Working with Microsoft Excel, Part 1

Doug Hennig

Microsoft Excel is one of the most widely-used applications ever. Because of its popularity, other applications often need to read from or write to Excel documents. This article, the first of two, discusses several mechanisms for outputting VFP data to Excel.

Most applications I've worked on had to interact with Microsoft Excel documents in one way or another. Outputting data to Excel is extremely common, as most people are comfortable working with Excel and often want to analyze or massage the data in their applications in ways a developer can't foresee. But importing data from Excel is also popular, both because it can be a common interchange format between different applications and because people often treat Excel like a database, with its row and column layout making data entry fast and easy.

Over the next couple of articles, I'll explore ways to get data from VFP into Excel documents and vice versa. This month, we'll look at several mechanisms and tools you can use to export data into Excel.

Built-in commands

The VFP COPY TO and EXPORT TO commands support creating Excel 2.0 (using the "TYPE XLS" clause) and 5.0 (using "TYPE XL5") files. Newer versions of Excel can open both types of files, but there are some problems:

- Both formats ignore memo fields.
- You may not be able to output all records: VFP 8 and 9 only output a maximum of 65,535 rows, and with VFP 7 and earlier it's only 16,384.
- Neither command can create an XLSX file (Excel 2007 and later).
- Excel 2.0 format doesn't handle date fields correctly: they give a "text date with 2-digit year" warning and have to be edited to work as actual date values.

The first and second points are usually the biggest problems for most people. If you can live with these issues, then these commands are your best solution since they're fast and take only one line of code.

Using OLE Automation

As you're probably aware, you can instantiate Excel using COM and use OLE Automation to write to an Excel document. It would be a lot of code to go through all fields and all records of a cursor and write them to Excel; it would also likely be quite slow. Code written by Cetin Basoz that he showed on TekTips (http://tinyurl.com/hv9ahd2) eliminates the need to do that by using a clever trick: creating an ADO recordset from the VFP data and using Excel's little-known CopyFromRecordSet method to read in the data.

Listing 1 shows the ExcelExportAutomation class. Instantiate the class and call the Export method, passing it the name of an Excel XLSX file to create from the table in the current workarea. If you want the document left open and Excel visible, set lShowExcel to .T. first. If you don't want field headings in the first row, set lIncludeHeadings to .F.

Listing 1. The ExcelExportAutomation class outputs to Excel using OLE Automation.

```
define class ExcelExportAutomation as Custom
  lShowExcel = .F.
    && .T. to leave the document open and
    && Excel visible
  lIncludeHeadings = .T.
    && .T. to put field names in the first
    && row
  cErrorMessage = ''
    && the text of any error that occurs

* Exports the table in the current workarea to
* the specified Excel file.

  function Export(tcFileName)
    local loExcel, ;
      loException, ;
      lcDatabase, ;
      lcDataSource, ;
      lcTable, ;
      llReturn
```

```
    try
      loExcel = ;
        createobject('Excel.Application')
    catch to loException
      This.cErrorMessage = 'Could not ' + ;
        'instantiate Excel. The error ' + ;
        'message is: ' + loException.Message
    endtry
    if vartype(loExcel) = 'O'
      with loExcel

* Determine the datasource (the DBC if there
* is one or the folder for the current table
* if not) and SQL statement (just USE TABLE in
* this case) to use.

        lcDatabase = cursorgetprop('Database')
        if empty(lcDatabase)
          lcDataSource = '"' + ;
            justpath(dbf()) + '"'
        else
          lcDataSource = '"' + lcDatabase + ;
            '"'
        endif empty(lcDatabase)
        lcTable = 'use "' + dbf() + '"'

* Create a workbook and output the data to it,
* then save the file.

        .WorkBooks.Add()
        loSheet = .ActiveWorkBook.ActiveSheet
        llReturn = ;
          This.VFP2Excel(lcDataSource, ;
          lcTable, loSheet.Range('A1'))
        if llReturn
          loSheet.UsedRange.Columns.Autofit()
          llReturn = ;
            This.SaveDocument(loExcel, ;
              .ActiveWorkBook, tcFileName)
        endif llReturn

* Either display Excel or close it.

        if This.lShowExcel and llReturn
          .Visible = .T.
        else
          .Quit()
        endif This.lShowExcel ...
      endwith
    endif vartype(loExcel) = 'O'
    return llReturn
  endfunc

* Fills the specified range with the data from
* the specified SQL statement or table.

  protected function VFP2Excel(tcDataSource, ;
    tcSQL, toRange)
    local loConn, ;
      loRS, ;
      lnI, ;
      lnRow, ;
      llReturn
    try
      loConn = ;
        createobject('ADODB.Connection')
      loConn.ConnectionString = ;
        'Provider=VFPOLEDB;Data Source=' + ;
        tcDataSource
      loConn.Open()
      loRS = loConn.Execute(tcSQL)
      if This.lIncludeHeadings
        for lnI = 0 to loRS.Fields.Count - 1
          toRange.Offset(0, lnI).Value = ;
            proper(loRS.Fields(lnI).Name)
          toRange.Offset(0, lnI).Font.Bold = ;
            .T.
        next lnI
        lnRow = 1
      else
        lnRow = 0
      endif This.lIncludeHeadings
      toRange.Offset(lnRow, ;
        0).CopyFromRecordSet(loRS)
      loRS.Close()
      loConn.Close()
      llReturn = .T.
    catch to loException
      This.cErrorMessage = 'Could not ' + ;
        'output data to workbook. The ' + ;
        'error message is: '  + ;
        alltrim(substr(loException.Message, ;
        at(':', loException.Message) + 1))
    endtry
    return llReturn
  endfunc

* Saves the workbook to the specified file.

  protected function SaveDocument(toExcel, ;
    toWorkbook, tcFileName)
    local llAlerts, ;
      lcFileName, ;
      llReturn

* Set DisplayAlerts to .F. to avoid any
* dialogs.

    try
      llAlerts = toExcel.DisplayAlerts
      toExcel.DisplayAlerts = .F.
    catch
    endtry

* Save the workbook.

    lcFileName = GetFullPath(tcFileName)
    try
      toWorkbook.SaveAs(lcFileName)
      llReturn = file(lcFileName)
    catch to loException
    endtry
    if not llReturn
      This.cErrorMessage = 'Could not ' + ;
        'save ' + lcFileName + ;
        iif(vartype(loException) = 'O', ;
        '. The error message is: ' + ;
        alltrim(substr(loException.Message, ;
        at(':', loException.Message) + 1)), ;
        '')
    endif not llReturn

* Restore DisplayAlerts.

    try
      toExcel.DisplayAlerts = llAlerts
    catch
    endtry
    return llReturn
  endfunc
enddefine
```

The code shown in **Listing 2** outputs the Employee sample table that comes with VFP to EmployeeOLE.XLSX.

Listing 2. Test code for ExcelExportAutomation.

```
open database (_samples + 'data\testdata')
use Employee
loExport = ;
  createobject('ExcelExportAutomation')
loExport.lShowExcel = .T.
if not loExport.Export('EmployeeOLE.xlsx')
  messagebox(loExport.cErrorMessage)
endif not loExport.Export('EmployeeOLE.xlsx')
```

There are a couple of drawbacks to ExcelExportAutomation:

- It doesn't work with a cursor, only a table.
- Like all OLE Automation mechanisms, it requires Excel installed on the machine, which may be problem if you're running the code on a server because many companies don't install Excel on a server, just workstations.

Using ADO

The Microsoft Access Database Engine, in addition to (as its name suggests) providing access to Microsoft Access databases, also provides access to Excel spreadsheets. Craig Boyd takes advantage of that in AppendXLSX.PRG, which you can download from http://tinyurl.com/jzolom3. Although its name suggests it's for reading from Excel (I'll discuss that in the next issue's article), it has a CopyToExcel function that writes to Excel. This function doesn't require Excel to be installed.

CopyToExcel accepts the following parameters:

- The name of the Excel file to create or update.
- The name of the worksheet to write to. Specify Sheet1$ to write to the default worksheet.
- The alias, workarea, or name of the table to export (if not specified, the current workarea is used). This can be a cursor or a table.
- A comma delimited list of columns to create in the worksheet (if not specified, the columns match the cursor fields).
- A comma delimited list of fields to copy from the cursor (if not specified, all fields are copied).
- A WHERE clause used when querying the cursor for data to be copied to the worksheet.

CopyToExcel uses a clever technique: using ADO to create a "table" (that is, a worksheet) in the document, then updating that table from the cursor using a CursorAdapter. I won't list the code here because it's fairly long; feel free to examine AppendXLSX.PRG yourself. I fixed a couple of bugs in the code; the fixed version is included with this month's downloads.

The code in **Listing 3** outputs the Employee sample table to EmployeeADO.XLSX. The resulting spreadsheet is very similar to EmployeeOLE.XLSX.

Listing 3. Test code for CopyToExcel.

```
open database (_samples + 'data\testdata')
use Employee
set procedure to AppendXLSX
lnRecords = CopyToExcel('EmployeeADO.xlsx', ;
  'Employees')
messagebox(lnRecords)
```

A couple of issues I found are:

- I'm not sure whether installing Microsoft Office automatically installs the Microsoft Access Database Engine—it wasn't installed on my system—but you can download the installer from http://tinyurl.com/jzurwra. Be aware that you may also have to install it on any customer systems you intend to use this technique on.
- Although CopyToExcel has code that deletes the default worksheet (Sheet1) if that isn't the one you're outputting to, that doesn't work so you end up with two worksheets. So you might as well use Sheet1 for output.

Using Excel binary format

In the March 2014 issue of FoxRockX, Rick Schummer discussed a VFPX project named FoxyXLS (http://vfpx.codeplex.com), written by Cesar Chalom. FoxyXLS writes to a BIFF3 format file, which is an older Excel binary format. I won't go into detail about FoxyXLS because Rick covered it thoroughly in his article.

Since FoxyXLS writes directly to an Excel file, it doesn't require that Excel be installed on the system. However, it has a few drawbacks other tools discussed in this article don't have:

- It doesn't have a specific method to write a cursor to an Excel document, so you have to write your own code to do that, such as the code shown in **Listing 4** (Execute.PRG, called from this code, using the Windows API ShellExecute function to open a file).
- It writes to XLS files, not the newer XLSX.
- It doesn't support memo fields properly; only the first 256 bytes are output.

Listing 4. This code outputs the Employees table to Excel using FoxyXLS.

```
open database (_samples + 'data\testdata')
use Employee
set procedure to FoxyXLS\FoxyXLS.prg
loExport = createobject('FoxyXLS')
```

```
lnFields = afields(laFields)
for lnI = 1 to lnFields
    loExport.AddCell(1, lnI, laFields[lnI, 1])
    loExport.SetColumnWidth(lnI, ;
        laFields[lnI, 3] * 10)
next lnI
lnRow = 2
scan
    for lnI = 1 to lnFields
        lcField = laFields[lnI, 1]
        if laFields[lnI, 2] = 'G'
            luValue = ''
        else
            luValue = evaluate(lcField)
        endif laFields[lnI, 2] = 'G'
        loExport.AddCell(lnRow, lnI, luValue)
    next lnI
    lnRow = lnRow + 1
endscan
loExport.WriteFile('EmployeeFoxyXLS.xls')
Execute('EmployeeFoxyXLS.xls')
```

Using Excel XML

One of the file formats Excel can read is XML. Since XML is just text, VFP can easily create a file in a format Excel can open. Since no OLE Automation is involved, Excel doesn't have to be installed, and because the XML supports formatting attributes, you can create a document that's nicer looking than just a raw spreadsheet. This mechanism is also fast since it simply uses VFP text processing functions.

There are several tools available to do this, including:

- ExcelXML, a VFPX project created by Rodrigo Bruscain and discussed by Rick Schummer in the May 2014 issue of FoxRockX.
- XLSXWorkbook, another VFPX project, by Greg Green.
- Vilhelm-Ion Praisach has several similar utilities, including ExportXLSX and DBF2XLSX which you can download from http://tinyurl.com/j9nxmwq.

Let's look at XLSXWorkbook. You can download it from http://tinyurl.com/zr6vxc5; the download includes a number of test programs and forms, but all of the actual code for the utility is in a single VCX, VFPxWorkbookXLSX.VCX. The download also includes extensive documentation in a PDF file.

To use XLSXWorkbook, start by instantiating the VFPXWorkbookXLSX class in VFPXWorkbookXLSX.VCX. XLSXWorkbook has a couple of methods that make quick work of creating a spreadsheet from VFP data: SaveTableToWorkbook, which saves the specified table to a workbook, and SaveGridToWorkbook, which save the content and some of the formatting of the specified grid to a workbook.

The code in **Listing 5** outputs the Employee sample table to EmployeeXLSXWookbook.XLSX. This file is very similar to those created using the other techniques.

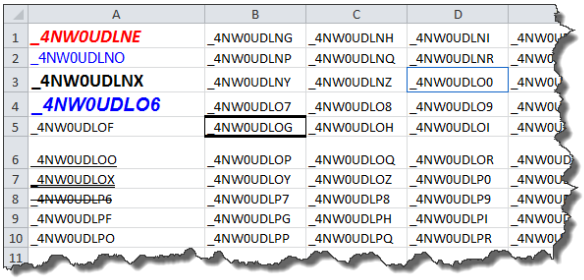
Listing 5. Test code for XLSXWorkbook.

```
open database (_samples + 'data\testdata')
use Employee
loExport = newobject('VFPXWorkbookXLSX', ;
    'VFPXWorkbookXLSX\VFPXWorkbookXLSX.vcx')
loExport.SaveTableToWorkbook(alias(), ;
    'EmployeeXLSXWookbook.xlsx', .T., .T.)
```

Creating a workbook from a grid is easy too, needing just a single line of code:

```
loExport.SaveGridToWorkbook(This.grdGrid, ;
    'MyWookbook.xlsx', .T., .T.)
```

However, XLSXWookbook provides a lot more control over the workbook if you wish. You can format cells (color, font, border, style, etc.), manage worksheets within the workbook, use custom numeric formatting, and so on. So, you don't have to create a plain spreadsheet if your users want something more attractive. The code in FormatXLSXWorkbook.PRG, shown in **Listing 6** and included with this month's downloads, creates the workbook shown in **Figure 1**.



	A	B	C	D
1	_4NW0UDLNE	_4NW0UDLNG	_4NW0UDLNH	_4NW0UDLNI
2	_4NW0UDLNO	_4NW0UDLNP	_4NW0UDLNQ	_4NW0UDLNR
3	_4NW0UDLNX	_4NW0UDLNY	_4NW0UDLNZ	_4NW0UDLO0
4	_4NW0UDLO6	_4NW0UDLO7	_4NW0UDLO8	_4NW0UDLO9
5	_4NW0UDLOF	_4NW0UDLOG	_4NW0UDLOH	_4NW0UDLOI
6	_4NW0UDLOO	_4NW0UDLOP	_4NW0UDLOQ	_4NW0UDLOR
7	_4NW0UDLOX	_4NW0UDLOY	_4NW0UDLOZ	_4NW0UDLP0
8	_4NW0UDLP6	_4NW0UDLP7	_4NW0UDLP8	_4NW0UDLP9
9	_4NW0UDLPF	_4NW0UDLPG	_4NW0UDLPH	_4NW0UDLPI
10	_4NW0UDLPO	_4NW0UDLPP	_4NW0UDLPQ	_4NW0UDLPR
11				

Figure 1. This formatted Excel workbook was created with XLSXWookbook.

Listing 6. This code creates a formatted Excel document.

```
#DEFINE BORDER_LEFT                     1
#DEFINE BORDER_RIGHT                    2
#DEFINE BORDER_TOP                      4
#DEFINE BORDER_BOTTOM                   8

loExport = newobject('VFPXWorkbookXLSX', ;
    'VFPXWorkbookXLSX\VFPXWorkbookXLSX.vcx')
with loExport
  lnWorkbook = ;
    .CreateWorkbook('ExcelTest.xlsx')
  lnSheet    = .AddSheet(lnWorkbook, ;
    'Test Sheet 1')

* Create some dummy data.

  for lnRow = 1 to 10
    for lnCol = 1 to 9
      .SetCellValue(lnWorkbook, lnSheet, ;
        lnRow, lnCol, sys(2015))
    next lnCol
  next lnRow

* Set the row and column heights.
```

```
  .SetRowHeight(lnWorkbook, lnSheet, 6, 25)
  .SetColumnWidth(lnWorkbook, lnSheet, 1, 25)
  for lnCol = 2 to 9
    .SetColumnWidth(lnWorkbook, lnSheet, ;
      lnCol, 14)
  next lnCol

* Set font, size, color, and style.

  .SetCellFont(lnWorkbook, lnSheet, 1, 1, ;
    'Calibri', 14, .T., .T., rgb(255, 0, 0))
  .SetCellFont(lnWorkbook, lnSheet, 2, 1, ;
    'Tahoma', , , , rgb(0, 0, 255))
  .SetCellFont(lnWorkbook, lnSheet, 3, 1, , ;
    14, .T.)
  .SetCellFont(lnWorkbook, lnSheet, 4, 1, ;
    'Arial', 14, .T., .T., rgb(0, 0, 255))
  .SetCellFont(lnWorkbook, lnSheet, 6, 1, , ;
    , , , , 'single')
  .SetCellFont(lnWorkbook, lnSheet, 7, 1, , ;
    , , 'double')

* Set borders.

  lnBorder = BORDER_LEFT + BORDER_RIGHT + ;
    BORDER_TOP + BORDER_BOTTOM
  .SetCellBorder(lnWorkbook, lnSheet, 3, 4, ;
    lnBorder, 'thin', rgb( 16, 100, 200))
  .SetCellBorder(lnWorkbook, lnSheet, 3, 6, ;
    lnBorder, 'thick', rgb(100, 150, 200))
  .SetCellBorder(lnWorkbook, lnSheet, 3, 8, ;
    lnBorder, 'double', rgb(200, 150, 100))
  .SetCellBorderEx(lnWorkbook, lnSheet, 5, ;
    2, 'thin', ,'thin', ,'thick', ,'thick')

* Save and open the workbook.

    .SaveWorkbook(lnWorkbook)
endwith
Execute('ExcelTest.xlsx')
```

Summary

Of the solutions I looked at, ExcelXML and XLSWookbook are the best for my needs, and I've used them both in production applications. They have the fewest drawbacks and the most flexibility in output.

In the next issue, we'll discuss importing from Excel into VFP data.

Doug Hennig is a partner with Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna.

Doug is co-author of "VFPX: Open Source Treasure for the VFP Developer," "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://vfpx.codeplex.com). He was a Microsoft Most Valuable Professional (MVP) from 1996 to 2011. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://tinyurl.com/ygnk73h).