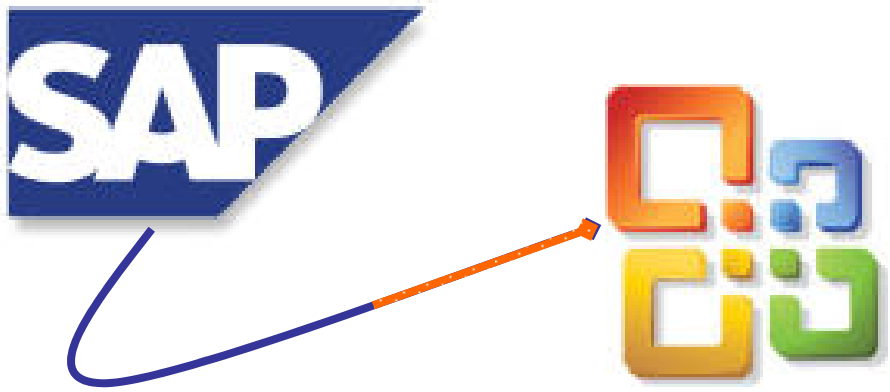
An
Easy Reference
for

OLE Automation

(Microsoft Word & Excel)



Serdar ŞİMŞEKLER
2004, Ankara TURKEY

Table of Contents

Purpose

The purpose of this tutorial is to provide a step-by-step guide to illustrate the use of Object Linking and Embedding (OLE) automation techniques within an ABAP program. It is generally recommended to use SAP DOI (Desktop Office Integration) for office integration, since it standardizes the procedure and handles the integration by a structured and robust service. However, in some cases, developers need simpler and more flexible methods. This tutorial does not aim to dive into profound technical facts about the underlying technology, nor does it cover all the related details about the topic. However, this tutorial may be used as a quick reference manual since it aims to draw a general scheme.

Prerequisites

Obviously, basic level ABAP programming skills are required to make use of this tutorial. Knowledge of the macro language of the application will be of great help and some knowledge of the OLE technology is recommended.

A Basics

Basically, utilizing OLE automation is achieved by creating OLE objects and calling their methods. Within ABAP, five basic statements are used for OLE automation. So, this first section will deal with those ABAP statements.

A.1 Data Definitions

For each entity of the OLE object, there must be a variable holding handle data for it. These handle variables should be of the type "`ole2_object`" which is defined in the type-pool "`ole2`". Hence, within your program you should include "`ole2incl`" which wraps the pool and then define your handle variables.

```
*--Include for OLE-enabling definitions
INCLUDE ole2incl .

*--Global variables
*--Variables to hold OLE object handles
DATA gs_word TYPE ole2_object .
.. ..
```

Code Part A.1 *Data definitions*

A.2 Creating an OLE Object

To create an OLE object, the ABAP statement "`CREATE OBJECT`" is used. Its syntax is: **`CREATE OBJECT obj class.`**

Here, "**`obj`**" is the handle variable for the base object and "**`class`**" is the specific identifier for the corresponding application.

e.g. `CREATE OBJECT gs_word 'WORD.APPLICATION'.`

If the creation is successful, the value of "sy-subrc" becomes "0" – otherwise it becomes some other value (i.e. "1", "2" or "3" with respect to the error type).

A.3 Calling a Method of an Object

After creating an OLE object, it is possible to call its methods to execute its functionality. This is achieved by using the ABAP statement "CALL METHOD OF". You can also pass required parameters using this statement.

The syntax is: **CALL METHOD OF obj m [= f] [EXPORTING p1 = f1 ... pn = fn].**

Here, "**obj**" is the object handle variable, "**m**" is the method name, "**f**" is the variable where the output of the method will be replaced and "**pn = fn**" assignments are used to pass parameters. The "**EXPORTING…**" part must be at the end of the statement. For the moment, parameter passing is done by giving their positions and the corresponding value.

e.g.
```
CALL METHOD OF gs_word 'Documents' = gs_documents .
CALL METHOD OF gs_selection 'TypeText' EXPORTING #1 = ip_text .
```

Successful method calls makes "sy-subrc" value "0," and unsuccessful cases make it some other value.

A.4 Setting a Property of an Object

To set a property of an OLE object, the ABAP statement "SET PROPERTY OF" is used. The syntax is: **SET PROPERTY OF obj p = f .**

Here, "**obj**" is the object handle variable, "**p**" is the property name and "**f**" is the value to be assigned.

e.g.
```
SET PROPERTY OF gs_word 'Visible' = 1 .
```

Operation result status is indicated at the system variable "sy-subrc"; "0" for successful operations and another value for erroneous cases.

A.5 Getting a Property of an Object

Getting the value of a property of an OLE object is obviously similar to setting it. For this, the ABAP statement "GET PROPERTY OF" is used. The syntax is: **GET PROPERTY OF obj p = f.**

Here, "**obj**" is the object handle variable, "**p**" is the property name and "**f**" is the variable to which the value of the property is assigned.

e.g.
```
GET PROPERTY OF gs_view 'Type' = gv_viewtype .
```

Again, operation result status is indicated at the system variable "sy-subrc"; "0" for successful operations and another value for erroneous cases.

A.6 Freeing an Object

Generally for performance issues, it is required to free the memory allocated for OLE objects. For this, the ABAP statement "FREE OBJECT" is used.

The syntax is: **FREE OBJECT obj.** where "**obj**" is the object handle variable.

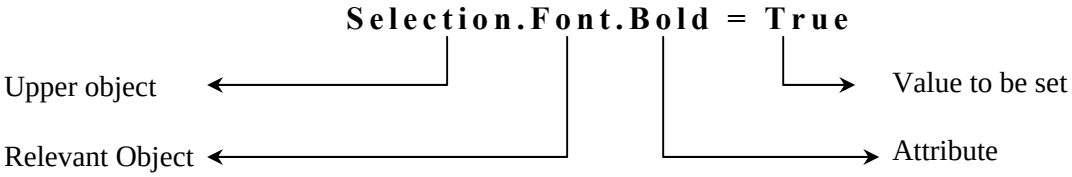
A.7 NO FLUSH Addition

Normally, OLE statements are buffered by the ABAP processor and executed at the frontend collectively before the first statement that is not of OLE context. Using this addition prevents this and postpones the execution until just before the first non-OLE statement coming after an OLE statement without NO FLUSH addition.

A.8 Knowing About Methods and Properties of an OLE Object

What a developer requires is generally to retrieve information about methods and properties that the OLE object bestows. Generally, it is a useful way to use the macro debugging of the application to figure out those. The relevant library of the application will also give useful information about these. What we require to figure out is the class chain whose instances we will create and make method calls. Recording and debugging a macro generally provides relevant object hierarchy to be called within the program.

Here is a simple VB code:



This line of macro code tells us that to set the attribute *bold* we must create OLE instances for *font* and *selection* and then set the property *bold* of *font* object.

So let's switch to ABAP and write relevant code:

i. Getting instance for *font*:

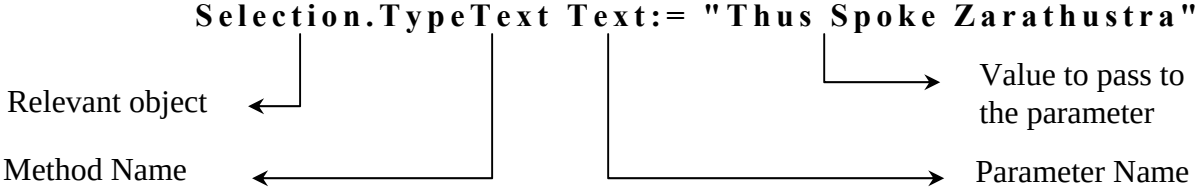
```
GET PROPERTY OF gs_selection 'Font' = gs_font .
```

ii. Setting attribute *bold*

```
SET PROPERTY OF gs_font 'Bold' = '1' .
```

Here, it is seen that to retrieve lower level instances we use "`GET PROPERTY OF`" statement. One will ask how to instantiate *selection* object which seems to be the topmost object, although in the whole picture it is not. This object is reached following the class hierarchy from the root OLE object created for the application. This procedure is illustrated in code parts in following sections.

Here is another VB code line calling a method of an object:



This VB code is adapted to ABAP as:

```
CALL METHOD OF gs_selection 'TypeText'
  EXPORTING
      #1 = 'Thus Spoke Zarathustra'.
```

B A General Scheme for Integration with Microsoft Word

Now, it is time to build an application that integrates with MS Word and uses some of its basic features. So, let's define the outline for its task as:

i. Add a proper title.
ii. Write some text.
iii. Insert a table.
iv. Write a second snippet of text.
v. Insert some identification text at the header.
vi. Set measurement unit for the document to 'CM.'

To be more clear, the code will be written in a non-modular way that will repeat reusable parts. For your further works, you can modularize all these. For example, all functional codes may be written as subroutines to be collected in a subroutine pool together. Or a function group can be implemented. In fact, the best way is to develop a class to encapsulate all.

Step 1 → Data declarations.

```
REPORT  zole_tutor_example_ms_word .

*--Include for OLE-enabling definitions
INCLUDE ole2incl .

*--Global variables
*--Variables to hold OLE object and entity handles
DATA gs_word TYPE ole2_object . "OLE object handle
DATA gs_documents TYPE ole2_object . "Documents
DATA gs_actdoc TYPE ole2_object . "Active document
DATA gs_application TYPE ole2_object . "Application
DATA gs_options TYPE ole2_object . "Application options
DATA gs_actwin TYPE ole2_object . "Active window
DATA gs_actpan TYPE ole2_object . "Active pane
DATA gs_view TYPE ole2_object . "View
DATA gs_selection TYPE ole2_object . "Selection
DATA gs_font TYPE ole2_object . "Font
DATA gs_parformat TYPE ole2_object . "Paragraph format
DATA gs_tables TYPE ole2_object . "Tables
DATA gs_range TYPE ole2_object . "Range handle for various ranges
DATA gs_table TYPE ole2_object . "One table
DATA gs_table_border TYPE ole2_object . "Table border
DATA gs_cell TYPE ole2_object . "One cell of a table
DATA gs_paragraph TYPE ole2_object . "Paragraph

DATA gv_pos(5) TYPE n . "Position information for table
```

Code Part B.1 *Data declarations*

Step 2 → Creating the OLE object and get main entities to handle variables.

```
START-OF-SELECTION .

*--Creating OLE object handle variable
  CREATE OBJECT gs_word 'WORD.APPLICATION' .
  IF sy-subrc NE 0 .
    MESSAGE s000(su) WITH 'Error while creating OLE object!'.
    LEAVE PROGRAM .
  ENDIF .
```

```
*--Setting object's visibility property
  SET PROPERTY OF gs_word 'Visible' = '1' .
*--Opening a new document
  GET PROPERTY OF gs_word 'Documents' = gs_documents .
  CALL METHOD OF gs_documents 'Add' .

*--Getting active document handle
  GET PROPERTY OF gs_word 'ActiveDocument' = gs_actdoc .
*--Getting applications handle
  GET PROPERTY OF gs_actdoc 'Application' = gs_application .
```

Code Part B.2 *Creating the OLE object*

Step 3 → Setting the measurement unit to 'CM.'

```
*--Setting the measurement unit
  GET PROPERTY OF gs_application 'Options' = gs_options .
  SET PROPERTY OF gs_options 'MeasurementUnit' = '1' . "CM
```

Code Part B.3 *Setting measurement unit*

Step 4 → Some header text.

```
*--Getting handle for the selection which is here the character at the
*--cursor position
  GET PROPERTY OF gs_application 'Selection' = gs_selection .
  GET PROPERTY OF gs_selection 'Font' = gs_font .
  GET PROPERTY OF gs_selection 'ParagraphFormat' = gs_parformat .

*--Setting font attributes
  SET PROPERTY OF gs_font 'Name' = 'Arial' .
  SET PROPERTY OF gs_font 'Size' = '10' .
  SET PROPERTY OF gs_font 'Bold' = '0' . "Not bold
  SET PROPERTY OF gs_font 'Italic' = '1' . "Italic
  SET PROPERTY OF gs_font 'Underline' = '0' . "Not underlined

*--Setting paragraph format attribute
  SET PROPERTY OF gs_parformat 'Alignment' = '2' . "Right-justified
  CALL METHOD OF gs_selection 'TypeText'
    EXPORTING
      #1 = 'This is an OLE example!'.

*--Setting the view to the main document again
  SET PROPERTY OF gs_view 'SeekView' = '0' . "Main document view
```

Code Part B.4 *Setting header content*

Step 5 → Writing the title.

```
*--Reseting font attributes for the title
  SET PROPERTY OF gs_font 'Name' = 'Times New Roman' .
  SET PROPERTY OF gs_font 'Size' = '16' .
  SET PROPERTY OF gs_font 'Bold' = '1' . "Bold
  SET PROPERTY OF gs_font 'Italic' = '0' . "Not Italic
  SET PROPERTY OF gs_font 'Underline' = '0' . "Not underlined

*--Setting paragraph format attribute
  SET PROPERTY OF gs_parformat 'Alignment' = '1' . "Centered
  CALL METHOD OF gs_selection 'TypeText'
    EXPORTING
      #1 = text-000.
*--Advancing cursor to the new line
  CALL METHOD OF gs_selection 'TypeParagraph' .
```

Code Part B.5 *Writing the title*

Step 6 → Writing some text.

```
*--Reseting font attributes for ordinary text
  SET PROPERTY OF gs_font 'Name' = 'Times New Roman' .
  SET PROPERTY OF gs_font 'Size' = '12' .
  SET PROPERTY OF gs_font 'Bold' = '0' . "Not bold
  SET PROPERTY OF gs_font 'Italic' = '0' . "Not Italic
  SET PROPERTY OF gs_font 'Underline' = '0' . "Not underlined

*--Setting paragraph format attribute
  SET PROPERTY OF gs_parformat 'Alignment' = '3' . "Justified
  CALL METHOD OF gs_selection 'TypeText'
    EXPORTING
      #1 = text-001.

*--Skip some lines
  DO 4 TIMES .
    CALL METHOD OF gs_selection 'TypeParagraph' .
  ENDDO .
```

Code Part B.6 *Writing some text*

Step 7 → Inserting a table and filling some of its cells.

```
*--Getting entity handles for the entities on the way
  GET PROPERTY OF gs_actdoc 'Tables' = gs_tables .
  GET PROPERTY OF gs_selection 'Range' = gs_range .
*--Adding a table with 3 rows and 2 columns
  CALL METHOD OF gs_tables 'Add' = gs_table
    EXPORTING
      #1 = gs_range " Handle for range entity
      #2 = '3' "Number of rows
      #3 = '2'. "Number of columns
*--Setting border attribute for the table
  GET PROPERTY OF gs_table 'Borders' = gs_table_border .
  SET PROPERTY OF gs_table_border 'Enable' = '1' . "With border

*--Filling the table with dummy data
*--Reseting font attributes for table content
  SET PROPERTY OF gs_font 'Name' = 'Garamond' .
  SET PROPERTY OF gs_font 'Size' = '11' .
  SET PROPERTY OF gs_font 'Bold' = '0' . "Not bold
  SET PROPERTY OF gs_font 'Italic' = '0' . "Not Italic
  SET PROPERTY OF gs_font 'Underline' = '0' . "Not underlined

*--Getting cell coordinates
  CALL METHOD OF gs_table 'Cell' = gs_cell
    EXPORTING
      #1 = '1' "first row
      #2 = '1'. "first column

*--Getting the range handle to write the text
  GET PROPERTY OF gs_cell 'Range' = gs_range .
*--Filling the cell
  SET PROPERTY OF gs_range 'Text' = 'OLE' .

*--Getting cell coordinates
  CALL METHOD OF gs_table 'Cell' = gs_cell
    EXPORTING
      #1 = '3' "third row
      #2 = '2'. "second column

*--Getting the range handle to write the text
```

```
  GET PROPERTY OF gs_cell 'Range' = gs_range .
*--Filling the cell
  SET PROPERTY OF gs_range 'Text' = 'OLE' .


*--Advancing the cursor to the end of the table
  GET PROPERTY OF gs_table 'Range' = gs_range .
  GET PROPERTY OF gs_range 'End' = gv_pos .
  SET PROPERTY OF gs_range 'Start' = gv_pos .
  CALL METHOD OF gs_range 'Select' .

*--Skip some lines
  DO 3 TIMES .
    CALL METHOD OF gs_selection 'TypeParagraph' .
  ENDDO .
```

Code Part B.7 *Some table work*

Step 8 → Adding some other text and indent its paragraph.

```
*--Reseting font attributes for ordinary text
  SET PROPERTY OF gs_font 'Name' = 'Times New Roman' .
  SET PROPERTY OF gs_font 'Size' = '12' .
  SET PROPERTY OF gs_font 'Bold' = '0' . "Not bold
  SET PROPERTY OF gs_font 'Italic' = '0' . "Not Italic
  SET PROPERTY OF gs_font 'Underline' = '0' . "Not underlined

*--Setting paragraph format attribute
  SET PROPERTY OF gs_parformat 'Alignment' = '3' . "Justified
*--Indent the paragraph once
  GET PROPERTY OF gs_selection 'Paragraphs' = gs_paragraph .
  CALL METHOD OF gs_paragraph 'Indent' .
  CALL METHOD OF gs_selection 'TypeText'
    EXPORTING
      #1 = text-002.
```

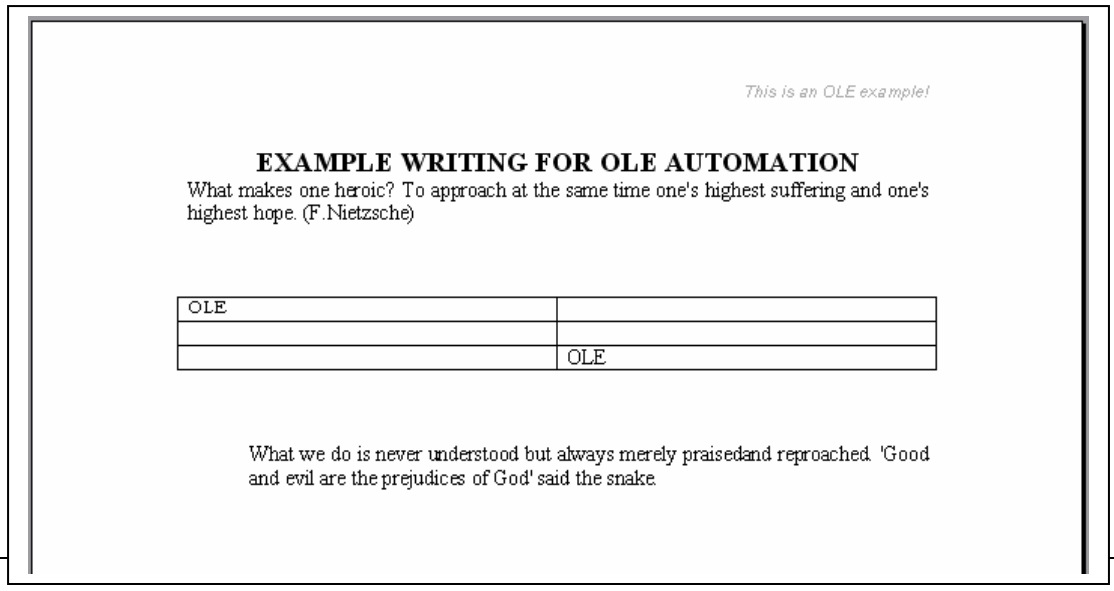
Code Part B.8 *Writing some indented text*

Step 9 → Freeing object handle variable to deallocate memory.

```
FREE OBJECT gs_word.
```

Code Part B.9 *Freeing object handle variable*

Result:



C A General Scheme for Integration with Microsoft Excel

Secondly, let's build an application that integrates with MS Excel and uses some of its basic features. So, let's define the outline for its task as:

i. User inputs the number of worksheets.
ii. For each sheet, user creates some data that is also the source for a chart.
iii. Formats cells.
iv. Draws the chart and relocates it to the proper place on the sheet.

Again, to be more clear, the code will be written in a non-modular way that will repeat reusable parts. For your further works, you can modularize all these. For example, all functional codes may be written as subroutines to be collected in a subroutine pool altogether. Or a function group can be implemented. In fact, the best way is to develop a class to encapsulate all. The order of method calls is important, so do not change their order.

Step 1 → Data declarations.

```
REPORT  zole_tutor_example_ms_excel .

INCLUDE ole2incl .

DATA: gs_excel TYPE ole2_object ,
      gs_wbooklist TYPE ole2_object ,
      gs_application TYPE ole2_object ,
      gs_wbook TYPE ole2_object ,
      gs_activesheet TYPE ole2_object ,
      gs_sheets TYPE ole2_object ,
      gs_newsheet TYPE ole2_object ,
      gs_cell1 TYPE ole2_object ,
      gs_cell2 TYPE ole2_object ,
      gs_cells TYPE ole2_object ,
      gs_range TYPE ole2_object ,
      gs_font TYPE ole2_object ,
      gs_interior TYPE ole2_object ,
      gs_columns TYPE ole2_object ,
      gs_charts TYPE ole2_object ,
      gs_chart TYPE ole2_object ,
      gs_charttitle TYPE ole2_object ,
      gs_charttitlechar TYPE ole2_object ,
      gs_chartobjects TYPE ole2_object .


DATA gv_sheet_name(20) TYPE c .
DATA gv_outer_index LIKE sy-index .
DATA gv_intex(2) TYPE c .
DATA gv_line_cntr TYPE i . "line counter
DATA gv_linno TYPE i . "line number
DATA gv_colno TYPE i . "column number
DATA gv_value TYPE i . "data

PARAMETERS: p_sheets TYPE i .
```

Code Part C.1 *Data declarations*

Step 2 → Initiate the do-loop and OLE automation base objects.

```
START-OF-SELECTION .

  DO p_sheets TIMES .

*--Forming sheet name
    gv_intex = sy-index .
    gv_outer_index = sy-index .
    CONCATENATE 'Excel Sheet #' gv_intex INTO gv_sheet_name .

*--For the first loop, Excel is initiated and one new sheet is added
    IF sy-index = 1 .
      CREATE OBJECT gs_excel 'EXCEL.APPLICATION' .
      SET PROPERTY OF gs_excel 'Visible' = 1 .
      GET PROPERTY OF gs_excel 'Workbooks' = gs_wbooklist .
      GET PROPERTY OF gs_wbooklist 'Application' = gs_application .
      SET PROPERTY OF gs_application 'SheetsInNewWorkbook' = 1 .
      CALL METHOD OF gs_wbooklist 'Add' = gs_wbook .

      GET PROPERTY OF gs_application 'ActiveSheet' = gs_activesheet .
      SET PROPERTY OF gs_activesheet 'Name' = gv_sheet_name .
*--For the rest of loops, other sheets are added
    ELSE .
      GET PROPERTY OF gs_wbook 'Sheets' = gs_sheets .
      CALL METHOD OF gs_sheets 'Add' = gs_newsheet .
      SET PROPERTY OF gs_newsheet 'Name' = gv_sheet_name .
    ENDIF .

    gv_line_cntr = 1 .  "line counter
```

Code Part C.2 *Looping and initializing, adding new worksheets*

Step3 → Write the title and format it.

```
*--Title
*--Selecting cell area to be merged.
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = 1
        #2 = 1.
    CALL METHOD OF gs_excel 'Cells' = gs_cell2
      EXPORTING
        #1 = 1
        #2 = 4.

    CALL METHOD OF gs_excel 'Range' = gs_cells
      EXPORTING
        #1 = gs_cell1
        #2 = gs_cell2.
    CALL METHOD OF gs_cells 'Select' .

*--Merging
    CALL METHOD OF gs_cells 'Merge' .

*--Setting title data
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_line_cntr
        #2 = 1.

    SET PROPERTY OF gs_cell1 'Value' = 'TITLE' .
```

```
*--Formatting the title
    GET PROPERTY OF gs_cell1 'Font' = gs_font .
    SET PROPERTY OF gs_font 'Underline' = 2 .
    SET PROPERTY OF gs_font 'Bold' = 1 .
    SET PROPERTY OF gs_cell1 'HorizontalAlignment' = -4108 .
    GET PROPERTY OF gs_cell1 'Interior' = gs_interior .
    SET PROPERTY OF gs_interior 'ColorIndex' = 15 .
    SET PROPERTY OF gs_interior 'Pattern' = -4124 .
    SET PROPERTY OF gs_interior 'PatternColorIndex' = -4105 .
```

Code Part C.3 *Writing and formatting the title*

Step 4 → Write some additional data for the title area and format them.

```
    gv_line_cntr = gv_line_cntr + 1 .

*--Writing some additional data for the title
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_line_cntr
        #2 = 1.

    SET PROPERTY OF gs_cell1 'Value' = 'Sheet No' .

    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_line_cntr
        #2 = 5.

    SET PROPERTY OF gs_cell1 'Value' = ':' .

    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_line_cntr
        #2 = 6.

    SET PROPERTY OF gs_cell1 'Value' = gv_intex .

*--Formatting the area of additional data 1
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = 1
        #2 = 1.
    CALL METHOD OF gs_excel 'Cells' = gs_cell2
      EXPORTING
        #1 = gv_line_cntr
        #2 = 5.

    CALL METHOD OF gs_excel 'Range' = gs_cells
      EXPORTING
        #1 = gs_cell1
        #2 = gs_cell2.
    CALL METHOD OF gs_cells 'Select' .

    GET PROPERTY OF gs_cells 'Font' = gs_font .
    SET PROPERTY OF gs_font 'Bold' = 1 .
```

```
*--Formatting the area of additional data 2
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = 1
        #2 = 5.
    CALL METHOD OF gs_excel 'Cells' = gs_cell2
      EXPORTING
        #1 = gv_line_cntr
        #2 = 5.

    CALL METHOD OF gs_excel 'Range' = gs_cells
      EXPORTING
        #1 = gs_cell1
        #2 = gs_cell2.
    CALL METHOD OF gs_cells 'Select' .

    GET PROPERTY OF gs_cells 'Columns' = gs_columns .
    CALL METHOD OF gs_columns 'AutoFit' .

*--Bordering title data area
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = 1
        #2 = 1.
    CALL METHOD OF gs_excel 'Cells' = gs_cell2
      EXPORTING
        #1 = gv_line_cntr
        #2 = 6.

    CALL METHOD OF gs_excel 'Range' = gs_cells
      EXPORTING
        #1 = gs_cell1
        #2 = gs_cell2.
    CALL METHOD OF gs_cells 'Select' .

    CALL METHOD OF gs_cells 'BorderAround'
      EXPORTING
        #1 = 1   "continuous line
        #2 = 4. "thick
```

Code Part C.4 *Some additional writing to the title area, formatting and bordering around the title area*

Step 5 → Put axis labels to the data area.

```
*--Putting axis labels
    gv_colno = 2 .
    gv_line_cntr = gv_line_cntr + 5 .
    gv_linno = gv_line_cntr - 1 .


    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_linno
        #2 = 1.

    SET PROPERTY OF gs_cell1 'Value' = 'X' .

    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_line_cntr
        #2 = 1.

    SET PROPERTY OF gs_cell1 'Value' = 'Y' .
```

Code Part C.5 *Axis Labels*

Step 6 → Generate some data.

```
*--Generating some data
    DO 3 TIMES .
      gv_value = gv_outer_index * sy-index * 10 .

      CALL METHOD OF gs_excel 'Cells' = gs_cell1
        EXPORTING
          #1 = gv_linno
          #2 = gv_colno.

      SET PROPERTY OF gs_cell1 'Value' = sy-index .

      CALL METHOD OF gs_excel 'Cells' = gs_cell1
        EXPORTING
          #1 = gv_line_cntr
          #2 = gv_colno.

      SET PROPERTY OF gs_cell1 'Value' = gv_value .

      gv_colno = gv_colno + 1 .

    ENDDO .
```

Code Part C.6 *Generating Data*

Step 7 → Set source data area for the chart.

```
*--Source data area
    gv_colno = gv_colno - 1 .

    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_linno
        #2 = 1.
    CALL METHOD OF gs_excel 'Cells' = gs_cell2
      EXPORTING
        #1 = gv_line_cntr
        #2 = gv_colno.

    CALL METHOD OF gs_excel 'Range' = gs_cells
      EXPORTING
        #1 = gs_cell1
        #2 = gs_cell2.
    CALL METHOD OF gs_cells 'Select' .
```

Code Part C.7 *Setting source data area for the chart*

Step8 → Draw the chart

```
    GET PROPERTY OF gs_application 'Charts' = gs_charts .
    CALL METHOD OF gs_charts 'Add' = gs_chart .
    CALL METHOD OF gs_chart 'Activate' .
    SET PROPERTY OF gs_chart 'ChartType' = '51' . "Vertical bar graph
    CALL METHOD OF gs_chart 'SetSourceData'
      EXPORTING
        #1 = gs_cells
        #2 = 1.

    SET PROPERTY OF gs_chart 'HasTitle' = 1 .
    GET PROPERTY OF gs_chart 'ChartTitle' = gs_charttitle .
    GET PROPERTY OF gs_charttitle 'Characters' = gs_charttitlechar .
    SET PROPERTY OF gs_charttitlechar 'Text' = 'Sample Graph' .
```

Code Part C.8 *Draw the chart*

Step 9 → Locate the chart onto the current worksheet.

```
*--Locate the chart onto the current worksheet
*--Activate current sheet
    CALL METHOD OF gs_excel 'WorkSheets' = gs_activesheet
      EXPORTING
        #1 = gv_sheet_name.
    CALL METHOD OF gs_activesheet 'Activate' .
    CALL METHOD OF gs_chart 'Location'
      EXPORTING
        #1 = 2
        #2 = gv_sheet_name.
```

Code Part C.9 *Locating the chart onto the current worksheet*

Step 10 → Reposition the chart to a proper place and finish the do-loop.

```
*--Reposition the chart on the worksheet (cut&paste)
    CALL METHOD OF gs_activesheet 'ChartObjects' = gs_chartobjects .
    CALL METHOD OF gs_chartobjects 'Select' .
    CALL METHOD OF gs_chartobjects 'Cut' .

*--Select new area
    gv_line_cntr = gv_line_cntr + 2 .
    CALL METHOD OF gs_excel 'Cells' = gs_cell1
      EXPORTING
        #1 = gv_line_cntr
        #2 = 1.
    CALL METHOD OF gs_excel 'Cells' = gs_cell2
      EXPORTING
        #1 = gv_line_cntr
        #2 = 1.

    CALL METHOD OF gs_excel 'Range' = gs_cells
      EXPORTING
        #1 = gs_cell1
        #2 = gs_cell2.
    CALL METHOD OF gs_cells 'Select' .
    CALL METHOD OF gs_activesheet 'Paste' .

  ENDDO .
```

Code Part C.10 *Repositioning the chart to a proper place and end of the do-loop counting sheet number*

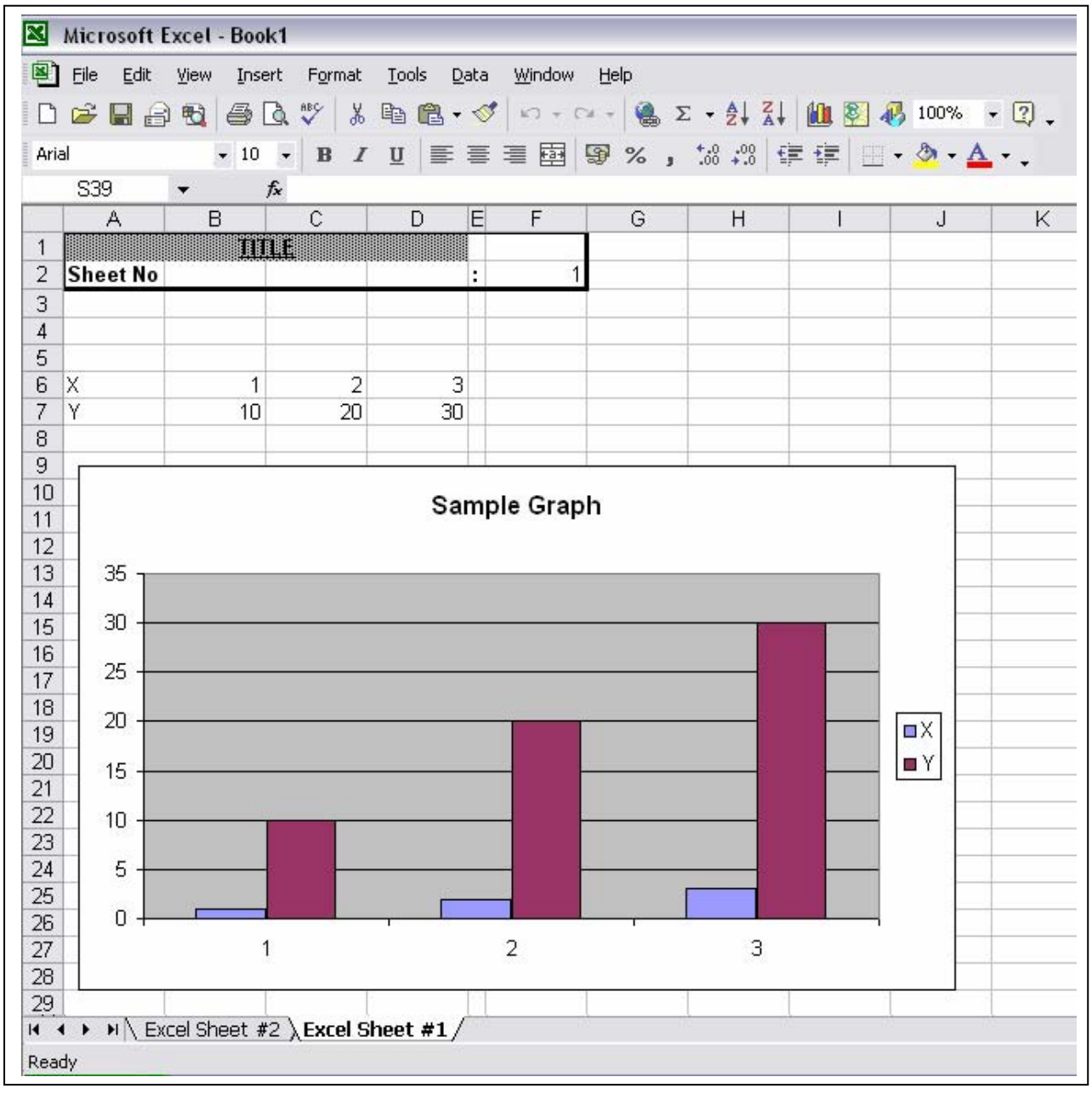
Step 11 → Free OLE objects to deallocate memory.

```
*--Deallocating memory
  FREE: gs_excel, gs_wbooklist, gs_application, gs_wbook,
        gs_activesheet,gs_sheets, gs_newsheet, gs_cell1,
        gs_cell2, gs_cells, gs_range, gs_font, gs_interior,
        gs_columns, gs_charts, gs_chart, gs_charttitle,
        gs_charttitlechar, gs_chartobjects .
```

Code Part C.11 *Deallocating the memory*

Result:

The result of the above program will be a number of worksheets having a title area, some generated data, and a chart related to those data.



D Conclusion

It is possible to write programs that provide integration with MS Office applications through simple OLE automation techniques. The way to interpret macro codes to ABAP is explained so that one can find and implement required functionality using means of the automated application. As stated before, for requirements that can be fulfilled by DOI, it is recommended using that. Rainer Ehre defines SAP Desktop Office Integration (SAP DOI) as "a technology that allows programmers to integrate desktop applications without needing extensive knowledge of the application's macro language, or even having to be acquainted with the application to any greater depth than the average end-user." For more information on SAP DOI, you can search for Ehre's published documents. You can also inspect demo programs for SAP DOI at the development class **SOFFICEINTEGRATION**. In an SAP R/3 system (after release 4.0), you can reach related help documents following: **Help → R/3 Library, BC – Basis → Component Integration → BC – Desktop Office Integration**.

Another component mentioned here is SAP BDS (Business Document Service) that provides utilities for important document services and has its own user interface: BDN – Business Document Navigator.

Consequently, it is highly recommended to make use of SAP DOI for office integration. For further questions on OLE automatio,n you can refer to the **SAP Developer Network ABAP Programming Forum** at http://www.sdn.sap.com .