

OLE and OCX

Overview

J OLE Automation Server

J OLE/OCX Client

Examples

Tutorial: J OLE Server for Excel

Tutorial: J OLE Client to Excel

Overview

OLE allows client programs to access server programs, enabling integration between a variety of software components. All OLE servers are usable by all OLE clients.

An OCX control, also known as an ActiveX control, or Microsoft Custom Control, is a special form of OLE control. The main difference is that OCX controls cause events, and are *in-process* server objects (DLL) whereas OLE controls are *local-server objects* (EXE).

Both OLE and OCX controls are application packages that can be accessed in a simple and consistent way from other Windows applications.

J Win95/NT supports OLE and OCX, enabling J to be used both as a calculation server and as a client for other Windows software such as Visual Basic and Excel. In particular, OLE makes it very easy to add the power of J to other applications.

OLE

A key part of OLE is that a server can run in the same task, or as a separate task, or even as a separate task on another network connected machine. A server in the same task is an *in-process server*, a server in another task on the same machine is a *local server*, and a server on another machine is a *remote server*. An in-process server is the most efficient; the task can call the in-process server's routines almost as efficiently as its own routines.

Servers provide functions, called *methods*, that can be accessed by a client. For example, the J server provides a method `Do` which executes a J sentence; and a method `Get` which retrieves the value of a variable. If a client runs the method `Do 'x = 3+5'`, this causes the sentence to be executed in the J server. The client can then call method `Get` to retrieve the value of `x`.

OLE Automation is an interface supported by many clients and servers. OLE Automation gives a client an interface to a server that mimics the server's end-user interface. This allows a client to automate what would otherwise have to be done manually by a user. Although this is useful, the real power comes when the server has a general purpose programming environment that can be used to build complex services that can be used from a client. OLE Automation then provides easy access from the client to the full power of the server.

J supports OLE Automation as follows:

J OLE Automation server. There are two servers:

- *JDLLServer* is an in-process J server
- *JEXEServer* is a local J server

J OLE Automation client. This supports servers such as Excel, that provide an OLE Automation interface to an application object that supports macro execution.

J OLE Automation Server

There are two servers:

- JEXEServer is the full J development system and is intended for use in developing applications. This is the same J.EXE file that provides the regular J development system.
- JDLLServer is the J interpreter only, and is intended for runtime applications. This is in file J.DLL.

An EXE server runs as a separate application from the client, i.e. is a local server. JEXEServer provides the full J development system, which makes it easy to develop and debug applications. As a developer, you have full access to both the client and the J server environments.

A DLL is part of the client application and uses the same memory space, i.e. is an in-process server. A client accesses DLL services almost as efficiently as it accesses its own native services. The JDLLServer is not as convenient as JEXEServer for development purposes, but it is very efficient and is ideal for runtime applications.

Typically, you develop your application using JEXEServer, and run it using the JDLLServer.

Note that JEXEServer and JDLLServer are 32bit servers and are designed for 32bit clients. It may be possible to use JEXEServer from 16bit clients, but this is not officially supported.

J.DLL can be called directly as an ordinary DLL, without using OLE. For more information, see online help *DLLs and Memory Management*.

Clients

Any application with OLE Automation controller support, such as Visual Basic, Delphi, Excel, or a Visual C++ application, can use JEXEServer and JDLLServer. Also, any application that can call 32bit DLLs can access JDLLServer.

Registration

JEXEServer and JDLLServer must be registered with your system before they can be used. To do so, you run JREG.EXE , which is stored in the same directory as J.EXE, i.e. select Start/Run and enter:

```
c:\j401\jreg.exe      (use the correct directory name)
```

If there are problems later when accessing the J servers it may be because they are no longer properly registered. You can always run JREG.EXE to register the servers again. In particular, you will need to do this if you move the J system files to another directory, or if for some reason, the Registry is damaged and you have to recover an old version.

Using the J OLE Automation servers

The steps are fairly straightforward, but may differ in minor ways from one client to another.

You should be familiar with both the J system and the client before tackling them in a client/server combination.

First load the client application and ensure it references the J servers. You need only select the server you intend to use, but when experimenting, you should check both:

Once the J servers are referenced, you can check the methods available, which are as follows.

Break	interrupt J execution
Clear	erases all definitions in J
Do	execute a J sentence
ErrorText/ErrorTextM	get error text (run after a J error)
Get/GetB/GetM	get the value of a J variable
IsBusy	returns 0 if J is ready to execute, else an error code
Log	display (1) or discard (0) the J EXE session log
Quit	causes J EXE server to close when last object is released
Set/SetB/SetM	set a value to a J variable
Show	show (1) or hide (0) the J EXE server
Transpose	return array data transposed

For details, see file system\examples\data\jdll.h.

Note that methods Show and Log are ignored by a JDLLServer.

Once the J servers have been referenced, their methods become available for use by the client. You should declare the J server as an object in the client, and can then reference that object in order to access the J server methods.

J OLE/OCX Client

J interacts with OLE and OCX controls using the Window Driver. You need a parent window to display the control, and you then create and use a OLE/OCX control in much the same way as for any other child control. You can use both standard and OLE/OCX controls on the same form.

```
wd 'cc tree ocx:comctl.treectrl.1'  
wd 'cc xl oleautomation:excel.application'
```

OCX and OLE controls are not distributed with J. Some OCX and OLE controls are included with Windows, and with software such as Visual Basic. In some cases, these are subsets or earlier versions of the full systems obtainable from the original manufacturer. For serious use, we recommend you purchase the most recent versions of the controls, which will then include full documentation and support.

To see which OCX controls are available, create a new form using the Form Editor, then click New for new control, and select class OCX.

OLE controls are not supported by the Form Editor and should be added directly in the `form_run` verb. The names of the OLE controls that are available are not visible from J.

An OCX or OLE control is a child on a form that contains an object named *base*. This base object can contain other objects. For example the base object could contain a font object. An object is identified by the child id and the name of the object. The name *temp* refers to a contained object that has been returned by a method or property. The temp object is automatically released and reset whenever a new object is returned. The `oleid` command gives the temp object a name and makes it permanent.

Each control has a specific list of *methods*, *properties* and *events* (for OCX controls). The properties define the way the control is set up. The events are the events that the OCX control can generate. When you create a control, you typically set various properties, and register various events that you want to be notified of. When the control is in use, you retrieve information from the control by either reading the value of its properties, or by receiving event notifications.

wd commands for OLE/OCX controls

`oledlg id`

Run the OCX property dialog. The state can be saved with the `olesave` command.

`oleenable id eventname [bool]`

Enable/disable OCX event. You must enable an event in order to trigger an event in J.

`oleget id objectname property`

Return value of a property. Objectname is *base*, *temp*, or a name set with `oleid`. If the result is an object, it is set as the temp object. This allows a series of wd commands that use the temp object to get the next object.

`oleinfo id`

Return information about events, methods, properties, and constants.

oleload id filename

Initialize properties from a file created by olesave. An oleload should only be done once before it is shown.

olemethod id objectname method parameters....

Run a method.

, is an elided parameter. A wd parameter of , is the same as " ", except it is treated as an elided parameter where appropriate.

Some methods distinguish between a numeric parameter and a string. A simple (not delimited) string that is an integer is treated as an integer. If you want 23 to be treated as a string, use "23".

If the result is an object, it is set as the temp object.

An object parameter is indicated by a simple parameter of the form:

object:formid.childid.objectname

A picture object parameter is indicated by a simple parameter of the form:

picture:filename

olesave id filename

Save properties in a file that can be used to initialize a control after it is created.

oleset id objectname property value

Set property value.

Objectnames

An OLE or OCX base object can contain other objects.

For example, many OCX controls contain a Font object that is returned by the font property.

```
wd 'oleget ocx base font' get font object as temp
```

```
wd 'oleget ocx temp fontsize' get font size
```

8.5

```
wd 'oleget ocx temp fontname' get font name
```

```
MS Sans Serif
```

```
wd 'oleset ocx temp MS Sans Serif;oleset ocx temp fontsize 20'  
set font
```

OCX Events

An OCX event is signalled as a button event. For example, form abc, OCX id spin, signals sysevent abc_spin_button. The name sysocx is assigned the OCX event name.

Example: Spinbutton Control

To illustrate these commands, we use the Outrider Systems SPIN32.OCX, a simple control that is ideal for experimentation. We assume you have this control installed on your system.

First try reading the OCX information (if this fails, you do not have the SpinButton installed). This returns a text description of properties, methods and events:

```
    load 'packages\ocx\ocxutil.js'           load OCX utilities
    ocxinfo 'spin.spinbutton'
```

NB. event: SpinDown
NB. prototype: void SpinDown ()
NB. help: Occurs when the user clicks one of the arrows...

NB. event: SpinUp
...

Next, load the spinbutton demo, as follows:

```
    load 'examples\ocx\misc\spin.js'
```

Click on the spinbutton to change the text field. Click on the About button to display the about box from Outrider, and the Dialog button to view and change the OCX properties.

Note that to see events from the spinbutton, you use the `oleenable` command when initializing the form, i.e.

```
spin_run=: 3 : 0
wd SPIN
showpay''
wd 'oleenable sx spindown'
wd 'oleenable sx spinup'
wd 'pshow;'
)
```

The About button invokes the OCX's `aboutbox` method:

```
spin_about_button=: 3 : 0
wd 'olemethod sx base aboutbox'
)
```

The Dialog button invokes the `oledlg` command:

```
spin_dialog_button=: 3 : 0
wd 'oledlg sx'
)
```

A press of the spinbutton, invokes the spinbutton handler:

```
spin_sx_button=: 3 : 0
if. sysocx-:'spinup' do.
    PAYNDX=: 4 |>:PAYNDX
else.
```

```
PAYNDX=: 4 |<:PAYNDX
end.
showpay''
)
```

This handler checks the value of `sysocx` and adjusts the text field accordingly.

Examples

Example: Graph control

File: examples\ocx\misc\graph.js has a simple example using Pinnacle's GRAPH32.OCX control.

For a fuller example:

```
load 'examples\ocx\graph\grafdemo.js'
```

Example: TreeView

This example assumes you have the TreeView OCX installed and assorted bitmaps.

Run the script examples\ocx\misc\tree.js to create a treeview control. An imagelist control is loaded with various bitmaps which are then used by the treeview control.

Example: Controlling Microsoft Word

It is possible to automate the word.basic object. This could be a useful way of solving some printing problems. The following loads Word, and reads in the J readme.wri file:

```
load 'examples\ocx\misc\word.js'
```

Example: Excel OLE Automation

```
wd 'pc excel'                parent to hold ole child
wd 'cc x oleautomation:excel.application' create excel.application object
wd 'oleset x base visible 1' set base visible property to 1
wd 'oleget x workbooks'      get workbooks object as temp
wd 'oleid x workbooks'       temp to permanent as workbooks
wd 'olemethod x workbooks open foo.xls' open workbook foo.xls
wd 'olemethod x workbooks add' run workbooks add method
                               add method returns temp as the new workbook object
wd 'oleget x temp name'      get workbook name
wd 'olemethod x base quit'   quit
```

The excel.application object is defined in the system registry to create a new instance (private copy) of excel. The excel.sheet object creates a new worksheet in a shared excel.

Run the script and then run excel". It is possible to access properties and methods directly, as well as run macros. The getobject verb shows how to support some of the simpler syntax of VB. Run excelquit" to close excel.

Tutorial: J OLE Server for Excel

This tutorial has been created with Excel 97 and J401. Expect different behavior with other versions!

It will help to have the Excel VBA help file VBAXL8.HLP (or similar) readily available.

Before running it, copy the file jsutils.xls from J directory system\examples\ole\excel to your Excel directory.

Introduction

The purpose of OLE Automation is to allow a client program to run functions in a server program, and the basic idea is pretty straightforward - simply load J from Excel, then send it the required J functions for execution. In practice it is helpful to create Excel macros that provide cover functions for the basic tasks such as loading J, reading cells for transmission to J and so on. Thus you typically program with a mixture of J functions and Excel macros.

Functions provided by an OLE Server are referred to as *methods*, see J OLE Automation Server

In Excel, you can enter these method names in upper or lowercase. When you enter names in Excel, it gives them its default capitalization. Here we use lowercase throughout.

Troubleshooting

You are going to be working with both J and Excel sessions active. It will be helpful to close down other applications to minimize screen clutter.

As you use OLE, commands sent from Excel may change the active focus to J. To enter new commands in Excel, click on the Excel session to change the active focus back to Excel.

Most of the time when things go wrong, you can simply shut down J and Excel and start again. Sometimes, the J server has been loaded but is not visible. You can check this by pressing Ctrl+Alt+Del simultaneously, which brings up the list of current applications loaded. If necessary, select J and click End Task.

Sometimes when you edit Excel macros, Excel closes down J - it closes the OLE Automation object which may in turn cause J to close. You will then need to re-open the J OLE Automation object. If J has closed and you try to run an OLE command, the error message is "Object variable not Set". This problem occurs only while you are developing Excel macros, and should not occur when your application is in use.

If you use the utilities in jsutil.xls, there will be no problem in trying to re-open the J OLE Automation object even if it is already open.

One of the "user-friendly" features of Excel is to change your entry in a cell if it thinks it may be incorrect. For example, "i.5" gets changed into "I.5". To get around this, enter more letters, then backspace and delete the extra entries, for example, instead of "i.5" try entering "ii.5".

Tutorial

Start by unloading all applications, then loading Excel. Arrange the window so that it covers only about half the screen. Open a new workbook if none is shown.

Bring up Visual Basic (Alt-F11 or Tools|Macro|Visual Basic...), and insert a new module sheet (Insert|Module).

With the module sheet visible, select menu item Tools|References and check both J DLL Server and J EXE Server, and click OK. In practice you need only check the server that will be used; also, if you use the jsutil.xls utilities described below, this reference will be done for you.

In the module sheet, enter:

```
Public js As Object
```

```
Sub jopen()  
Set js = CreateObject("jexeserver")  
End Sub
```

The first statement declares the name `js` that will be used for the JEXEServer. The function `jopen` will be used to load the JEXEServer. Note that you can only run this once - you will get an error at this point if you try to open the server twice.

Loading J

Next open up the Immediate window for experimentation (if not already open). To do so, select menu item View|Immediate. You can enter a series of commands in this Window - when you press Enter, Excel runs the command in the line where the cursor is.

To load J (it will not be visible), enter:

```
jopen
```

To show J, enter:

```
js.show 1
```

This means: run the `show` method of `js`, i.e. of the JEXEServer, with argument `1`.

The J OLE Automation Server should now be visible. Maximize the `ijx` window within the J session, then arrange the windows so that both Excel and J are visible. Note that not only is the J Server visible, but if you click on it to give it focus then you have full access to the regular J development system.

In the Excel Immediate window, experiment with show:

```
js.show 0          this hides the window  
js.show 1          this shows it again
```

Next set on logging - this tells J to display commands sent by Excel in the J window:

```
js.log 1
```

Sending commands to J

The required function is do, which takes a J sentence as its argument. Note that Excel strings are delimited by the double quote, so that J quotes can be entered as is, and need not be doubled. Try:

```
js.do "i.4 5"
```

```
js.do " 1!:1<'c:\autoexec.bat' "
```

You should see the statements and results in the J window.

Retrieving values from J

The function get retrieves a value from J, as a Variant datatype. Variants cannot be displayed directly in the Immediate window, but can be assigned to a worksheet range. For example:

Set value of x in J:

```
js.do "x=: i.4 5"
```

Retrieve value of x into Excel variant y:

```
js.get "x",y
```

Set value of y into the worksheet:

```
Worksheets("sheet1").Range("a1:e4")=y
```

Now close the Immediate window and switch to Sheet1 to see that the value of y has been written in.

Utilities

Now lets take a look at the J OLE utilities in jsutil.xls.

Close Excel (no save) and the J OLE Automation server. Reload Excel and open a new workbook if none is shown. Use Visual Basic, Insert|Module to create a new module as before, then select Tools|Reference and add jsutil.xls (you will likely have to Browse to the file).

Note that jsutil.xls should already reference the J Servers, so you do not need to reference them specifically in any workbook that includes this as a module. You might confirm that jsutil.xls does indeed reference the J Servers.

The utilities available are:

jlopen	open JDLLServer
jxopen	open JEXEServer
jcnd (string)	execute J command, return result as variant
jcndc string,r,c,h,w	execute J command, store result in active sheet at row,col,height,width
jcndr string,range	execute J command, store result in active sheet at range
jdo string	execute J command
jget(x)	get J noun x

jloadprofile	load standard J profile
jlog boolean	log on/off (EXE only)
jshow boolean	show on/off (EXE only)

You can customize these or add your own utilities.

Loading J automatically

In the Module, enter an auto_open subroutine as follows:

```
Sub auto_open()
jxopen
jshow 1
jlog 1
End Sub
```

This sub will be run each time this workbook is opened. It opens the JEXEServer, shows the J session and logs commands sent from J.

In practice, only the first command jxopen is needed, and if you are using the JDLLServer instead, you would need only:

```
Sub auto_open()      (do not enter this now!)
jdopen
End Sub
```

Now put the cursor on the name auto_open and press F5 to run it. The J session should display.

Now check that auto_open works correctly when you load the book. Switch back to Excel, save the book as test.xls and close Excel - note that the J session will close as well. Reload Excel, and open test.xls - you should see the J session again. Arrange the windows so that both Excel and J are visible.

jcmd

In Excel, switch to Sheet1 and in cell B3 enter:

```
=jcmd("/+2 3 5 7")
```

(You may find it more convenient to enter this in the Formula Bar, rather than directly in the cell.)

The statement should be executed in J, and the result (17) displayed in Excel.

Try:

In cell B5 enter: 12

In cell B6 enter: 15

In cell B7 enter: =jcmd(B5 & "*" & B6)

B7 displays the result (180). Note that if you now change B5 or B6, then B7 will be recalculated.

In general, `jcnd` can be used for calculations which return a single value to be displayed in the current cell. The right argument is the sentence to be sent to J.

This method is really only suitable for simple calculations. Typically, you will want to run calculations that return a range of results to Excel and you set up such calculations by invoking an Excel macro explicitly, for example, by selecting Tools|Macro|Run or pressing an assigned hot-key, or else by setting the `OnEntry` property for the worksheet.

jcndc, jcndr

These utilities execute a J expression, displaying the result in a range in the active sheet. Function `jcndc` specifies the range as 4 numbers: topleft row, column, number of rows, number of columns. Function `jcndr` specifies the range in the traditional alphanumeric notation, for example: C6:E10.

We will create a macro run to test these and subsequent expressions. Switch to the module and enter:

```
Sub run()  
jcndc "?3 4$10", 2, 3, 3, 4  
End Sub
```

Next, return to the worksheet, select Tools|Macro, highlight `run` and click Options. Enter `Ctrl-r` as the shortcut key and click OK. Close the Macro dialog, switch to Sheet1 and press `Ctrl-r`. The macro should run and display the result. Press `Ctrl-r` again to re-run the macro.

jsetc, jsetr

These utilities set values in J, from a range in the active sheet. As with `jcndc` and `jcndr` above, `jsetc` specifies the range as 4 numbers and function `jsetr` specifies the range in the traditional notation. Switch to the module and edit `run` to:

```
Sub run()  
jsetr "Y", "D3:F4"  
End Sub
```

Switch to Sheet1 and as before use Tools|Macro to select `Ctrl-r` as a shortcut key for the macro. In the worksheet, press `Ctrl-r`. Then click on the J session and display Y (these are random numbers so the exact values will likely differ):

```
Y  
+--+--+  
| 4 | 8 | 8 |  
+--+--+  
| 7 | 3 | 1 |  
+--+--+
```

OnEntry

Change `run` to:

```
Sub run()  
jsetc "Y", 2, 3, 3, 4
```

```
jcmdc "+^>Y", 7, 3, 3, 4  
End Sub
```

Switch to Sheet1, use Tools|Macro to select Ctrl-r as the shortcut key, then in the worksheet, press Ctrl-r. The macro will read the numbers in the upper range and display the sum scan in the lower range. Now if you change one of the numbers in the upper range, for example E2, you must press Ctrl-r to update the lower range. To get Excel to update the lower range automatically, create and run a function that sets the OnEntry property. In the module, enter:

```
Sub setentry()  
Worksheets("sheet1").OnEntry = "run"  
End Sub
```

Put the cursor on the function name setentry and press F5 to run it. Switch back to sheet1 and try changing values in the upper range - the lower range will be automatically re-calculated.

JDLL

The commands described work the same way using the JDLL. To experiment, switch to the module, comment out auto_open and enter a new version as:

```
Sub auto_open()  
jdopen  
setentry  
End Sub
```

Next close Excel (saving changes), re-open it and load the test.xls workbook. Switch to Sheet1 and try changing values in the upper range.

Tutorial: J OLE Client to Excel

This tutorial has been created with Excel 97 and J401. Expect different behavior with other versions!

It will help to have the Excel VBA help file VBAXL8.HLP (or similar) readily available.

When you create a new book in Excel, by default the book has 16 worksheets. Since we will experiment with adding worksheets under program control, we suggest that you change the default to 1 worksheet (use the Tools|Options|General dialog).

Introduction

In theory, any Excel function can be called directly from J. In practice, some Excel functions have an unusual syntax that is either awkward or impossible to call from J, for example the ChartWizard method. However, you can always get around this by creating a corresponding Excel macro that you can call from J. Moreover it makes sense to use Excel macros anyway - there is no point in trying to duplicate in J a series of Excel function calls that could be just as easily, or more easily, programmed in Excel.

Thus you typically program with a mixture of J functions and Excel macros.

Excel Hierarchy

The various parts of Excel such as the Workbooks, Worksheets and Charts (all known as *objects*) are organized in a hierarchy. Objects have *methods* (functions) and *properties* (variables). References to Excel objects, methods and properties must include their position in the hierarchy, for example:

```
Application.Workbooks("Book1").Worksheets("MySheet").ChartObjects.Item(1).Chart.PlotArea.Width
```

Now this naming convention gets a little tedious to enter, so Excel allows you to simplify it a little. For example, if MySheet happens to be the active sheet, you could instead use:

```
Activsheet.ChartObjects.Item(1).Chart.PlotArea.Width
```

J does not support this method of referencing names. Instead, for each reference you provide two names - the first being the position in the object hierarchy. Thus if the name abc represented Activsheet.ChartObjects.Item(1).Chart.PlotArea then the equivalent J reference would be:

```
abc width
```

How do you assign names in J to positions in the object hierarchy? To start off with, there are two reserved names. The name `base` represents the root of the hierarchy, equivalent to `Application` in Excel. Thus the following are equivalent:

<code>base visible</code>	J
<code>Application.Visible</code>	Excel

The name `temp` is assigned to the current position in the hierarchy. For example, if you have just created a new worksheet `Sheet1`, then the following are equivalent:

<code>temp activate</code>	<code>J</code>
<code>Worksheets("Sheet1").Activate</code>	<code>Excel</code>

Next, at any point, you can assign a name to the `temp` position. Thus if you assigned the name `sh1` to `temp` at this point, you could then use:

```
sh1 activate
```

The idea is that you assign names to positions that you expect to revisit, while `temp` can be used for positions that you are just passing through.

Note that Excel names are not case-sensitive, but when programming, Excel automatically converts your entries to its standard capitalization. From `J`, you can use any case, and here we use lowercase throughout.

Troubleshooting

You are going to be working with both `J` and `Excel` sessions active. It will be helpful to close down other applications to minimize screen clutter.

As you use OLE, commands sent from `J` may change the active focus to `Excel`. To enter new commands in `J`, click on the `J` session to change the active focus back to `J`.

If the OLE link goes wrong somewhere, you can simply close down the `Excel` session, and reset the `J` session. The `J` OLE interface uses the Window Driver, so you should enter `wd 'reset'` to reset it. Of course, normally you would shut down `Excel` and reset `J` under program control.

Sometimes you send a command to `Excel` that appears to hang up, while the `Excel` session flashes. This happens when `Excel` displays a dialog box that requires user intervention, for example an error message or a prompt to save changes on exit. In such cases, switch to `Excel` and respond to the dialog box before continuing.

At other times, `Excel` will hang up when it is waiting for user entry to be completed. For example, if you highlight a cell and start editing its contents, then switch to `J` and try an OLE command, the system will hang until you go back to `Excel` and complete the cell editing.

While `Excel` is fairly efficient at the tasks you are likely to use it for, you might inadvertently give it a task that takes a long time. For example, suppose you create a chart from data in a spreadsheet, then send a command from `J` to update that data. After each cell is updated, `Excel` will re-draw the chart - as many times as there are cells! While this happens, everything is locked up, and you will have to wait, or shut `Excel` down. (This particular problem is solved by erasing the chart before you update the data, then re-creating it after the update.)

Tutorial

Start by unloading all applications, then loading `J`. Maximize the `ijx` window in the `J` session, then arrange the `J` session window so it covers only about half the screen.

Opening up Excel

Create a parent to hold the Excel OLE Automation control:

```
wd 'pc xlauto'
```

Create the Excel OLE Automation control. This may take a few seconds, because it loads Excel into memory (it will not be visible).

```
wd 'cc xl oleautomation:excel.application'
```

All J OLE commands from now on will refer to the `xl` control. Note that the names `xlauto` and `xl` used here are not required - you can use your own names. However, the utilities included with J also use these names, so it is recommended that you stick with them. Note also that `xl` is short for Excel and not 'x', 'one'.

At this point, Excel has been loaded, but is not visible. Excel has a `Visible` property that can be set to display it. This property is part of the `Application` object, and hence the Excel call to use it would be:

```
Application.Visible = 1
```

In J, the `Application` object is named `base`, therefore to set it, enter:

```
wd 'oleset xl base visible 1'
```

This means: execute `oleset` on control `xl`, setting the `visible` property of `base` to 1.

If Excel opens full screen, shrink it down so that both the J and Excel windows are visible.

Now Excel is visible, but has no workbook open. To create a new workbook in Excel, you use the `Add` method of the `Workbooks` object. Note that `Add` is a method of the `Workbooks` object (as well as several other objects), but is it not a method of the `Application` object - `Application.Add` will not work! Therefore the first step in J is to get access to the `Workbooks` object. To do so, enter:

```
wd 'oleget xl base workbooks'
```

This command should complete successfully, but display no result. However, internally, J has assigned the `Workbooks` object to `temp`, and this can now be used to invoke the `Add` method:

```
wd 'olemethod xl temp add'
```

This should have created a new workbook. Try entering it again to add another workbook:

```
wd 'olemethod xl temp add'  
|domain error  
| wd'olemethod xl temp add'
```

This time you get a domain error - try: `wd 'qer'`. What happened is that the `temp` name really is temporary - it refers to the current position in the Excel hierarchy, which is constantly changing as you move about Excel. In this case, when you added the new workbook, `temp` changed to that workbook - which does not have an `Add` method!

Therefore, in order to add another workbook, you have to assign `temp` to `Workbooks` again:

```
wd 'oleget xl base workbooks'  
wd 'olemethod xl temp add'
```

Of course, this quickly becomes tedious - therefore the proper treatment here is to assign a name to the `Workbooks` object, so that you can just use that name in future. To do so, use the `oleid` command:

```
wd 'oleget xl base workbooks'  
wd 'oleid xl wb'
```

Now you can use `wb` to create several books:

```
wd 'olemethod xl wb add'  
wd 'olemethod xl wb add'  
wd 'olemethod xl wb add'
```

Closing Excel

Now lets try closing down Excel. The `Application` object in Excel has the `Quit` method to close down. `Quit` will prompt, should there be any unsaved changes. Try switching to Excel, then entering some values into one of the spreadsheets. Ensure that you have completed your entries (press `Enter` if Excel is waiting for you to complete the entry of a cell), then switch back to `J` and enter:

```
wd 'olemethod xl base quit'
```

Excel will start flashing, and if you try to enter anything in the `J` window, it eventually displays a "Server Busy" dialog box. Click on Excel, and respond to the "Save changes in 'Book'?" prompt. Eventually, Excel will close. You should now reset the `J` Window Driver with:

```
wd 'reset'
```

Utilities

This is a good time to look at the Excel OLE utilities, to do so enter:

```
load 'system\examples\ole\excel\xlutil.ijs'  
names ''
```

This defines several utilities, the main ones being:

<code>xlopen</code>	create Excel OLE automation object
<code>xlshow</code>	show/hide Excel OLE automation object
<code>xlexit</code>	exit Excel OLE automation object (saves)
<code>xlget</code>	cover for <code>oleget</code> - get object

xlset	cover for oleset - set object parameter
xlcmd	cover for olemethod - invoke method
xlid	cover for oleid - assign id to current position
xlread	read cell
xlreadr	read range
xlwrite	write cell
xlwriter	write range
xlsetchart	set chart range

The verb `xlopen` opens up Excel. It:

- creates the parent window `xlauto`
- creates the Excel OLE automation control `xl`
- names `wb` as the Workbooks object
- loads the macro file `examples\ole\excel\jmacros.xls` (which is hidden)

Try:

```
xlopen ' '
```

Note that Excel is not shown, indeed you may want to use Excel without it ever being visible. To make it visible, enter:

```
xlshow ' '
```

Verb `xlcmd` runs an OLE method. Since `wb` has been named in `xlopen`, it can be used directly. To add a workbook:

```
xlcmd 'wb add'
```

Take a look at the workbook name:

```
xlget 'temp name'
Book1
```

Try changing the workbook name:

```
xlset 'temp name Mybook'
|domain error
| xlset'temp name Mybook'

wd 'qer'
ole - Workbook does not have writeable Name property : 12
```

What is happening is that in Excel, you can only change the name of a workbook by saving it. Thus, the following saves the workbook, and also renames it:

```
xlcmd 'temp saveas Mybook'
```

This may return -1, which really is the result from Excel!

(If you already have saved Mybook, Excel will prompt you to overwrite it.)

Accessing the Worksheet

To access the worksheet, we first have to get the Worksheets object, which belongs to the workbook. We will use the Worksheets object a few times, so will give it a name:

```
xlget 'temp worksheets'  
xlid 'ws'
```

We can try adding new worksheets:

```
xlcmd 'ws add'  
xlcmd 'ws add'  
xlcmd 'ws add'
```

Next we access the first sheet using the Item method, and assign the name sh1:

```
xlget 'ws item sheet1'  
xlid 'sh1'  
xlget 'sh1 name'  
Sheet1
```

Be careful to distinguish sh1 which is the name used by J for a position in the Excel object hierarchy, from Sheet1, which is the name used by Excel for the current worksheet. You can change the worksheet name:

```
xlset 'sh1 name Mysheet'
```

If this worksheet is hidden behind another (which will be the case if you followed the above steps exactly) you can activate it with:

```
xlcmd 'sh1 activate'
```

Now lets try writing to a specific cell. In Excel you can use cell references of the form 2 3 or old-style alphanumeric references such as B3; the former are easier to program. First reference a cell, using the Cells property:

```
xlget 'sh1 cells 2 3'
```

Then set the value of temp as required. The new value should appear in the spreadsheet:

```
xlset 'temp value 123'  
  
xlget 'temp value'  
123
```

Reading and Writing Ranges

In practice, you will typically want to read and write a range of cells. It would be tedious to do so one cell at a time; unfortunately, the form in which Excel reads and writes range data is not available to J. The solution is to use the utilities xlreadr (read range) and xlwriter (write

range) that call appropriate macros from jmacros.xls. The right argument is the workbook, worksheet, topleft cell position and number of rows and columns. The left argument of `xlwriter` is the data to be written. Try:

```
(i.3 4) xlwriter 'mybook.xls mysheet 2 2'
```

Verb `xlreadr` returns data as a boxed array of character strings:

```
xlreadr 'mybook.xls mysheet 3 3 2 3'  
+----+----+  
|5|6 |7 |  
+----+----+  
|9|10|11|  
+----+----+
```

```
". &> xlreadr 'mybook.xls mysheet 3 3 2 3'  
5 6 7  
9 10 11
```

Finally, use `xlexit` to close Excel (you may be prompted to save):

```
xlexit''
```

Data passing

Data parameters sent using these utilities are limited to 65K, which suffices for most purposes. The best way to pass data longer than this is via a temporary file. Thus J can write a file then send an OLE command to Excel to read it.