

SPREZZATURA

45 St Mary's Road • Ealing • LONDON W5 5RQ

Tel: 020 8912 1010 Fax: 020 8912 1011

info@sprezzatura.com

SENL

MARCH 2007

Sprezzatura's Electronic Newsletter

For Revelation developers by Revelation developers

MAKING DATABASES HAPPEN

SPREZZATURA

Contents

- WELCOME..... 3**
- COMPILER ENHANCEMENTS – ANDREW P MCAULEY 5**
 - Displaying comments.....8
 - Saving expanded code10
 - Ensuring comments remain in Outputted Code11
 - Squeezing the last ounce of performance from a routine.....12
 - Ensuring Repository Tracking13
 - Multiline Concatenation/Nested inserts15
- WINDOWS COMMON (AND NOT-SO-COMMON) CONTROLS AND OPENINSIGHT – CARL PATES 16**
 - WINCONTROL and the Windows ClassName.....16
 - The OI CLASSNAME property.....17
 - Window Messaging17
 - Creating a WINCONTROL control.....18
 - Using a WINCONTROL.....18
 - Wrapping a WINCONTROL.....19
 - Example wrapper - zzx_ProgressBar()..... 20
 - Finding out more21
- MAIL MERGING FROM ADVANCED REVELATION – ANDREW P MCAULEY 22**
 - Creating the data file to merge from22
 - Creating the merge document.....22
 - Record the actual merge macro26
 - Drive the production of the Merge from AREV28
- CELEBRITY INTERVIEW - BRYAN SHUMSKY - REVELATION R&D – SENIOR DEVELOPER..... 29**
- ZZ_IDE_DUMP – A NEW UTILITY – AARON KAPLAN..... 31**
- AN INTRODUCTION TO THE USE OF OLE CONTROLS FROM OPENENGINE – PART 2 - ANDREW MCAULEY35**
 - Exploring Using Macros35
 - ChangeFileOpenDirectory37
 - Documents.Open39
 - ActivePrinter.....40
 - Application.Printout41
 - Performing a Mail Merge.....41
 - Starting the OLE Application.....43
 - Conclusion52
- CHANGING UNIVERSAL DRIVER FILES TO 2.1 FORMAT – AARON KAPLAN 55**
- PERIPHERAL TRIVIA 56**
 - COPYRIGHT NOTICE56

SPREZZATURA

Welcome to this issue of SENL! We know it'll be a slight shock getting this quote "so soon after the last issue" but we had so many requests to get the OLE article finished and so many cool things have been happening that we thought we'd get this out quickly and then sit back and review how we're going to proceed with the journal from hereon in.

We're not long back from conference and as has been reported this was one of the most successful conferences ever. The location was fantastic – Seattle has a wealth of things to interest people like myself who are interested in decent tasting beers – and the T-Shirt stores give Sacramento a run for its money. (Though Sacramento wins by a nose). Breakfast at Pike's Place Market is such a cool way to start the day and the range of good food places was awesome – "awesome like a hot dog" to quote an English comedian. ... "And your President's going to be going (American voice) "Can you tell me, astronaut, can you tell me what it's like?" "It's awesome, sir." "What, like a hot dog?" "Like a hundred billion hot dogs, sir." – Eddie Izzard.

The conference itself had a range of technical information ranging from the introductory through to the seriously technical. This time it seemed that people had done their homework more on the sort of presentations that they'd like to see and unlike in previous years no one was seen to walk out of a technical track complaining that the presentation was too technical.

The social side of the conference once again added extra value as people we hadn't seen for years appeared. It's always fun hanging with our peers and this conference was no exception. The themed party this year provided an excuse for a noisy clique of Sprezz bods and ***** bods (names changed to protect the guilty) to hang around practicing their cockney rhyming slang and out joking each other. By the end of the evening onlookers were drawn to the group like smaller planets being sucked into a black hole. We haven't laughed so much in a long time – thanks to those who helped make the evening and all y'all know who you are ☺

We've had a number of requests to publish our proceedings publicly from people who weren't able to attend the conference for various reasons – for these people we have some bad news and some good news!

The bad news is that it is policy that Conference Presentations aren't released to non-attendees until some considerable time has elapsed since the show. This makes sense if you think about it. The presentations are created for the show – that's the driving force behind their production. People take the time and trouble to attend the show to get this information. The show costs money and a great deal of time to put on and as a rule doesn't even break even for Revelation. The less people who go the more of a loss is made. If you didn't need to attend to get the materials then the show would make even more of a loss as more delegates stayed away. This would probably lead to cancellation of sponsorships and ultimately cancellation of the show. In turn this means that no one would prepare presentations – so everyone would lose out!

The good news is that Revelation have given us permission to document the meat of one of the most technical presentations of the conference – Carl's presentation on using Windows Common Controls in OI 7.1+. The reason for this is that delivering "Windows Standard" software is necessary in today's climate and people expect certain features.

SPREZZATURA

With Carl's help you'll be able to spruce up your applications in no time! So a big vote of "Thanks" to Revelation for their permission on this!

As I type we've just started the beta cycle for version 8.0 of the product and this truly is a landmark release for a LOT of people. One of the most significant inclusions is going to be AREV32 – a console app version of AREV but with all of the 16 bit DOS limitations removed. So if you've got a client who just WOULDN'T move to OpenInsight because of the work involved you can finally move them without changing them! Put simply, AREV32 uses OpenInsight's "CTO" technology to pretend to be an AREV app running under OI. That immediately removes the 64K limit and the EMS problems! There is very little work involved in actually moving to AREV32 and more details will be made available as we get closer to the release date. Suffice to say for now that your clients will have to buy Developer Class licenses of OI and that they'll then be running AREV within OI. That ideally positions them for a phased transition to OpenInsight!

To celebrate the introduction of AREV32 we've decided to include an AREV tip to complement our OLE article – multiple feline fur removers and all that. This is a cute little trick which allows you to perform mail merges from AREV using Word for Windows... didn't Cogent do something with WordPerfect like that all those years ago? Plus ca change plus ca la meme chose...

The toolset has been extended to take advantage of the addition of Drag and Drop into the product and finally now that we're in Beta Carl is free to catch up on other projects. And we've got a LOT of partially finished projects here at Sprezz Towers... one of these is showcased in Aaron's article on our new Linear Hash Dump utility. We're just working out new licensing packages and expect to be releasing a slew of free software into the Rev community by summer.

We're starting to hear rumours of a UK Revelation Conference to be held in London in May. This will only be a single day and a single track apparently but it is hoped to have at least 5 presentations – at least 3 of which will be seriously technical. So if any of our global readership are planning a London holiday this year think about May!

It would seem that not everyone is on the Revelation EMEA newsletter mailing list because at the Sprezz presentations in Seattle people kept asking about code snippets such as \$USES. These compiler enhancements were the subject of a technical article in the August '06 newsletter. As the authors of the original piece we've expanded it slightly for this SENL but you might want to check out the Newsletter back issues in case there's anything else you've been missing! They can be found at <http://www.revsoft.co.uk/devnewsletters.htm>.

In closing it will doubtless come as a huge surprise to you after reading this SENL that we've finally gotten around to buying SnagIt for screen dumping – what a cool tool! So our apologies in advance if we've become a little "snap happy".

Enjoy this SENL we truly hope that you'll find something in it that is of value to you.

Regards

Your friends at Sprezz

SPREZZATURA

Compiler Enhancements – Andrew P McAuley

The compiler is the compiler is the compiler and has been pretty much the same since the introduction of Revelation B. Sure along the way new opcodes have been added to support new system common and new at variables but it has remained pretty much the same until very recently! The release of 7.2.1 brought with it a number of compiler enhancements which serve to make life easier for the serious developer. This article documents these changes.

Precompiling

The introduction of the CTO provided the much needed excuse to enhance the compiler with precompiling technology. This is something that we've been asking to have included in the compiler since the year dot. Our motives are a tad obscure – reasoning it would be fun to allow people to compile in their native language for example. However the CTO provided a practical excuse for its inclusion.

Put simply precompilation allows the developer to stipulate that a program (or programs) be run before the main compiler is run. This allows the developer to make changes to the program before it is compiled. As a simple example of what could be achieved, the developer could write a program that translated French words into their English equivalent before compiling.

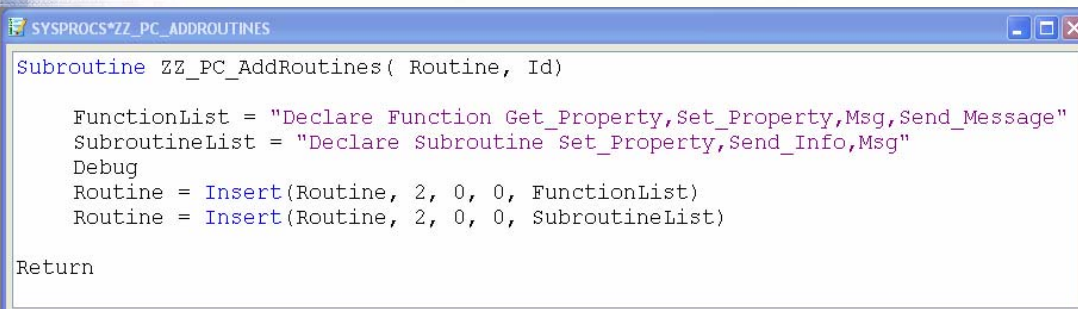
A new row has been added into SYSENV called CFG_PRECOMPILER. This is a value mark delimited list of programs to call as a precompiler. Note that there is no practical limit to how many such precompilers can be called but bear in mind that once the programs are entered into CFG_PRECOMPILER they will be called for the compilation of *all* events and programs.

If you'd like to be a little more selective in when and if you call your precompiler you may make use of the new PRECOMP #PRAGMA predirective. This allows you to specify that a specific program should be called as a precompiler when compiling this routine or event.

The syntax is

```
#PRAGMA PRECOMP PROGRAMNAME
```

In all cases the precompiler should be written to accept two parameters. The first is the source code itself and the second is the name of the routine being compiled. So if you decided that you were bored of always having to declare certain functions and subroutines you could write a precompiler such as this :-



```

SYSPROCS*ZZ_PC_ADDROUTINES
Subroutine ZZ_PC_AddRoutines( Routine, Id)

  FunctionList = "Declare Function Get_Property,Set_Property,Msg,Send_Message"
  SubroutineList = "Declare Subroutine Set_Property,Send_Info,Msg"
  Debug
  Routine = Insert(Routine, 2, 0, 0, FunctionList)
  Routine = Insert(Routine, 2, 0, 0, SubroutineList)

Return
  
```

SPREZZATURA

Note that the debug is there just for the purposes of this article! So now let's write a routine to make use of this precompiler :-

```
SYSPROCS*DELETEME
Subroutine DeleteMe(Void)

    #PRAGMA PRECOMP ZZ_PC_AddRoutines

    A = Get_Property("EDITOR", "CLASSNAME")
    Msg(@Window, A)

Return
```

And let's compile this routine – which because of the above debug will drop us into the debugger.

The screenshot shows the OpenInsight Debugger interface. The 'Source' window displays the following code:

```
00001: Subroutine ZZ_PC_AddRoutines( Routine, Id)
00002:
00003:     FunctionList = "Declare Function Get_Property,Set_Property,Msg,Send_Message"
00004:     SubroutineList = "Declare Subroutine Set_Property,Send_Info,Msg"
00005:     Debug
00006:     Routine = Insert(Routine, 2, 0, 0, FunctionList)
00007:     Routine = Insert(Routine, 2, 0, 0, SubroutineList)
00008:
00009:     Return
```

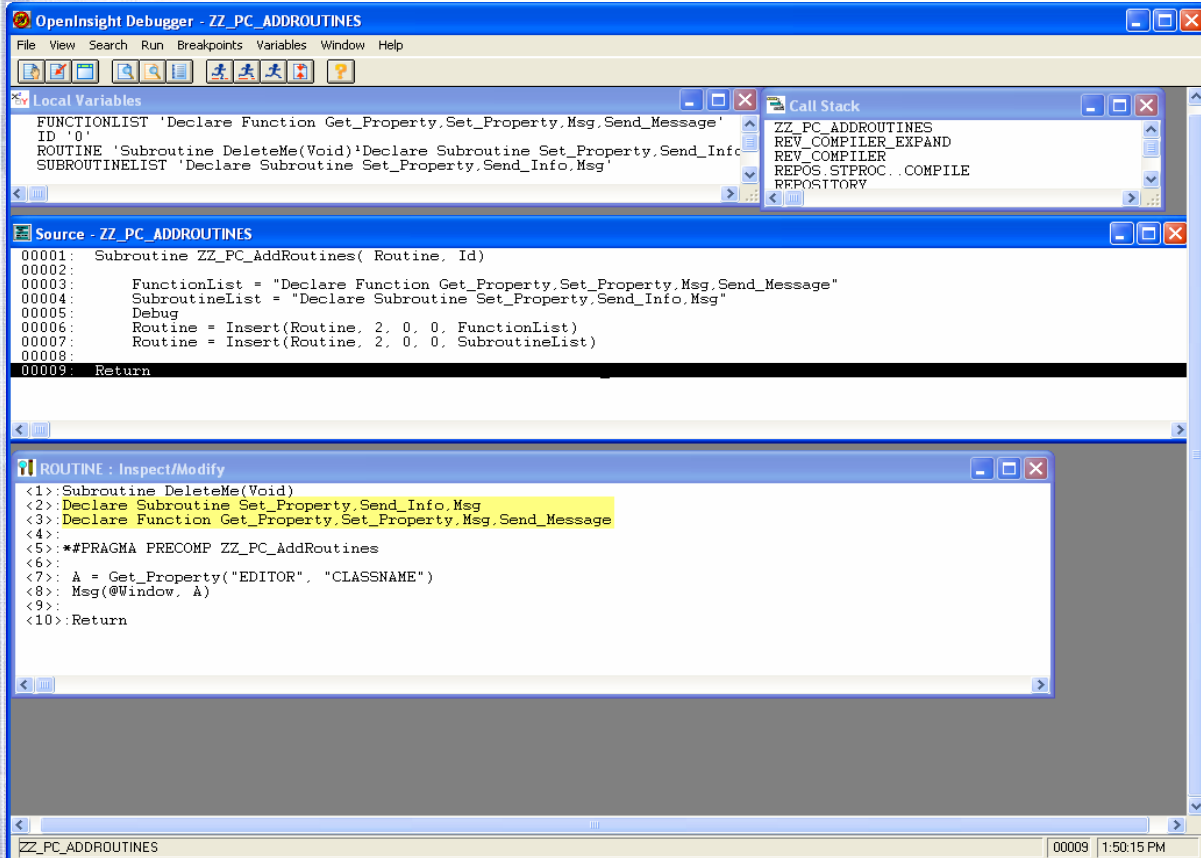
The 'ROUTINE : Inspect/Modify' window shows the source code for the 'DeleteMe' routine:

```
<1>:Subroutine DeleteMe(Void)
<2>:
<3>:##PRAGMA PRECOMP ZZ_PC_AddRoutines
<4>:
<5>: A = Get_Property("EDITOR", "CLASSNAME")
<6>: Msg(@Window, A)
<7>:
<8>:Return
```

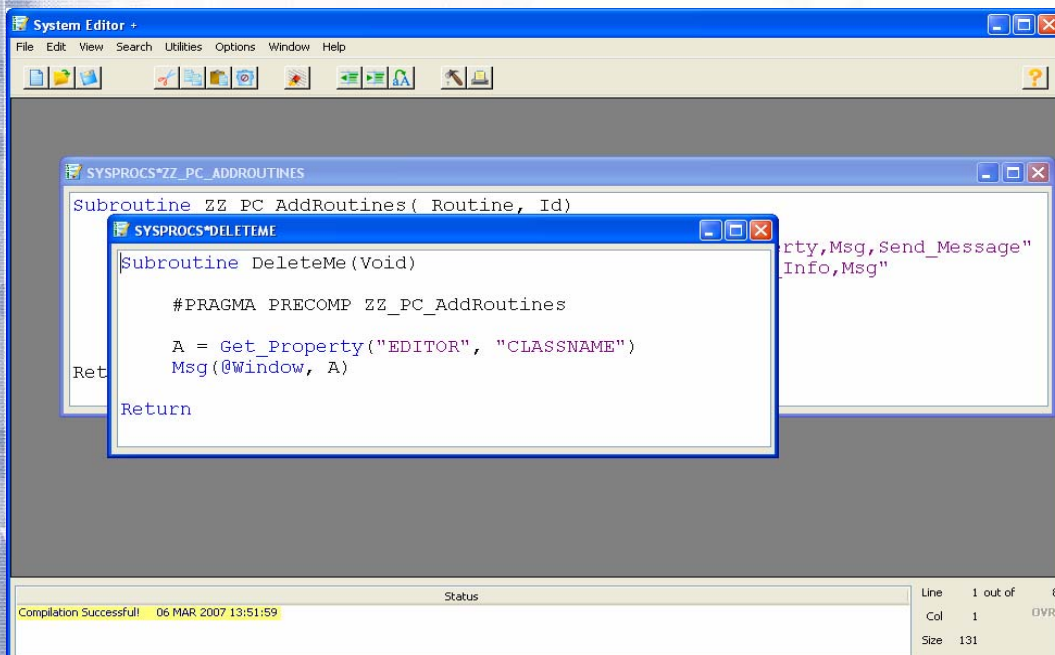
A blue arrow points from the text "See how the Pragma is now commented out" to the line 3 in the 'ROUTINE : Inspect/Modify' window.

SPREZZATURA

A little side point to mention here that will get cheers from a number of developers – the debugger now remembers your MDI child placement preferences between sessions. So we'll step over the next two lines and see our code inserted into the output.

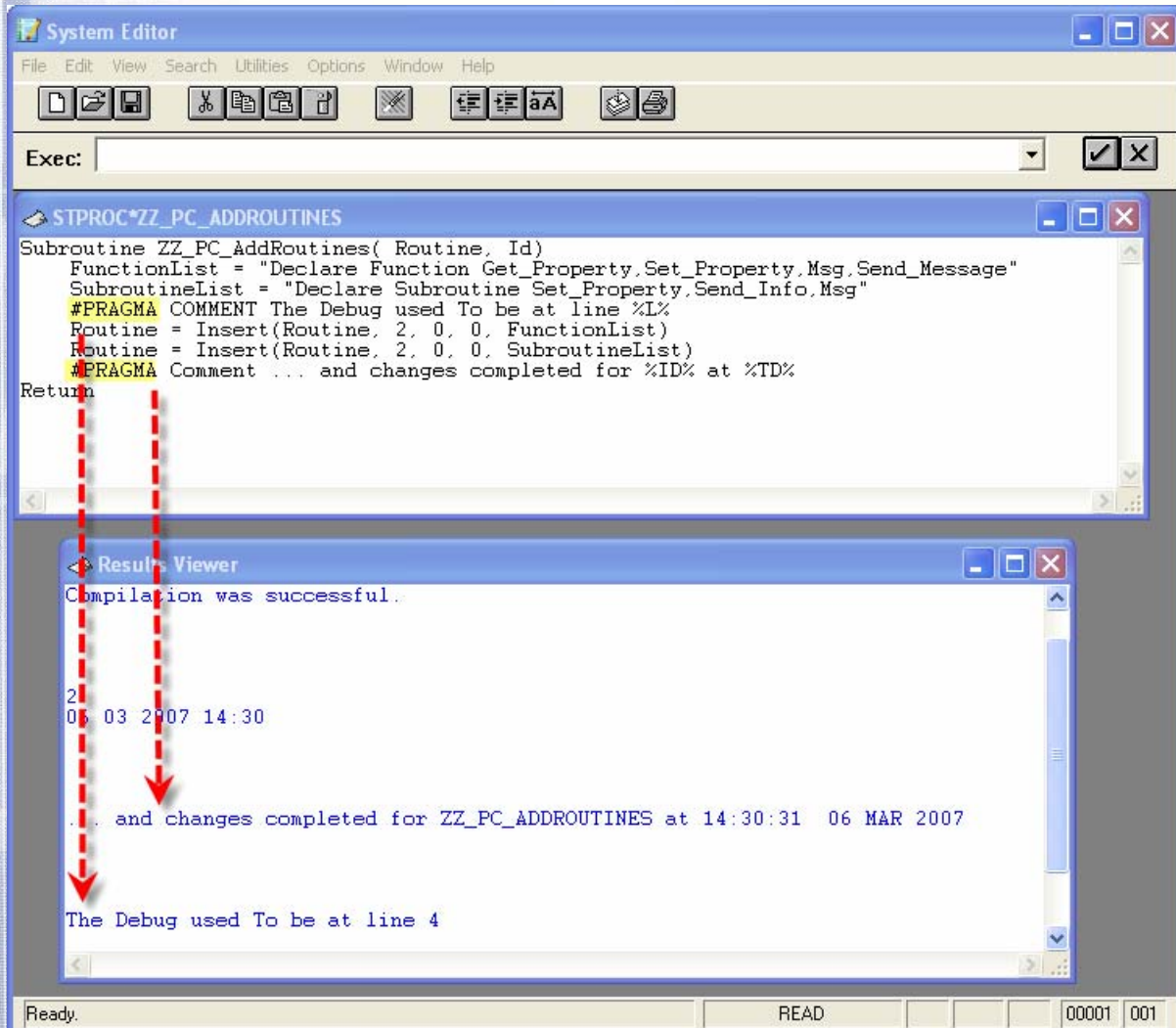


And carry on – compilation completes.



SPREZZATURA

At the moment this only works with the old editor not with the Editor+. So revisiting the code above and adding comment pragmas then compiling we get

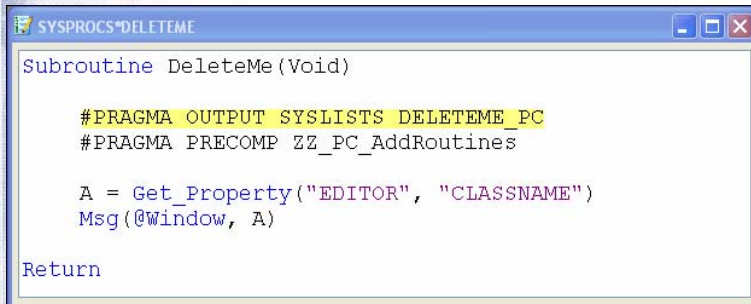


SPREZZATURA

Saving expanded code

If you wish to save off the final result of all of the precompilation options so that you may examine it you may use the OUTPUT pragma. This directs OpenInsight to save the ultimate program that will be compiled into the specified file and row. The syntax is:

```
#PRAGMA OUTPUT FileName RowId
```



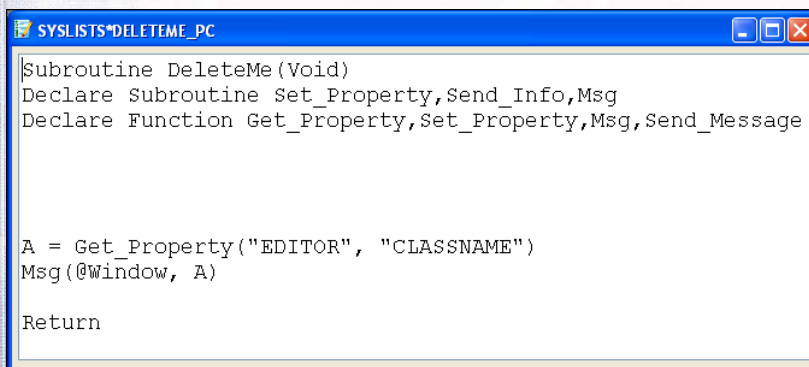
```
SYSPROCS*DELETEME
Subroutine DeleteMe(Void)

#PRAGMA OUTPUT SYSLISTS DELETEME_PC
#PRAGMA PRECOMP ZZ_PC_AddRoutines

A = Get_Property("EDITOR", "CLASSNAME")
Msg(@Window, A)

Return
```

And the resultant code in SYSLISTS



```
SYSLISTS*DELETEME_PC
Subroutine DeleteMe(Void)
Declare Subroutine Set_Property,Send_Info,Msg
Declare Function Get_Property,Set_Property,Msg,Send_Message

A = Get_Property("EDITOR", "CLASSNAME")
Msg(@Window, A)

Return
```

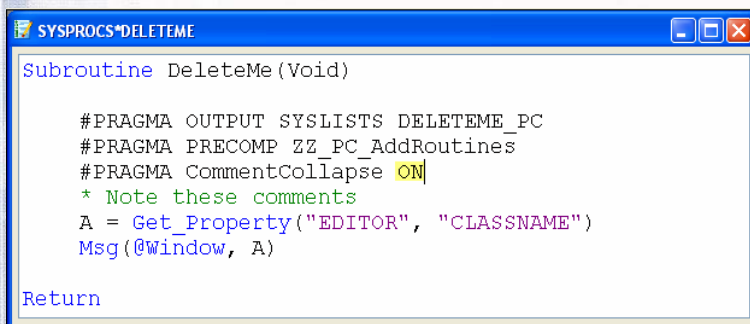
SPREZZATURA

Ensuring comments remain in Outputted Code

By default the system removes the text of comments before submitting the program to the compiler. This makes compilation slightly faster as the compiler does not have to process comment strings. This means that your saved OUTPUT code won't contain comments. To ensure the comments remain you use the COMMENTCOLLAPSE pragma. This has a value of ON or OFF. The default setting is ON. To ensure comments remain use OFF:

```
#PRAGMA COMMENTCOLLAPSE ON|OFF
```

So with Collapse ON (the default)

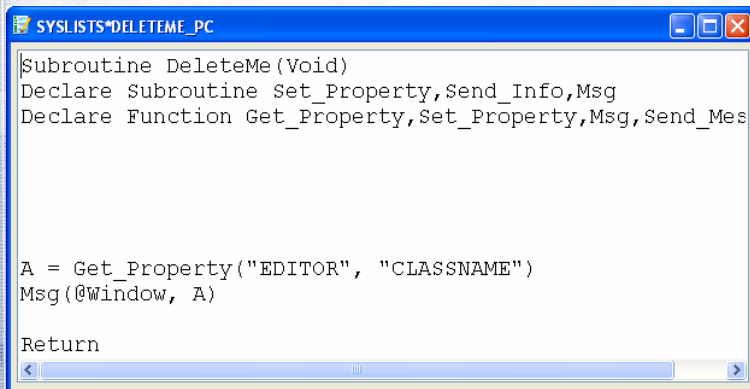


```
Subroutine DeleteMe (Void)

#PRAGMA OUTPUT SYSLISTS DELETEME_PC
#PRAGMA PRECOMP ZZ_PC_AddRoutines
#PRAGMA CommentCollapse ON
* Note these comments
A = Get_Property("EDITOR", "CLASSNAME")
Msg(@Window, A)

Return
```

The resultant output is

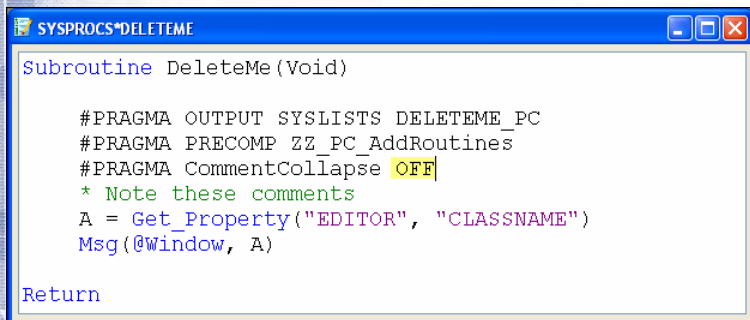


```
Subroutine DeleteMe (Void)
Declare Subroutine Set_Property, Send_Info, Msg
Declare Function Get_Property, Set_Property, Msg, Send_Mes

A = Get_Property("EDITOR", "CLASSNAME")
Msg(@Window, A)

Return
```

And with it OFF



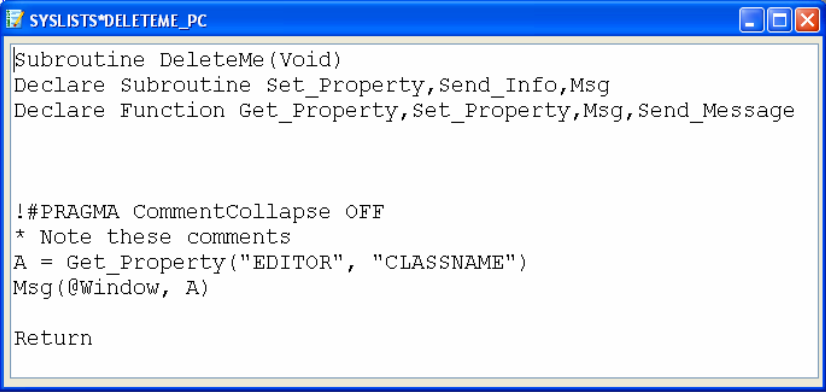
```
Subroutine DeleteMe (Void)

#PRAGMA OUTPUT SYSLISTS DELETEME_PC
#PRAGMA PRECOMP ZZ_PC_AddRoutines
#PRAGMA CommentCollapse OFF
* Note these comments
A = Get_Property("EDITOR", "CLASSNAME")
Msg(@Window, A)

Return
```

SPREZZATURA

The resultant output is



```
SYSLISTS*DELETEME_PC
Subroutine DeleteMe(Void)
Declare Subroutine Set_Property,Send_Info,Msg
Declare Function Get_Property,Set_Property,Msg,Send_Message

!#PRAGMA CommentCollapse OFF
* Note these comments
A = Get_Property("EDITOR", "CLASSNAME")
Msg(@Window, A)

Return
```

Squeezing the last ounce of performance from a routine

When programs are compiled a linemark is inserted between each line to assist in debugging. This means that a 1,000 line program will have 1,000 linemark opcodes. If performance is critical then these may be removed using the LMCOLLAPSE pragma. This takes an ON or an OFF argument and the syntax is:

```
#PRAGMA LMCOLLAPSE ON|OFF
```

This is especially useful when \$Inserting large blocks of code. When debugging this would stop at each line of the \$Insert. Using LMCOLLAPSE would allow the programmer to step over the \$Insert with one keystroke. For example:

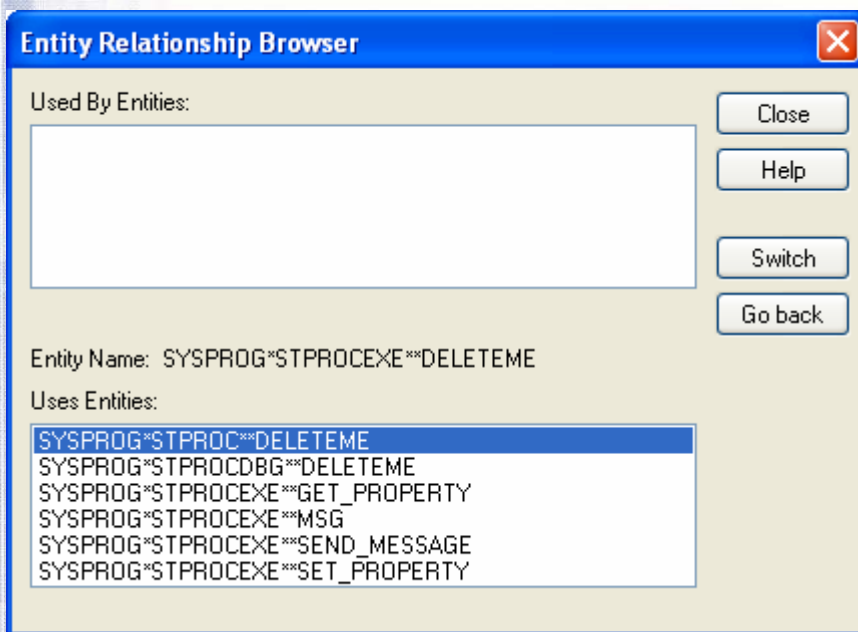
```
/// Turn on linemark collapse for inserted code
#pragma lmcollapse on
$insert sysprocs, fserrors_100
$insert syserrors_1000
/// Turn off the $insert linemark collapse
#pragma lmcollapse off
```

SPREZZATURA

Ensuring Repository Tracking

The compiler has been enhanced to keep track of the "Uses" relationship when a subroutine or function is called. Historically this only happens when the DECLARE statement is used. Now the act of CALLING a routine ensures that it is tracked.

So if we were to look at the Entity Relationships window for the DELETEME object code we'd see that it uses several stored procedure executables :-



If you wish to link in other entities to ensure complete deployment using the RDK or checkout you can now tell the compiler that this routine uses specific entities. For example, say your program is referenced in an external document and you want to ensure that this document is deployed with the program. You may now accomplish this using the \$USES compiler predirective to ensure that this is tracked in the repository.

This has the syntax:

```
$USES EntityID
```

where entity ID can have one of the following three forms:

APPLICATION_ID*TYPE*CLASS*ENTITYID where APPLICATION_ID is the application name

@APPID*TYPE*CLASS*ENTITYID where the literal @APPID will be replaced with @AppId<1>

TYPE*CLASS*ENTITYID where @AppId<1> will be put on the beginning for you.

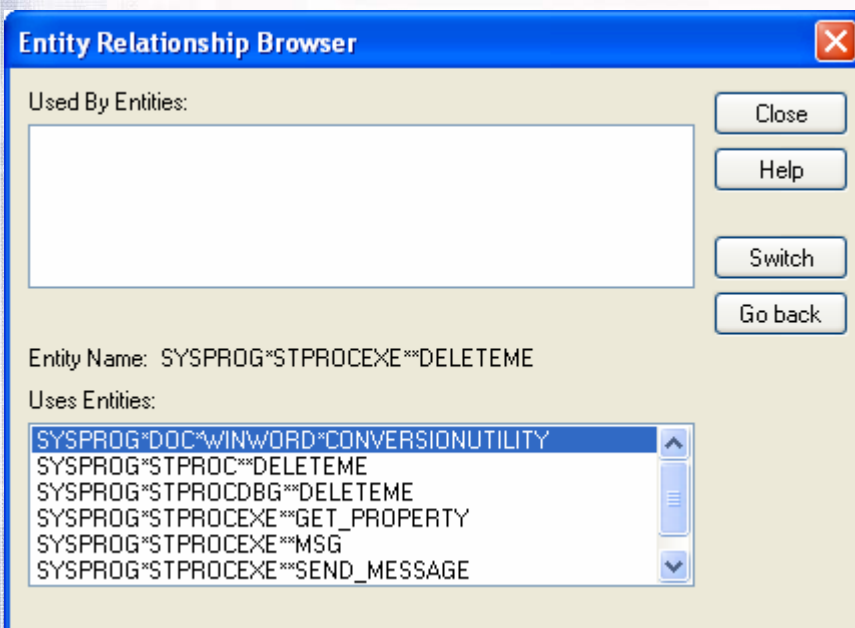
For example:

```
$USES @APPID*DOC*WINWORD*CONVERSIONUTILITY
```

SPREZZATURA

```
SYSPROCS*DELETEME  
Subroutine DeleteMe (Void)  
  
#PRAGMA OUTPUT SYSLISTS DELETEME_PC  
#PRAGMA PRECOMP ZZ_PC_AddRoutines  
#PRAGMA CommentCollapse OFF  
$USES @APPID*DOC*WINWORD*CONVERSIONUTILITY  
  
A = Get_Property("EDITOR", "CLASSNAME")  
Msg(@Window, A)  
  
Return
```

Compiling this and examining the Entity Relationships gives me



The Entity Relationship Browser window displays the following information:

- Used By Entities:** (Empty list)
- Entity Name:** SYSPROG*STPROCEXE**DELETEME
- Uses Entities:**
 - SYSPROG*DOC*WINWORD*CONVERSIONUTILITY
 - SYSPROG*STPROC**DELETEME
 - SYSPROG*STPROCDBG**DELETEME
 - SYSPROG*STPROCEXE**GET_PROPERTY
 - SYSPROG*STPROCEXE**MSG
 - SYSPROG*STPROCEXE**SEND_MESSAGE

Buttons on the right side include: Close, Help, Switch, and Go back.

SPREZZATURA

Multiline Concatenation/Nested inserts

Two longstanding compiler bugs have been fixed. It is now possible to nest inserts without messing up line counts. In addition you may now have multiple lines concatenated into a single line to improve program legibility.

For example:

```

SYSPROCS*DELETEME
Subroutine DeleteMe(Branch)
  #PRAGMA Output SYSLISTS DELETEME_PC
  Locate Branch In "ACTIVATED,ARRANGEICONS,BUTTONDOWN, |
                  BUTTONUP,CALCULATE,CASCADE,CLEAR, |
                  CLOSE,CREATE,DDEEROR,Delete,DRAGDROP, |
                  DRAGEND,DRAGOVER,DRAGSTART,DROFFILES, |
                  DSOABS,DSOCLEAR,DSOCOMMIT,DSODELETE, |
                  DSOEXECUTE,DSOFIRST,DSOINSERT, |
                  DSOINSTANCE,DSOLAST,DSONEXT,DSOPREV" |
                  Using "," Setting Pos Then
  On Pos Gosub ACTIVATED,ARRANGEICONS,BUTTONDOWN, |
              BUTTONUP,CALCULATE,CASCADE,CLEAR, |
              CLOSE,CREATE,DDEEROR,Delete,DRAGDROP, |
              DRAGEND,DRAGOVER,DRAGSTART,DROFFILES, |
              DSOABS,DSOCLEAR,DSOCOMMIT,DSODELETE, |
              DSOEXECUTE,DSOFIRST,DSOINSERT, |
              DSOINSTANCE,DSOLAST,DSONEXT,DSOPREV
  End
Return

```

Which compiles as

```

SYSLISTS*DELETEME_PC
Subroutine DeleteMe(Branch)
Locate Branch In "ACTIVATED,ARRANGEICONS,BUTTONDOWN,BUTTONUP,CALCULATE,CASCADE,CLEAR,CLOSE,CREATE,DDEEROR,Delete,DRAGDROP,DRAGEND,DRAGOVER,DRAGSTART,DROFFILES,DSOABS,DSOCLEAR,DSOCOMMIT,DSODELETE,DSOEXECUTE,DSOFIRST,DSOINSERT,DSOINSTANCE,DSOLAST,DSONEXT,DSOPREV" Using "," Setting Pos Then
On Pos Gosub ACTIVATED,ARRANGEICONS,BUTTONDOWN,BUTTONUP,CALCULATE,CASCADE,CLEAR,CLOSE,CREATE,DDEEROR,Delete,DRAGDROP,DRAGEND,DRAGOVER,DRAGSTART,DROFFILES,DSOABS,DSOCLEAR,DSOCOMMIT,DSODELETE,DSOEXECUTE,DSOFIRST,DSOINSERT,DSOINSTANCE,DSOLAST,DSONEXT,DSOPREV
End
Return
ACTIVATED:
ARRANGEICONS:

```

So as you can see – quite a lot of movement on the compiler front!

SPREZZATURA

Windows Common (and not-so-common) Controls and OpenInsight –

Carl Pates

Out of the box OpenInsight (OI) provides developers with access to all the essential user interface controls, such as edit controls, listboxes, checkboxes and so on. For the majority of tasks this is quite sufficient, but a situation invariably arises when the standard set of control types is just not enough and a more elegant solution must be found – the user interface must be extended.

In some cases this may simply be a matter of using one of the many “new” controls supplied with 32-bit and subsequent versions of Windows (the so-called “Common Controls” such as Progress Bars and TreeView controls) while in other cases it means using a “custom control” supplied by a 3rd party and written in another lower-level toolset such as Visual C++ or Delphi.

Currently OI developers have the following options for extending the user interface:

- Wait for Revelation to implement and expose the required control
- Use an OLE control
- Write a DLL to implement or wrap the desired custom control

With the release of OI 7.2 a new method of embedding controls has been added to OI that provides access to virtually any Windows control, or in the case of OIL, any control that is supported by the MainWin runtime environment. This is achieved by using a new OI control type – the WINCONTROL control. Support for this feature has been significantly enhanced for OI 8.0.

WINCONTROL and the Windows ClassName

Every control in Windows is based on a predefined template called a “Window Class”. The class has a unique name, called the *ClassName*, used to identify it. When an application wishes to create a control it specifies the *ClassName* as part of the creation process. The OI WINCONTROL does something similar: It allows OpenInsight to create a control based solely on its Windows *ClassName*, rather than the OI-specific name which is normally used (and which is translated internally to a Windows *ClassName* at runtime in any case).

Some examples of Windows and their actual class names are:

Control Type	OI Type Name	Windows ClassName
Single line edit control	EDITFIELD	Edit
Multiline edit control	EDITBOX	Edit
Bitmap	BITMAP	Static
Rich Edit	RICEDITBOX	RichEdit20W
Tab control	TABCONTROL	SysTabControl32
Progress Bar	n/a	msctls_progress32
OpenInsight Form	WINDOW	RTI_PrCl

SPREZZATURA

Before an application can use a Window Class it must “register” it. This makes sure that all of the above class information is associated with the ClassName when used. In the case of OpenInsight the following applies:

All the basic GUI controls are available to all Windows applications. There is no need to register them.

OI registers the Common Controls at start-up so there is no further work for the developer to do to use them.

Custom control classes must be registered with OI before they can be used. Most custom control authors provide an easy way to do this via exported function in a DLL that is called before use.

The OI CLASSNAME property

OI 8.0 saw the introduction of a new property called CLASSNAME which returns the Windows ClassName of a control. Prior to this the only way to find it was using the GetClassName() Windows API function.

Window Messaging

Windows is an event-driven system, which means that controls wait for input and data to be passed to them and then respond. This process is enabled by a low-level interface that applications can interact with known as the “Window Message” interface.

Input is sent to a control in the form of a “message” which then passes it to a special function known as a “Window Procedure” (or “WndProc”) to be handled. Every window class has a window procedure, and every window created with that class uses that same window procedure to respond to messages.

The system sends a message to a window procedure by passing the message data as arguments to the procedure which then interprets and processes them accordingly.

A Windows Message is comprised of four components:

The target window handle. This identifies the control the message is sent to and therefore the window procedure used to process it.

The message ID. This is a numeric value that identifies the purpose of the message.

Two numeric parameters that specify additional information pertaining to the message, such as the location of data. Their exact meaning depends on the message itself.

Within OI we have the capability to use the Windows API SendMessage and PostMessage functions to communicate with the control and the WINMSG event to listen for messages that are sent to the controls we create on our forms.

Armed with this we now have the basic tools we need to access and implement WINCONTROL functionality.

SPREZZATURA

Creating a WINCONTROL control

The process for creating a WINCONTROL control is fairly simple, although not as easy as it might be due to the fact that it is not yet supported by the OI Form Designer. There are four simple steps:

Build a control structure or copy one from an existing control (e.g. from an ORIG_STRUCT property)

Change the control type to:

WINCONTROL.<classname>

(where <classname> is the window ClassName of the desired control)

Pass the structure to Utility("CREATE")

Update the Window Common area

Example: Creating a Win32 ProgressBar

```
$insert ps_Equates
equ PROGRESS_CLASS$ to "msctls_progress32"
* // myCtrl is a simple editline used a "placeholder"
ctrlStruct = get_Property( myCtrl, "ORIG_STRUCT" )
ctrlStruct<0,PSPOS_TYPE$> = "WINCONTROL." : PROGRESS_CLASS$
call utility( "DESTROY", myCtrl )
call utility( "CREATE", ctrlStruct )
```

Using a WINCONTROL

To use a WINCONTROL control we use the Windows API SendMessage or PostMessage functions, both of which are already prototyped for us in OI. Using either involves the following steps:

Get the target control HANDLE

Determine the message ID

Determine the message parameters

Send the message

SPREZZATURA

Much of the work using a WINCONTROL involves trawling Microsoft's documentation to find what styles and messages you need to use to interact with it. Remember that you're operating much "closer to the metal" when using a WINCONTROL than you are when you're using a standard OI control so you'll be doing some extra legwork. One thing you'll notice about the MS documentation is that it mostly refers to constants by their name – to find their actual value you'll need to look in the Windows header files that are supplied with Windows Platform SDK (and which you can download freely from the Microsoft website).

For the following example we had to consult both the ProgressBar documentation on the MSDN website and the commctrl.h header file.

Example: Using the Win32 ProgressBar control to set the bar position

```
equ PBM_SETPOS$ to 0x0402 ; * from commctrl.h

* // myCtrl is a WINCONTROL progress bar

hwndCtrl = get_Property( myCtrl, "HANDLE" )

newPos = 50

call sendMessage( hwndCtrl, PBM_SETPOS$, newPos, 0 )
```

Wrapping a WINCONTROL

One consequence of using WINCONTROL controls is that it may begin to make your code look cluttered and complex – they certainly don't have the clean get/set property interface offered by "normal" OI and OLE controls for example.

However, that doesn't stop you from providing your own wrapper and here at Sprezzatura this is exactly what we're doing. For each WINCONTROL type that we want to use we create the following:

- ☑ A core routine to encapsulate all the Utility and Windows API SendMessage calls.
- ☑ A "Get Property" style routine to provide a string-based interface for accessing properties.
- ☑ A "Set Property" style routine to provide a string-based interface for updating properties.
- ☑ A "Send Message" style routine to provide a string based interface for calling methods.

The properties and methods that the last three property/method routines expose are translated into the appropriate window messages in the core routine. Deciding what property and method names map onto which windows message is something you must decide when writing the wrapper, but common sense should dictate this in most circumstances. A good rule of thumb is that window messages that get and set persistent control data should be exposed as properties, while other window messages should probably be exposed as methods.

SPREZZATURA

Example wrapper - zzx_ProgressBar()

The zzx_ProgressBar package is Sprezzatura’s wrapper around the Win32 ProgressBar control. It allows you to create and manipulate a progress bar control via a string based interface in the manner described above. It is composed of four routines:

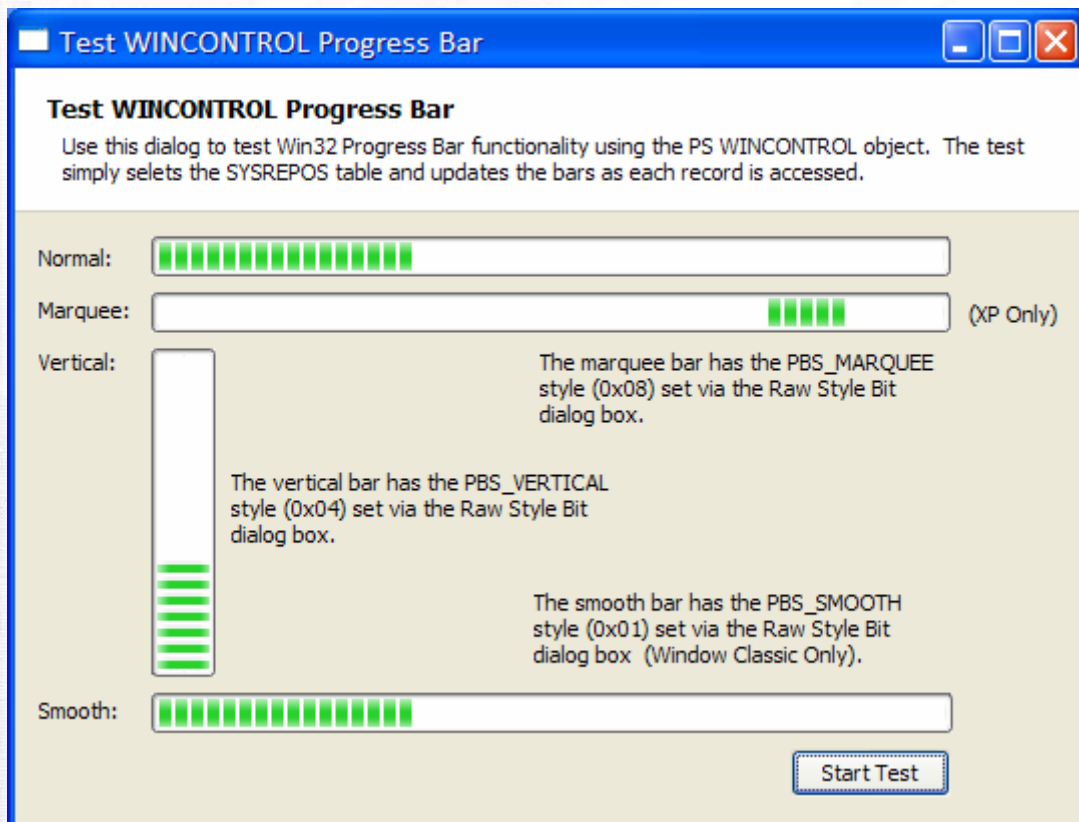
zzx_ProgressBar() – the core routine

prb_Get_Property – A “get property” function to extract Progress Bar property values. This has exactly the same interface as the normal OI Get_Property function.

prb_Set_Property – A “set property” function to update Progress Bar property values. This has exactly the same interface as the normal OI Set_Property function.

prb_Call_Method – A “Send Message” function to execute Progress Bar methods. This has exactly the same interface as the normal OI Send_Message function.

As part of this SENL issue we are releasing the source code for each of these routines so you can see how a WINCONTROL can be wrapped and used from within OI. You will also find a window called PRB_TEST that demonstrates how to use the progress bar wrapper functions. The source code is available on request from us at info@sprezzatura.com.



SPREZZATURA

Finding out more

As you've probably guessed by now this is not a small topic. However it is covered extensively on the MSDN website should you wish to find out more, specifically in the Win32 User Interface section on Windowing:

<http://msdn2.microsoft.com/en-us/library/ms632586.aspx>

And more specifically in the sections on Classes, Window Procedures, and Messages:

<http://msdn2.microsoft.com/en-us/library/ms632596.aspx>

<http://msdn2.microsoft.com/en-us/library/ms632593.aspx>

<http://msdn2.microsoft.com/en-us/library/ms632590.aspx>

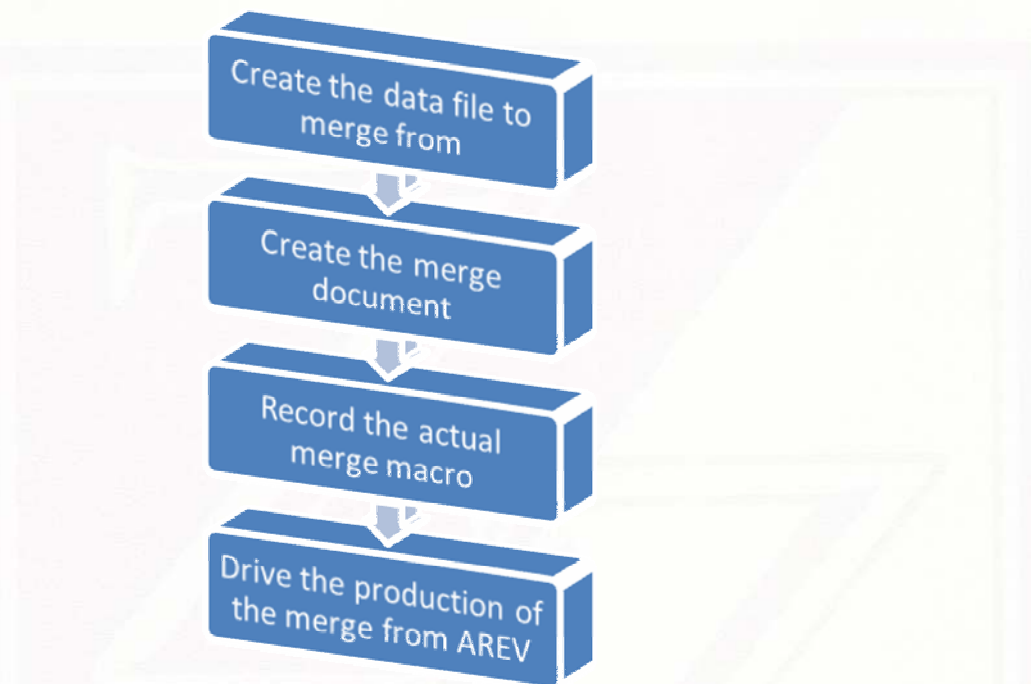
If you download the Windows Platform SDK you will have access to the core Windows header files. Probably the most useful ones in respect of OI and WINCONTROL programming are:

- winbase.h
- wingdi.h
- winuser.h
- commctrl.h

SPREZZATURA

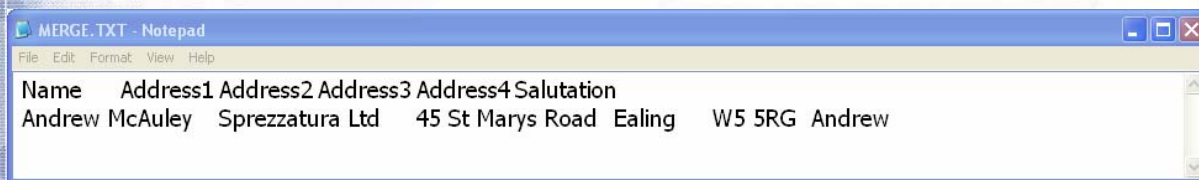
Mail Merging From Advanced Revelation – Andrew P McAuley

At a client site recently we were faced with creating a complex document to replace an existing report for legislative purposes. Needless to say the idea of cutting lots of PCL didn't appeal and having had so much success with using OLE within OpenInsight my thoughts naturally turned to using Word for Windows as my reporting vehicle of choice. What would be required could be split into four discrete areas :-



Creating the data file to merge from

This is actually very straightforward. The easiest format for mail merging is simply a tab delimited text file, where the first line is the headings to merge and the second and subsequent lines are the data to merge. This can be created by hand using NOTEPAD.EXE for proof of concept.



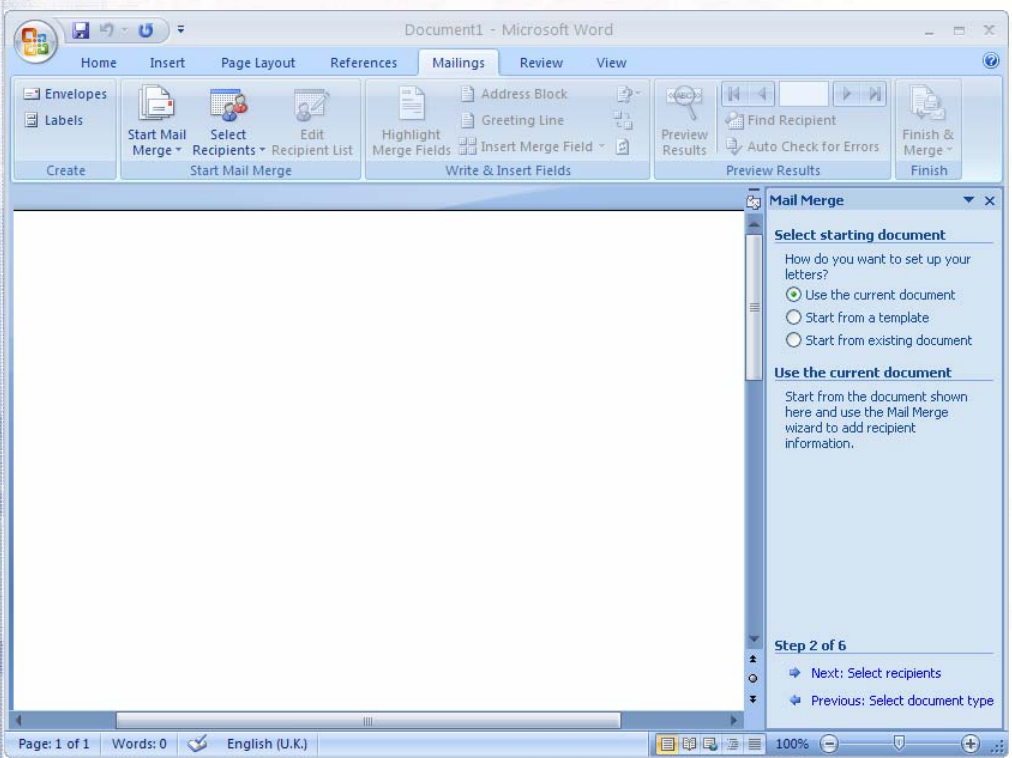
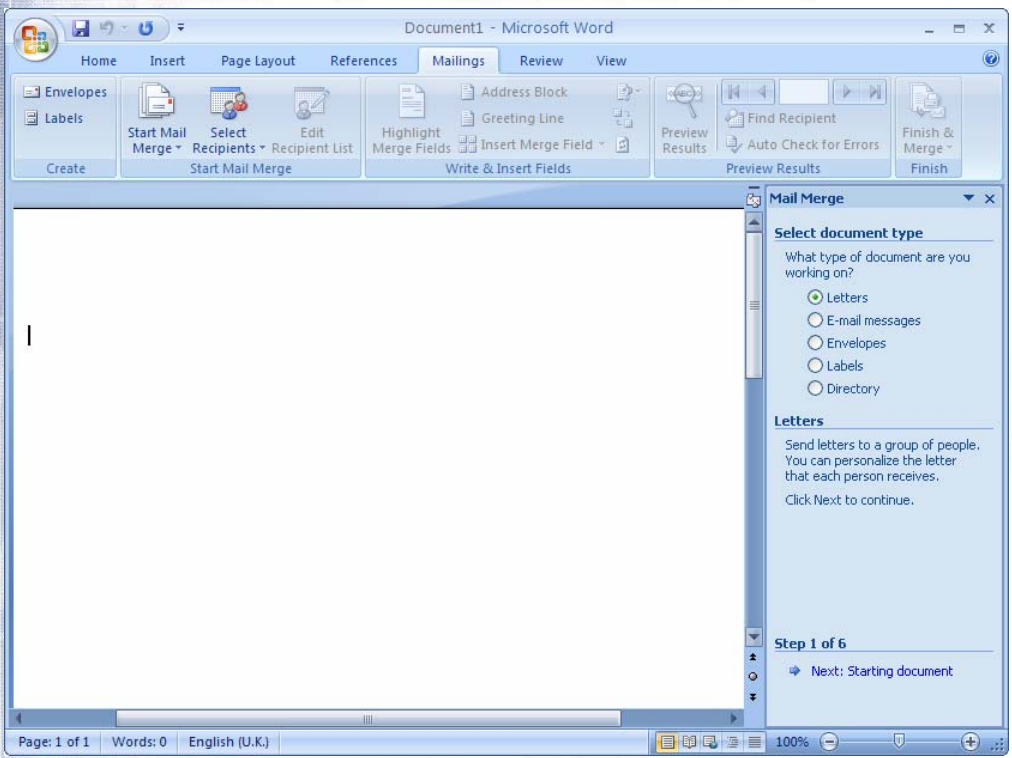
To create this we simple enter Name, press tab, Address1 press tab, Address2 press tab etc.

Creating the merge document

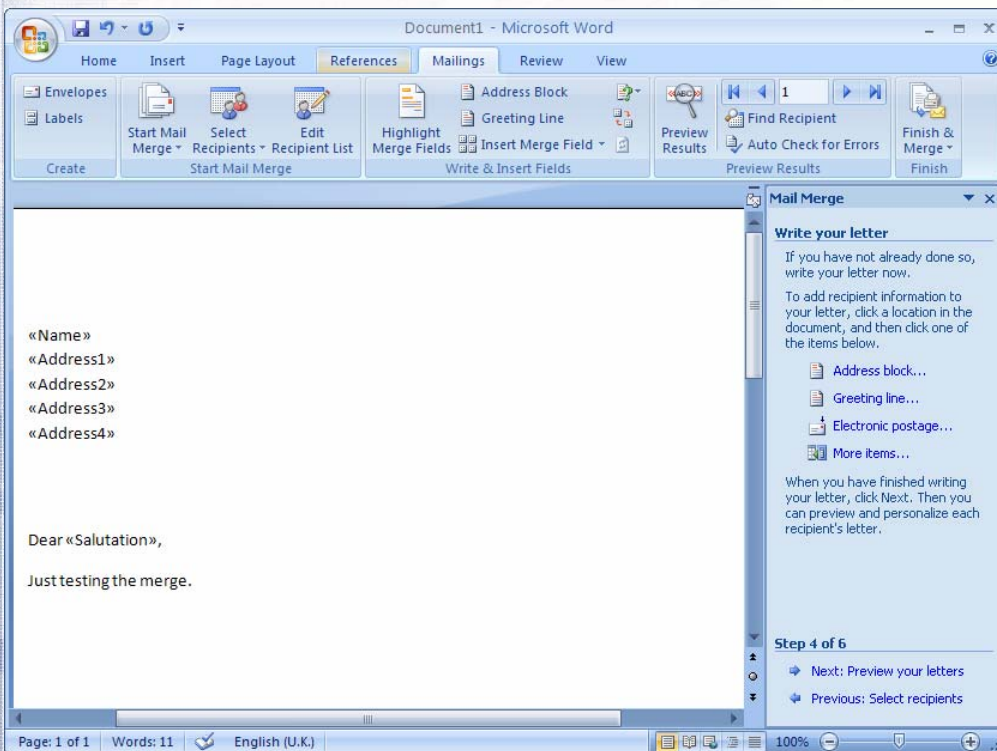
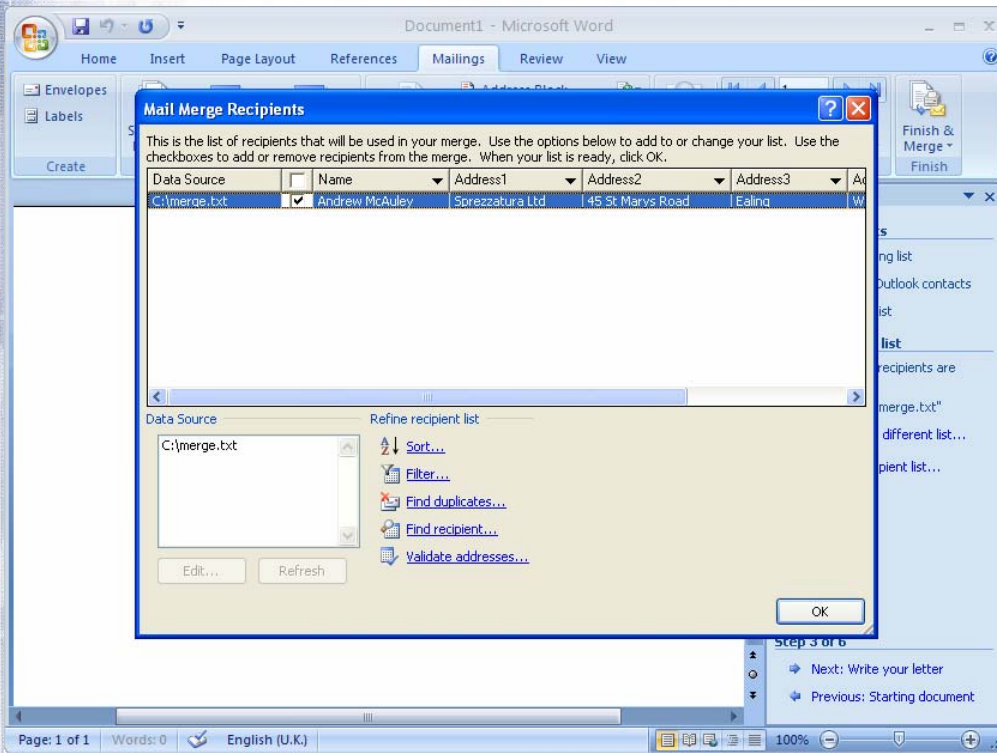
Again this is straightforward – simply create a standard mail merge document using as the data source the mail merge data file created above.

We'll use the Mail Merge Wizard provided with Word and whiz through the first three steps :-

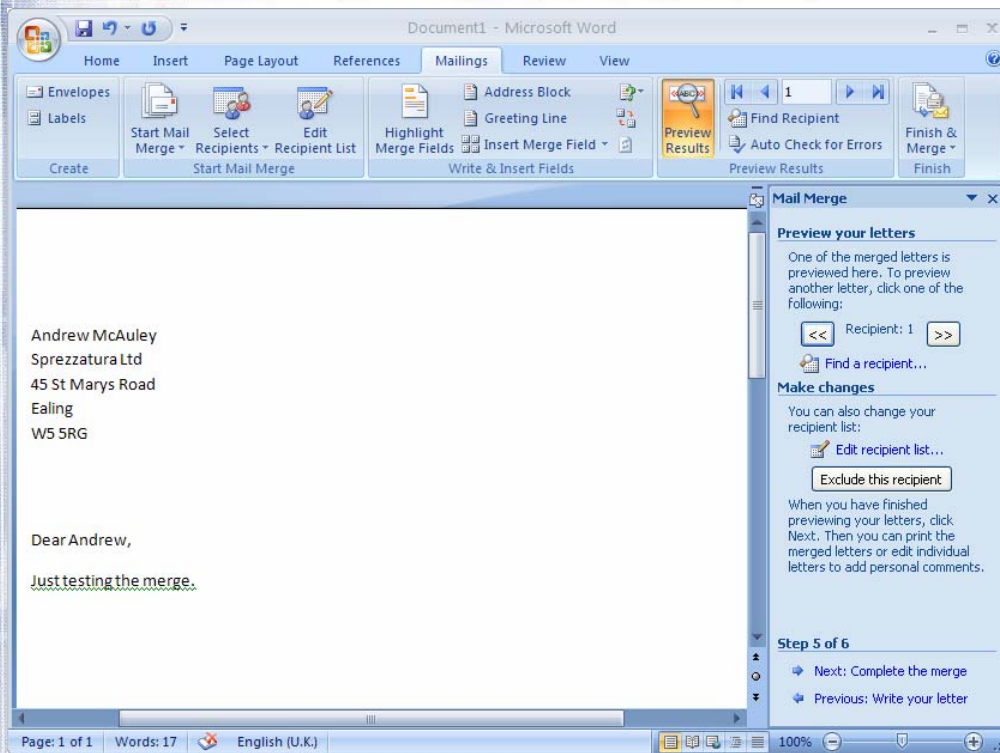
SPREZZATURA



SPREZZATURA

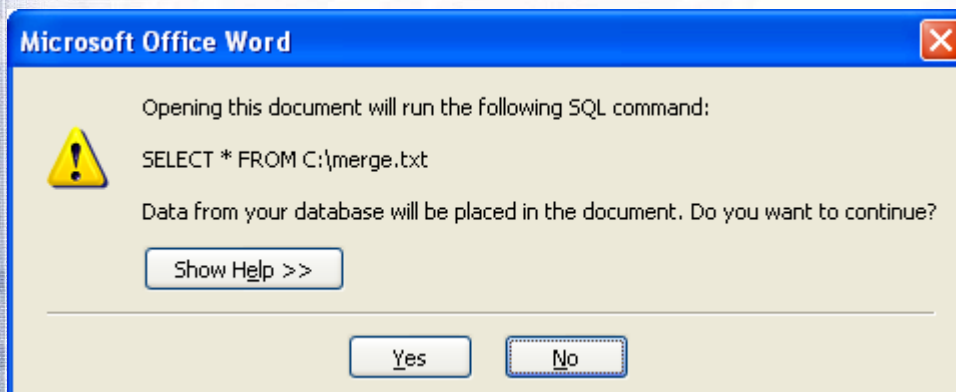


SPREZZATURA



And save the Merge as c:\AREVM.DOC.

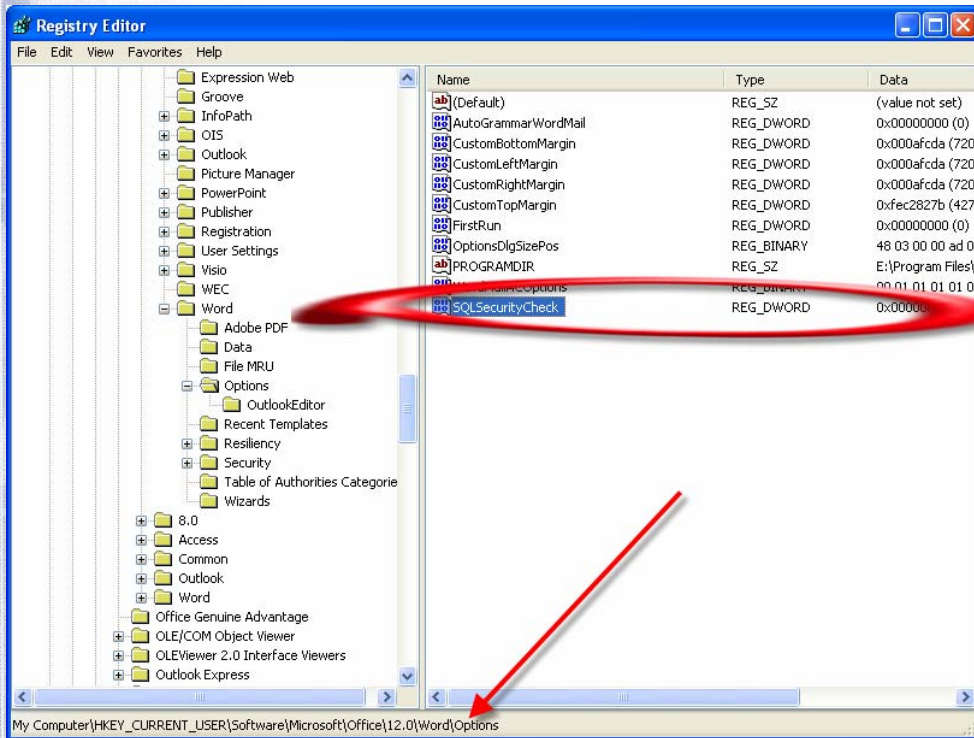
Closing the document and then reopening it shows us a slight problem. The following Dialog appears



This is undesirable because it will block our automated merge process later – so we need to prevent this dialog from appearing...

SPREZZATURA

This can be done using REGEDIT to add the value SQLSecurityCheck to the appropriate version of Office- this should be set to 0 :-



Record the actual merge macro

This macro needs to do the following actions :-

- Open

 - Open the document to be merged
- Merge

 - Perform the merge to the printer
- Close

 - Close the document and quit Word

Simply recording the macro was *nearly* enough but the last couple of lines highlighted also needed to be added to close the document and quit Word. Stating the obvious you can't record a Macro which involves closing Word as then the Macro is killed!

SPREZZATURA

Sub AREVMERG()
,

' AREVMERG Macro
,

' Macro recorded 06/03/2007 by Andrew McAuley
,

```
ChangeFileOpenDirectory "C:\"  
Documents.Open FileName:= ""AREVM.doc"", _  
    ConfirmConversions:=False, ReadOnly:=False, AddToRecentFiles:=False, _  
    PasswordDocument:="", PasswordTemplate:="", Revert:=False, _  
    WritePasswordDocument:="", WritePasswordTemplate:="", Format:= _  
    wdOpenFormatAuto, XMLTransform:=""  
With ActiveDocument.MailMerge  
    .Destination = wdSendToPrinter  
    .SuppressBlankLines = True  
    With .DataSource  
        .FirstRecord = wdDefaultFirstRecord  
        .LastRecord = wdDefaultLastRecord  
    End With  
    .Execute Pause:=False  
End With  
ActiveDocument.Close _  
SaveChanges:=wdDoNotSaveChanges  
Application.Quit  
End Sub
```

SPREZZATURA

Drive the production of the Merge from AREV

This requires a knowledge of command line options for Word for Windows and knowledge of a new command called START that spawns a new command prompt. Our AREV command then becomes

```
PcExecute "CMD /C START /MIN WINWORD /MMERGE"
```

PcExecute	Runs a DOS command
CMD	Loads a command processor
/C	Close the command processor after execution of command
START	Spawn the new command session
/MIN	Minimise it so it cannot be seen
WINWORD	Launch Word for Windows
/MMERGE	Run the Macro MERGE when it opens

Now all we have to do is update the merge data file from AREV and run the line above. Magically our mail merge will start to spew out of the printer without the user even being aware that Word has been invoked!



SPREZZATURA

Celebrity Interview - Bryan Shumsky - Revelation R&D, Senior Developer

Who are you?

I'm Bryan Shumsky (though I imagine you knew that already :-)) Perhaps here you're looking for a mini-CV? I've worked in the multivalued market since 1986, first as a database administrator, and then as a software developer. Most recently, I worked for Via Systems as the Director of Engineering.

Who do you work for?

I'm a Revelation Software employee, and report directly to Mike Ruane.

What is your role in the RevSoft community?

I'm involved in development and maintenance of the base OpenInsight product, as well as the networking products. I've had a lot to do with CTO (the Character to OpenInsight green-screen interface), and AREV32, in addition to a whole set of tools that interact with them (OECGI2, OI4WEB, etc.)

How long have you been involved with RevSoft?

I've been working for RevSoft for the last 18 months. Before that, I'd done some development work in AREV (!) as an application developer many, many years ago.

Which products do you use?

OpenInsight 8.0 and AREV32 primarily, though I'm rather unfamiliar with many of the GUI parts of the product (as I'm a relative newcomer from the 'traditional' multivalued field) but I'm also using Java, Visual Studio .NET, and older Visual Studio products for C, C++, and Visual Basic development. [\(Quite the polyglot – Ed\)](#)

What do you most like about RevSoft products?

Their openness and flexibility. It makes it possible to do all kinds of things that neither the original developers, nor us, planned for in advance.

What do you most dislike about RevSoft products?

I don't think there's anything in particular - if there was, I'd just go and change it :-)

If you HAD to use another database what would it be and why?

Well, now, why would I HAVE to? :-)

What are your favourite 3 books and why?

I'm an avid reader (mostly humour, science and science fiction), so it's very hard to just pick three. However, my 'gut reaction' would be:

Bill Bryson - Notes From A Small Island - as a fellow Anglophile, and an avid traveler, it's the perfect book for me; Isaac Asimov - Foundation - an important and influential series in science fiction that I fondly remember first reading years (and years) ago; Hmmm, number 3...number 3...Oh, of course - Advanced Revelation Developer Series System Subroutines

SPREZZATURA

What are your favourite 3 CDs/Albums and why?

Meatloaf - Bat Out Of Hell - I guess I like the grand, operatic sound combined with rock and roll; Queen – Night at the Opera -- see above; Air - Talkie Walkie - I heard some of these songs on a car radio in a foreign country (I'm not even sure _which_ country anymore) and thought "my, that's catchy!" (I think it was "Sufin' on a rocket") And then had to spend a fair amount of time tracking down the album. It's a very different sound.

What are your three favourite films and why?

Well, here I'm going to cheat, and assume by 'films' you also meant to include 'DVDs', as the wife and I don't get to go to the cinema that much any more. And if we're going to stretch the definition a bit, I'll include:

Buffy the Vampire Slayer (TV Series) - pick any of the series 1-7, they were all great; Star Wars (the original movie, later entitled Episode IV) - yes, it's "bubblegum" science fiction, but it's FUN; Happy Feet - I've always loved penguins, and (if we weren't living in Arizona) I would seriously consider building a penguinarium in our basement (if we had a basement)

What event in history would you most like to have been present at?

That's a tough one. When one thinks of 'historical events', they're either (a) not really a single, discrete moment in time, but rather a _series_ of events, or (b) they're usually not so much fun for the participants (though, then again, maybe that's just my American history talking...) How far back can I go? Maybe the moment when life first started on Earth in some brackish pond...

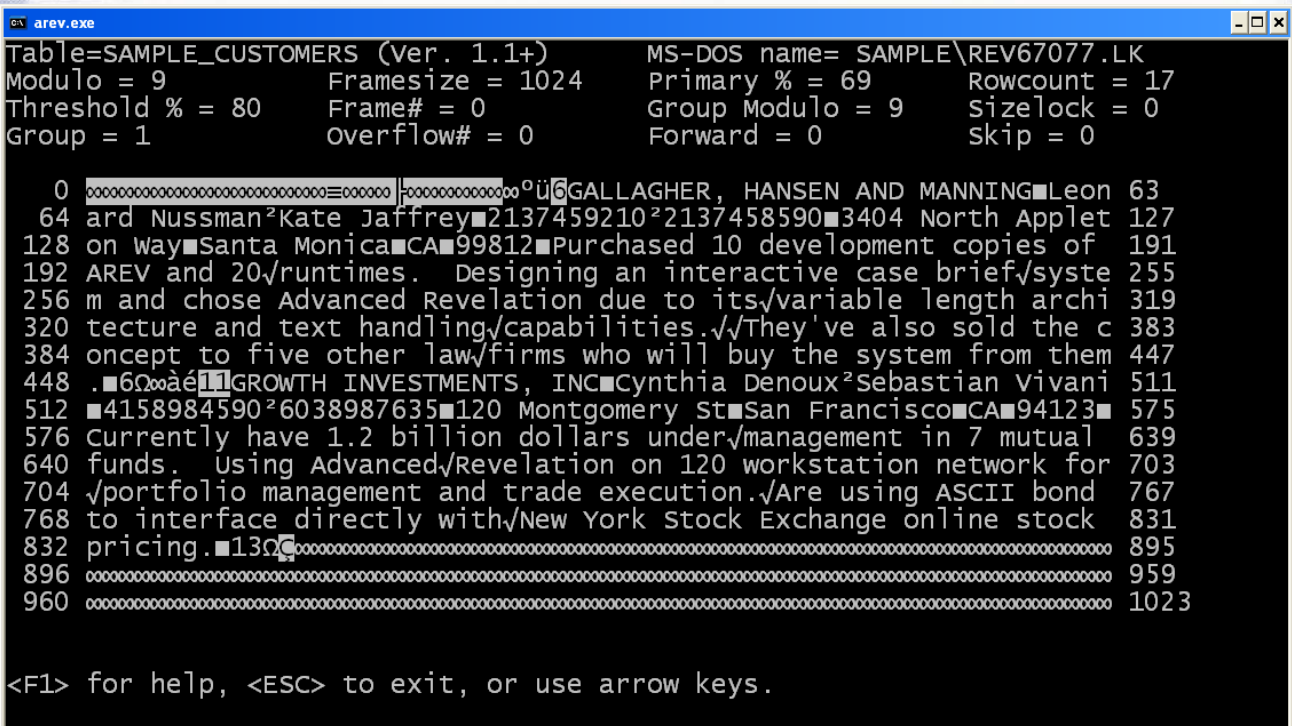
And finally Your motto/witty aphorism

I guess my unspoken motto would be: Always decide where you're having lunch before you do any other business for the day. (One must have ones priorities, after all...)

SPREZZATURA

ZZ_IDE_DUMP – A New Utility – Aaron Kaplan

When we decided to talk about Linear Hash structures at the recent user conference in Seattle, we realised that OpenInsight doesn't ship with a DUMP window. Those of you who came to OpenInsight from Advanced Revelation will remember this utility. It was especially useful when GFEs needed to be fixed but it was also useful at other times – not least for understanding what was actually going on at the physical level of the linear hash file. Just for old time's sake here's what it looked like!

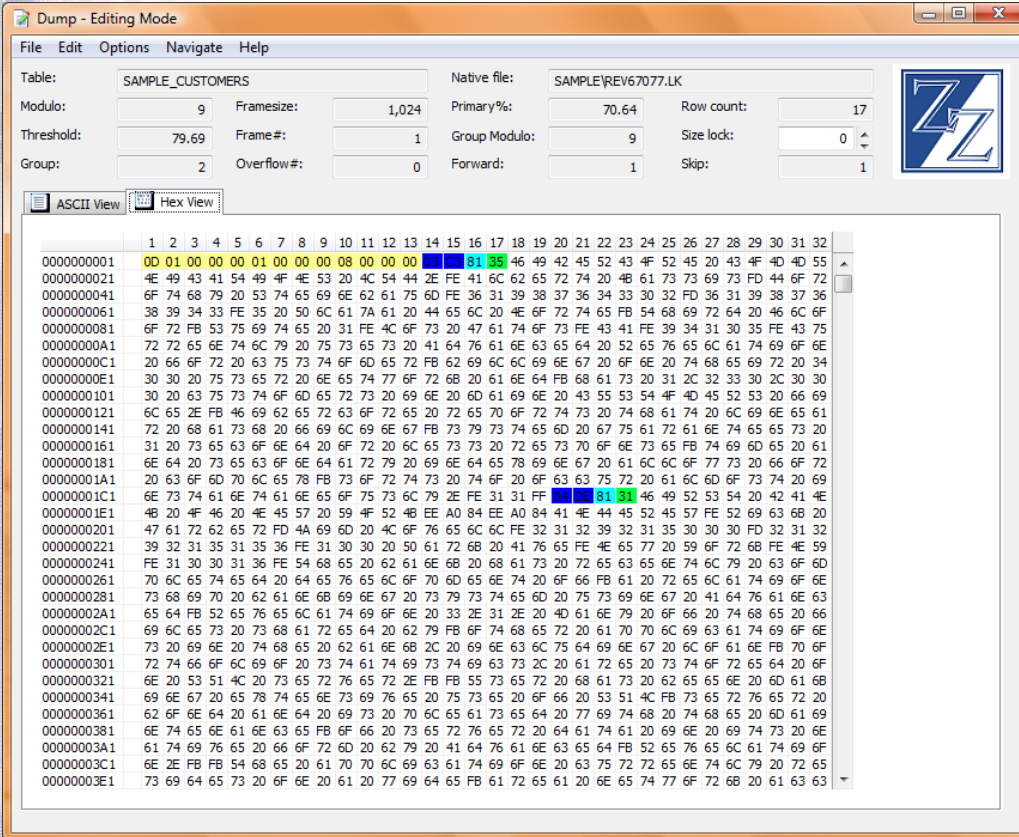


In addition DUMP in AREV doesn't understand the new Universal Driver file formats and so it can't be used against this new structure. So for these two reasons - the lack of UD support and the lack of ANYTHING in OpenInsight we thought we'd best write our own version of the dump!

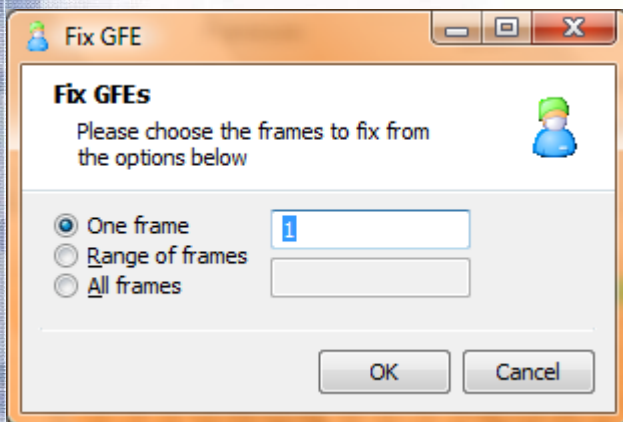
Naturally our starting point was the old AREV dump – after all if it ain't broke why fix it? But we also wanted to take a bit more advantage of the Windows User Interface which provides more usable screen real estate. So as you can see in the following screen shot we have more space for the native file name. We can afford to put commas between our thousands and we can display the primary % and threshold % as decimals which the old dump had no room for. The other minor alteration to the header display is that we've added a spin control to allow the incrementing and decrementing of the size lock. The AREV DUMP used Reverse Video to good effect to show the header section of the frame and to highlight the individual row ids. We've extended this concept and used colours instead. As you can see the header section is in yellow, the row length is in blue, the key length is in cyan and the key is in green. Naturally these colours are user configurable as we'll see later.

SPREZZATURA

This is where the Windows Interface makes things so much easier. Instead of showing a small section of the file in Hex we provide a second tab that allows the entire frame to be examined in hex. This makes the header structures much easier to understand.



As you'd expect, we have options to fix GFEs and compress the overflow – although to ensure that we keep your data safe we use Revelation's own routines to actually perform the work.



SPREZZATURA

Naturally we've also allowed for the ability to edit data. By turning on editing (available from the menu) the system changes into edit mode. Notice the icon change from a locked bolt to a pencil. We wanted to ensure that editing was as "fool proof" as possible, so the system won't allow you to edit a key – after all you might change it to something that hashes to another frame and thereby "lose" the data. We also wanted to make it easy for you to insert "control characters" so rather than you actually editing the data in place we allow you to click on the character you wish to edit and we then provide a dialog for you to choose your character from.

Legend

Option	Value
Character to use below space	.
Character to use above 246	¶
Header colour	8454143
Row length colour	16711680
Id length colour	16776960
Id colour	4259584
Data colour	16777215

Our intention is to make this utility available later this year. If you'd be interested in a beta copy then please email your request to support@sprezzatura.com.

SPREZZATURA

An Introduction to the Use of OLE controls from OpenEngine – Part 2 -

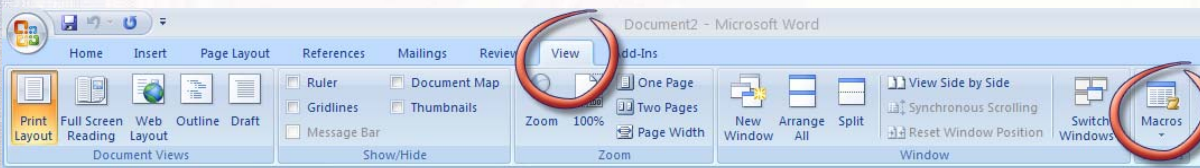
Andrew McAuley

In the last issue we examined the Basic+ statements available to us for manipulating OLE controls from the OpenEngine. We also looked at ways of exploring the wealth of OLE objects available for us to use from within our own programs. In this issue we're going to look at a practical example of using OLE for Mail Merging.

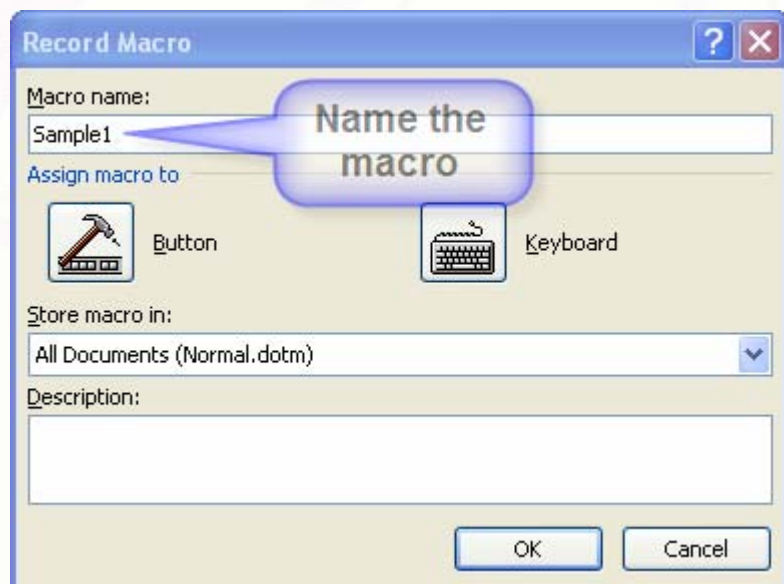
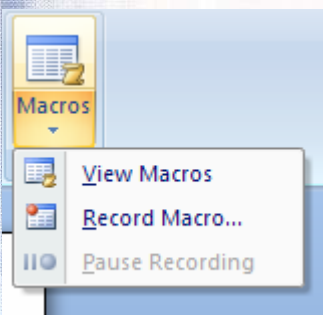
Exploring Using Macros

One of the easiest ways of exploring automation using OLE is to acquire a rudimentary understanding of OLE Automation using a more common scripting language and then to convert this into the Basic+ equivalent. By far the easiest way to achieve this is by using the Macro Recording capabilities of the software you are working with and then to examine this to see how to achieve your desired goal. So let's start with a very simple example – opening a file and printing it. I'll open the file we created earlier for the AREV example.

Firstly we need to start recording the Macro

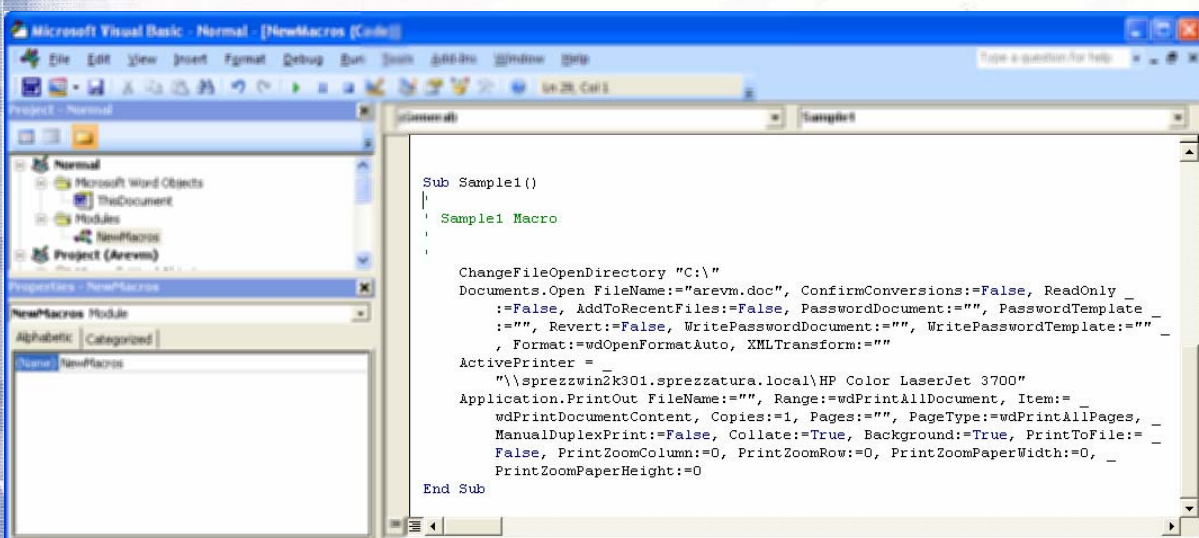
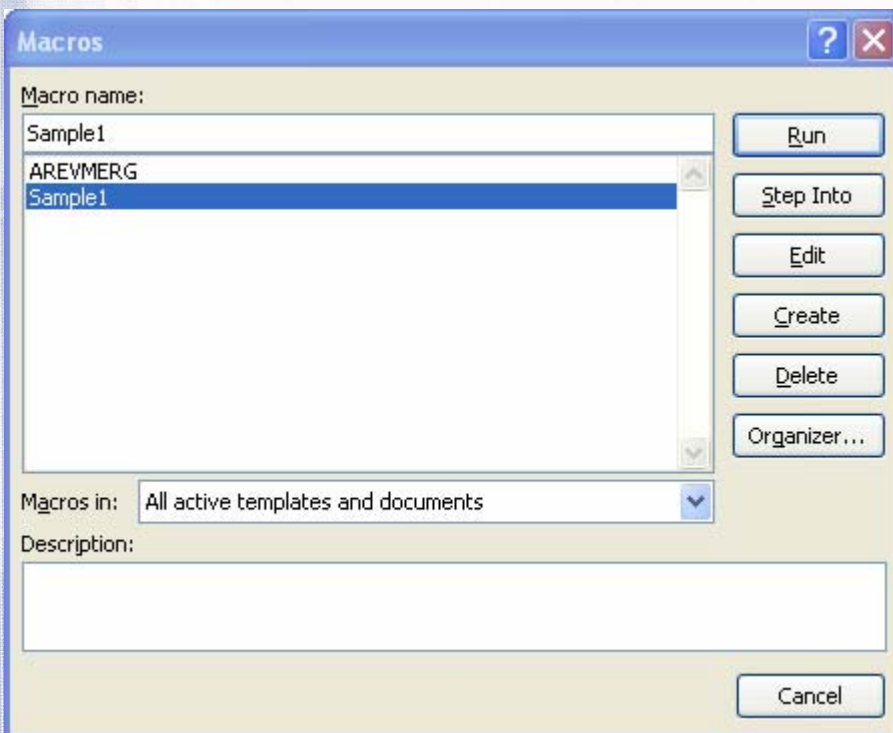


So we choose the "Record Macro" option



SPREZZATURA

And open the file and print it. We can then stop the macro recording and examine what has been generated for us.



SPREZZATURA

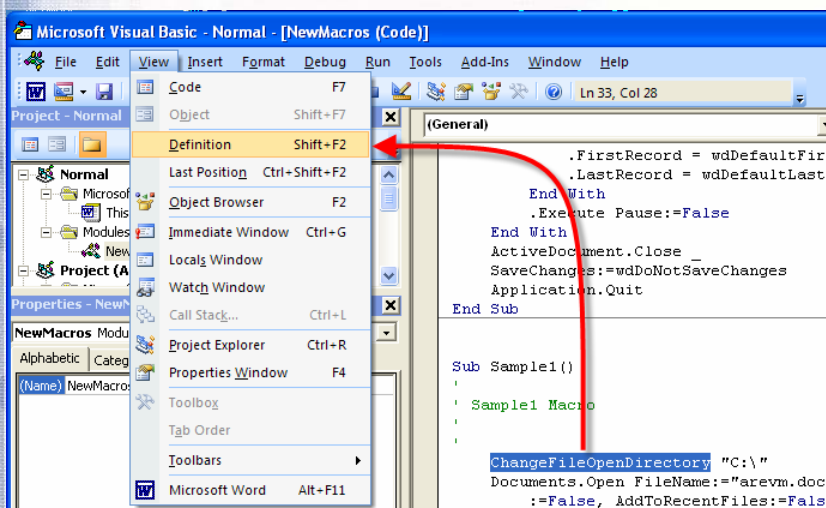
Now looking at this it is obvious that the macro has recorded four “actions” :-

- Change
 - Change the directory
- Open
 - Open the file
- Change
 - Set the printer
- Print
 - Print out the document

Now to examine what this is actually doing we can make use of the IDE used to examine the Macro itself in conjunction with the MSDN online library. Let’s illustrate this step by step.

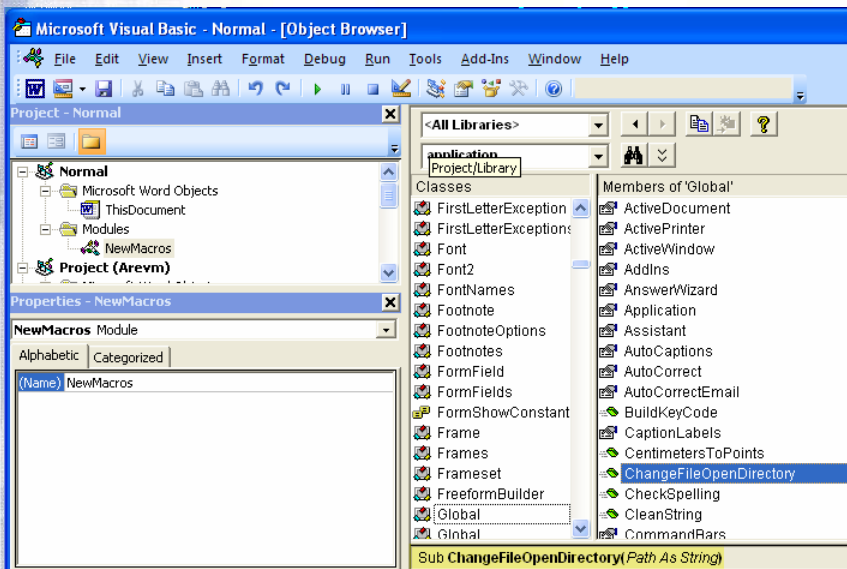
ChangeFileOpenDirectory

Firstly to find out what this is we can highlight the word and choose the menu option View/Definition

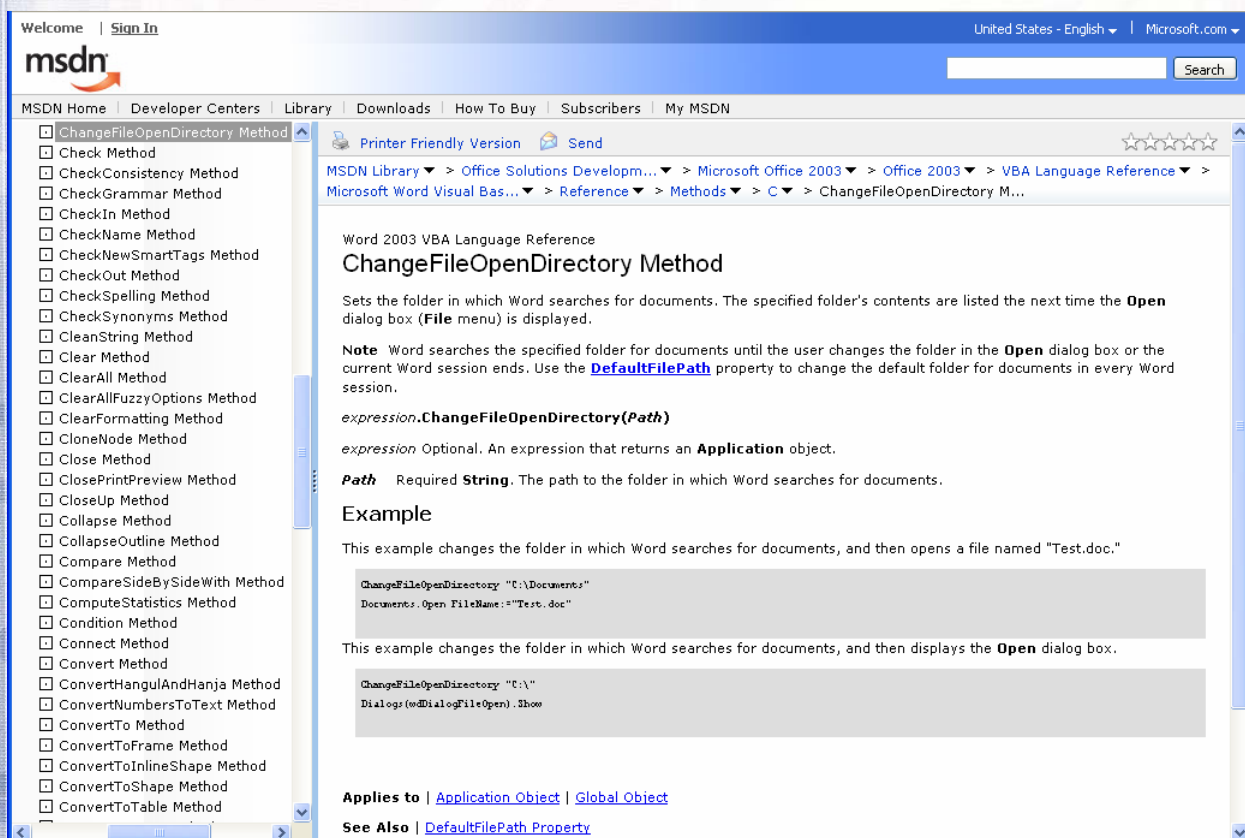


SPREZZATURA

This opens the Object Browser and shows us that ChangeFileOpenDirectory is a “Sub” – in OpenInsight terms a “Method” – so we know that to emulate this in Basic+ we’d need to use OLECallMethod.



To find out what this method is used for we turn to the MSDN online library where it is defined thusly :-

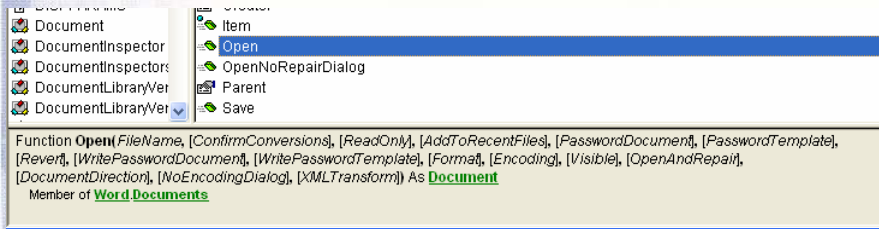


From this we can (with the benefit of an additional helping of hindsight) work out that we don’t actually need to do this as we don’t want to force the File Open Dialog to use that Directory – rather when WE automate it we’ll provide the full path to the relevant file.

SPREZZATURA

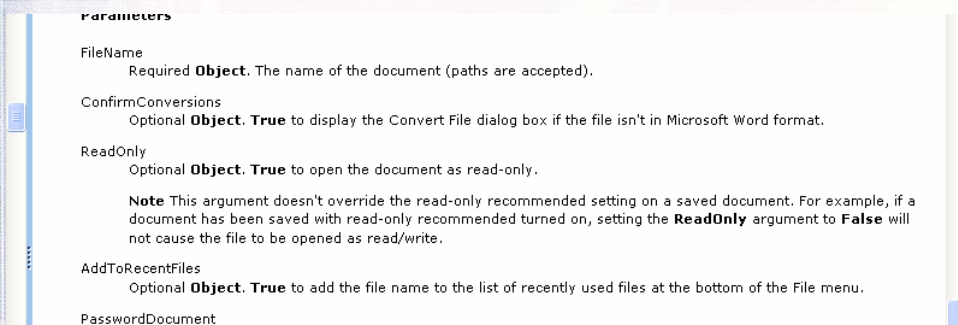
Documents.Open

Employing the same technique as before to get to the Object Browser and we see that this is a Function – again exposed using OLECallMethod.

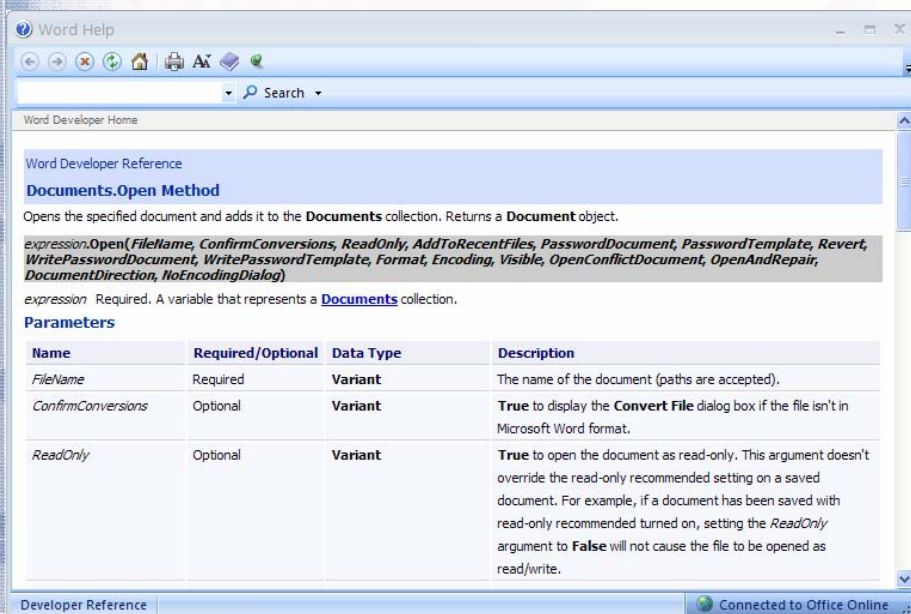


The object browser even helpfully provides us with the optional parameters – and as you can see **most** of the parameters are optional – indicated by being enclosed in [].

MSDN again comes up trumps with this providing more details of what the parameters are used for.

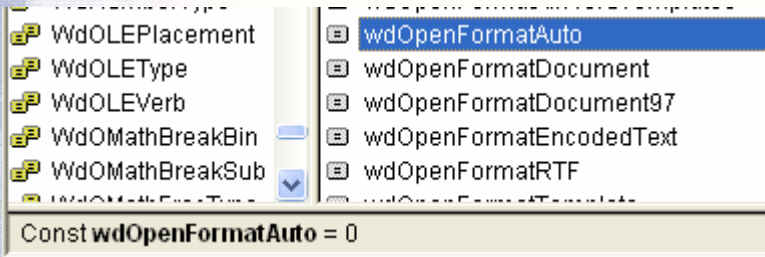


Even more straightforwardly – pressing F1 at this point brings up the online help



SPREZZATURA

Of course it's not *quite* that straightforward. Looking at "Format:=wdOpenFormatAuto" we can see that reference is being made to some form of constant – *wdOpenFormatAuto*. If we were going to reference this ourselves we'd need to know the value associated with this constant. But fortunately object browser springs to the rescue again showing that *wdOpenFormatAuto* has a value of 0.

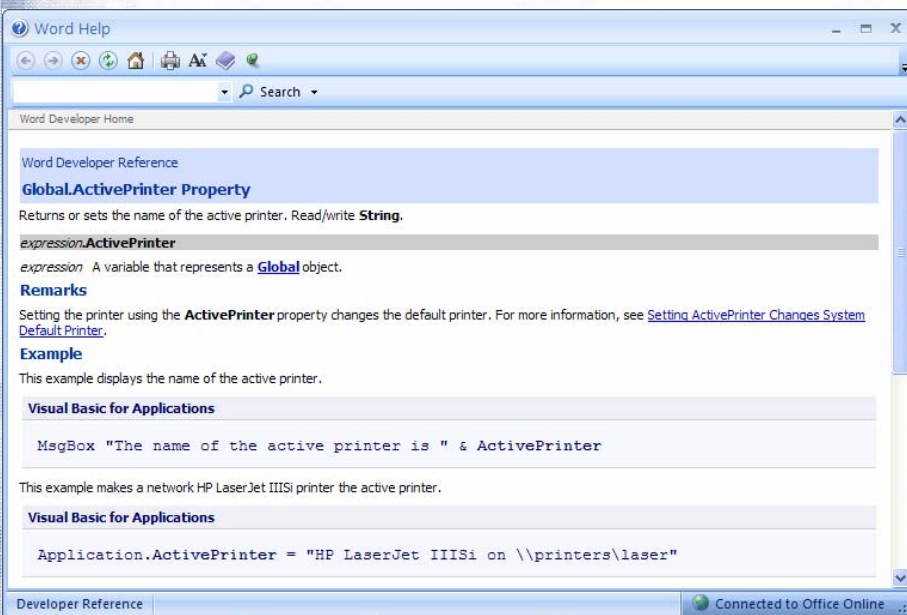


Even better it lists all the other possible values for this constant such as *wdOpenFormatRTF* being 3.



ActivePrinter

Using the object browser for this we see that this is a PROPERTY of the Application



So mapping into Basic+ we know that we'd use the OLEPutProperty routine to set this.

SPREZZATURA

Application.Printout

Combining all of the above and using the object browser, it's a simple task to see that again this is a method, that wdPrintAllDocument has a value of 0 but could have other values to print ranges, the current page etc etc.

Performing a Mail Merge

So let's put all of our new found knowledge to use and create a Subroutine that is passed the name of the Mail Merge document to print, along with the name of the data file to use in the merge. Let's also optionally have this either show the merged documents for amendment and subsequent printing OR just print out the document directly to the printer.

The first caveat as we begin our development is that at every step of the process we need to check for potential errors as there is no point in continuing if errors exist. It's at times like this that we could *really* use an OnError GoSub construct. As it is we'll just have to GoSub ErrorHandler after every OLE related call.

The second caveat is that although OpenInsight provides us with a way of returning the OLE error, it doesn't provide a mechanism for converting this error number into meaningful text. Needless to say it is not impossible to write your own such translator and at Sprezz we've embedded this into one of our generic utility functions – you'll see references to this in the code.

So the first thing we need is our own error handling subroutine...

```
CheckError:

    FullErrorText = ""
    OleStatus = OleStatus()
    If OleStatus Then
        FullErrorText = ZZx_Utility("WINERROR", OleStatus)
    End

Return
```

First off we'll show you what we mean about error numbers – let's deliberately cause an OLE error by trying to start Word using a non-existent class.

```
wordApp = OleCreateInstance("Word.Applications")

Gosub CheckError
```


SPREZZATURA

System Monitor

```
run zz_ole_word_automation
```

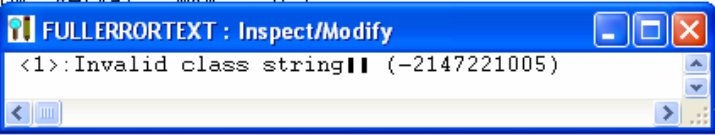
This firstly uses OleStatus() to get the error number

```
00080: documentHandle = ""
00081: myMerge
00082: wordApp
00083: document
00084: RetVal =
00085: Return
00086:
00087:
00088: CheckError:
00089: Debug
00090: FullErrorText = ""
00091: OleStatus = OleStatus()
00092: If OleStatus Then
00093:     FullErrorText = ZZx_Utility("WINERROR", OleStatus)
00094: End
00095:
00096: Return
00097:
```



Then uses our in house utility to get a meaningful description

```
00083: documentHandle2 = ""
00084: RetVal = Msg(@Window, RetVal, @Em, "D")
00085: Return
00086:
00087:
00088: CheckError:
00089: Debug
00090: FullErrorText = ""
00091: OleStatus = OleStatus()
00092: If OleStatus Then
00093:     FullErrorText = ZZx_Utility("WINERROR", OleStatus)
00094: End
00095:
00096: Return
```



Of course if you don't have our utility you can always Google the number and get something like this

Never seen it before but it is perfectly reasonable behaviour and works the same in all versions of VBScript. -2147221005 is the ole automation error number for Invalid class string.

-2147221005 (800401F3) Invalid class string.

Right so now we've shown how the error works let's guide you through the process of constructing the merge.

Remember we're now dealing with instantiating Windows OLE Components so at every stage we'll be making sure we get rid of what we've created.

SPREZZATURA

What we'll do is create a routine called ZZ_OLE_Word_Automation which takes four parameters – the name of the document to merge, the name of the data file to merge, the name of the file to save the resultant merge document as (if blank it won't be saved) and finally a flag to indicate whether the merged document should be printed and closed or left on the screen.

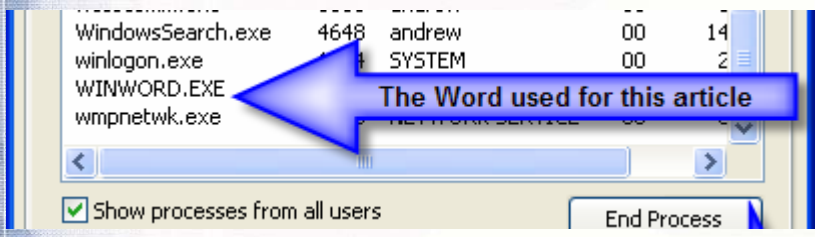
Starting the OLE Application

To start an OLE application so that we can control it we use the **OleCreateInstance** routine. Note that in line with what we've just said we'll be destroying the object after creating it. So that we can actually track what's going on we'll incorporate strategic DEBUG statements. So the first section of code looks like the following. Note that we're going to debug after the create instance to check that it worked and then again after the QUIT command. Stating the obvious – the OleCreateInstance creates an instance of the Word Application and returns the *handle* in wordApp. We then subsequently use this *handle* to tell Windows which application to close.

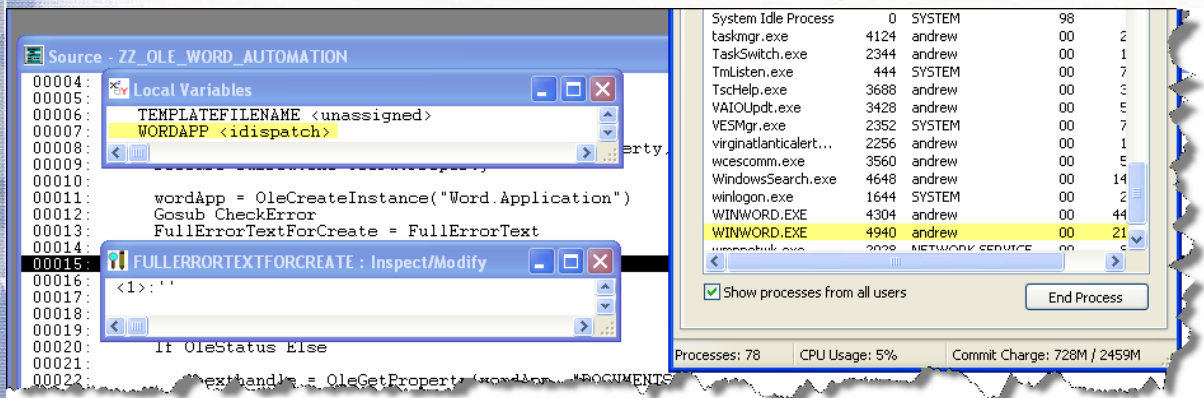
```

wordApp = OleCreateInstance("Word.Application")
Gosub CheckError
FullErrorTextForCreate = FullErrorText
Debug
Quit = OleCallMethod(wordApp, "QUIT")
Gosub CheckError
FullErrorTextForQuit = FullErrorText
Debug
Return
    
```

So before we run this program the relevant section of the task manager looks like this



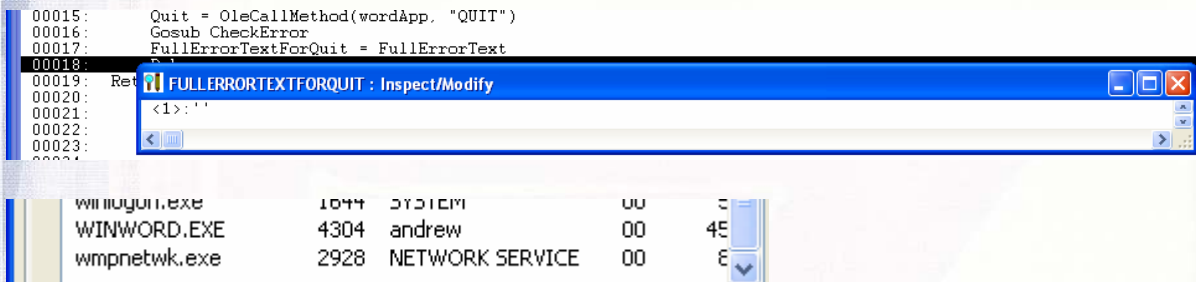
So running this and stopping after the creation of the OLE instance and examining the relevant variables we get



SPREZZATURA

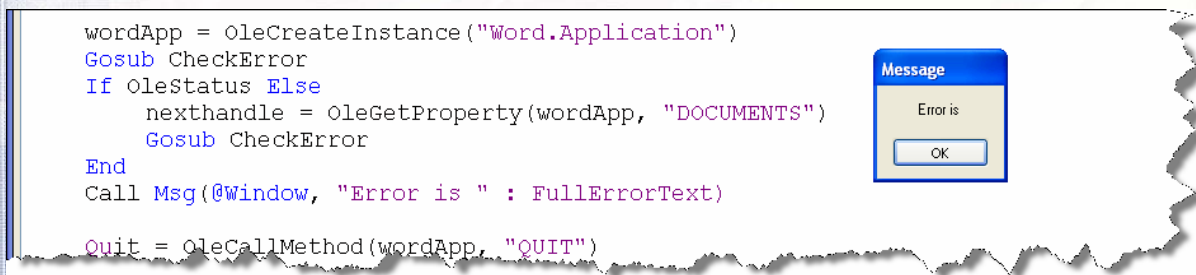
Note that at this stage wordApp contains an "idispatch" pointer – this is an internal reference which basically confirms that the operation worked. You can't use the debugger to inspect the variable however. This shows that we have successfully "instantiated" Word for Windows as does the second "WINWORD.EXE" in the task manager.

Now we move on to call the "QUIT" method against the handle for the word application and sure enough no errors register and the second copy of WINWORD disappears from the task manager.

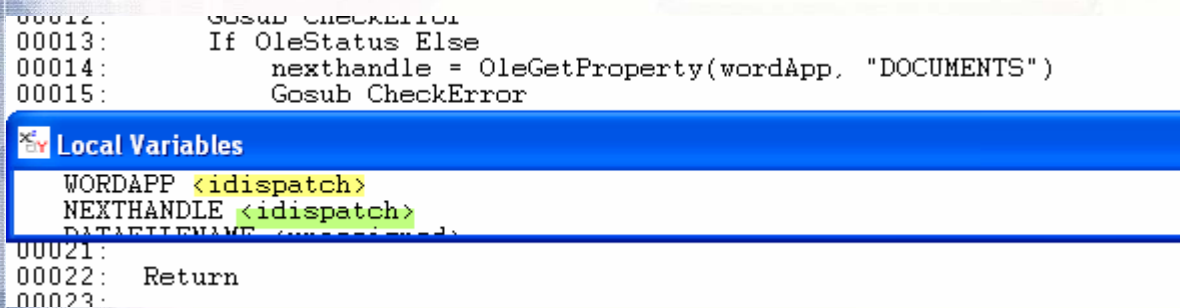


So our routine is successfully starting and stopping Word!

Next up we have to get a handle to the DOCUMENTS property of the Word application so that we can give it instructions. To understand all of this you're really going to have to become familiar with the Object Model of the Word application. As ever MSDN is your friend.



So now we have TWO idispatch pointers – one for the Word application and one for the Documents property of the Word Application



So we're still making progress a step at a time.

SPREZZATURA

Let's now try opening the appropriate document – we'll modify the code to do the open

```

11 OleStatus Else
    nexthandle = OleGetProperty(wordApp, "DOCUMENTS")
    Gosub CheckError
End
If OleStatus Else
    documentHandle = OleCallMethod(nextHandle, "OPEN", MergeDocument)
    Gosub CheckError
End
Debug
    
```

And run the program telling it which file to open



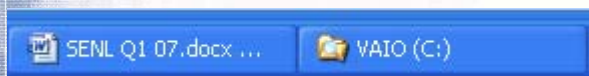
Local Variables

```

WORDAPP <idispatch>
NEXTHANDLE <idispatch>
DOCUMENTHANDLE <idispatch>
DATAFILENAME <unassigned>
DATASOURCE ''
DOCUMENTHANDLE2 <unassigned>
FILETOSAVEAS <unassigned>
FULLERRORETEXT ''
MERGEDOCUMENT 'C:\AREVM.DOC'
MYMERGE <unassigned>
OLEDISPATCH <unassigned>
OLESTATUS '0'
OPTIONS ''
PARAM1 <unassigned>
PARAM2 <unassigned>
PARAM3 <unassigned>
QUIT <unassigned>
        
```

Image Name	PID	User Name	CPU	Mem
svchost.exe	1988	NETWORK SERVICE	00	4
Switcher.exe	3272	andrew	00	10
System	4	SYSTEM	00	
System Idle Process	0	SYSTEM	91	
taskmgr.exe	4124	andrew	02	3
TaskSwitch.exe	2344	andrew	00	1
TmListen.exe	444	SYSTEM	00	7
TscHelp.exe	3688	andrew	00	3
VAIOUptd.exe	3428	andrew	00	5
VESMgr.exe	2352	SYSTEM	00	7
virginatlanticalert...	2256	andrew	00	1
wcescomm.exe	3560	andrew	00	5
WindowsSearch.exe	4648	andrew	00	14
winlogon.exe	1644	SYSTEM	00	5
WINWORD.EXE	240	andrew	00	35
WINWORD.EXE	4304	andrew	00	37

But painfully Word ISN'T on the task bar so we can't see if it worked or not!



So the next trick – solely for debugging purposes – is to tell Word that we want to be able to see it! By default OLE objects we instantiate don't appear unless we ask them too so let's add that in :-

```

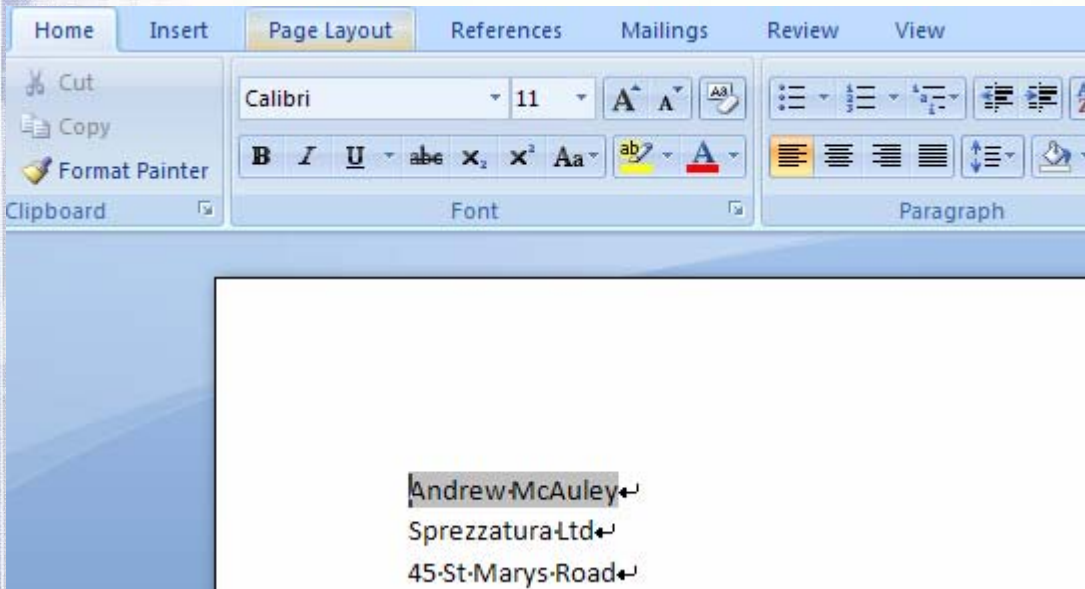
00016: End
00017: If OleStatus Else
00018:     OlePutProperty(wordApp, "VISIBLE", 1)
00019:     Gosub CheckError
00020: End
00021: If OleStatus Else
00022:     documentHandle = OleCallMethod(nextHandle, "OPEN", MergeDocument)
00023:     Gosub CheckError
00024: End
00025:
    
```

SPREZZATURA

We've asked it to be "VISIBLE" and sure enough it's on the task bar



And visible



OK so to review – from within OI we're launching Word and opening a document. Now to do the mail merge...

Hopefully by now you're starting to trust that we know what we're talking about – so having opened the correct document to mail merge we need to grab the handle to the MAILMERGE property so that we can tell it what the data source of the MAILMERGE object is :-

```

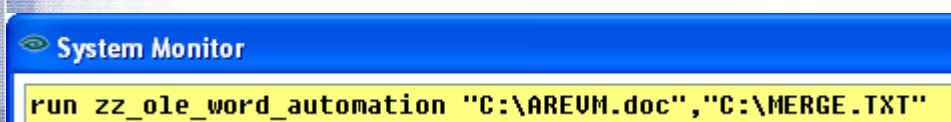
If OLEstatus Else
    myMerge = OleGetProperty(documentHandle, "MAILMERGE")
    Gosub CheckError
End

```

once

again we check that all is OK and if it is we set the data source

This time we run passing in the data source



And again all runs OK

SPREZZATURA

```

Local Variables
-----
DATASOURCE 'C:\MERGE.TXT'
FULLERRORETEXT ''
RES ''
DATAFILENAME (unassigned)
00029:      End
00030:
00031:      If OLEStatus Else
00032:          res = OleCallMethod(myMerge, "OPENDATASOURCE", DataSource
00033:          Gosub CheckError

```

No errors.

So we now need to tell the mail merge where it should create its output – should it go to the printer or to a new document?

```

If OLEStatus Else
    res = OleCallMethod(myMerge, "OPENDATASOURCE", DataSource)
    Gosub CheckError
End

If OLEStatus Else
    OLEPutProperty(myMerge, "DESTINATION", 0)
    Gosub CheckError
End

```

Of course to understand what we're looking for we need to use the object browser to see what the possible values are for destination and choose accordingly

Classes	Members of 'WdMailMergeDestination'
WdGranularity	wdSendToEmail
WdGutterStyle	wdSendToFax
WdGutterStyleOld	wdSendToNewDocument
WdHeaderFooterInd	wdSendToPrinter
WdHeadingSeparat	

Const **wdSendToNewDocument** = 0

And now that's all set up we tell it to actually DO the mail merge

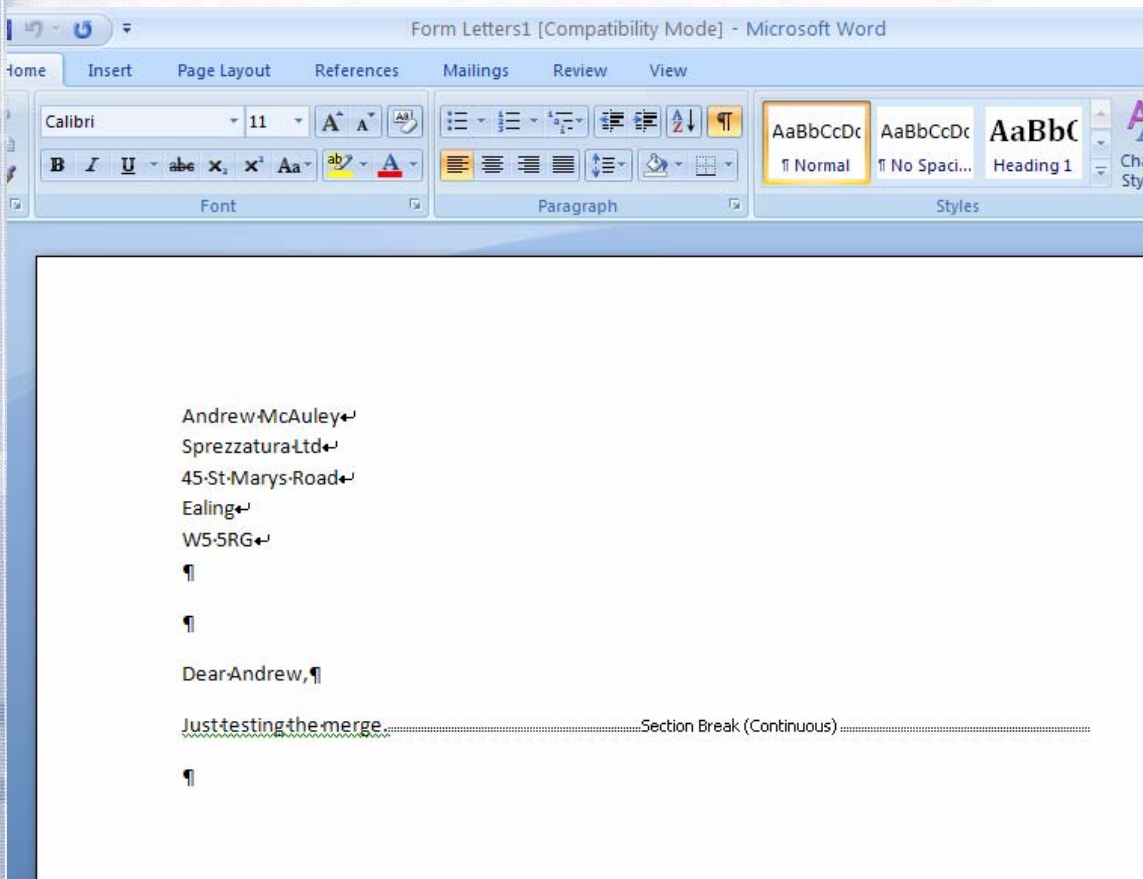
SPREZZATURA

```

If OLEstatus Else
    OLEPutProperty(myMerge, "DESTINATION", 0)
    Gosub CheckError
End

If OLEstatus Else
    res = OleCallMethod(myMerge, "EXECUTE")
    Gosub CheckError
End
    
```

Note that this time the merge document is actually created on screen



Now at this point we have two documents – the mail merge document and the results thereof. So firstly let’s close the merge document – we know the handle for this as we explicitly opened it :-

SPREZZATURA

```

If OLEStatus Else
    res = OleCallMethod(myMerge, "EXECUTE")
    Gosub CheckError
End

If OLEStatus Else
    res2 = OleCallMethod(documentHandle, "CLOSE", 0)
    Gosub CheckError
End

```

And now that is closed there is only one document open – the new one we have just mail merged to. The problem is that we don't have any kind of reference for this document. The good news is that as it is the only Word document left it has become the "Active" document – so we can get a handle for it that way :-

```

If OLEStatus Else
    documentHandle2 = OleGetProperty(wordApp, "ACTIVEDOCUMENT")
    Gosub CheckError
End

```

Having this handle we can now save the results file under another name if this was required :-

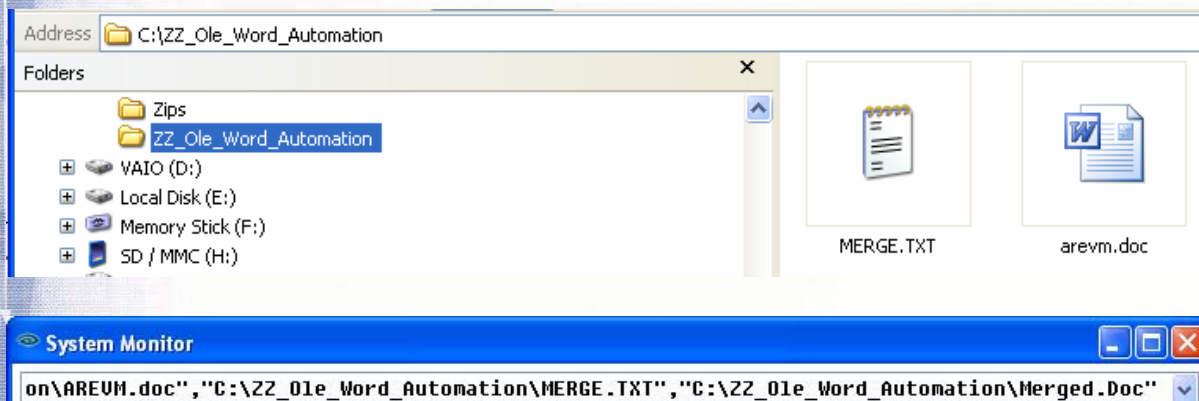
```

If OLEStatus Else
    documentHandle2 = OleGetProperty(wordApp, "ACTIVEDOCUMENT")
    Gosub CheckError
End

If OLEStatus Else
    If Len(FileToSaveAs) Then
        res3 = OleCallMethod(documentHandle2, "SAVEAS", FileToSaveAs)
        Gosub CheckError
    End
End

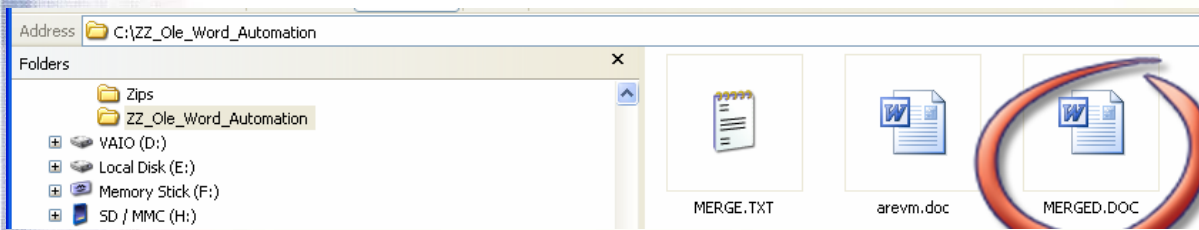
```

We'll modify the call slightly to isolate the file being created. So we'll move our test files to a new subdirectory and make the calls accordingly



SPREZZATURA

So we're passing in a file to save as. We run this and sure enough



As we've opted to use the destination of a new merged document we'll need to print the document directly. By recording a Macro we can easily see that the required command is PRINTOUT so we can modify our code accordingly

```

If Len(FileToSaveAs) Then
    res3 = OleCallMethod(documentHandle2, "SAVEAS", FileToSaveAs)
    Gosub CheckError
End
End

If OLEstatus Else
    If PrintAndClose Then
        result = OleCallMethod(wordApp, "PRINTOUT",1)
        Gosub CheckError
        Quit = OleCallMethod(wordApp, "QUIT")
    End
End

```

And when we run our code now using



Sure enough we get a saved document AND a printout. The final piece of the puzzle is to change our VISIBLE statement to be conditional upon the PrintAndClose flag :-

```

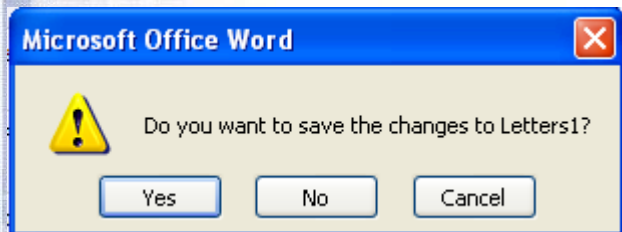
If OLEstatus Else
    If PrintAndClose Else
        OlePutProperty(wordApp, "VISIBLE", 1)
        Gosub CheckError
    End
End

```



SPREZZATURA

Now if we run with this we get the printout **BUT** we have one final hurdle. The merged document has changed – so Word asks us if we want to save the changed document. This *forces* the process to become visible to the user.



Naturally enough once more the object browser and the online help come to our rescue. Looking up the help for QUIT we see

Word Developer Reference

Application.Quit Method

Quits Microsoft Office Word and optionally saves or routes the open documents.

expression.Quit(SaveChanges, Format, RouteDocument)

expression Required. A variable that represents an [Application](#) object.

Parameters

Name	Required/Optional	Data Type	Description
<i>SaveChanges</i>	Optional	VARIANT	Specifies whether Word saves changed documents before closing. Can be one of the WdSaveOptions constants.

And using the object browser to examine the wdSaveOptions constants we get

The screenshot shows the Visual Studio Object Browser. The 'Library' is set to 'Word' and the 'Class' is 'WdSaveOptions'. The 'Members of WdSaveOptions' list includes:

- wdDoNotSaveChanges
- wdPromptToSaveChanges
- wdSaveChanges

At the bottom, a tooltip for the constant `Const wdDoNotSaveChanges = 0` is shown, indicating it is a member of `Word.WdSaveOptions`.

SPREZZATURA

So we can modify our code to be as follows :-

```

If OLEstatus Else
  If PrintAndClose Then
    result = OleCallMethod(wordApp, "PRINTOUT",1)
    Gosub CheckError
    Quit = OleCallMethod(wordApp, "QUIT",0)
  End
End
End

```

And running the code seamlessly prints our merge to the printer.

Conclusion

Hopefully in the course of these two articles we've been able to show that using OLE can be a lot easier than it at first appears. Naturally there are additional complexities – this is a beginner's tutorial after all – but as you become more familiar with the medium and the support available to you with online resources we're sure that you'll find that you can become effective. And if you are stuck then don't forget that here at Sprezz Towers we're always ready to provide cost effective Pay as You Go support!

The complete code listing for this exercise follows :-

```

Subroutine ZZ_OLE_Word_Automation(MergeDocument, DataSource, FileToSaveAs,
PrintAndClose)
/*
  Author      AMcA
  Date        March 2007
  Purpose     To automate a mail merge using OLE
              MergeDocument -> Name of the document To merge
              DataSource      -> Name of the data file To merge from
              FileToSaveAs    -> If this contains a filename the resultant
                              merge will be saved as this
              PrintAndClose  -> If true the document will be printed Then
                              Word will be closed. For the purposes of this
                              routine If this is true Then Word will not
                              be made visible
*/

Declare Function OleCreateInstance, OleGetProperty, OleStatus, zzx_Utility,
Msg
Declare Subroutine OlePutProperty

wordApp = OleCreateInstance("Word.Application")
Gosub CheckError
If OleStatus Else

```

SPREZZATURA

```
nexthandle = OleGetProperty(wordApp, "DOCUMENTS")
Gosub CheckError

End

If OLEStatus Else
    If PrintAndClose Else
        OlePutProperty(wordApp, "VISIBLE", 1)
        Gosub CheckError
    End
End

If OleStatus Else
    documentHandle = OleCallMethod(nextHandle, "OPEN", MergeDocument)
    Gosub CheckError
End

If OLEStatus Else
    myMerge = OleGetPProperty(documentHandle, "MAILMERGE")
    Gosub CheckError
End

If OLEStatus Else
    res = OleCallMethod(myMerge, "OPENDATASOURCE", DataSource)
    Gosub CheckError
End

If OLEStatus Else
    OLEPutProperty(myMerge, "DESTINATION", 0)
    Gosub CheckError
End

If OLEStatus Else
    res = OleCallMethod(myMerge, "EXECUTE")
    Gosub CheckError
End

If OLEStatus Else
    res2 = OleCallMethod(documentHandle, "CLOSE", 0)
    Gosub CheckError
End

If OLEStatus Else
    documentHandle2 = OleGetProperty(wordApp, "ACTIVEDOCUMENT")
    Gosub CheckError
End

If OLEStatus Else
    If Len(FileToSaveAs) Then
```

SPREZZATURA

```
res3 = OleCallMethod(documentHandle2, "SAVEAS", FileToSaveAs)
Gosub CheckError
```

```
End
```

```
End
```

```
If OleStatus Else
```

```
  If PrintAndClose Then
```

```
    result = OleCallMethod(wordApp, "PRINTOUT", 1)
```

```
    Gosub CheckError
```

```
    Quit = OleCallMethod(wordApp, "QUIT", 0)
```

```
  End
```

```
End
```

```
Return
```

```
CheckError:
```

```
FullErrorText = ""
```

```
OleStatus = OleStatus()
```

```
If OleStatus Then
```

```
  FullErrorText = ZZx_Utility("WINERROR", OleStatus)
```

```
End
```

```
Return
```



Changing Universal Driver Files to 2.1 Format – Aaron Kaplan

The release of the Universal Driver 3.0 gave Linear Hash the ability to use larger frame sizes and larger OS file sizes. To accommodate these changes, modifications to the primary frame header were required. These changes made files created with the UD3 unusable with earlier drivers. Sometimes developers have reason to access files created with the UD3 on earlier systems. Perhaps they are sharing data with systems that have moved to the UD, they are receiving deployments from people or they have reverted to a prior version because of [DEP issues](#). As promised, we have created a utility to allow you to revert these files back to a pre UD3 header.

```
success = zz_revertLHfiles( directory, osErrorList, skippedFileList )
```

Parameters

Input	<table border="1"> <tr> <td data-bbox="335 806 574 952">Directory</td> <td data-bbox="574 806 1436 952">This contains a directory path to convert. All LK files in this directory will be examined for conversion</td> </tr> </table>	Directory	This contains a directory path to convert. All LK files in this directory will be examined for conversion																										
Directory	This contains a directory path to convert. All LK files in this directory will be examined for conversion																												
Output	<table border="1"> <tr> <td data-bbox="335 952 574 1433"> osErrorList </td> <td data-bbox="574 952 1436 1433"> Returns an @FM delimited list of files that could not be processed because access was denied by the operating system. Each file returned has an @VM delimited status list <table border="1"> <tr> <td data-bbox="574 952 686 1433">< 1, 1 > Error Type - Defines the type of OS Error that occurred</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td>equ ERR_OSOPEN_ERR\$</td> <td>to 1</td> </tr> <tr> <td>equ ERR_OSREAD_ERR\$</td> <td>to 2</td> </tr> <tr> <td>equ ERR_OSWRITE_ERR\$</td> <td>to 3</td> </tr> </table> <table border="1"> <tr> <td data-bbox="574 952 686 1433">< 1, 2 > OS File Name (REV12345.LK)</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td data-bbox="574 952 686 1433">< 1, 3 > Frame number, if applicable</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td data-bbox="574 952 686 1433">< 1, 4 > Error code as returned from status()</td> <td data-bbox="686 952 1436 1433"></td> </tr> </table> Please note that files in this list with write errors may now be corrupt. </td> </tr> <tr> <td data-bbox="335 1433 574 1798"> SkippedFileList </td> <td data-bbox="574 1433 1436 1798"> Returns an @FM delimited list of files that could not be processed because access was denied by the operating system. Each file returned has an @VM delimited status list <table border="1"> <tr> <td data-bbox="574 1433 686 1798">< 1, 1 > OS File Name (REV12345.LK)</td> <td data-bbox="686 1433 1436 1798"></td> </tr> <tr> <td data-bbox="574 1433 686 1798">< 1, 2 > Reason this file was not converted.</td> <td data-bbox="686 1433 1436 1798"></td> </tr> <tr> <td>equ ERR_FRAME_SIZE_TO_LARGE\$</td> <td>to 1</td> </tr> <tr> <td>equ ERR_FRAME_HAS_EXT_INFO\$</td> <td>to 2</td> </tr> <tr> <td>equ ERR_FRAME_NOT_UD3\$</td> <td>to 3</td> </tr> </table> </td> </tr> </table>	osErrorList	Returns an @FM delimited list of files that could not be processed because access was denied by the operating system. Each file returned has an @VM delimited status list <table border="1"> <tr> <td data-bbox="574 952 686 1433">< 1, 1 > Error Type - Defines the type of OS Error that occurred</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td>equ ERR_OSOPEN_ERR\$</td> <td>to 1</td> </tr> <tr> <td>equ ERR_OSREAD_ERR\$</td> <td>to 2</td> </tr> <tr> <td>equ ERR_OSWRITE_ERR\$</td> <td>to 3</td> </tr> </table> <table border="1"> <tr> <td data-bbox="574 952 686 1433">< 1, 2 > OS File Name (REV12345.LK)</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td data-bbox="574 952 686 1433">< 1, 3 > Frame number, if applicable</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td data-bbox="574 952 686 1433">< 1, 4 > Error code as returned from status()</td> <td data-bbox="686 952 1436 1433"></td> </tr> </table> Please note that files in this list with write errors may now be corrupt.	< 1, 1 > Error Type - Defines the type of OS Error that occurred		equ ERR_OSOPEN_ERR\$	to 1	equ ERR_OSREAD_ERR\$	to 2	equ ERR_OSWRITE_ERR\$	to 3	< 1, 2 > OS File Name (REV12345.LK)		< 1, 3 > Frame number, if applicable		< 1, 4 > Error code as returned from status()		SkippedFileList	Returns an @FM delimited list of files that could not be processed because access was denied by the operating system. Each file returned has an @VM delimited status list <table border="1"> <tr> <td data-bbox="574 1433 686 1798">< 1, 1 > OS File Name (REV12345.LK)</td> <td data-bbox="686 1433 1436 1798"></td> </tr> <tr> <td data-bbox="574 1433 686 1798">< 1, 2 > Reason this file was not converted.</td> <td data-bbox="686 1433 1436 1798"></td> </tr> <tr> <td>equ ERR_FRAME_SIZE_TO_LARGE\$</td> <td>to 1</td> </tr> <tr> <td>equ ERR_FRAME_HAS_EXT_INFO\$</td> <td>to 2</td> </tr> <tr> <td>equ ERR_FRAME_NOT_UD3\$</td> <td>to 3</td> </tr> </table>	< 1, 1 > OS File Name (REV12345.LK)		< 1, 2 > Reason this file was not converted.		equ ERR_FRAME_SIZE_TO_LARGE\$	to 1	equ ERR_FRAME_HAS_EXT_INFO\$	to 2	equ ERR_FRAME_NOT_UD3\$	to 3
osErrorList	Returns an @FM delimited list of files that could not be processed because access was denied by the operating system. Each file returned has an @VM delimited status list <table border="1"> <tr> <td data-bbox="574 952 686 1433">< 1, 1 > Error Type - Defines the type of OS Error that occurred</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td>equ ERR_OSOPEN_ERR\$</td> <td>to 1</td> </tr> <tr> <td>equ ERR_OSREAD_ERR\$</td> <td>to 2</td> </tr> <tr> <td>equ ERR_OSWRITE_ERR\$</td> <td>to 3</td> </tr> </table> <table border="1"> <tr> <td data-bbox="574 952 686 1433">< 1, 2 > OS File Name (REV12345.LK)</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td data-bbox="574 952 686 1433">< 1, 3 > Frame number, if applicable</td> <td data-bbox="686 952 1436 1433"></td> </tr> <tr> <td data-bbox="574 952 686 1433">< 1, 4 > Error code as returned from status()</td> <td data-bbox="686 952 1436 1433"></td> </tr> </table> Please note that files in this list with write errors may now be corrupt.	< 1, 1 > Error Type - Defines the type of OS Error that occurred		equ ERR_OSOPEN_ERR\$	to 1	equ ERR_OSREAD_ERR\$	to 2	equ ERR_OSWRITE_ERR\$	to 3	< 1, 2 > OS File Name (REV12345.LK)		< 1, 3 > Frame number, if applicable		< 1, 4 > Error code as returned from status()															
< 1, 1 > Error Type - Defines the type of OS Error that occurred																													
equ ERR_OSOPEN_ERR\$	to 1																												
equ ERR_OSREAD_ERR\$	to 2																												
equ ERR_OSWRITE_ERR\$	to 3																												
< 1, 2 > OS File Name (REV12345.LK)																													
< 1, 3 > Frame number, if applicable																													
< 1, 4 > Error code as returned from status()																													
SkippedFileList	Returns an @FM delimited list of files that could not be processed because access was denied by the operating system. Each file returned has an @VM delimited status list <table border="1"> <tr> <td data-bbox="574 1433 686 1798">< 1, 1 > OS File Name (REV12345.LK)</td> <td data-bbox="686 1433 1436 1798"></td> </tr> <tr> <td data-bbox="574 1433 686 1798">< 1, 2 > Reason this file was not converted.</td> <td data-bbox="686 1433 1436 1798"></td> </tr> <tr> <td>equ ERR_FRAME_SIZE_TO_LARGE\$</td> <td>to 1</td> </tr> <tr> <td>equ ERR_FRAME_HAS_EXT_INFO\$</td> <td>to 2</td> </tr> <tr> <td>equ ERR_FRAME_NOT_UD3\$</td> <td>to 3</td> </tr> </table>	< 1, 1 > OS File Name (REV12345.LK)		< 1, 2 > Reason this file was not converted.		equ ERR_FRAME_SIZE_TO_LARGE\$	to 1	equ ERR_FRAME_HAS_EXT_INFO\$	to 2	equ ERR_FRAME_NOT_UD3\$	to 3																		
< 1, 1 > OS File Name (REV12345.LK)																													
< 1, 2 > Reason this file was not converted.																													
equ ERR_FRAME_SIZE_TO_LARGE\$	to 1																												
equ ERR_FRAME_HAS_EXT_INFO\$	to 2																												
equ ERR_FRAME_NOT_UD3\$	to 3																												

Please use caution when using this utility and ensure you have a full backup of your system before running this against any active files. Files may be locked at the OS level by the NT service, so you should stop the service before running this program. Additionally, all users should be out of the system when the files are converting. The logged in network user will require full access to the LK and OV files at the operating system level.

The Checkout for this utility can be found at <http://www.sprezzatura.com/downloads/revertlhfiles.zip>



Peripheral Trivia

As this issue of S/ENL was put to bed we fed the inner man with:

TV:	Heroes again!
Book:	Pro Visual C++/CLI and the .NET 2.0 Platform. Stephen R G Fraser
CD:	Three Dead Trolls In a Baggie – “Skit Happens”
WEB:	http://www.prosperon.co.uk/lansurveyor-network-maps.htm

Join us :	Send Mail to Admin@Sprezzatura.com with subject SUBSCRIBE SENL or complete our online registration form .
Leave Us:	Send Mail to Admin@Sprezzatura.com with subject UNSUBSCRIBE SENL
Change of Address:	Leave at the old address & join at the new one
Web Info:	http://www.sprezzatura.com/
Tell us what you'd like to see in S/ENL:	info@sprezzatura.com

COPYRIGHT NOTICE

S/ENL - ©2007 Sprezzatura Ltd. All rights reserved.. Portions copyright Microsoft Corporation Inc. Portions copyright Revelation Technologies, Inc. No portion of this journal (other than code segments) may be reproduced by any means, be it photocopied, digitised, transcribed, transmitted, reduced to any electronic medium or machine readable form, nor translated into any other language without the prior written consent of Sprezzatura Ltd or Sprezzatura, Inc. The moral rights of the authors have been asserted.

Disclaimer - Whilst every effort is made to ensure the accuracy of the information contained herein, neither Sprezzatura Ltd nor Sprezzatura Inc. can accept liability for the failure of anything documented herein to work nor for damage resulting from the application of methods/techniques learned herein.

TRADEMARK NOTICE

OpenInsight is a trademark of Revelation Technologies Inc. trading as Revelation Software. Microsoft, Windows™, and MS-DOS are registered trademarks of Microsoft Corporation. All other product names are trademarks or registered trademarks of their respective owners. Printed in the United Kingdom.

S/ENL Volume 4 Issue 4, 10th March 2007.

Please encourage your correspondents to send e-mail to admin@sprezzatura.com with SUBSCRIBE SENL in the subject line to get their own free subscription. Everyone is welcome! Tell your friends about S/ENL.