Appendix A: Microsoft Word Automation Program

This article was contributed by Poonam Bajaj.
http://www.codeguru.com/interface/WordAutomation.html

Environment: VC++ 6.0, Windows 2000 with MS-Office installed.

In this article, I will describe the implementation of a C++ class that automates Microsoft Word application. The class uses low-level COM and is quite handy in its usage. The idea came into my mind when I answered two queries on this topic on a discussion forum in just one day!

I named the class 'CAutoWord'. It can open a Word documment file and print its contents on the default printer. Everything works in the background. A typical application can be an NT service which generally runs without any UI. It has following exposed methods.

1. CAutoWord() : Constructor function that initializes COM.
2. int InitAutomation() : Initializes the automation. In case of failure, it returns -1 else 0 is returned.
3. int PrintDocument() : This function will load the file and print the contents on the default printer. Negative return value indicates that an error occurred.

More functions can be added using the same framework.

Usage Scenario

The code snippet given below demonstrates its usage.

```
#include "AutoWord.h"

int PrintWordDocument(CString szFilePath)
{
 // Instantiate an object of CAutoWord
 CAutoWord      AutoWord;

 // Set filepath
 int iRetVal = AutoWord.InitAutomation();
 if (iRetVal != 0)
 {
  printf("LoadDocument operation failed.");
 }

 // Set filepath
 char* strFilePath = szFilePath.GetBuffer(szFilePath.
                                          GetLength));

 // Print the WORD document
 iRetVal = AutoWord.PrintDocument(strFilePath);
 if (iRetVal == 0)
 {
```

```
  AfxMessageBox(szFilePath+" was sent to printer.");
 }
 else
 {
  AfxMessageBox("Print operation failed.");
 }

 return iRetVal;
}
```

Implementation

Following is CAutoWord class declaration :

```
#include <oaidl.h>

class CAutoWord
{
 void Destroy(); // Cleaup function

public:
 // Opens, Prints and closes a document
 int PrintDocument(char* strFilepath);

 // Initializes the automation class and prepares
 // it for use.
 int InitAutomation();

 // Constructor : Initializes COM libraries
 CAutoWord();

 // Destructor : Calls Detroy() and UnInitializes COM.
 virtual ~CAutoWord();

private:
 // Keeps value of Dispatch pointer to
 // 'Word.Application' instance
 IDispatch* m_pDispApp;

 // Keeps a pointer to 'Documents' property of
 // m_pDispApp interface.
 IDispatch* m_pDocuments;
};
```

Most of the work is done InitAutomation() and PrintDocument() functions.

InitAutomation() creates an instance of the 'Word.Application' object and obtains the pointer to the instance's IUnknown interface.

```
IUnknown* pUnk;
HRESULT hr = ::CoCreateInstance( clsid,
                                 NULL,
                                 CLSCTX_LOCAL_SERVER,
                                 IID_IUnknown,
                                 (void**) &pUnk);
```

Calling QueryInterface() on IUnknown gives a pointer to the IDispatch interface (m_pDispApp).

```
hr = pUnk->QueryInterface(IID_IDispatch,
                          (void**)&m_pDispApp);
```

Using GetIDsOfNames(), we get Dispatch Id of 'Documents' property of m_pDispApp interface.

```
LPOLESTR szDoc = L"Documents";
hr = m_pDispApp->GetIDsOfNames(IID_NULL,
                               &szDoc,
                               1,
                               LOCALE_SYSTEM_DEFAULT,
                               &dispID);
```

Now, Invoke() function is called that gives us Dispatch pointer to its 'Documents' property. The pointer is saved in m_pDocuments member variable.

```
hr = m_pDispApp->Invoke(dispID,
                        IID_NULL,
                        LOCALE_SYSTEM_DEFAULT,
                        DISPATCH_PROPERTYGET,
                        &dp,
                        &varRetVal,
                        NULL,
                        NULL);
...
m_pDocuments = varRetVal.pdispVal;
```

PrintDocument() method gets the Dispatch Id of 'Open' method of m_pDocuments interface. Then the invoke function is called with the Word Document's filepath as a parameter. This returns a pointer to the Dispatch interface to this Word document. We call it pDocument.

```
LPOLESTR szOpenDoc = L"Open";
HRESULT hr = m_pDocuments->GetIDsOfNames(IID_NULL,
                                         &szOpenDoc,
                                         1,
                                         LOCALE_SYSTEM_DEFAULT,
                                         &dispOpenID);
hr = m_pDocuments->Invoke(dispOpenID,
                          IID_NULL,
                          LOCALE_SYSTEM_DEFAULT,
                          DISPATCH_METHOD,
                          &dpOpen,
                          &varRetVal,
                          &excepInfo,
                          NULL);
...
IDispatch* pDocument = varRetVal.pdispVal;
```

After pDocument is obtained, it's 'PrintOut' method is called. This actually prints out the contents of the Word Document on the default printer and the document is closed.

```
LPOLESTR szPrintDoc = L"PrintOut";
hr = pDocument->GetIDsOfNames(IID_NULL,
                              &szPrintDoc,
                              1,
                              LOCALE_SYSTEM_DEFAULT,
                              &dispPrintID);
```

```
hr = pDocument->Invoke(dispPrintID,
                       IID_NULL,
                       LOCALE_SYSTEM_DEFAULT,
                       DISPATCH_METHOD,
                       &dpPrint,
                       &varRetVal,
                       NULL,
                       NULL);
...
LPOLESTR szCloseDoc = L"Close";
hr = pDocument->GetIDsOfNames(IID_NULL,
                              &szCloseDoc,
                              1,
                              LOCALE_SYSTEM_DEFAULT,
                              &dispCloseID);
hr = pDocument->Invoke(dispCloseID,
                       IID_NULL,
                       LOCALE_SYSTEM_DEFAULT,
                       DISPATCH_METHOD,
                       &dpClose,
                       &varRetVal,
                       &excepInfo,
                       NULL);
```

The **Destroy()** method simply quits the Word Application's instance.

```
LPOLESTR szQuit = L"Quit";
HRESULT hr = m_pDispApp->GetIDsOfNames(IID_NULL,
                                       &szQuit,
                                       1,
                                       LOCALE_SYSTEM_DEFAULT,
                                       &dispQuit);
hr = m_pDispApp->Invoke(dispQuit,
                        IID_NULL,
                        LOCALE_SYSTEM_DEFAULT,
                        DISPATCH_METHOD,
                        &dpQuit,
                        &varRetVal,
                        &excepInfo,
                        NULL);
```

The Demo project uses CAutoWord class to print the Word files.

Source code for CautoWord C++ class

```
// Word Automation Class in C++
//
// Author : Poonam Bajaj
//
// Date  : 24th December, 2000
//
// Header file : AutoWord.h

#include <ole2.h>
#include <comdef.h>

#include "AutoWord.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CAutoWord::CAutoWord()
{
        m_pDispApp = NULL;
        m_pDocuments = NULL;

   //Initialize the COM libraries
   ::CoInitialize(NULL);
}

CAutoWord::~CAutoWord()
{
        Destroy();

        // Uninitialize com libraries
        CoUninitialize();
}

// Sets the file path as specified by the caller.
int CAutoWord::InitAutomation()
{
        // Get the CLSID for Word's Application Object
        CLSID clsid;
        CLSIDFromProgID(L"Word.Application", &clsid);

        // Create an instance of the Word application and obtain the pointer
        // to the application's IUnknown interface
        IUnknown* pUnk;
        HRESULT hr = ::CoCreateInstance( clsid, NULL, CLSCTX_LOCAL_SERVER,
IID_IUnknown, (void**) &pUnk);
        if (FAILED(hr))
        {
                OutputDebugString("Error in creating Word application instance\n");
                Destroy();
                return -1;
        }

        // Query IUnknown to retrieve a pointer to the IDispatch interface
```

```
	hr = pUnk->QueryInterface(IID_IDispatch, (void**)&m_pDispApp);

	// Get pointer to Documents interface.
	DISPPARAMS dp = { NULL, NULL, 0, 0 };
	DISPID dispID;
	LPOLESTR szDoc = L"Documents";
	IDispatch* pDocuments = NULL;
	VARIANT varRetVal;
	hr = m_pDispApp->GetIDsOfNames(IID_NULL, &szDoc, 1,
LOCALE_SYSTEM_DEFAULT, &dispID);
	hr = m_pDispApp->Invoke(dispID, IID_NULL, LOCALE_SYSTEM_DEFAULT,
DISPATCH_PROPERTYGET, &dp, &varRetVal, NULL, NULL);
	if (!SUCCEEDED(hr))
	{
		Destroy();
		return -1;
	}

	// keep the Dispatch pointer in the member variable for later use.
	m_pDocuments = varRetVal.pdispVal;

	return 0;
}

// Prints the WORD document if everything goes fine
// otherwise error is returned
int CAutoWord::PrintDocument(char *strFilePath)
{
	// Open the document
	VARIANT varRetVal;
	EXCEPINFO excepInfo;		// this variable contains exception info if any Invoke call
fails
	VARIANTARG varg;
	varg.vt = VT_BSTR;
	varg.bstrVal = _bstr_t(strFilePath); // this is the MS-word document filename, must be
changed to a valid filename that exists on disk
	DISPPARAMS dpOpen = { &varg, NULL, 1, 0 };
	DISPID dispOpenID;
	LPOLESTR szOpenDoc = L"Open";
	HRESULT hr = m_pDocuments->GetIDsOfNames(IID_NULL, &szOpenDoc, 1,
LOCALE_SYSTEM_DEFAULT, &dispOpenID);
	hr = m_pDocuments->Invoke(dispOpenID, IID_NULL, LOCALE_SYSTEM_DEFAULT,
DISPATCH_METHOD, &dpOpen, &varRetVal, &excepInfo, NULL);
	if (FAILED(hr))
	{
		OutputDebugString("Error opening the document\n");
		Destroy();
		return -1;
	}

	IDispatch* pDocument = varRetVal.pdispVal;

	// Call PrintOut method of the opened Document
	DISPPARAMS dpPrint = { NULL, NULL, 0, 0 };
	DISPID dispPrintID;
	LPOLESTR szPrintDoc = L"PrintOut";
```

```
        hr = pDocument->GetIDsOfNames(IID_NULL, &szPrintDoc, 1,
LOCALE_SYSTEM_DEFAULT, &dispPrintID);
        hr = pDocument->Invoke(dispPrintID, IID_NULL, LOCALE_SYSTEM_DEFAULT,
DISPATCH_METHOD, &dpPrint, &varRetVal, NULL, NULL);
        if (FAILED(hr))
        {
                OutputDebugString("The document could not be printed\n");
                Destroy();
                return -1;
        }

        // Close the document now.
        DISPPARAMS dpClose = { NULL, NULL, 0, 0 };
        DISPID dispCloseID;
        LPOLESTR szCloseDoc = L"Close";
        hr = pDocument->GetIDsOfNames(IID_NULL, &szCloseDoc, 1,
LOCALE_SYSTEM_DEFAULT, &dispCloseID);
        hr = pDocument->Invoke(dispCloseID, IID_NULL, LOCALE_SYSTEM_DEFAULT,
DISPATCH_METHOD, &dpClose, &varRetVal, &excepInfo, NULL);

        return 0;
}

void CAutoWord::Destroy()
{
        // If just a fake call, return
        if (m_pDispApp == NULL)
                return;

        VARIANT varRetVal;
        EXCEPINFO excepInfo;          // this variable contains exception info if any Invoke call
fails

        // Variables to store 'Quit' related parameters
        DISPPARAMS dpQuit = { NULL, NULL, 0, 0 };
        DISPID dispQuit;
        LPOLESTR szQuit = L"Quit";

        //Quit the App
        HRESULT hr = m_pDispApp->GetIDsOfNames(IID_NULL, &szQuit, 1,
LOCALE_SYSTEM_DEFAULT, &dispQuit);
        hr = m_pDispApp->Invoke(dispQuit, IID_NULL, LOCALE_SYSTEM_DEFAULT,
DISPATCH_METHOD, &dpQuit, &varRetVal, &excepInfo, NULL);
        if (FAILED(hr))
        {
                OutputDebugString("Error in quitting WinWord.exe\n");
        }

        m_pDispApp = NULL;
}
```

AutoWord.h

```
// AutoWord.h: interface for the CAutoWord class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_AUTOWORD_H__01AFA8C4_EFCD_4EA8_87E2_972E79BBE65D__INCLU
DED_)
#define
AFX_AUTOWORD_H__01AFA8C4_EFCD_4EA8_87E2_972E79BBE65D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <oaidl.h>

class CAutoWord
{
        void Destroy();
public:
        int PrintDocument(char* strFilepath);
        int InitAutomation();
        CAutoWord();
        virtual ~CAutoWord();

private:
        IDispatch* m_pDispApp;
        IDispatch* m_pDocuments;
};

#endif
// !defined(AFX_AUTOWORD_H__01AFA8C4_EFCD_4EA8_87E2_972E79BBE65D__INCLUD
ED_)
```

Appendix B: Microsoft Excel Automation Using Visual Basic

Microsoft Knowledge Base Article – 219151
http://support.microsoft.com/default.aspx?kbid=219151

SUMMARY

This article demonstrates how to create and manipulate Excel by using Automation from Visual Basic.

MORE INFORMATION

There are two ways to control an Automation server: by using either late binding or early binding. With late binding, methods are not bound until run-time and the Automation server is declared as Object. With early binding, your application knows at design-time the exact type of object it will be communicating with, and can declare its objects as a specific type. This sample uses early binding, which is considered better in most cases because it affords greater performance and better type safety.

To early bind to an Automation server, you need to set a reference to that server's type library. In Visual Basic, this is done through the References dialog box found under the Project | References menu. For this sample, you will need to add a reference to the type library for Excel before you can run the code. See the steps below on how to add the reference.

Building the Automation Sample

1. Start Visual Basic and create a new Standard EXE project. Form1 is created by default.

2. Select Project|References to bring up the References dialog. Scroll down the list until you find "Microsoft Excel 10.0 Object Library" and select the item to add a reference to Excel 2002. If the item does not appear in the list, make sure you have Excel 2002 properly installed.

Note If you are automating Microsoft Office Excel 2003, the type library appears as "Microsoft Excel 11.0 Object Library" in the References list. If you are automating Microsoft Excel 2000, the type library appears as "Microsoft Excel 9.0 Object Library" in the References list , and if you have Microsoft Excel 97, it will appear as the "Microsoft Excel 8.0 Object Library."

3. Click OK to close the References dialog.

4. Add a CommandButton to Form1.

5. In the code window for Form1, insert the following code:

```
Option Explicit


Private Sub Command1_Click()

   Dim oXL As Excel.Application

   Dim oWB As Excel.Workbook

   Dim oSheet As Excel.Worksheet
```

```
    Dim oRng As Excel.Range


    'On Error GoTo Err_Handler

' Start Excel and get Application object.
    Set oXL = CreateObject("Excel.Application")
    oXL.Visible = True

' Get a new workbook.
    Set oWB = oXL.Workbooks.Add
    Set oSheet = oWB.ActiveSheet

' Add table headers going cell by cell.
    oSheet.Cells(1, 1).Value = "First Name"
    oSheet.Cells(1, 2).Value = "Last Name"
    oSheet.Cells(1, 3).Value = "Full Name"
    oSheet.Cells(1, 4).Value = "Salary"


' Format A1:D1 as bold, vertical alignment = center.
    With oSheet.Range("A1", "D1")
        .Font.Bold = True
        .VerticalAlignment = xlVAlignCenter
    End With
```

```
' Create an array to set multiple values at once.
   Dim saNames(5, 2) As String
   saNames(0, 0) = "John"
   saNames(0, 1) = "Smith"
   saNames(1, 0) = "Tom"
   saNames(1, 1) = "Brown"
   saNames(2, 0) = "Sue"
   saNames(2, 1) = "Thomas"
   saNames(3, 0) = "Jane"

   saNames(3, 1) = "Jones"
   saNames(4, 0) = "Adam"
   saNames(4, 1) = "Johnson"

' Fill A2:B6 with an array of values (First and Last Names).
   oSheet.Range("A2", "B6").Value = saNames

' Fill C2:C6 with a relative formula (=A2 & " " & B2).
   Set oRng = oSheet.Range("C2", "C6")
   oRng.Formula = "=A2 & "" "" & B2"

' Fill D2:D6 with a formula(=RAND()*100000) and apply format.
   Set oRng = oSheet.Range("D2", "D6")
   oRng.Formula = "=RAND()*100000"
```

```
    oRng.NumberFormat = "$0.00"

  ' AutoFit columns A:D.
    Set oRng = oSheet.Range("A1", "D1")
    oRng.EntireColumn.AutoFit

  ' Manipulate a variable number of columns for Quarterly Sales
Data.
    Call DisplayQuarterlySales(oSheet)

  ' Make sure Excel is visible and give the user control
  ' of Microsoft Excel's lifetime.
    oXL.Visible = True
    oXL.UserControl = True

  ' Make sure you release object references.
    Set oRng = Nothing
    Set oSheet = Nothing
    Set oWB = Nothing
    Set oXL = Nothing

Exit Sub
Err_Handler:
  MsgBox Err.Description, vbCritical, "Error: " & Err.Number
End Sub
```

```
Private Sub DisplayQuarterlySales(oWS As Excel.Worksheet)

    Dim oResizeRange As Excel.Range

    Dim oChart As Excel.Chart

    Dim iNumQtrs As Integer

    Dim sMsg As String

    Dim iRet As Integer


  ' Determine how many quarters to display data for.

    For iNumQtrs = 4 To 2 Step -1

      sMsg = "Enter sales data for" & Str(iNumQtrs) & "
quarter(s)?"

      iRet = MsgBox(sMsg, vbYesNo Or vbQuestion _

        Or vbMsgBoxSetForeground, "Quarterly Sales")

      If iRet = vbYes Then Exit For

    Next iNumQtrs



    sMsg = "Displaying data for" & Str(iNumQtrs) & " quarter(s)."

    MsgBox sMsg, vbMsgBoxSetForeground, "Quarterly Sales"


  ' Starting at E1, fill headers for the number of columns
selected.

    Set oResizeRange = oWS.Range("E1",
"E1").Resize(ColumnSize:=iNumQtrs)


    oResizeRange.Formula = "=""Q"" & COLUMN()-4 & CHAR(10) &
""Sales"""
```

```
    ' Change the Orientation and WrapText properties for the
headers.

      oResizeRange.Orientation = 38

      oResizeRange.WrapText = True


    ' Fill the interior color of the headers.

      oResizeRange.Interior.ColorIndex = 36


    ' Fill the columns with a formula and apply a number format.

      Set oResizeRange = oWS.Range("E2",
"E6").Resize(ColumnSize:=iNumQtrs)

      oResizeRange.Formula = "=RAND()*100"

      oResizeRange.NumberFormat = "$0.00"


    ' Apply borders to the Sales data and headers.

      Set oResizeRange = oWS.Range("E1",
"E6").Resize(ColumnSize:=iNumQtrs)

      oResizeRange.Borders.Weight = xlThin


    ' Add a Totals formula for the sales data and apply a border.

      Set oResizeRange = oWS.Range("E8",
"E8").Resize(ColumnSize:=iNumQtrs)

      oResizeRange.Formula = "=SUM(E2:E6)"

      With oResizeRange.Borders(xlEdgeBottom)

         .LineStyle = xlDouble

         .Weight = xlThick
```

```
        End With


    ' Add a Chart for the selected data

        Set oResizeRange =
oWS.Range("E2:E6").Resize(ColumnSize:=iNumQtrs)

        Set oChart = oWS.Parent.Charts.Add

        With oChart

           .ChartWizard oResizeRange, xl3DColumn, , xlColumns

           .SeriesCollection(1).XValues = oWS.Range("A2", "A6")

              For iRet = 1 To iNumQtrs

                 .SeriesCollection(iRet).Name = "=""Q" & Str(iRet) &
"""""

              Next iRet

           .Location xlLocationAsObject, oWS.Name

        End With


    ' Move the chart so as not to cover your data.

        With oWS.Shapes("Chart 1")

           .Top = oWS.Rows(10).Top

           .Left = oWS.Columns(2).Left


        End With


    ' Free any references.

        Set oChart = Nothing

        Set oResizeRange = Nothing
```

```
End Sub
```

6. Press F5 to run the project.

Appendix C: Microsoft Excel Automation Using Microsoft Visual J++

Microsoft Knowledge Base Article – 169796
http://support.microsoft.com/default.aspx?kbid=169796

SUMMARY

This article illustrates how to call a COM object like Excel from Java. It provides code samples that show how you can make an Excel application visible and open an existing Excel file. There are two code samples. One illustrates how to invoke Excel 7.0 from Java and the other shows how to invoke Excel 8.0 from Java. You will find the classes and interfaces to be quite different in Excel 8.0.

MORE INFORMATION

Before using code samples 1 and 2, follow these steps below:

1. Use the Java Applet Wizard to create a default Applet or Application.
2. Run JAVATLB or JACTIVEX (utility similar to JAVATLB and supported by the SDK for Java 2.0x) on the Microsoft Excel 7.0 Object Library if you are using Excel 7.0 or the Microsoft Excel 8.0 Object Library if you are using Excel 8.0. This will create Java descriptions of the Excel Object. If you are using Excel 97, run the JACTIVEX tool on EXCEL8.OLB **NOTE**: If you are using JACTIVEX, then you will need to build this sample with the JVC.EXE that ships with the SDK for Java 2.0 or later. You can download the SDK for Java from the following Microsoft Web site:
http://www.microsoft.com/java/

3. Use the import statement to import the classes of the Excel Object Library into your Java project.
4. Insert either one of the two code snippets in your Java Project to automate Excel from Java.

Code Sample 1

```
import xl5en32.*;

import com.ms.com.*;

public void TestExcel()

{

    _Global gbl = (_Global) new _ExcelApplication();

    Variant param = gbl.Application();

    Application app = (Application) param.getDispatch();

    param.putInt(xl5en32.Constants.xlVisible);

    app.putVisible(param);


    Variant vtemp = new Variant();

    vtemp.noParam();
```

```
    Variant vtWorkbooks = Dispatch.get(gbl,"Workbooks");
    Workbooks wbs = (Workbooks)vtWorkbooks.getDispatch();

    Variant vtName = new Variant();
    VtName.putStringRef("C:\\book1.xls");

    Variant vtEmpty = new Variant();
    vtEmpty.noParam();
    wbs.Open(vtName,vtEmpty,vtEmpty,vtEmpty,vtEmpty,vtEmpty,vtEmpty,
            vtEmpty,vtEmpty,vtEmpty,vtEmpty,vtEmpty,vtEmpty);
}
```

Code Sample 2

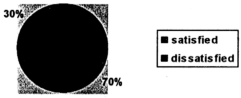
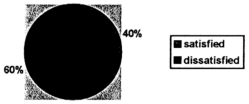
```
import excel8.*;
import com.ms.com.*;

_Global globXL=null;
_Application appXL=null;
Workbooks books=null;
_Workbook book = null;

try{
   globXL = (_Global)new Global();
   appXL = (_Application)globXL.getApplication();
   appXL.putVisible(0,true);  // in Excel 97 use:
                              // appXL.setVisible(0,true);
```

```
    books = (Workbooks)appXL.getWorkbooks();


    Variant vTemp = new Variant();

    vTemp.putString("c:\\book1.xls");


    Variant vOptional = new Variant();

    vOptional.noParam();

    book =

(_Workbook)books.Open("c:\\book1.xls",vOptional,vOptional,vOptional,
vOption

al,vOptional,vOptional,vOptional,vOptional,vOptional,vOptional,vOpti
onal,vO

ptional,0);


    }

    catch(ComFailException e)

    {

       System.out.println(e.getMessage());

    }
```

The above code will make the Excel application visible, and it will open up an existing Excel file.

Appendix D: System Testing Results – collected from 10 system users

Table D1 Test System functionality: Segregating Excel document

Feature	Overall Comments Gathered
Enter the exact words/ phrases found in a data cell of the excel source file.	Satisfied – match with expected result (7 users) 
Enter any words/phrases (words that cannot be found in the Excel source file) as any of the part1-part7 headers.	Satisfied – match with expected result (4 users) 
Enter value for starting cell and ending cell (format: A1)	Satisfied – match with expected result (3 users) 

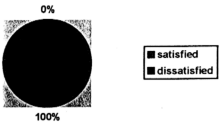
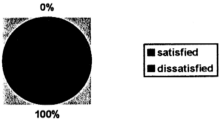
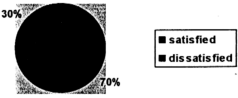
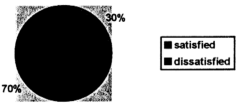
Press the submit button	Satisfied – match with expected result (10 users) 
Click on the segregated Microsoft Excel file links	Satisfied – match with expected result (10 users) 

Table D2 Comments Gathered for System Functionality – Segregating Excel Document

Feature	Comment
Enter the exact words/ phrases found in a data cell of the excel source file.	Users expect the system to present a pop-up window (display the content of Excel document) instead of just prompting an error message: Can't locate the words, please enter the exact words found in the Excel source file. The pop-up window will help users to search for words that are needed for segregation.
Enter value for starting cell and ending cell (format: A1)	Users find it difficult to remember Excel cell name.

Table D3 Test System Functionality: Combining Excel Documents

Feature	Overall Comments Gathered
Select the option vertically combine, fill up two or more Excel files and press submit button	Satisfied – match with expected result (7 users) 
Select the option horizontally combine, fill up two or more Excel files and press submit button	Satisfied – match with expected result (7 users) 
Select either the option horizontally or vertically, browse and fill up two or more Excel files and specify the starting cell, ending cell. Press submit button.	Satisfied – match with expected result (3 users) 
Do not select any option and press submit button	Satisfied – match with expected result (10 users)

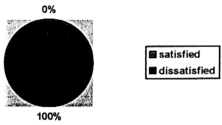
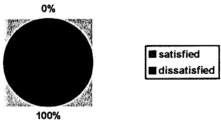
	
Click on the combined Microsoft Excel file link	Satisfied – match with expected result (10 users) 

Table D4 Comments Gathered for System Functionality – Combining Excel Documents

Feature	Comment
Select the option vertically combine, fill up two or more Excel files and press submit button	Three users expect that the system will produce a combined Excel document which preserves the formatting / layout design in each Excel document.
Select the option horizontally combine, fill up two or more Excel files and press submit button	Users expect every single formatting feature / layout design is preserved in the combined Excel document.
Select either the option horizontally or vertically, browse and fill up two or more Excel files and specify the start cell, end cell. Press submit button.	Users find it difficult to remember the start cell and end cell in each Excel document that they want to combine.

Table D5 Overall Comments and Suggestions

Feature	Comment
Excel start cell and end cell could be used to indicate a range of Excel data cells	Not user-friendly enough. Hard to remember the exact cell name to indicate the range of Excel data that need to be segregated or combined.
Unique words/phrases contained in an Excel document must be provided in order to segregate the document accordingly	Question raise from system users where there could be more than one cell in an Excel document that contains same words / phrases. This feature could only be applied to simple Excel document that doesn't have duplicate data in any cell.

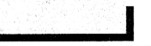
Suggestions:

- Users may wish that the OODA System for Microsoft Excel Files could be extended to support Microsoft Words files as well.
- It is suggested by users that any parts in an Excel document that is combined or segregated should have a comment to indicate the source of where the data is obtained.

Appendix E: Microsoft Excel 2000 Object Model

You can use these diagrams as a handy shortcut to finding the object you want to work with and understanding how that object fits into the overall object model exposed by an application. The following table shows how objects and collections are represented in the diagrams.

Table E1 Representation of Object and Collection for Microsoft Excel 2000 Object Model

This type of item	Is designated this way
Object	
Collection	

A group of similar objects can be combined in the hierarchy as a *collection*. You can work with a member of a collection as a single object or as a member of that collection. For example, the Microsoft Excel object model exposes the **Application** object as the top-level object in its object model hierarchy. The **Application** object has a **Workbooks** collection that contains a **Workbook** object for each currently open workbook. Similarly, each **Workbook** object has a **Worksheets** collection that represents the worksheets in a workbook, and so on.

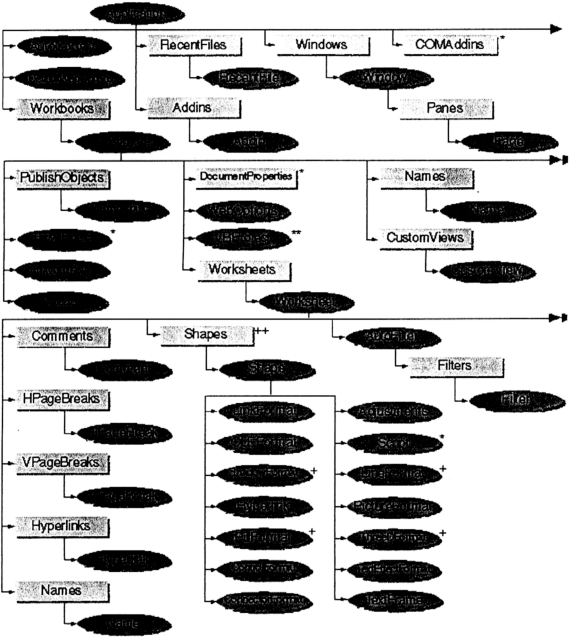


Figure E1 Microsoft Excel 2000 Object Model

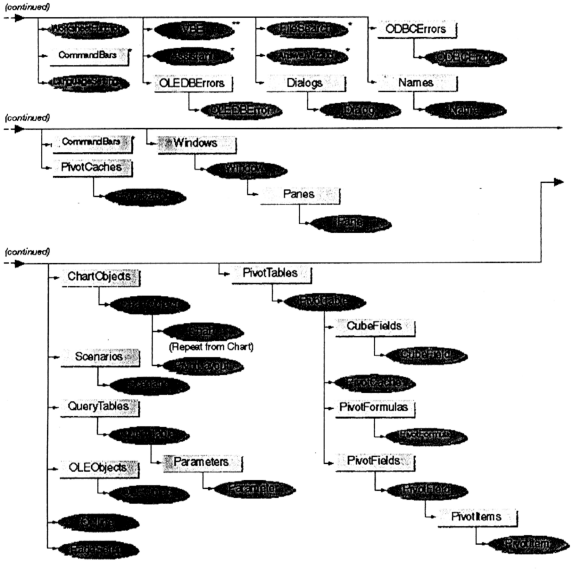


Figure E1 Microsoft Excel 2000 Object Model (continued)

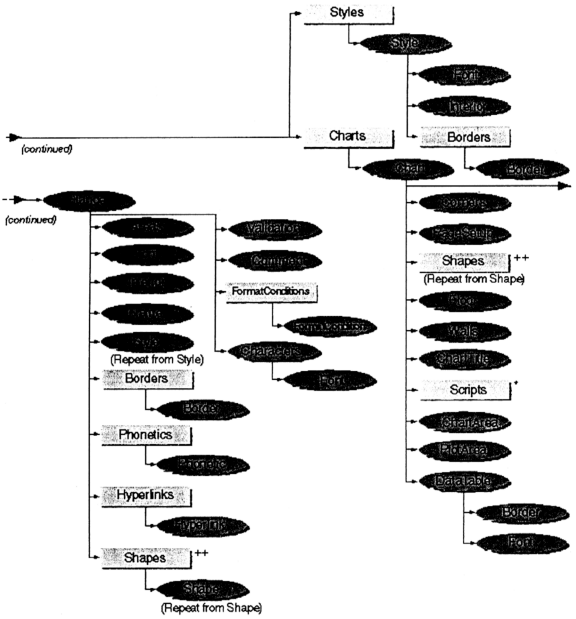


Figure E1 Microsoft Excel 2000 Object Model (continued)

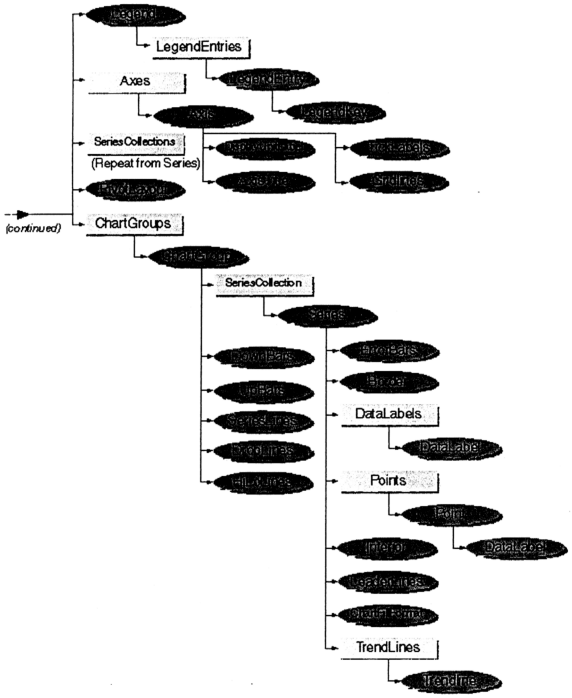


Figure E1 Microsoft Excel 2000 Object Model (continued)

Appendix F: Screen Shots

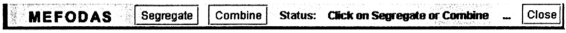


Figure F1 Startup menu screen

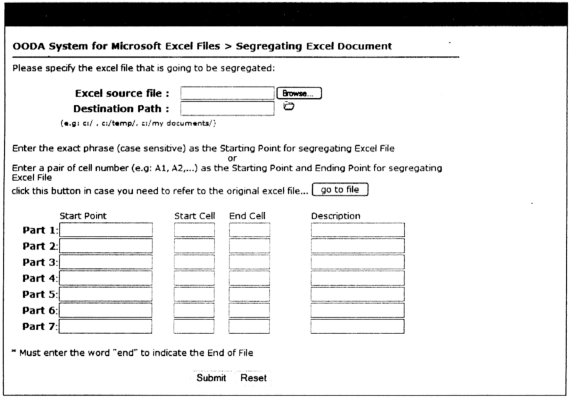


Figure F2 Segregating Excel document screen

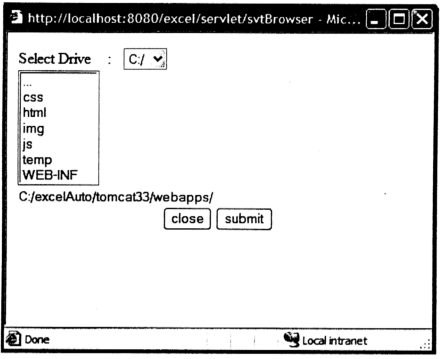


Figure F3 Browse folder screen

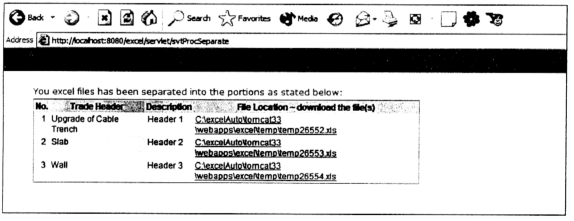


Figure F4 Segregating Excel result screen

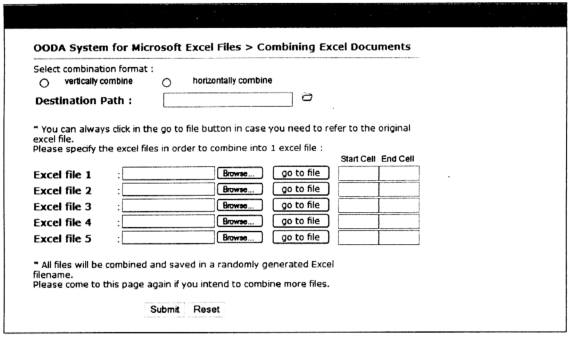


Figure F5 Combining Excel documents screen

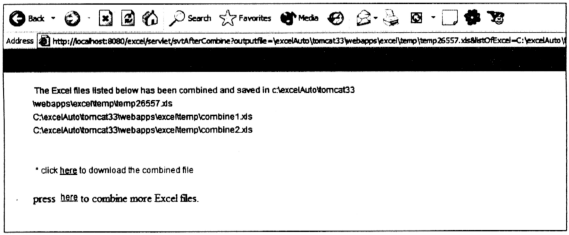


Figure F6 Combining Excel result screen

Appendix G: Example of Excel files – Segregating Function

PROJECT: CADANGAN PENGUBAHSUAIAN DAN PEMBANGUNAN SEMULA DI ATAS LOT 2066 & 2067 JALAN TAMPOI, MUKIM TEBRAU, JOHOR BAHRU, JOHOR UNTUK CIDB MALAYSIA - REFURBISHMENT TO PART OF BLOCK B

PROJECT NO : 9098

PROGRESS CLAIM NO : FOUR (4) - FINAL

TNB Substation

Item	Description	B.Q Qty	B.Q Unit	B.Q Amount (Rm)
	Upgrade of Cable Trench			
1	Hack carefully existing brickwall and concrete Slab and to enlarge cable trench including carting away and remove debris of site to Contractor's own dump.		Item	
2	Construct and built-up cable trench wall and slab with 115mm thick brickwall including plastering and forming rebate to receive cover slab.	76	m^3	
3	Provide precast concrete cover slab for cable trench.			
	a) Overall size 1200mm x 600mm	7	m	
	b) Overall size 900mm x 600mm	28	m	
4	To fill existing trench with mass concrete.	7	m3	
5	Ditto with approved sand.	11	m3	
	Slab			
6	Make good existing ground slab including cleaning, patching, filling up holes and cavities with cement and sand rendering smooth to flush with existing slab.		Item	
	Wall			
7	Take down carefully existing window and frame overall size approximately 5000mm x 1500mm high including carting away and remove debris off site to contractor's own dump.		Item	
8	Take down carefully existing window and frame overall size approximately 3000mm x 2000mm high including carting away and remove debris off site to contractor's own dump.		Item	
9	Construct new external and internal 115mm thick half brickwall including all necessary damp proof course and bonding ties and including plastering.	47	m^3	

Figure G1 Example Excel file for segregating purpose

After Segregated the Excel file into three portions

Portion 1

		Upgrade of Cable Trench				
	1	Hack carefully existing brickwall and concrete Slab and to enlarge cable trench including carting away and remove debris of site to Contractor's own dump.		Item		
	2	Construct and built-up cable trench wall and slab with 115mm thick brickwall including plastering and forming rebate to receive cover slab.	76	m^2		
	3	Provide precast concrete cover slab for cable trench.				
		a) Overall size 1200mm x 600mm	7	m		
		b) Overall size 900mm x 600mm	28	m		
	4	To fill existing trench with mass concrete.	7	m3		
	5	Ditto with approved sand.	11	m3		

Figure G2 Example of segregated result : Part 1

Portion 2

		Slab				
	6	Make good existing ground slab including cleaning, patching, filling up holes and cavities with cement and sand rendering smooth to flush with existing slab.		Item		

Figure G3 Example of segregated result : Part 2

Portion 3

	Wall				
7	Take down carefully existing window and frame overall size approximately 5000mm x 1500mm high including carting away and remove debris off site to contractor's own dump.		Item		
8	Take down carefully existing window and frame overall size approximately 3000mm x 2000mm high including carting away and remove debris off site to contractor's own dump.		Item		
9	Construct new external and internal 115mm thick half brickwall including all necessary damp proof course and bonding ties and including plastering.	47	m²		

Figure G4 Example of segregated result : Part 3

Appendix H: Example of Excel files – Combining Function

Wall				
item	desc	qty	unit	price
1.1	item 1	1	nos	3000.5
1.2	item 2	2	nos	3333
1.3	item 3	3	nos	2222

Figure H1 Example Excel file1 for combining purpose

Trade2				
item	desc	qty	unit	price
2.1	bitem1	3	m	33
2.2	bitem2	4	km	55
2.3	bitem3	5	kg	88
2.4	bitem4	6	lt	99

Figure H2 Example Excel file2 for combining purpose

Trade3				
item	desc	qty	unit	price
3.1	citem1	22	m	249
3.2	citem2	3	lt	933
3.3	citem3	5	kg	988
3.4	citem4	1	gn	777

Figure H3 Example Excel file3 for combining purpose

Vertically Combine

Wall

item	desc	qty	unit	price
1.1	item 1	1	nos	3000.5
1.2	item 2	2	nos	3333
1.3	item 3	3	nos	2222

Trade2

item	desc	qty	unit	price
2.1	bitem1	3	m	33
2.2	bitem2	4	km	55
2.3	bitem3	5	kg	88
2.4	bitem4	6	lt	99

Trade3

item	desc	qty	unit	price
3.1	citem1	22	m	249
3.2	citem2	3	lt	933
3.3	citem3	5	kg	988
3.4	citem4	1	gn	777

Figure H4 Result of vertically combine

Horizontally Combined

Wall				Trade2				Trade3			
item desc	qty	unit	price	item desc	qty	unit	price	item desc	qty	unit	price
1.1 item 1	1	nos	3000.5	2.1 bitem1	3	m	33	3.1 citem1	22	m	
1.2 item 2	2	nos	3333	2.2 bitem2	4	km	55	3.2 citem2	3	lt	
1.3 item 3	3	nos	2222	2.3 bitem3	5	kg	88	3.3 citem3	5	kg	
				2.4 bitem4	6	lt	99	3.4 citem4	1	gn	

Figure H5 Result for horizontally combine

Appendix I: Setup for OODA System for Microsoft Excel Files

Step 1: Distribute startup folder to system users
Step 2: Create a folder namely excelAuto in C: drive
Step 3: Copy all files/folder from startup folder in to C:\excelAuto\ folder
Step 4: Once copied, double click on C:\excelAuto\Autorun.bat file
Step 5: The OODA System for Microsoft Excel Files is launched
Step 6: Choose either segregate or combine button in the menu
Step 7: Use the close button to exit the system