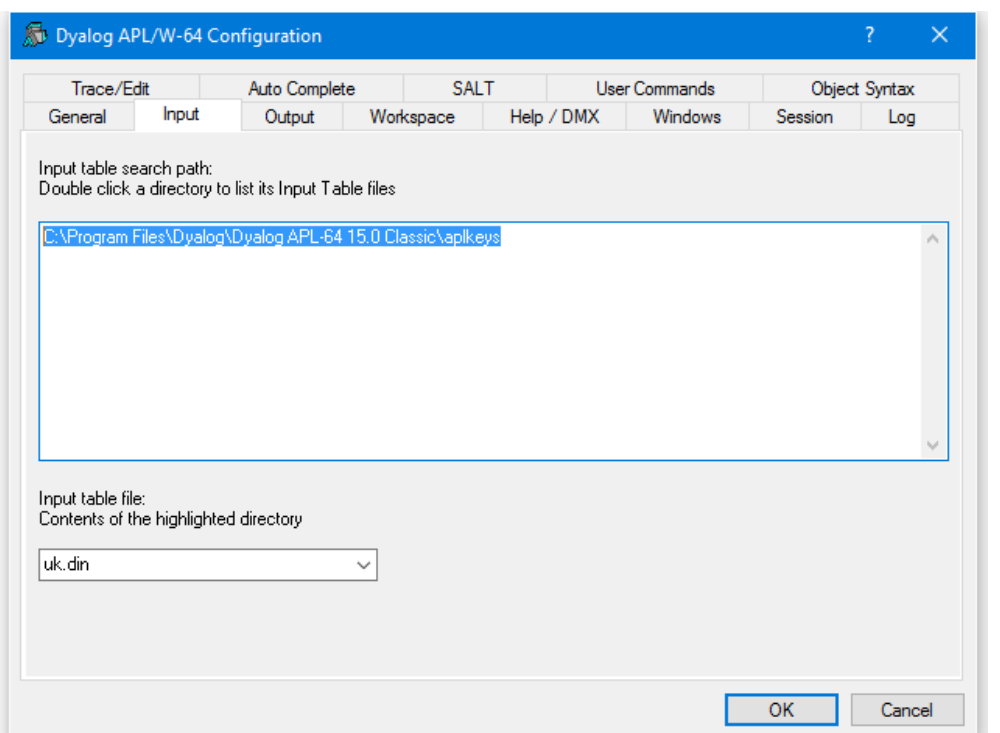


Table: Configuration dialog: Keyboard

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)

2.1.4 Output Tab (Classic Edition Only)

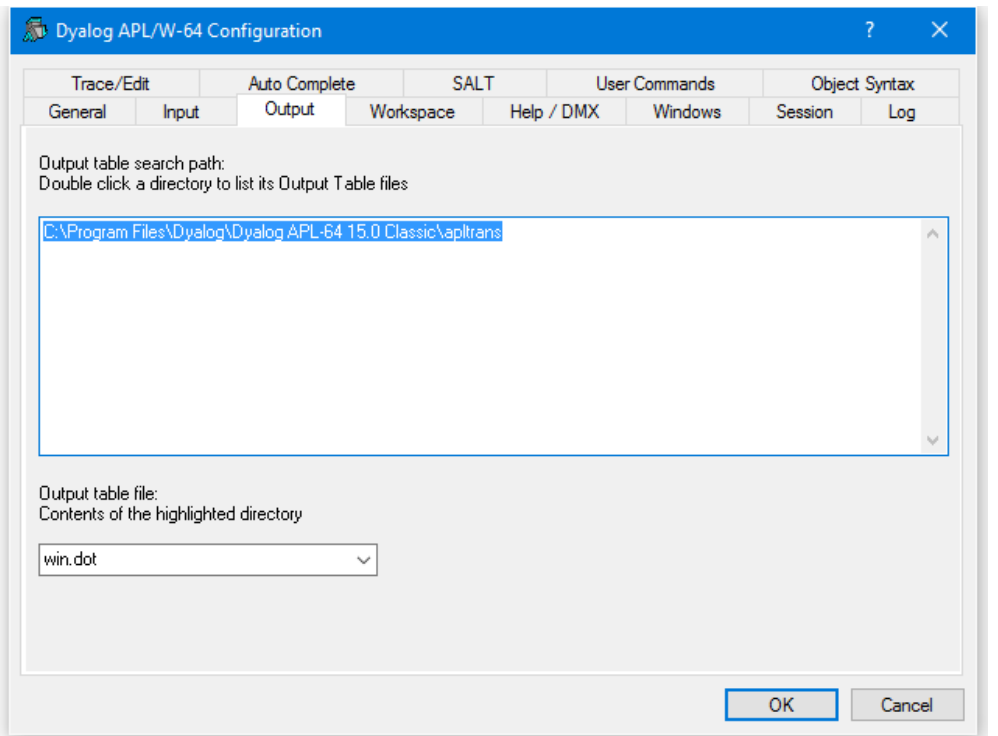
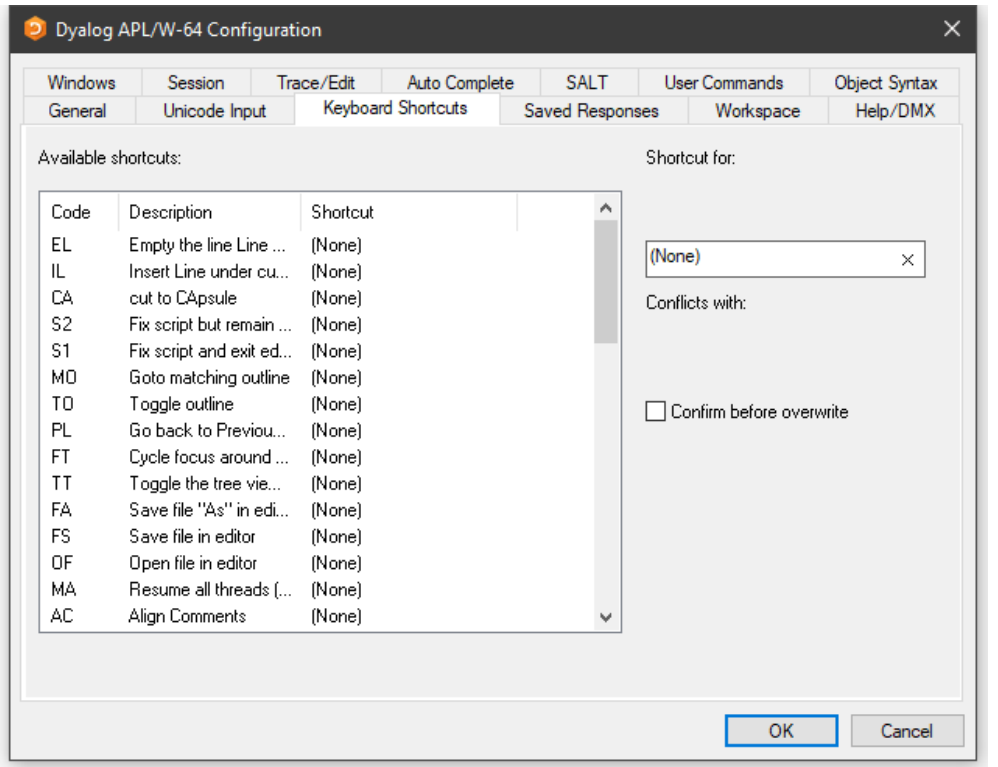


Table: Configuration dialog: Output

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)

2.1.5 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 (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.

2.1.6 Workspace Tab

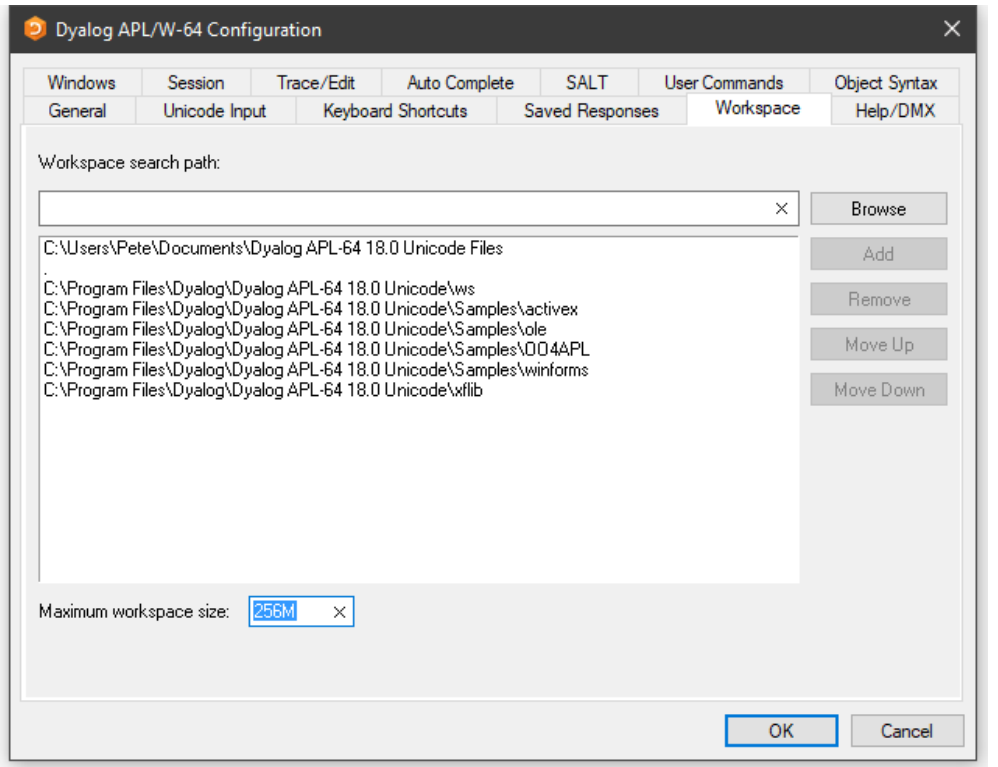


Table: Configuration dialog: Workspace

Label	Parameter	Description
Workspace search path	*WSPath*	A list of directories to be searched for the specified workspace when the user executes `)LOAD .`
Maximum workspace size	*MaxWS*	The maximum size of the workspace.

2.1.7 Help/DMX Tab

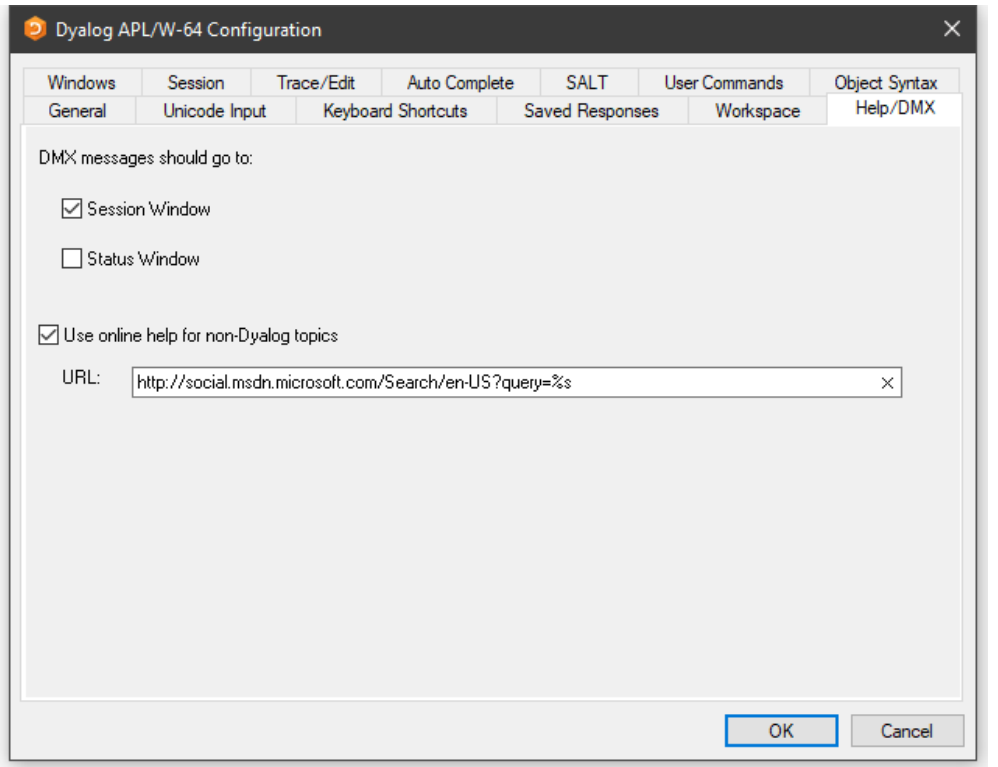


Table: Configuration dialog: Help/DMX

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	*UseExternalHelpURL*	If this option is checked, APL will look for help for external objects at Microsoft's documentation center, which is identified by the specified URL.
URL	*ExternalHelpURL*	The URL for the documentation centre.

2.1.8 Windows Tab

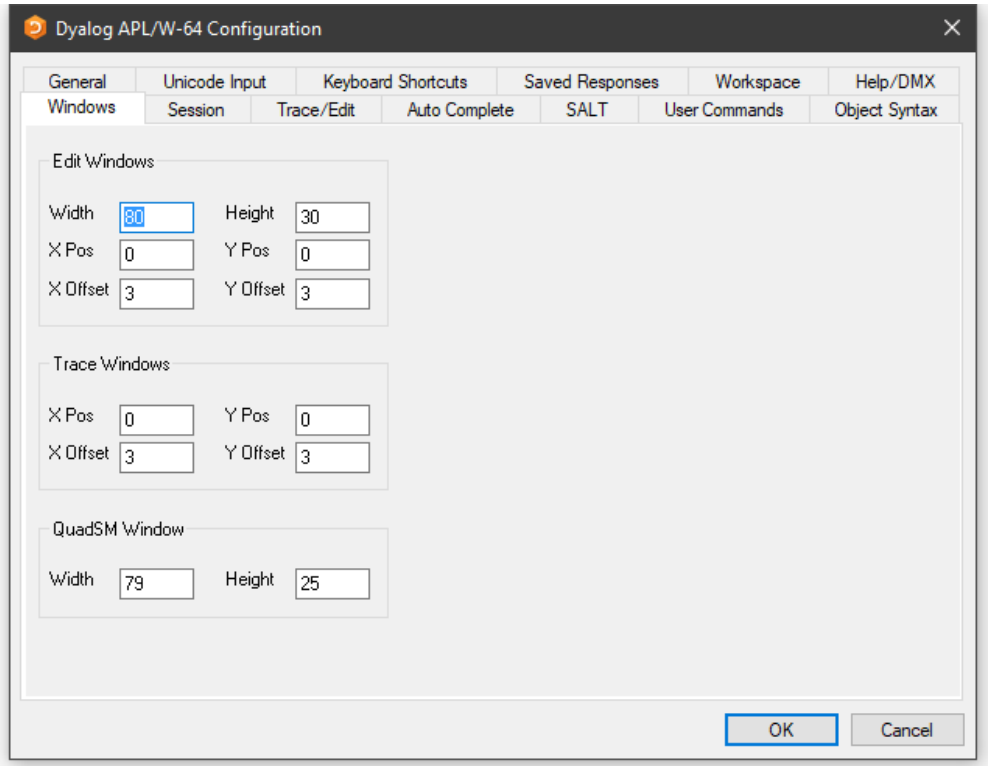


Table: Configuration dialog: Windows (Edit Windows)

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: 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: Configuration dialog: Windows (QuadSM Window)

Label	Parameter	Description
Width	*SM_Cols*	The width of the QSM and prefect windows.
Height	*SM_Rows*	The height of the QSM and prefect windows.

2.1.9 Session Tab

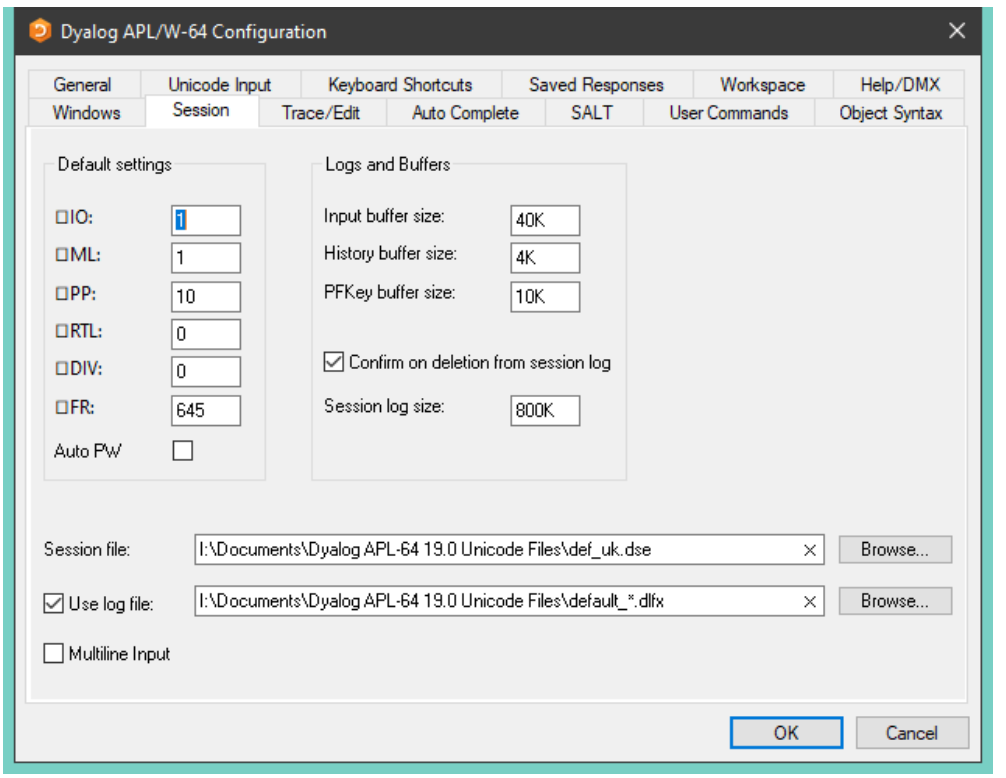


Table: Configuration dialog: Session

Label	Parameter	Description
□IO	*Default_IO*	The default value of □IO in a `clear ws`.
□ML	*Default_ML*	The default value of □ML in a `clear ws`
□PP	*Default_PP*	The default value of □PP in a `clear ws`.
□RTL	*Default_RTL*	The default value of □RTL in a `clear ws`.
□DIV	*Default_DIV*	The default value of □DIV in a `clear ws`.
□WX	*Default_WX*	The default value of □WX in a `clear ws`.
Auto PW	*Auto_PW*	If checked, the value of □PW is dynamic and depends on the width of the Session Window.
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)
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.
Session file	*Session_File*	The name of the Session file in which the definition of your session (□SE) is stored.
Use log file	*Log_File_InUse*	Specifies whether or not the Session log is saved in a session log file

Label	Parameter	Description
Use log file	*Log_File*	The full pathname of the Session log file
Multiline Input	*Dyalog_LineEditor_Mode*	Specifies whether or not multi-line input is enabled in the Session.

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).

2.1.10 Trace/Edit Tab

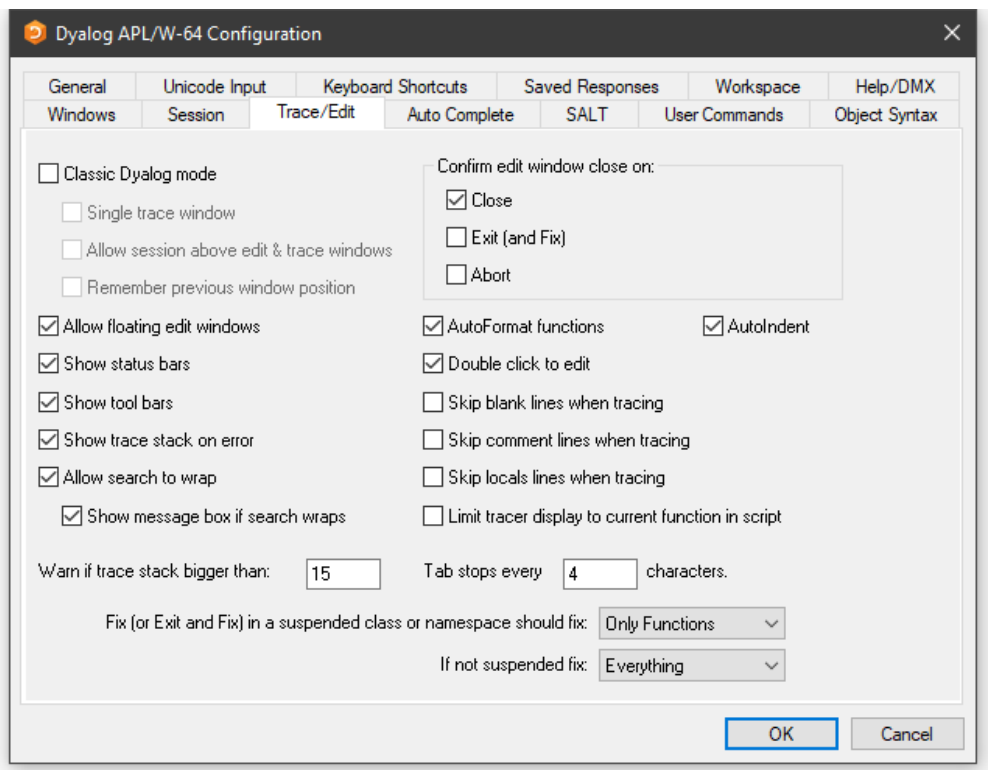


Table: Configuration dialog: Trace/Edit

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 re-docked 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
Allow search to wrap	*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.
Show message box if text wraps	*WrapSearchMsgBox*	Specifies whether or not a message box is displayed to inform the user when the search wraps.
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 *Fix* or *Exit*
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	*SkipLines*	If enabled, this causes the Tracer to automatically skip blank lines.
Skip comment lines when tracing	*SkipLines*	If enabled, this causes the Tracer to automatically skip comment lines.
Skip locals lines when tracing	*SkipLines*	If enabled, this causes the Tracer to automatically skip locals lines.
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 …	*InitFullScriptSusp*	See Fixing Scripts below
If not …	*InitFullScriptNormal*	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.

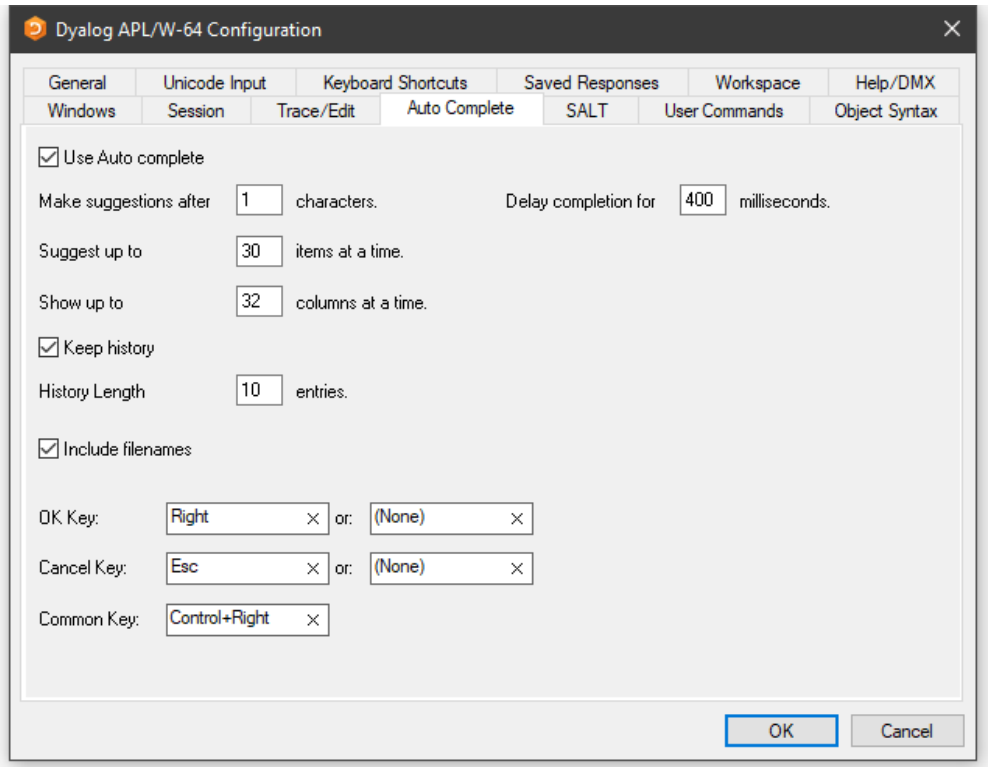
These two actions always appear in the Editor File menu, but you can specify which is associated with the (Esc) key by selecting the appropriate option in the drop-downs labelled:

- 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 Editor (The File Menu, editing a script)) will change according to which behaviour applies. Note that if you specify a keystroke for in the *Keyboard Shortcuts* tab, this will be associated with the unselected action.

2.1.11 Auto Complete Tab



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: 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
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 *common prefix* .

2.1.12 SALT Tab

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

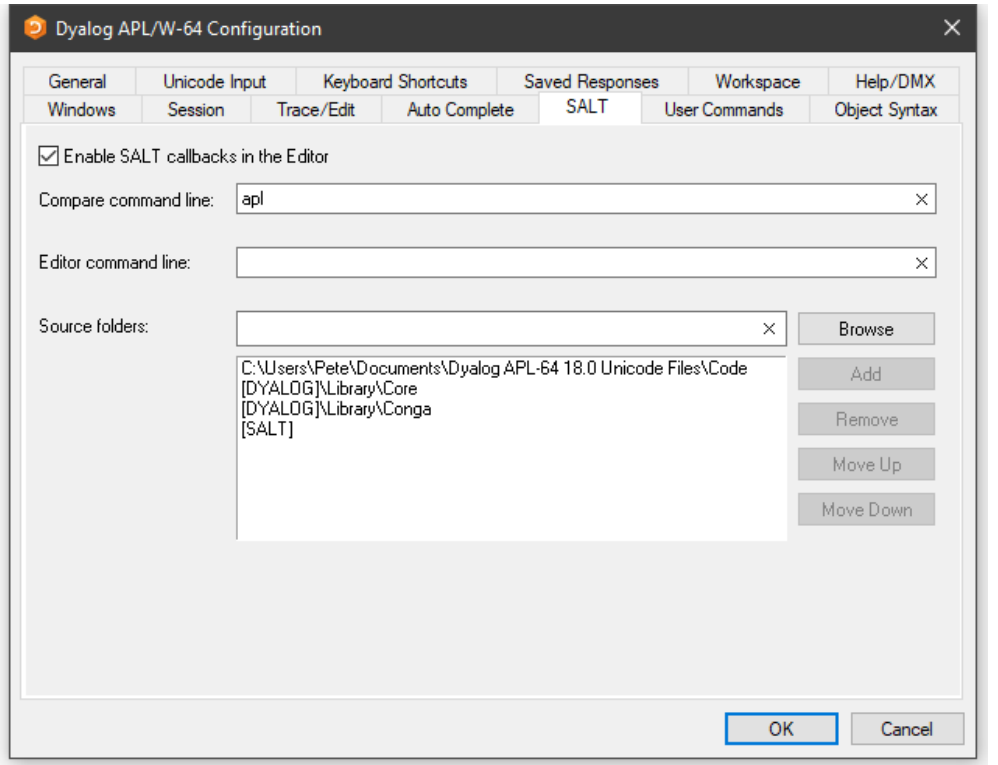
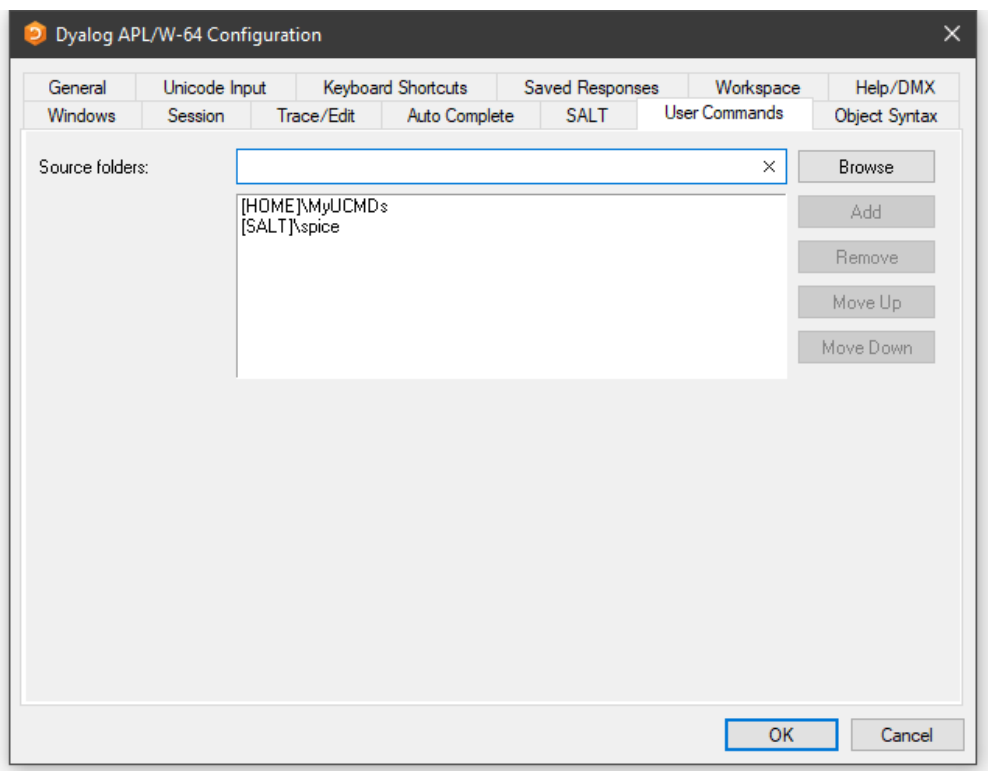


Table: Configuration dialog: SALT

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 command line:	Editor	Name of the program to be used to edit script files (default "Notepad").
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

2.1.13 User Commands Tab

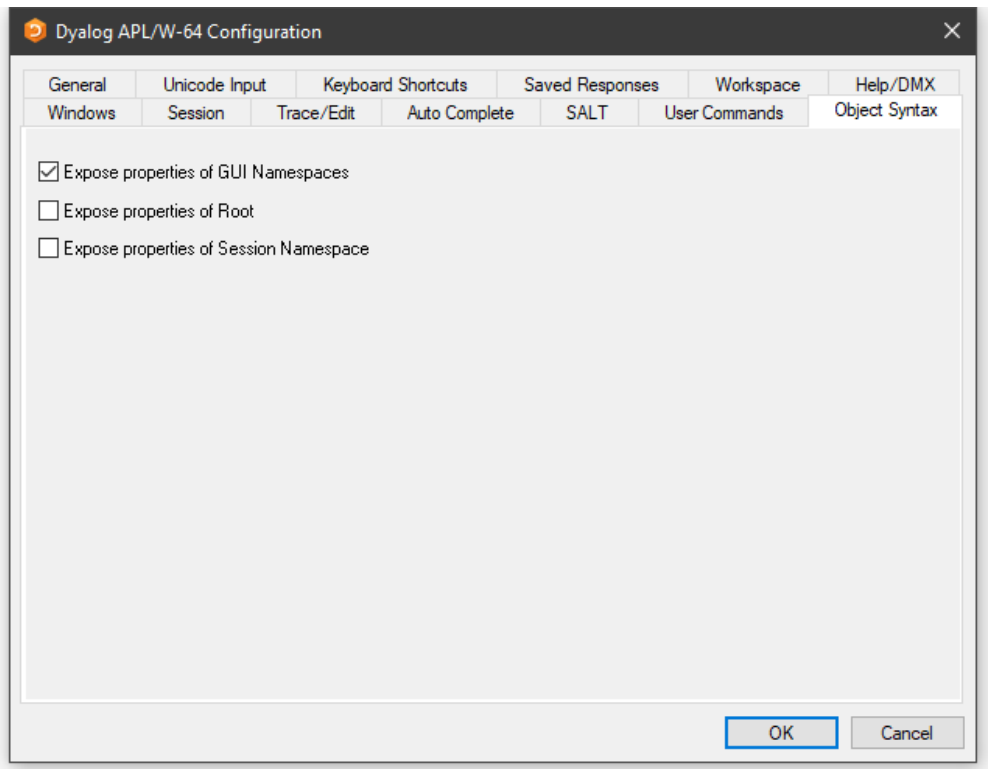


This page is used to specify and organise a list of folders that contain User-Command files. When you issue a User Command, these folders will be searched for the source of the command in the order in which they appear in this list.

Table: 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.

2.1.14 Object Syntax Tab

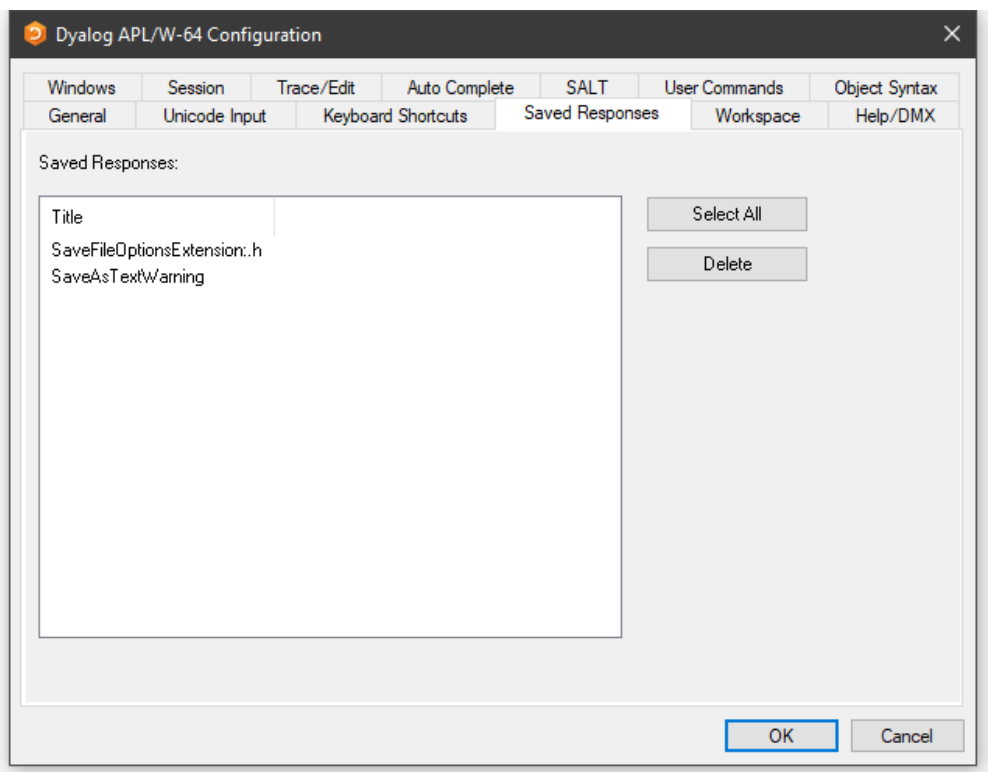


The *Object Syntax* tab of the *Configuration* dialog is used to set your *default preferences* for Object Syntax. Use *Options/Object Syntax* to change the settings for the current workspace.

Table: Configuration dialog: Object Syntax

Label	Parameter	Description
Expose properties of GUI Namespaces	*Default_WX*	Specifies the value of ⎕WX in a clear workspace.
Expose properties of Root	*PropertyExposeRoot*	Specifies whether or not the names of properties, methods and events of the Root object are exposed.
Expose properties of Session Namespace	*PropertyExposeSE*	Specifies whether or not the names of properties, methods and events of the Session object are exposed.

2.1.15 Saved ResponsesTab

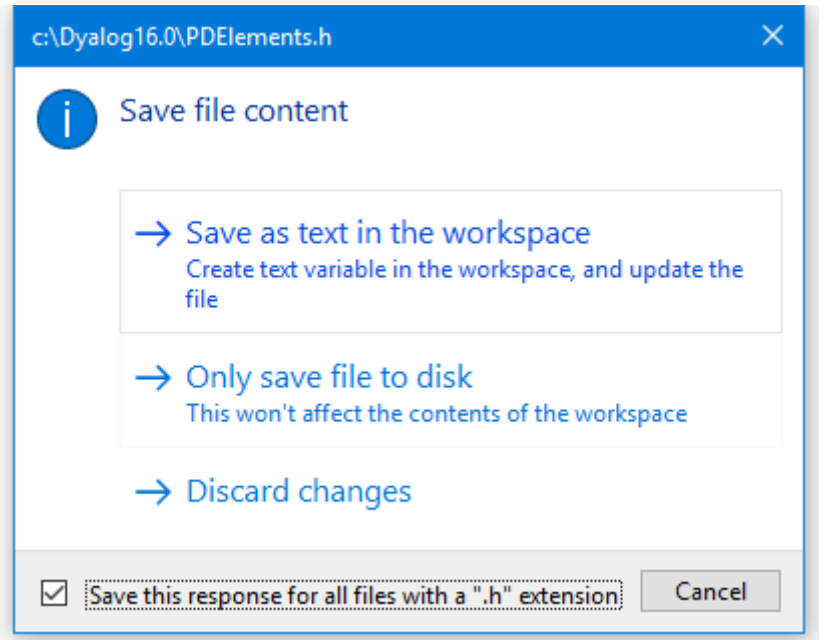


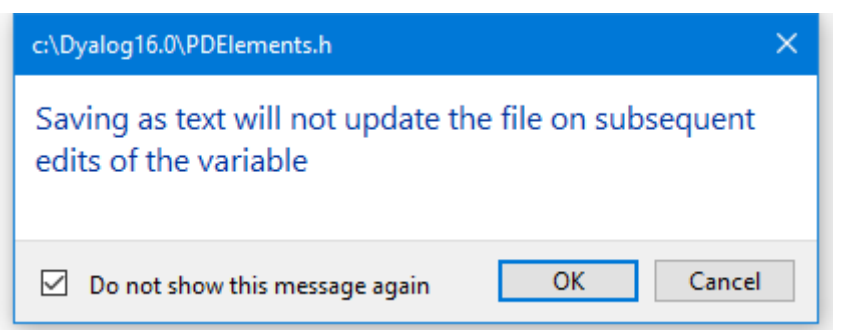
The *Saved Responses* tab of the *Configuration* dialog is used to remove preferences that the user has previously established.

In the example illustrated above, the user has at some point chosen to save a text file with a `.h` extension as text in the workspace and, by checking the option *Save this response for all files with a ".h" extension*, saved this as a preference for all such text

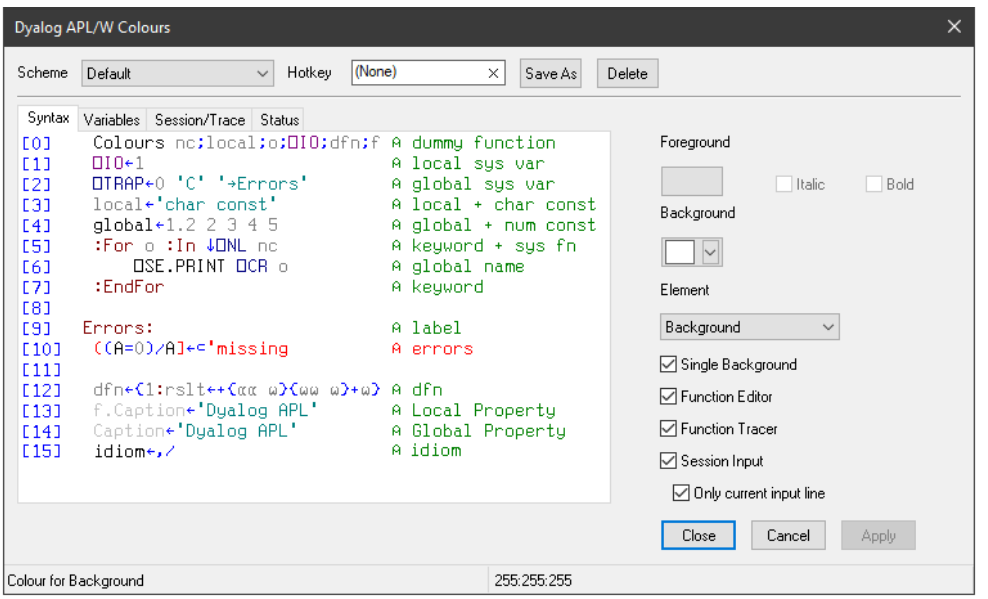
files. Similarly, the user has checked the option *Do not show this message again* when responding to the warning dialog *Saving as text will ....*

If the user wishes to reverse these decisions, even temporarily, it is necessary to select the corresponding option /preference name(s) and click *Delete*. The names are intended to be self-explanatory and are not listed here.





2.2 Colour Selection Dialog



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

- Syntax colouring in the Session window
- Variables
- Edit and Trace windows
- Status window

To choose for which of which of these items you want to define colours by selecting the appropriate tab.

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

Colour 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.

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.

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.

Table: Colour Selection

Label	Description
Schemes	Choose the scheme you want to edit using this dropdown box.
HotKey	To associate a hot key with the currently selected colour scheme, click here, and then make the desired keystroke. To disassociate a hot key, use .
Save As	Click to overwrite the current colour scheme or save as a new one.
Delete	Click to delete the currently selected colour scheme.
Foreground	Choose the foreground colour from the colour picker
Italic	Enable/disable italic foreground
Bold	Enable/disable bold foreground
Single Background	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.

2.3 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

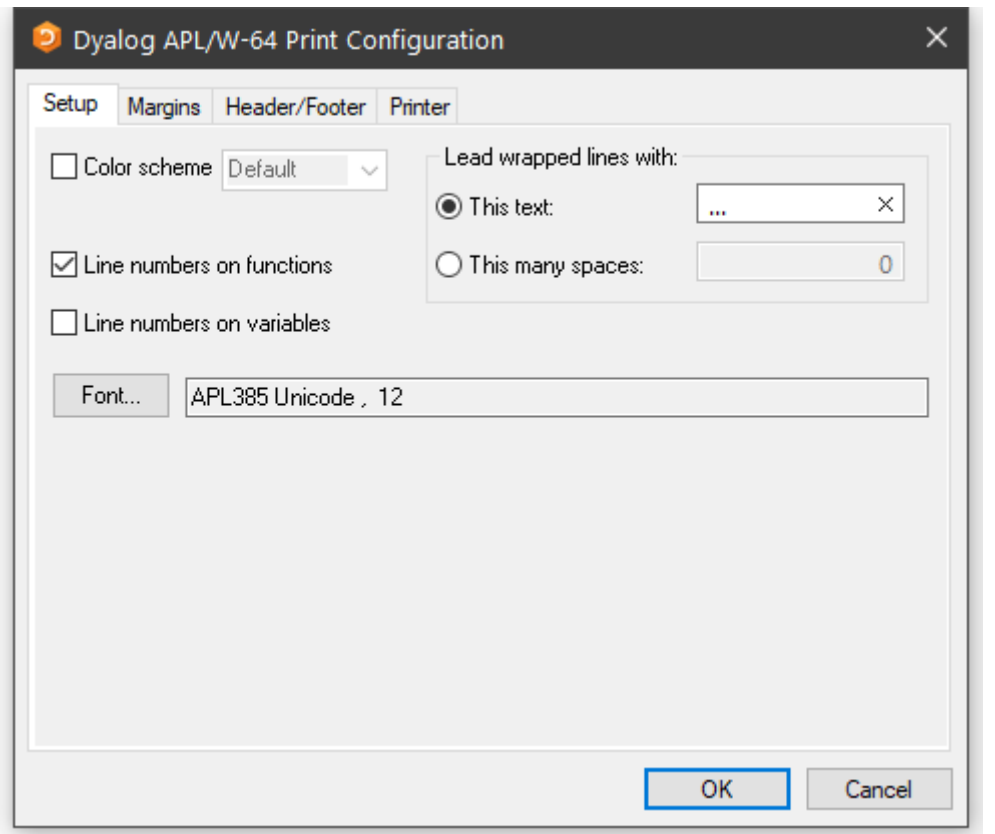


Table: Print Configuration dialog: Setup

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 `⎕IO` .
Font	Font	Click to select the font to be used for printing. Note that only fixed-pitch fonts are supported.

Margins Tab

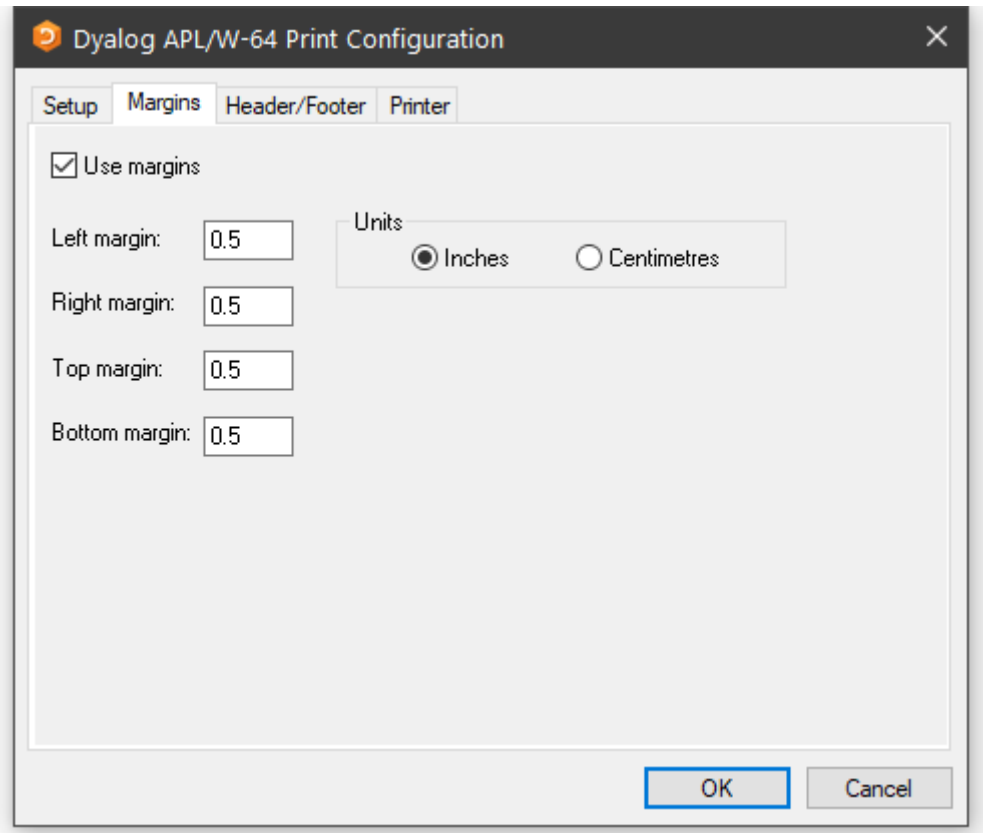


Table: Print Configuration dialog: Margins

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

Header/Footer Tab

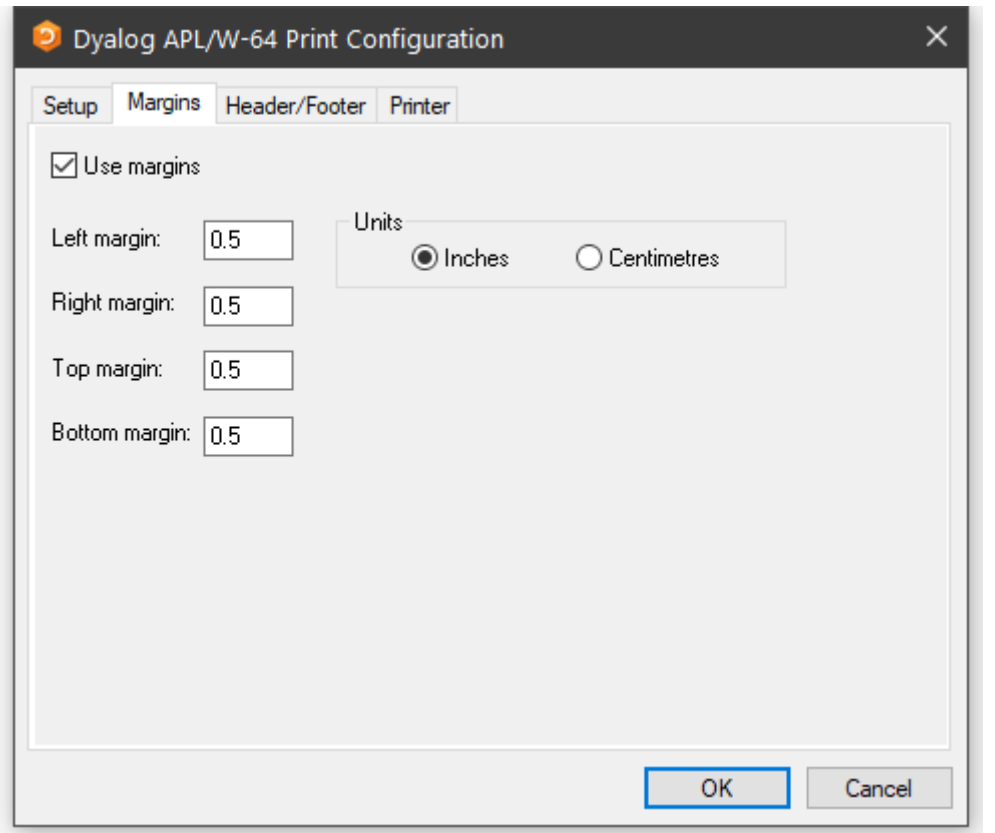


Table: Print Configuration dialog: Header/Footer

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, NA 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, for example, {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

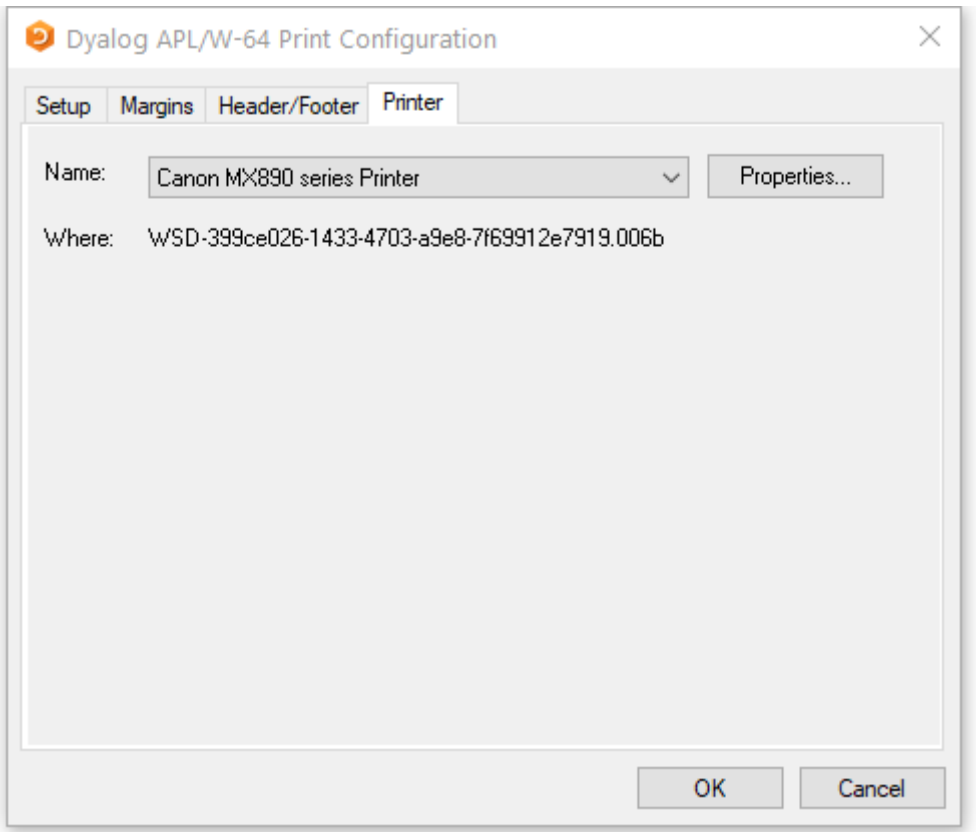


Table: Print Configuration dialog: Print

Label	Parameter	Description
Name	PrinterField	The name of the printer to be used for printing from Dyalog.
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.