



- ▶ OLE Automation and COM
- ▶ Writing an OLE Client

Writing an OLE Client - Browsing COM Objects

This facility allows you to browse Type Libraries associated with OLE Servers (and other COM objects) that are installed on your computer or loaded in the workspace.

It allows you to explore the COM objects installed on your computer, load their Type Libraries into the workspace, and browse the Properties, Methods and Events provided by these objects.

OLEClient Object

You can access an OLE Automation Server using the OLEClient object. When you create an OLEClient, you specify the name of the Server as the `ClassName` property for the object.

For example:

```
'EX' WC'OLEClient' 'Excel.Application'
```

The effect of this statement is to create a new namespace, which in this case is called *EX*, connected to an instance of the OLE Server object. You can obtain the names of the methods, properties and events exposed by the object using the `)METHODS`, `)PROPS` and `)EVENTS`.

For example:

```
'DB' WC'OLEClient' 'DAO.DBEngine'
)CS DB
#.DB
)METHODS
AddRef BeginTrans CommitTrans CompactDatabase
CreateDatabase
CreateWorkspace DefaultPassword DefaultUser
FreeLocks
GetIDsOfNames GetTypeInfo GetTypeInfoCount
...
)PROPS
Errors Properties Version Workspaces
```

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



▶ OLE Automation and COM

▶ Writing an OLE Client

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

Syntax Rules for an OLE Client

When you create an instance of an OLEClient object, the methods and the properties of the external OLE object may be used/referenced as if they were functions and variables in the OLEClient namespace.

In principle, a method is called like an APL function and a property is referenced or set like an APL variable.

However, OLE is in fact heavily influenced by the Visual Basic programming language and APL syntax is very much stricter than Visual Basic syntax. For example, you may call a Visual basic function with or without arguments. The same is not true in APL in which a niladic function is syntactically different from a monadic function. A second major difference is that OLE does not support the APL concept of a dyadic function.

The following set of rules has been adopted to accommodate OLE and Visual Basic syntax.

Calling Methods

You invoke a method in an OLE object by calling the method as if it were a defined function in your workspace.

If a method takes no parameters, you invoke it as if it were a niladic function.

If a method takes parameters, you invoke it as a monadic function; each element of the argument corresponds to each of the method's parameters.

If a method takes only *optional* parameters, you may call it without specifying any parameters by supplying an empty numeric vector (zilde) as the argument.

Single arguments are treated specially. Specifically, if you call an OLE method with a single argument that is a character vector or an array whose rank is greater than one, you do not need to enclose it. The argument will automatically be enclosed if required.

For example, the OpenDatabase method in the DAO.DBEngine OLE server may be called with a single parameter that specifies the name of the database to be opened. You may call it from APL with either of the following two expressions:

```
OpenDatabase 'c:\example.mdb'
OpenDatabase c'c:\example.mdb'
```

Note that when calling OLE methods, you may only use characters and numbers; special APL data types such as `⎕OR` may not be used even if you are calling functions in an APL server.



- ▶ OLE Automation and COM
- ▶ Writing an OLE Client

Arrays and Pointers

Many parameters to OLE methods are specified by pointers. If, for example, the parameter type is VT_BSTR, it means that the calling routine must supply a pointer to (i.e. the address of) a character string.

Similarly, if the parameter type is defined to be VT_VARIANT, it means that the parameter is the address of an arbitrary array (the VT_VARIANT data type actually maps nicely onto a Dyalog nested array).

The rule is that if a pointer is required, APL will provide it automatically; you do not have to do so. Instead, all you do is supply the value.

Optional Parameters

Methods are often defined to have optional parameters. For example the parameters defined for the OpenDatabase method provided by the DAO.DBEngine OLE object are:

Name	VT_BSTR
[Exclusive]	VT_VARIANT
[ReadOnly]	VT_VARIANT
[Connect]	VT_VARIANT

To call the corresponding APL function, you may supply a nested array that contains 1,2, 3or 4 elements corresponding to these parameters.

The parameters to some methods are all optional. This means that the method may be called with or without any parameters. As APL does not support this type of syntax, the special value \emptyset (zilde) is used to mean "0 parameters".

For example, the parameters for the Idle method provided by DAO.DBEngine are defined to be:

[Action] VT_VARIANT

This means that the method takes either no arguments or one argument. To call it with no argument, you must use \emptyset (zilde), **for example:**

Idle \emptyset

Note that you cannot therefore call a function in an APL server with a single argument that is an empty numeric vector.

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



- ▶ OLE Automation and COM
- ▶ Writing an OLE Client

Output Parameters

You may encounter parameters whose data type is defined explicitly as a pointer to something else, for example VT_PTR to VT_UI4 specifies a pointer to an unsigned 4-byte integer.

In these cases, it usually means that the calling routine is expected to pass an address into which the OLE method will place a value.

When you call the function you must use data of the type pointed to.

The result of the function is then a vector containing the result returned by the method followed by the (new) values of the output parameters. This is similar to the mechanism used by [N.A.](#)

Named Parameters

Visual Basic syntax allows you to specify parameters by position or by name; rather like [WC](#) and [WS](#). For example the parameters defined for the OpenDatabase method provided by the DAO.DBEngine OLE object are:

Name	VT_BSTR
[Exclusive]	VT_VARIANT
[ReadOnly]	VT_VARIANT
[Connect]	VT_VARIANT

You could call this method from Visual Basic using the syntax:

```
Set Db =
OpenDatabase (Name:="c:\example.mdb", ReadOnly:=True)
```

You may do the same thing from Dyalog, using [WS](#) syntax. For example, the equivalent call from APL would be:

```
OpenDatabase('Name' 'c:\example.mdb')('Exclusive' 1)
```

Note that you may only use named parameters if they are supported by the method. Many methods do not allow them.

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



- ▶ OLE Automation and COM
- ▶ Writing an OLE Client

Methods that return Objects

Object hierarchies in OLE are not static, but are created dynamically by calling methods that return objects as their result.

If the data type of its result is VT_DISPATCH or VT_COCLASS, or VT_PTR to VT_DISPATCH, the result of the method is a namespace reference to another OLEClient object.

Alternatively, a method that returns an object must be called with the name of the (new) namespace as its left argument. Note that OLE methods do not themselves accept left arguments, so this extension does not conflict with OLE conventions.

For example, GetMethodInfo tells us that the syntax for the OpenDatabase method provided by the OLE object DAO.DBEngine is as follows:

```

↑DB.GetMethodInfo 'OpenDatabase'
Opens a specified databas VT_DISPATCH
e                               H
Name                             VT_BSTR
[Exclusive]                       VT_VARIANT
[ReadOnly]                         VT_VARIANT
[Connect]                          VT_VARIANT
    
```

The data-type of the result is VT_DISPATCH, so it returns an object; indeed the help for the method tells us that it returns a Database object. The method can be called from APL as follows:

```

DB1←OpenDatabase c'example.mdb'
or
DB2' OpenDatabase c'example.mdb'
    
```

In the first case, DB1 is a a namespace reference to an OLEClient (database) object. In the second case, DB2 is the name of a namespace to be associated with the same object. In either case, you can reference properties and methods directly using dot syntax.

It is not actually necessary to assign the result to a name. If you only need to reference a single property or method in the result object, you may do so using parentheses. For example, the following expression first gets a Database object by calling OpenDatabase, then gets a Recordset object by calling its OpenRecordset method, then finally gets the data by calling its GetRows method.

```

DATA←⊘((DB.OpenDatabase<DATABASE).Op
enRecordset<QUERY).GetRows 999
    
```

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



- ▶ OLE Automation and COM
- ▶ Writing an OLE Client

Referencing Properties as Variables

Nearly all properties exposed by an OLE object may be referenced as variables.

To query the value of a property, you simply reference it. To set the value of the property, you assign a new value to it.

The data type of the variable is reported by the *GetPropertyInfo* method. Conversion between APL data types and OLE data types is performed automatically.

If you attempt to set the value of a property variable to an something with an inappropriate data type, APL will generate a *DOMAIN ERROR*.

If you set the value to something of the correct data type, APL will pass it through the OLE interface. However, the OLE object may itself reject the new value. In this case, APL will also generate a *DOMAIN ERROR*. However, the OLE error information may be obtained from the *LastError* property of the object itself or of *Root*.

Referencing Properties as Functions

Some properties are implemented internally by an OLE object as functions that take arguments. Typically, there will be two functions (with the same name as the property), a Set function to set the value of the property and a Get function to retrieve it. Note that in OLE there is no name conflict because internally all functions are called via function pointers and not by name.

Most of the properties that exhibit this behaviour are in fact *array properties* and the corresponding Set and Get functions require an index to be specified. However, array properties are very different from APL arrays. For example, their indices do not have to be contiguous or even numeric. The index is in effect a parameter and not a true index in the APL sense.

In these cases, the property must be treated as it it were a function.

To obtain the value of the property, you must call the property as a monadic function, specifying the required index (or other information) as the argument.

To set the value of the property, you must property as a dyadic function, specifying the required index (or other information) as the right argument and the new value as the left argument.

Note that *)PROPS* and *PropList* report the names of all of the properties of an object, regardless of whether the property is implemented as a variable or as a function.

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



- ▶ OLE Automation and COM
- ▶ Writing an OLE Client

Properties as Objects

Dyalog permits an object hierarchy to be represented by a namespace hierarchy. In other words, the relationship between one object and another is a parent-child relationship whereby one object owns and contains another.

Visual Basic has no such mechanism and the relationship between objects has to be specified in another way. This is commonly done using properties. For example, an object view of a bicycle could be a hierarchy consisting of a bicycle object that contains a Frame object, a FrontWheel object and a RearWheel object. In Visual Basic, you could represent this hierarchy as a Bicycle object having Frame, FrontWheel and RearWheel *properties* which are (in effect) pointers to the sub-objects. The properties are effectively used to tie the objects together.

An extension of this idea is the Visual Basic Collection object. This is a special type of object, that is somewhat similar to an array. It is used where one object may contain several objects of the same type. For example, a Wheel object could contain a Spokes collection object which itself contains a number of individual Spoke objects. Collection objects are usually implemented as properties.

When you reference the value of an object property, you will get a namespace reference.

Using the bicycle analogy, you could recreate the object hierarchy in the APL workspace as follows:

```
'BIKE' ⌈WC'OLEClient' 'EG.Bicycle'
FRONT←BIKE.FrontWheel
REAR←BIKE.RearWheel
```

The result would be a namespace called *BIKE*, and two unnamed namespaces referenced by *FRONT* and *REAR* each containing the specific properties, methods and events exposed by the three corresponding objects.

Note however, that in this example *BIKE*, *FRONT* and *REAR* are all top-level namespaces; a proper parent/child representation can be achieved by making *FRONT* and *REAR* child namespaces of *BIKE*.

For example:

```
BIKE.FRONT←BIKE.FrontWheel
BIKE.REAR←BIKE.RearWheel
```

This example illustrates that when you work with an OLE object, you have a choice whether to represent an object hierarchy as a namespace *tree* or just as a collection of otherwise unrelated namespaces.

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