What is OLE?

Object Linking and Embedding

I shall expand on the simple answer above! The purpose of OLE is to allow a 'client' application to edit certain types of object data without having to know how to do this itself. Although this process is not documented in the User Manual, the Style Guide or the Programmers' Reference Manual, it would be explained in the documentation for such a 'client' application.

The process was developed by Computer Concepts in 1993 and even allows 'server' applications such as Draw and Paint (which do not support OLE) to be made unconscious use of in this way.

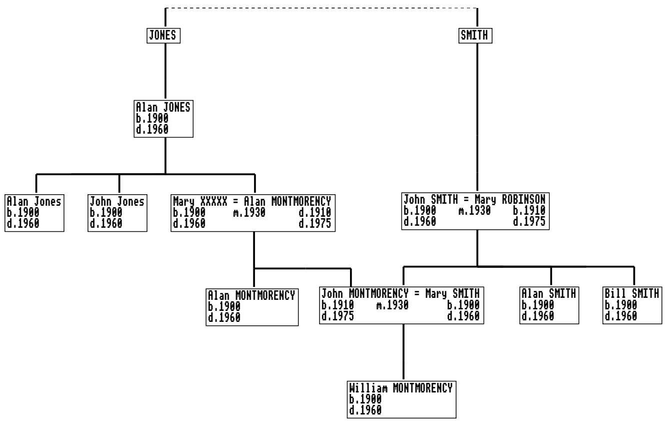
Such a server application would receive the same message that would be broadcast by the Filer when a file in a Filer window is double-clicked. The file itself would be in a location known to the client, probably in its 'Scrap' directory. Provided that the edited file is saved back to the same location (a constraint which also applies to server applications that know about OLE), the 'client' application would, in principle, be able to update the embedded object each time it is saved.

Embedded objects

For an embedded object to be capable of OLE it needs to have a filetype that identifies the serving application: for example a Draw file defines the !Draw application as one that can edit Draw files and a Draw file can be embedded in many applications as a vector graphic image.

Generally speaking, applications can identify the type of data in a file from its filetype or from its content (many types of file contain header information in the first few words characteristic of the type of data they contain). In RISC OS it is fairly rare to use the latter method.

!FamTree.Example.Smith+Jones


JONES
SMITH
Alan JONES b.1900 d.1960
Alan Jones b.1900 d.1960
John Jones b.1900 d.1960
Mary XXXXX = Alan MONTMORENCY b.1900 m.1930 d.1910 d.1960 d.1975
John SMITH = Mary ROBINSON b.1900 m.1930 b.1910 d.1960 d.1975
Alan MONTMORENCY b.1900 d.1960
John MONTMORENCY = Mary SMITH b.1910 m.1930 b.1900 d.1975 d.1960
Alan SMITH b.1900 d.1960
Bill SMITH b.1900 d.1960
William MONTMORENCY b.1900 d.1960

The graphic above is a 'TreeData' file, type &1E6, it contains both the graphic image that should be displayed and, embedded in a tag, the genealogical information from which the image was generated.

Let us say that you have a 'server' application that uses some internal data to produce a Draw graphic. A good and very simple example is TableMate: it saves its data in a 'TblMate' file (type &BCF).

Double-clicking a 'TblMate' file performs the action you would expect: it runs TableMate Designer II (if it is not already running) and loads the file for editing and re-saving.

Examining a 'TblMate' file shows that its internal format is that of a Draw file. Impression, ArtWorks, Ovation Pro and TechWriter know (through hard coding) that files of type &BCF contain Draw data and will embed them as a vector graphic.

Impression and TechWriter will also examine the header of a file of unknown file type and will recognise a file containing Draw data and load it as a vector graphic. They will also respond to a CTRL-double-click over such an object by saving the object data as a 'Scrap' file and sending the same message that the Filer would send if a file in a Filer window was double-clicked (and after no response, run the application concerned first).

Ovation Pro will respond to a CTRL-double-click over such an object (i.e. one that it recognises) in approximately the way described. ArtWorks does not.

A Drawfile that is not a Draw file?

Clearly if a 'TblMate' file contained only a vector graphic image then the internal formatting information would be difficult to recreate. The Draw file format allows extra data to be included in a Draw file held within a 'tag' object - the 'tag' object contains a 'tag' number (allocated by ROOL) plus a single Draw object. In the space following this Draw object and up to the end of the 'tag' object is stored data that should be understood by the owner of that tag number.

So TableMate can thus reserve space to save its internal data as well as creating a Draw vector graphic image. Other examples of such data embedded in a Draw format file are type &D91 (Equasor) and type &1E6 (TreeData used by FamTree).

In all these cases, only the tag information is considered when the file is loaded and the vector graphic image is recreated from scratch.

OLE needs to be 'two-way'

What has been described so far is a one-way process: the client application tells the server where the file is that is to be edited. It is all very well to know where you put the file but how do you know when it has been altered or re-saved? You could just keep looking at it to see whether the file datestamp has changed but you might find it incomplete or open with data partially written to it.

The OLE_Support module provides a way for the object to be linked: it will use 'up calls' (SWI OS_UpCall) to monitor when the file has been modified and notify the client by a Wimp Message.

The support module has two methods for doing what is required: originally it required the client to broadcast the message that a file was to be edited and once this had been done the client was expected to start the module and use the OLE_LinkFile SWI to ask for the file to be watched.

An improved method can also be used: the presence or absence of a system variable OLEServer$Type_ttt determines whether the server application handling files of type ttt is aware of the OLE protocol.

If the server is unaware (no variable exists) then the client application should start up the OLE_Support module and issue a SWI call OLE_SimulateSession. This will create the system variable for this file type, identifying the OLE_Support module itself as the task to start up if the session cannot be opened.

Now the client can use a Message_OLEOpenSession message in place of a Message_DataOpen message to identify the file to be edited. This will start up the server application if not already running.

```
OLEServer$Type_1E6 : -N FamTree -R run ADFS::HD4.$.Apps.!FamTree.!Run
OLEServer$Type_xxx : -N OLESupport -R /Desktop_OLESupport
```

The client calls the SWI OLE_SimulateSession if no system variable has been set up (server unaware of OLE) and the support module will set the variable for filetype xxx. The client can then reuse its normal message passing and variable scanning code for that file type. The application it will start up is the OLESupport module task. When this module recives a Message_OLEOpenSession, it will respond as normal with an acknowladgement. It will however also send a DataOpen message using information passed in OpenSession to get the 'real' but 'non-compliant' application to load the data. If this fails it will attempt to run the file. Having created a link, the module will then keep track of the file through up calls and inform the client everytime the real server saves to the file with an OLEFileChanged message.

When notified by the module that the file has been modified, the client should update the object being displayed accordingly.

Practical effect of OLE

A file that contains a Draw graphic is loaded into a client application which displays it as a vector graphic image. If the user CTRL-double-clicks over this object then it is loaded by an application that can edit such files allowing the edited version to be saved back into the client.

The subtle point here is that the editing application does NOT use the Draw data in the file but only the tag data and recreates the vector graphic image itself from scratch using the tag data only.

Using equations in Impression, TechWriter or Ovation Pro is greatly simplified: create the equation in Equasor and save the file in a frame. The graphic is displayed. CTRL-double-click and you can edit the equation in Equasor and save it back.

The same method with TableMate allows tables to be included and edited. Impression and TechWriter can also cope with such graphics in newly written applications: a FamTree file can be included as a graphic and reopened in FamTree to move the boxes in the family tree around.

Artworks will load Equasor or TableMate files as vector graphics (these file types are 'hard coded' in ArtWorks to be treated as Draw files) but it cannot support OLE.

Conclusion

This is a useful prorocol but, to say the least, is not very well publicised.

The module OLESupport is bundled with TechWriter, Messenger Pro, Impression Style (free to download) and Impression Publisher Plus (and X).

Chris Hall *chris@svrsig.org*