



▶ Graphical User Interface

▶ Namespaces Technology

Namespaces

Dyalog namespaces allow you to partition your workspace into self-contained subsystems.

Namespaces also provide the foundation for Object Oriented (OO) technology in Dyalog, and act as the basis for interfaces to other Windows OO technologies including .NET, COM and the Windows Graphical Interface (GUI).

Overview

Namespace technology is a unique Dyalog feature and one of the most significant advantages of the product.

A namespace is an object which may contain functions, operators, variables *and other objects*. Namespaces are therefore inherently hierarchical.

Namespaces may or may not have a GUI component. In other words, all GUI objects are namespaces but namespaces are not necessarily GUI objects.

A function in a namespace executes within that namespace; and normally only sees other functions and variables in the same namespace. Functions and variables with the same name in the parent workspace or in other namespaces are effectively shadowed.

Benefits

- Namespaces allow you to structure an application into separate **self-contained** units that are protected from one another. This allows you to break down a complex application into smaller sub-systems which improves robustness and ease of maintenance.
- Namespaces provide the mechanism for **object-oriented** features such as encapsulation. You can implement a self-contained object (with or without a GUI component) as a namespace. The object can subsequently be incorporated into any Dyalog APL application without requiring any knowledge of its inner workings and without any possibility of interference with the application's own code and data.

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



- ▶ Graphical User Interface
- ▶ Namespaces Technology

Manipulating Namespace Objects

Namespaces can be created using the system command `)NS` or the equivalent system function `[NS`.

You switch from one namespace to another using `)CS` or, dynamically and under program control, with the system function `[CS`. Whilst you are in a given namespace, any new objects you create become part of that namespace and any names to which you refer are taken to be within that namespace.

In addition, the `:With` control structure provides an elegant and concise means to switch in and out of namespaces in a defined function.

Namespaces are destroyed using `)ERASE` or `[EX` or as a result of localization.

Using Namespace Objects

Like a file in DOS or UNIX, an object may be referenced by its name or by its path name depending upon where you are at the time. A path name specifies the location of an object as well as its own name. For example, to refer to the function `PRINT` in the `WSDOC` namespace you may do so as shown in the following table:

Reference	From
<code>PRINT</code>	within the <code>WSDOC</code> namespace
<code>WSDOC.PRINT</code>	the root namespace (the parent of <code>WSDOC</code>)
<code>#.WSDOC.PRINT</code>	any other namespace

If you reference a function name that is not defined in the current namespace, the system searches for the function in a list of namespaces specified by the `[PATH` system variable. This allows you to store utilities in namespaces but avoids the need to use path names to call them. You need not even know where they are. To avoid accidental references to sub-utilities, function names may be exported or hidden from the `[PATH` mechanism using the system function `[EXPORT`.

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



▶ Graphical User Interface

▶ Namespaces Technology

GUI Namespaces

In Dyalog, all GUI objects are namespaces. This means that you can store the call-back functions, utilities and data variables that are used by a GUI object within that object. This feature, known in object oriented programming as encapsulation, allows you to hide the inner workings of an object within itself and protect its functions and variables from interfering with or clashing with others.

Managing Namespaces

Like any other APL object, a namespace may be copied in its entirety from a saved workspace. You may also copy specific objects from namespaces in saved workspaces by specifying their full pathnames. All copied objects are copied into the namespace in which the `)COPY` or `□CY` is executed.

Namespaces as OLE Objects

Namespaces provide the fundamental mechanism for the implementation of OLE Automation in Dyalog Version. (Also see the section on [OLE Automation and COM](#))

Furthermore, the binding between OLE objects and Dyalog namespaces is not only simple and elegant; it is practical and powerful too.

To create your own OLE Server, you simply create a namespace of type OLEServer. The functions within the namespace represent your object's methods, and the variables its properties. You can choose which functions and variables you wish to export as methods and properties and which are private and internal. Client applications, such as Visual Basic or Excel, even use the same dot syntax to access your exported functions and variables. What could be more natural?

To access an OLE Server, you create a namespace of type OLEClient as an instance of an OLE object. You can then use the methods and properties of the object with essentially the same dot syntax that is used by Visual Basic for Applications (VBA).

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



- ▶ Graphical User Interface
- ▶ Namespaces Technology

Namespace References

Namespaces may be referenced directly, just like variables, functions and operators.

For example:

```

)NS X
#.X
FOO X  A X is argument to function FOO
Y←X   A new reference to X.
    
```

References Explained

A namespace reference, or **ref** for short, is a separate data type in addition to number and character. The display form of a *ref* shows its namespace type.

Notice that *refs* are *references* to namespaces, so that if you make a copy, it is the *reference* that is copied, not the namespace itself.

This is sometimes referred to as a *shallow* as opposed to a *deep* copy.

```

X.B←'OLD'      A B is a vector in X
.
Y←X            A Y is a ref to X.
Y.B←'NEW'     A Changing Y's B,
X.B           A Changes X's B.
NEW
    
```

Similarly, a *ref* passed to a defined function is call-by-reference, so that modifications to the content or properties of the argument namespace using the passed reference, persist after the function exits. **For example:**

```

[]SE.xxx←'OLD' A xxx in []SE namespace.
▽ foo ref     A function with ref argt
[1] ref.xxx←'NEW' A change xxx in arg space
▽
foo []SE     A call foo with ref to []S
[]SE.xxx    A new value in arg space
NEW
    
```

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:
+ 44 (0) 1256 830 030

Fax:
+ 44 (0) 1256 830 031

e-mail:
sales@dyalog.com



- ▶ Graphical User Interface
- ▶ Namespaces Technology

Name Class

A name assigned to a simple scalar *ref* has name class 9, whereas one assigned to an *array containing refs* has name class 2.

```
'f1' □WC 'Form'
'ns1' □NS ''
□NC 'ns1'  A name class of a pure namespace
9
N←ns1
□NC 'N'    A name class of a scalar ref
9
□NC 'f1'   A name class of a Form
9
F←f1
□NC 'F'    A name class of a scalar ref
9
refs←N F   A vector of refs.
□NC 'refs' A nameclass of vector.
2
F2←2▷refs
□NC 'F2'
9
```

Namespace Syntax

As functions, and in general expressions, can return *refs*, the namespace reference syntax (*dot syntax*) *X.Y* allows arbitrary expressions:

```
'person1' □NS ''
'person2' □NS ''
person1.Name←'Peter'
person2.Name←'Paul'
people←person1 person2
```

The expression `▷people` returns a *ref* to namespace *person1*. You can use this result in place of the namespace name itself to reference the value of *Name*.

Dyalog Ltd
 South Barn
 Minchens Court
 Minchens Lane
 Bramley
 Hampshire
 RG26 5BH
 United Kingdom

Phone:
 + 44 (0) 1256 830 030

Fax:
 + 44 (0) 1256 830 031

e-mail:
 sales@dyalog.com



▶ Graphical User Interface

▶ Namespaces Technology

```
first↔people
First.Name
```

Peter

Or more simply:

```
(>people).Name
```

Peter

Notice that parentheses are used to delineate *ref*-returning expressions to the left of a dot:

Furthermore, expressions between dots can be arbitrarily complex. For example, the following expression displays the canonical representation of all the functions in `□SE`.

```
□SE.(□CR"↵□NL 3)  a functions in □SE.
```

Using OLE

Namespace reference syntax allows you to traverse an OLEClient object hierarchy.

The following example shows how simple it now is to execute an SQL query on a database using the Microsoft DAO.DBEngine OLE Server.

```
▽ DATA↔DATABASE SQL QUERY;DB;DBS;RCS
[1]  a Uses the OLE Server DAO.DBEngine to
    perform a
[2]  a query on an MS Access database
[3]
[4]  'DB'□WC'OleClient' 'DAO.DBEngine.35'
[5]
[6]  :Trap 11
[7]      DBS↔DB.OpenDatabase↔DATABASE
[8]      RCS↔DBS.OpenRecordset↔QUERY
[9]      DATA↔RCS.GetRows 999
[10] :Else
[11]     DATA↔'DB'□WG'LastError'
[12] :EndTrap
```

▽

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



▶ Graphical User Interface

▶ Namespaces Technology

SQL[7] calls the *OpenDatabase* method which returns a *Database* object. This is directly assigned to the *ref*, *DBS*. Similarly, *SQL*[8] calls the *OpenRecordset* function (using the *ref* *DBS*), which returns a *Recordset* object that is assigned to a second *ref* *RCS*. Finally, *SQL*[9] calls the *GetRows* method using the *ref* *RCS*.

Expressions that return namespaces do not need to be assigned to names, but can be used directly to call methods and reference properties. For example, we can combine lines [7], [8] and [9] into a single statement as shown in *SQL1*[6] below.

```

      ▽ DATA←DATABASE SQL1 QUERY;DB;□TRAP
[1]   a Shorter version of SQL
[2]
[3]   'DB'□WC'OleClient' 'DAO.DBEngine.35'
[4]
[5]   :Trap 11
[6]   DATA←□((DB.OpenDatabase←DATABASE).OpenRecordset←QUERY).GetRows 999
[7]   :Else
[8]       DATA←DB.LastError
[9]   :EndTrap
      ▽
  
```

Both these functions are provided in the OLEAUTO workspace.

Namespaces Arguments

Certain system functions and syntax elements that work on namespaces accept namespace refs as arguments as well as their names (as character vectors). These are *□CS*, *□OR*, and *□* (dyadic).

Expressions that return refs can be used as the right argument to *□CS*

```

      □CS 2>people      a change to second ref.
#.person1
  
```

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



Graphical User Interface

Namespaces Technology

The control structure *:With* may take a *ref* as argument.

```
:With [SE] ◊ ... a change to session space.
```

Dyadic execute may take a *ref* as left argument.

```
[se] '[nl] 9' a namelist in [se]
```

Unnamed Namespaces

```
'W' [WC] 'OLECLIENT' 'Word.Application'  
REL←W.Documents.Open 'myword.doc'
```

In the above statement, the expression *W.Documents* returns an unnamed namespace that is associated with the Documents object in Word.

Notice that assignment does not imply naming. The statement:

```
DOCS←W.Documents
```

Assigns a namespace reference to *DOCS* but does not give it a name. This is subtly different from the expression:

```
'DOCS' [WC] W.Documents
```

which creates a namespace called *DOCS* that is associated with the Documents object in Word. *[WC]* creates a namespace; assignment makes a *ref*.

Creating Unnamed Namespace

The monadic form of *[NS]* allows you to create unnamed namespaces explicitly.

Monadic *[NS]* creates a new unnamed namespace and returns a *ref* to it:

```
x←[NS]'' a create and name space.
```

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



- ▶ Graphical User Interface
- ▶ Namespaces Technology

It is not necessary to *name* a temporary namespace (which you must remember to localise).

For example:

```
:With []NS''      A change to new temp space.
      []CY'myws'  A copy in workspace.
      ...        A work within temp space.
:EndWith         A discard temp space.
```

Coding Examples

```
here←[]CS''      A ref to current space.
parent←here.##   A ref to parent space.
▽ ref←{ref}up levels A ref to some ancestor.
[1]   :If 0=[]NC'ref' A left arg defaults to
[2]       ref←[]CS'' A ... current space.
[3]   :EndIf
[4]   :While levels>0
[5]       ref←ref.##
[6]       levels←-1
[7]   :EndWhile
▽
(up 3).top←99 A set var 3 levels up.
up←{
      A dfn coding of 'up'.
      α←[]CS'' A default current space.
      ω≤0:α    A done: return ref.
      α.## ▽ ω-1 A up and decrement.
}
[]CS up 3      A change up 3 levels.
▽ ref←root sesh A ref to []SE or #.
[1] ref←sesh># []SE
▽
(root 1).xxx←99 A set variable in []SE.
[]SE.1 1      A use []SE's []IO.
ref2.[]'xxx←99' A execute in namespace
.
ref2 []'xxx←88' A .. .. ..
```

Dyalog Ltd
 South Barn
 Minchens Court
 Minchens Lane
 Bramley
 Hampshire
 RG26 5BH
 United Kingdom

Phone:
 + 44 (0) 1256 830 030

Fax:
 + 44 (0) 1256 830 031

e-mail:
 sales@dyalog.com