Module7: OLE Clients

OLE Clients drive OLE Servers via Object Linking and Embedding. This leads to very powerful connections between applications such as Dyalog APL and Microsoft Office.

§ 7.1 Inside Microsoft Word

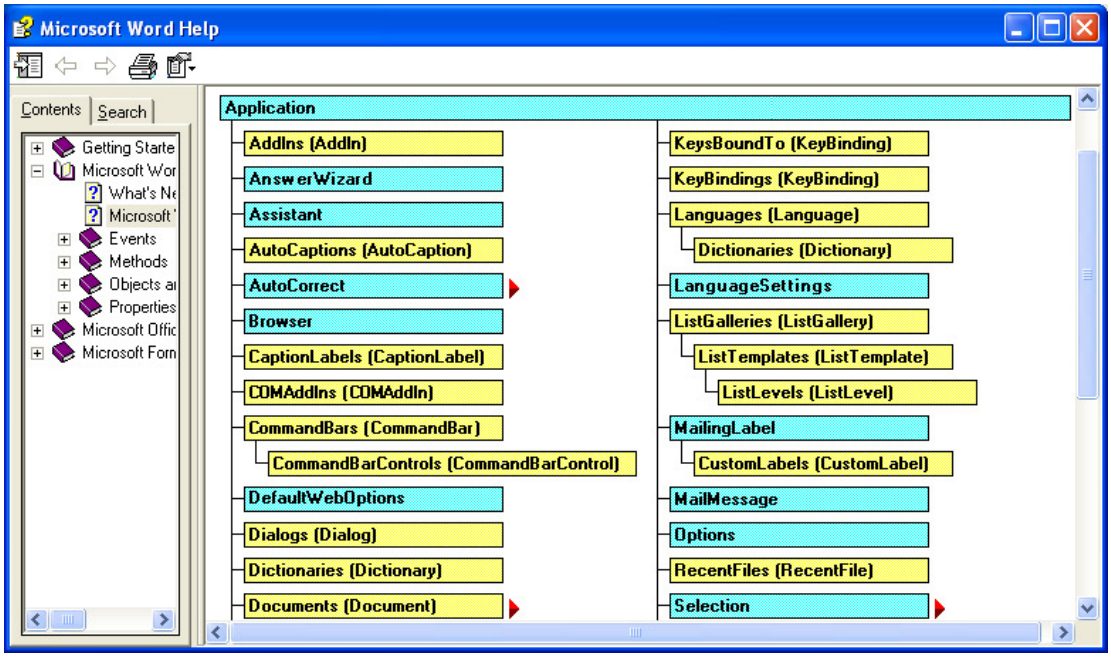
§§ 7.1.1 Registry Entries, Object Models and Type Libraries

If Microsoft Office is installed on your computer, then Word.Application will appear in the list `#.OLEServers`. The corresponding class ID agrees with that in the registry under HKEY_CLASSES_ROOT\Word.Application\CLSID. The registry entry then points to the most up-to-date version of Word currently installed, such as Word.Application.9.

This can be investigated further to find that HKEY_CLASSES_ROOT\CLSID contains key {000209FF-0000-0000-C000-000000000046}\LocalServer32
which contains the name of the program used for OLE automation:
C:\PROGRA~1\MICROS~2\Office\WINWORD.EXE /Automation

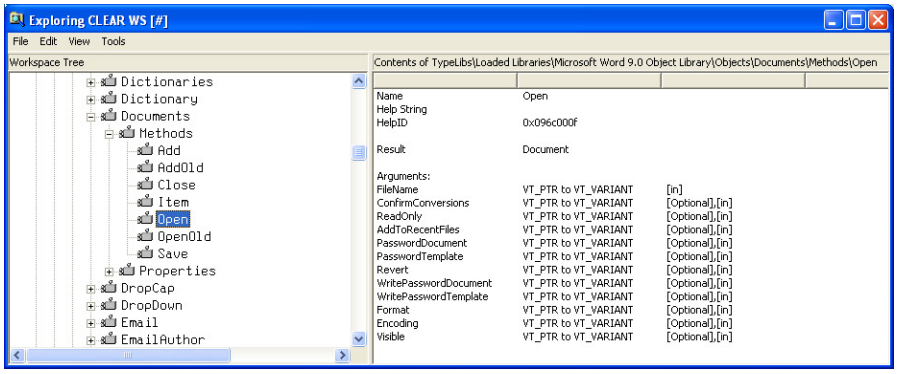
The Word program is vastly more complicated than our server of Module 6, and understanding its object structure and contents is a major task. The Word 9.0 VBA help file (VBAWRD9.CHM) is most helpful in this regard. The help file is particularly useful to programmers wishing to call Word object methods and properties from different environments such as VB or APL.

The Word object model reveals all the potential object hierarchies in a Word application. The properties and methods associated with each type of object are described in the help file, often with an example VBA call. This is close enough to the APL equivalent to be most useful when programming Word-linked APL applications.



To access an OLE server, you create a namespace of type `OLEClient` as an instance of the OLE server. The `ClassName` property of the `OLEClient` identifies the server and has to be set at create time. It has value `'Word.Application'` in the case of Microsoft Word.

7.1.1.1 Create an `OLEClient` object with `ClassName` set to `'Word.Application'`. Open the workspace explorer and browse the loaded type libraries. For example, explore the Microsoft Word Object Library and look at Objects\Documents\Methods\Open. Relate this to MS Word [File][Open].



7.1.1.2 Change into the `OLEClient` space, and compare the above with the result of

```
↑Documents.GetMethodInfo'Open'
```

Thus there are a number of ways to find out the calling syntax and argument types for methods in Word.

§§ 7.1.2 Digging into Word

7.1.2.1 In a clear workspace, start Word as an OLE Client and change space into the Word application.

```
⎕CS'WRD'⎕WC'OLEClient' 'Word.Application'
```

Set the `Visible` property to `1`. Write a function called `∇show∇` which displays its Rarg in the session. Set the `Event` property for all events to `∇show∇`.

```
Event←'All' 'show'
```

Look at the `EventList` property and try to fire an event which will show in the session. In Word, select [File][Exit] and note the `Quit` event in the session.

7.1.2.2 In a clear workspace, trace the function below and identify the methods and properties being used.

```
      ∇ WordExample;WRD
[1]     :With 'WRD'⎕WC'OLEClient' 'Word.Application'
[2]         Visible←1
[3]         :With Documents
[4]             :With Add 0
[5]                 :With Content
[6]                     Text←,(50 50⍴⎕A),3⊃⎕TC
[7]                 :End
[8]                 SaveAs'c:\myword.doc'
[9]             :End
[10]        :End
[11]        Quit 0
[12]    :End ∇
```

At each change of space, check the name of the current namespace and the methods and properties available in that space. Right click on methods `Save`, `SaveAs` and `Quit` to view their calling information. In the workspace explorer, investigate the `#.WRD` object and its children.

Notice that some of the reported namespace names are surrounded with brackets, *eg*

```
      )ns
#.WRD.[Documents].[_Document]
```

This indicates that these namespaces have not actually been given any name at create time. `Documents` is a property that returns a reference to a collection object. This reference is sufficient for `:With` to be able to deal with the collection, which is essentially a vector of objects (see Module11 on Arrays of Objects). `Add` is a method that returns a reference to a new document object. The result could be assigned to a name if a name is required.

Information about properties in the current space may be found using `GetPropertyInfo` method of `OLEClient` objects. eg In the Documents collection, `GetPropertyInfo'Count'↳VT_I4`.

§§ 7.1.3 Demonstrating the Power of OLE

7.1.3.1 In a clear workspace, create an `OLEClient` for the OLE server Word.Application. Enter the `Documents` collection and `Open` file C:\myword.doc. Make Word `Visible`.

7.1.3.2 Enter the `Documents` collection and `Add` a new document. Trace the following function snippet which adds and fills and colours a `Table`.

```
:With Tables
    :With Add(#.WRD.Selection.Range,3,5)
        :For x :In ⍳3
            :For y :In ⍳5
                (Cell(x,y)).Range.InsertAfter'Cell(',(⍕x),',',(⍕y),')'
            :EndFor
        :EndFor
        Columns.AutoFit
        :With Rows
            :With Item 1
                Select
                Alignment←1
                :With #.WRD.Selection.Font
                    Bold←1
                    Color←256 256 256⊥⌽255 127 0 ⍝ Orange
                :End ⍝ font
            :End  ⍝ row
        :End  ⍝ rows collection
    :End  ⍝ table
:End  ⍝ tables collection
```

7.1.3.3 While still within the new document, enter some text after the table using something like the following snippet, noting the parentheses and the reduced number of `:With`s.

```
:With Paragraphs.(Item Count)
   Range.Text←1000⍴100↑20↓⎕AV
:End ⍝ paragraph
```

7.1.3.4 While still within the document, add some suitable text to the first cell in the table and convert the text into a table (within a table) as in the following snippet:

```
:With Range(0,0)
   Text←,(⍕(⍕¨⍳10),'#',¨10 2⍴⍕¨⍳30),3⊃⎕TC
   Select  ⍝ this selects all
   DefaultTableSeparator←'#'
   ConvertToTable'#'
```

```
    :With Tables
        :With Item 1
            :With Columns
                :With Item 1
                    Select
                    :With #.WORD.Selection.Font
                        Bold←1
                        Color←256 256 256⊥⍬0 0 255⍝ Blue
                    :End ⍝ font
                :End  ⍝ column
            :End  ⍝ columns collection
        :End  ⍝ table
    :End  ⍝ tables collection
:End  ⍝ range
```

Tip: A useful way of constructing such code, apart from valuable help from VBAWRDx.CHM, is to record a macro in Word which takes the steps that you want your program to take, and then use **Alt+F11** to examine the VBA code.

Tip2: You might find that after creating and destroying (erasing or expunging) an `OLEClient` and saving the WS that the size of the WS has grown considerably. This happens because the TypeLibs, visible in WS Explorer have been saved too. They may be removed before saving by running the Root methods

```
#.DeleteTypeLib¨⊃¨#.ListTypeLibs
```

§ 7.2 Manipulating Microsoft Excel from the Inside

§§ 7.2.1 Recognising the Object Model

The object model for Excel is very similar in appearance to that of Word. The model for Excel 9.0 is to be found in help file VBAXL9.CHM. It is advisable to use the correct version of this help file for your particular Excel version.

As with Word, the registry entries for the Excel.Application `ClassName` can be investigated. More importantly from the point of view of writing APL-Excel OLE applications, the TypeLibs in WS Explorer, or `GetMethodInfo`, `GetPropertyInfo`, `GetEventInfo` and `GetTypeInfo`, should be used to obtain information about Excel functionality.

§§ 7.2.2 Digging into Excel

7.2.2.1 Create an `OLEClient` with `ClassName` Excel.Application. Make the application `Visible`. Set the application `Caption` to 'L&G Excel'. Look at the values of the application properties `MemoryFree`, `MemoryUser`, `MemoryTotal`, `LibraryPath`, `TemplatesPath`, `Path`, `Name`, `UserName`, `Value`, `Version`, `Height`, `Width` and `WindowState`.

7.2.2.2 With the `Workbooks` collection, `Add` a new workbook and look at the values of properties `Author`, `Path`, `Name`, `FullName` and `UserStatus`. Run `GetMethodInfo` on methods `SaveAs` and `Close` and then save the file as something like 'C:\myexcel.xls' and run the method `Close`. Then `Quit` the application.

7.2.2.3 Write a function that will start Excel (visibly), enter the `Workbooks` collection and `Open` the `workbook` 'C:\myexcel.xls'. In the application space, assign the `Range` between A1 and B2 in the `ActiveSheet` to a name for the reference and then assign the `Value` (or later the `Value2`) property of the ref to a suitable matrix.

7.2.2.4 Under this different style of programming, trace the following function and explore the references.

```
      ∇ OLEExcel2;XL
[1]     ⎕CS'#.XL'⎕WC'OLECLIENT' 'Excel.Application'
[2]     Visible←1
[3]     Wkb←Workbooks.Open'c:\myexcel.xls'
[4]     Wks←Wkb.Worksheets.Item 1
[5]     Rng←Wks.Range'A1:B10'
[6]     Rng.Value2←?10 2⍴100
[7]     Ch←Charts.Add ⍬
[8]     Ch.ChartType←xlColumnClustered
[9]     Ch.SetSourceData #.XL.Rng
[10]    ∇
```

§§ 7.2.3 Gaining full Control of Excel

Like Word, Excel is a big program with many dark corners. Thankfully, the task of learning how to use the OLE interface mirrors quite closely the task of learning how to use Excel itself. (The task of learning WORDBASIC, used with the APL shared variable approach to DDE communication with Word, was closely tied to the menus of Word, but now with OLE much more of the functionality of Word is exposed.) Each little step in Excel which can be incorporated in an APL program acts as a stepping stone for ever more ambitious and detailed communications with Excel.

The Word document of the monumental Dyalog APL *Object Reference* manual itself embodies an example of Word-APL OLE. Each description of an object in the manual opens with a section containing **Purpose, Parents, .., Methods**. The contents of these sections are mustered and positioned in the Word document via OLE in an APL workspace, thanks to the tireless work of Peter Donnelly.

7.2.3.1 Write a function `∇putMatrix∇` which takes a matrix Rarg and an optional `Offset` Larg and places the matrix in an Excel worksheet, offset by the given number of rows and columns.

Tip: `⎕A,(,⎕A∘.,⎕A),(,⎕A∘.,⎕A∘.,⎕A)` generates the names of the first 18278 column names in an Excel worksheet (see a generalised version in the DFns Module12☺.

§ 7.3 Linking to other Servers

§§ 7.3.1 Outlook

If Outlook.Application is in you list of `OLEServers`, then you can create an `OLEClient` with this `ClassName`. In that space you will find a method called `CreateItem`. This method returns an object whose type is determined by the Rarg. Available types are to be found in the list of Enums in the WS Explorer TypeLibs Loaded Libraries, in the Microsoft Outlook 9.0 Object Library. The Enums section has an entry called OlItemType. This contains a list of possible types and the Enum appropriate in each case. For example, to create a mail item use Enum olMailItem which has value zero. Thus the `CreateItem` method can take a Rarg of 0 or `olMailItem`. (`olMailItem` is an invisible keyword in the WS, not listed under `)VARS` or `)PROPS` and with `⎕NC'olMailItem'↳0`)

7.3.1.1 Enter the following two lines and assign the properties `Subject` and `Body` to suitable values. The `Body` of a message is a character string, as may be deduced from the result of `GetPropertyInfo'Body'`. Each new line in the `Body` should be terminated with a linefeed character. If `⎕ML<3` then `⎕TC[1+⎕IO]≡⎕AV[2+⎕IO]` and this is the linefeed character needed.

```
⎕CS'Outlook'⎕WC'OLEClient' 'Outlook.Application'
⎕CS CreateItem olMailItem
```

The current namespace contains a property that returns a `Recipients` collection. As usual, in this collection space there is an `Add` method that returns an object of appropriate type. The `Add` method type library calling information describes the Rarg of `Add` as VT_BSTR. It is in fact an enclosed character string which corresponds to an entry in your Outlook address book or a raw email address. For example

```
Recipients.Add⊂'Karen Shaw'
```

will check your address book and resolve the entry, if possible. If you are lucky, it might be resolved to karen.shaw@monadic.com or karen.shaw@dyadic.com or perhaps to briony.williams@triadic.com. Or you could add the recipient's email address directly as

```
Recipients.Add⊂'karen.shaw@dyadic.com'
```

Any number of recipients may be added in this way.

You can tell if a name was resolved successfully from the result of the niladic method; `Resolve`. If it does not get resolved properly and `~Resolve` then the message may be removed by means of the `Remove` method in the `Recipients` collection. The Rarg of this method is the item number of the recipient object in the collection. If there is only one recipient object in there, then Rarg is 1.

At this point the `Type` of the message may be changed. (Note the `Type` keyword conflict resolution.) The default `Recipeients.Type` is 1, or `olTo`, but it could be any of a number of `Type`s, their Enums being those in the group `OlMailRecipientType`. (Double clicking on these key words in the APL session displays their contents.)

The names of those to whom the message will be address is returned by the `To` property in the unnamed [_MailItem] namespace. If some of the recipients were of Type `olBCC` then the property `BCC` returns the list of names.

At this point all that needs to be done is to run the niladic, non-result returning method `Send`.

7.3.1.2 The mail item namespace contains a property called `Attachments` that returns a collection of attachment objects. Attach a file to an email by calling the `Add` method of this collection after checking the method's calling information using the property sheet obtained by right-clicking on the method name in the session, or by way of the WS Explorer.

```
Attachments.Add'C:\myword.doc'
```

§§ 7.3.2 Microsoft Internet Explorer

Have you ever wanted an APL function which takes a url as its argument and returns the retrieved html text as its result? The InternetExplorer.Application may be used for this, as Tommy Johansen has shown to the dyalogusers@yahoogroups.com mailbox group.

7.3.2.1 Start Internet Explorer as an OLEServer by

```
⎕CS'IE'⎕WC'OLEClient' 'InternetExplorer.Application'
```

Your program might need a delay (`⎕DL`) of a few seconds at this point to give time for the server to initialise properly. There is a `Busy` property in the application that should be checked after each significant action and a short delay included in the program `:While Busy. Visible` may be set to `1` if you wish to see the net activity.

7.3.2.2 Find an address of interest and `Navigate` to that address, eg

```
Navigate'http://www.simcorp.com'
```

or

```
Navigate'http://finance.yahoo.com/q/ecn?s=IBM'
```

or

```
Navigate'http://finance.yahoo.com/p?v&k=pf_1'
```

7.3.2.3 In order to obtain the body section of the resulting html, first get the DispHTMLDocument object returned by the `Document` property and then the get the DispHTMLBody object returned by the document's `body` property. Note the lower case spelling of `body` and of many of the object's other properties and methods. The `outerHTML` property of the DispHTMLBody object contains the <BODY>..</BODY> character string.

§§ 7.3.3 Beyond

It is possible, through OLE, to link with any of the servers listed by the Root property, `#.OLEServers`. For example, if you have DAO.dbEngine installed then it is possible to read data from Microsoft Access files directly into an APL workspace. Or if you have CrystalReports installed you can communicate directly with that. All sorts of applications, from computer-animated *synthespians* (examples of which may be downloaded from http://www.microsoft.com/MSAGENT/downloads/user.asp) to music and radio players, may be incorporated in your APL applications by object linking and embedding.

As a final example, the call to `⎕CMD` to open Notepad

```
⎕CMD'Notepad' ''
```

may be replaced by

```
'WSS'⎕WC'OLEClient' 'WScript.Shell'
WSS.Run'Notepad'
```

or

```
WSS.Exec'Notepad'
```

Remember to clean up the workspace by

```
#.DeleteTypeLib¨⊃¨#.ListTypeLibs
```

before saving if you don't want to save all the TypeLibrary information in the workspace.

7.3.3.1 Ask for the next module on **ActiveX Controls** ☺.