Pharmasug 2011 - Paper AD21

Importing Excel 'Data' Into SAS Datasets Without Involving SAS

Stephen Hunt, ICON Clinical Research, Redwood City, CA

ABSTRACT

Everyone's favorite supplimentary data-storage utility isn't actually a database: Excel™, despite its quirks and eccentricies, has had no serious competition for ease of use and ubiquity across a host of industries. Therefore, the neccessity of importing information from Excel into SAS® has spawned literally hundreds of papers on the subject. So why do we need yet another? In a nutshell, virtually all solutions for moving Excel 'data' into SAS fall along a continuum: The 'minimalist' approach assumes it's best to pull data as pure text and then post-process one's way out of any problems that arise. In contrast, the 'technologist' approach assumes that either the SAS Import facility is available and able to import any Excel data via a few mouseclicks, or a SAS map file will be sufficient to import any workbook saved as an XML Worksheet. However, most approaches seem to have assorted drawbacks. For instance, many seem to have at least some technical limitations, whether it's special characters, loss of data as a result of cell formatting, lack of variable attributes, etc.. Additionally, very few techniques include import-validation to ensure that incoming data conforms to expected attributes. This paper attempts to address these items by using an entirely different approach for importing Excel workbooks.

INTRODUCTION

This paper will attempt to demonstrate a simple, 4-step method for transforming Excel data into ready-to-libname, attribute-full, data sets (as XML) without the benefit of having SAS anywhere present for the transformation. The only tool necessary is the Microsoft Office suite on a machine running the Windows™ OS. Additionally, the reliability of this method in comparison with other accepted SAS import techniques will be assessed.

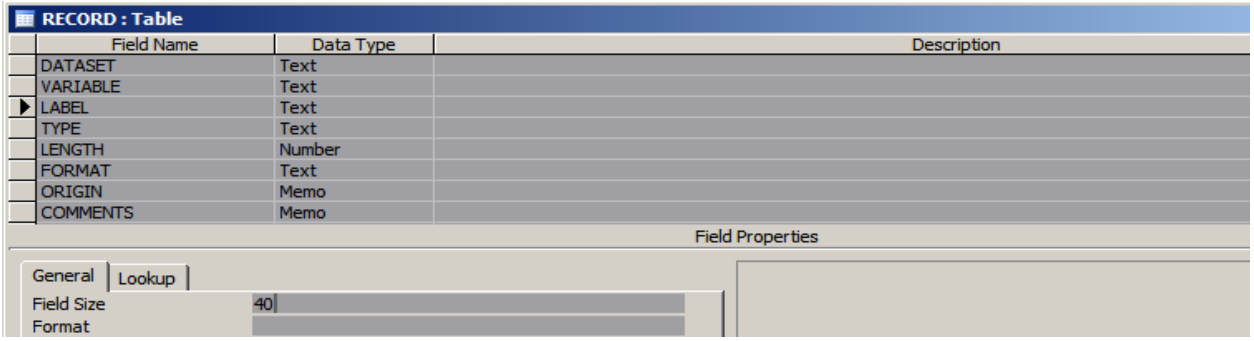
STEP 1: CREATE A SHELL DATABASE (OR IMPORT A SCHEMA) USING MS ACCESS

The most difficult part of this process may be the initial task of defining your own schema (and Microsoft actually makes this process quite easy). Fortunately, Access™ is more than happy to walk you through the process. To begin, you need to create an empty 'shell' database to represent the attributes of the spreadsheet you'll eventually import. For instance, lets say we want to import an Excel sheet with the following rows/columns:

	A	B	C	D	E	F	G	H
1	Dataset	Variable Name	Label	Type	Length	Controlled Terminology	Origin	Comments
2	D_DOSE	STUDYID	Study Identifier	Text	20	$20.	ADSL	= STUDYID
3	D_DOSE	SITE	Site Number	Integer	8	4	ADSL	= SITE
4	D_DOSE	SUBJID	Subject Identifier	Integer	8	8	ADSL	= SUBJID
5	D_DOSE	USUBJID	Unique Subject identifier	Text	12	$12.	ADSL	= USUBJID

Our first step is to create an Access database that could contain these fields: In other words, to define fields and attributes in Access that anticIpate the data to be imported. This can be easily done by selecting 'Create table in Design view' in an otherwise empty Access database:

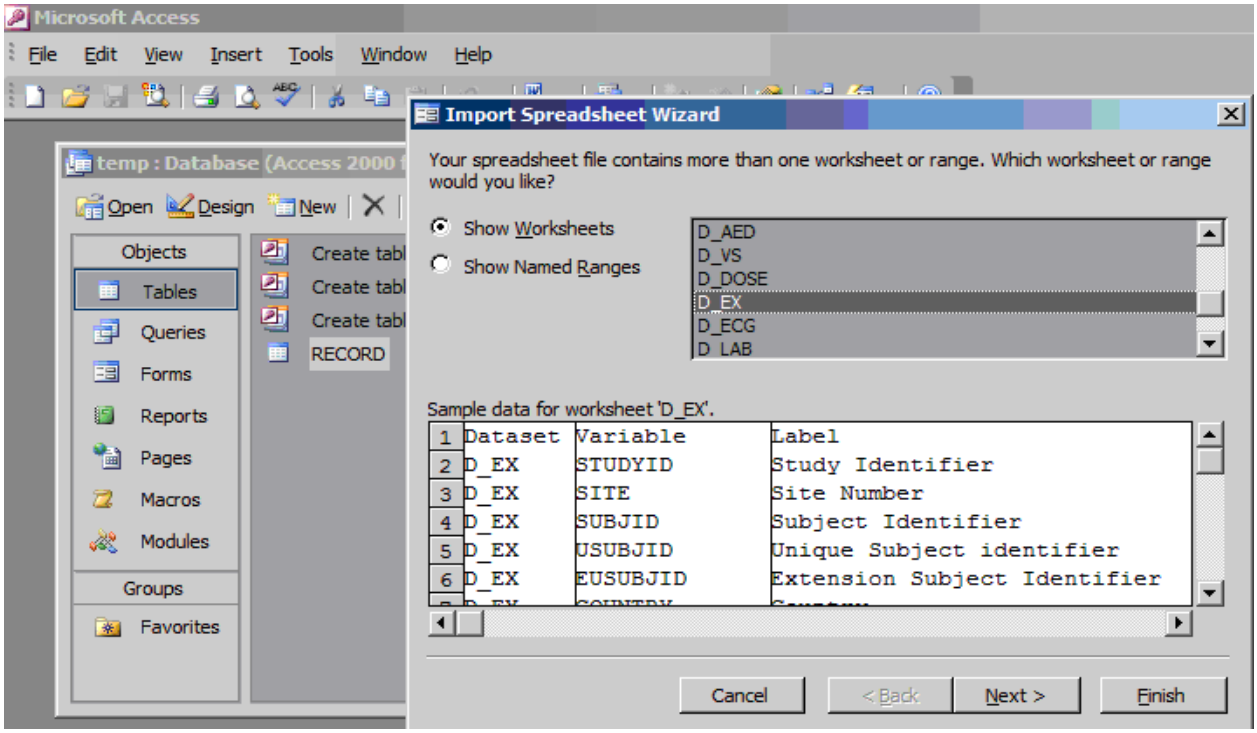
Next, we define all the fields to be imported and their respective attributes:



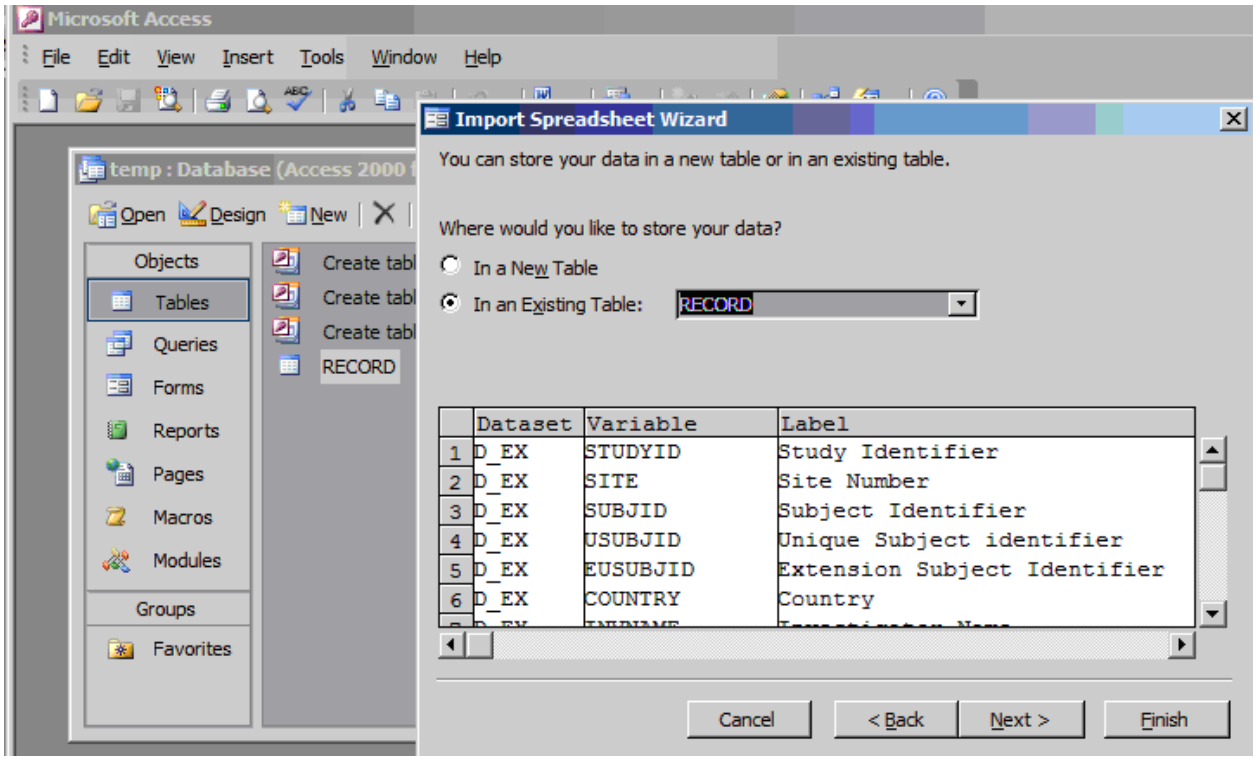
For our purposes in this paper, I'll simplify the attribute definitions my merely focusing on variable type and text variable length, although Access can apparently handle many more attribute details.

STEP 2: IMPORT YOUR EXCEL DATA INTO THE SHELL ACCESS DATABASE

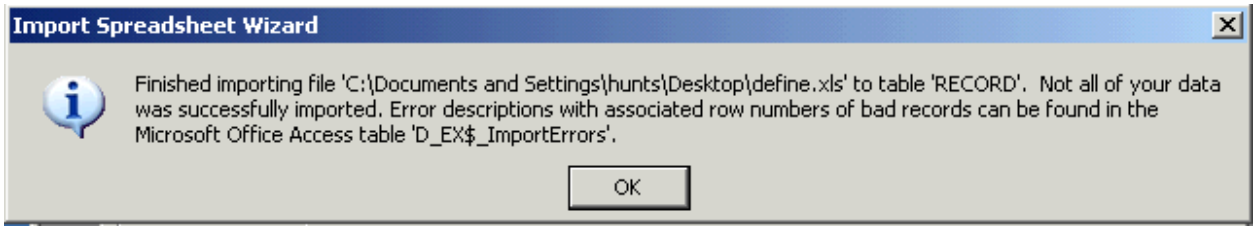
Once we've defined our attributes, we can then import the Excel sheets directly into our newly created table(s). Again, MS Access's import 'Wizard' makes this process practically painless:



The key to preserving your newly-created attributes in the incoming data is to dump it into your existing shell rather than allowing Access to create a new table by default:

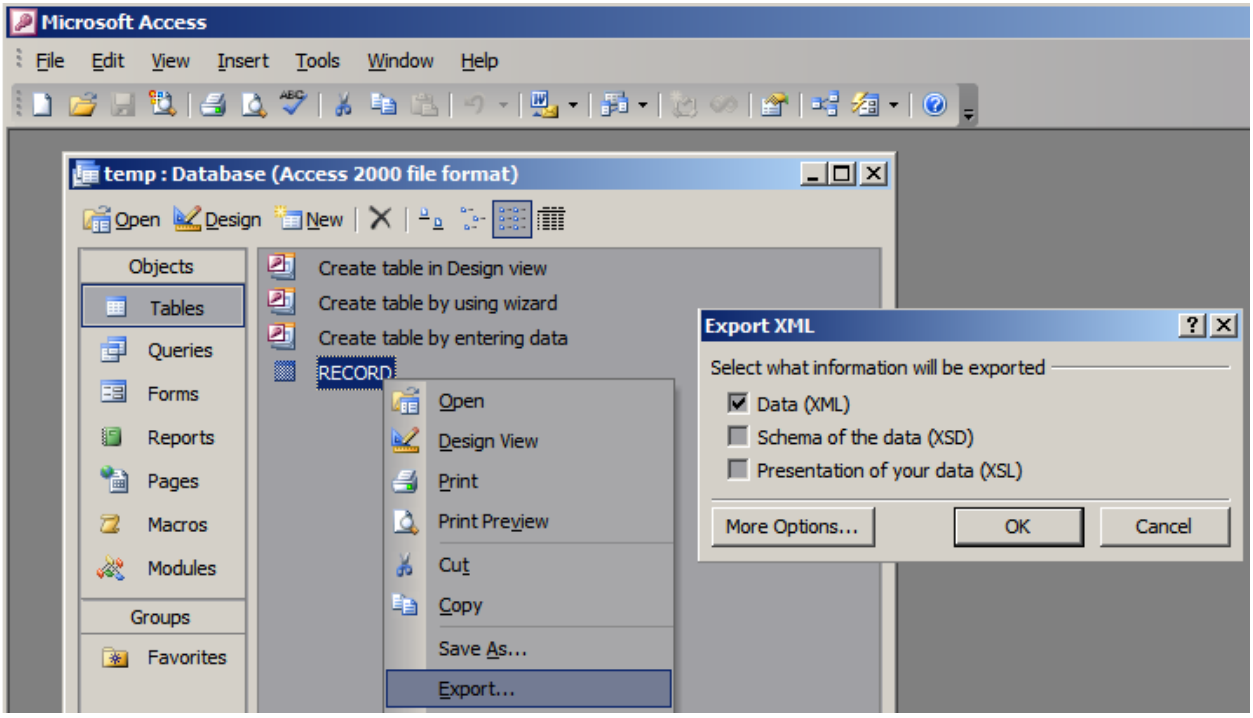


This is the stage where Access actually works for us to 'validate' the incoming data. For instance, attempting to import text into Access columns that have already been defined as numeric results in a warning from Access and renders the cells empty as a result. This is actually desirable behavior, since SAS reacts in a similar fashion when importing CSV files where input variables have already been defined via length or attribute statements. To demonstrate the complaint that Access raises when confronted with incompatible data types, we temporarily entered 'ADSF' into one of the cells in the LENGTH column, which had been pre-defined as numeric in our database shell.



STEP 3: EXPORT THE ACCESS DATABASE TO XML

This is yet another task where Access is more than happy to comply. In this case, we don't really need an Access schema or stylesheet (although use of a schema can be helpful for repetitive/dynamic importing – more on this later), so we check the 'Data' box to export a well-formed XML document only.



STEP 4: GENERATE A SAS 'MAP' FILE USING THE ATTRIBUTES PROVIDED BY AN ADODB.RECORDSET CONNECTION TO THE ACCESS DATABASE

SAS XML mapper isn't necessary to create a map file. However, a SAS 'map' file **is** necessary to ensure dataset attributes are assigned correctly when using XML as SAS data. Fortunately, there are only 3 components that need to be defined for each variable in a map file:

(1) The node path within the XML document,
(2) The variable type,
(3) The variable length.

As such, although it's relatively simple to write your own SAS map file manually as a text document, a much more reliable and efficient method is to use the following short VBScript file, which creates a ADODB.RECORDSET from the Access database and then uses the variable attributes stored there to dynamically generate a SAS map file. Using a list of ADODB recordset field properties found here (http://support.microsoft.com/kb/193947), we can actually translate any and all variable attributes from Access into terms that SAS uses in XML mapper. Here's the full script for creating a SAS map file from an Access database:

```
table_name="RECORD"  ' PLUG IN THE ACCESS TABLE NAME HERE.
mdb_file="temp.mdb"       ' PLUG IN THE ACCESS DATABASE/FILE NAME HERE. NO FURTHER CHANGES NEEDED AFTER THIS POINT.

dim fso, wshell
set fso=createobject("scripting.filesystemobject")
set wshell=createobject("wscript.shell")

currdir=fso.getfolder(wshell.currentdirectory)
database=currdir & "\" & mdb_file

set conn = createobject("ADODB.Connection")
conn.ConnectionString="Provider=Microsoft.Jet.OLEDB.4.0;" & "Persist Security Info=False;" & "Data Source=" & database
conn.Open

set x = conn.Execute("SELECT * from " & table_name)  ' Push the database into a recordset.

function saveschema(x)
  mapname=currdir & "\" & table_name & ".map"
  set mapfile=fso.createtextfile(mapname,true)

  mapfile.writeline "// MAP FILE REQUIRED BY SAS FOR DATA IMPORT OF XML, SINCE SAS DOES NOT KNOW WHAT TO DO WITH XSD SCHEMAS."
  mapfile.writeline "<SXLEMAP version='1.1' name='SAS_MAP'>"
  mapfile.writeline "  <TABLE name='" & table_name & "'>"
  mapfile.writeline "  <TABLE-PATH>/dataroot/" & table_name & "</TABLE-PATH>"

  numf=x.fields.count-1

  'Translate ADODB field types into SAS variable and data types.
  for i=0 to numf
    fname=ucase(x.fields(i).name)
    if x.fields(i).type=2 or x.fields(i).type=3 or x.fields(i).type=16 or x.fields(i).type=17 or x.fields(i).type=18 or x.fields(i).type=19 or x.fields(i).type=20 or  _
      x.fields(i).type=21 or x.fields(i).type=7 or x.fields(i).type=133 or x.fields(i).type=131 then
      ftype="numeric"
      flen=8
      if x.fields(i).type=2 or x.fields(i).type=3 or  x.fields(i).type=16 or x.fields(i).type=17 or x.fields(i).type=18 or x.fields(i).type=19 or x.fields(i).type=20 or  _
        x.fields(i).type=21 then
        dtype="integer"
      elseif x.fields(i).type=7 or x.fields(i).type=133 then
        dtype="date"
      elseif x.fields(i).type=131 then
        dtype="float"
      end if
    else
      ftype="character"
      dtype="string"
      flen=x.fields(i).definedsize
      if flen>2000 then
        flen=2000   'Access allows for far greater variable lengths than SAS. Adjusting to max of 2000 here.
      end if
    end if

    mapfile.writeline "     <COLUMN name='" & fname & "'>"
    mapfile.writeline "       <PATH>/dataroot/" & table_name & "/" & fname & "</PATH>"
    mapfile.writeline "       <TYPE>" & ftype & "</TYPE>"
    mapfile.writeline "       <DATATYPE>" & dtype & "</DATATYPE>"
    mapfile.writeline "       <LENGTH>" & flen & "</LENGTH>"
    mapfile.writeline "     </COLUMN>"
  next

  mapfile.writeline "   </TABLE>"
  mapfile.writeline "</SXLEMAP>"
  mapfile.close
end function
saveschema x
```

To reiterate, the above code is to be copied/pasted into a text document and saved with the extension '.vbs' into the same folder where the Access database is present (.mdb file). By merely modifying the name of the file and the database name within it in this VBS file (i.e., the first 2 lines of code), you then only need to double-click on the file (VBS is self-executing) to generate a map file from *any* Access database.

The last piece of the puzzle, reading the XML file into SAS, is simple and well-covered in a number of other papers, but is reprinted below for convenience. Here's how SAS can read in well-formed XML via filename/libname statements:

```
filename myxml 'c:\record.xml';
filename sxlemap 'c:\record.map';

libname myxml xml xmlmap=sxlemap;

data work.myex;
   set myxml.record; *** RECORD IS THE NAME OF THE ORIGINATING ACCESS TABLE.;
run;
```

COMPARING RESULTS

So, how does the above approach stack up against the established, SAS-sanctioned methods? For a test, we compared sheets imported using the technique in this paper against what is likely SAS's best recommendation of saving Excel sheets as an XML Worksheet and importing them using their excelxp.map file. The following is a summary of observed differences:

(1) SAS's approach doesn't pre-supply any variable attributes. As with proc import, attributes are detected automatically by SAS from among a pre-specified number of records. As such, the attributes that SAS applies may vary between identically structured sheets that just happen to contain differing records. With our approach, variable attributes are fixed:

Data Set Summary

Dataset	Created	Modified	NVar	NObs
WORK.MYEX	16NOV10:16:07:37	16NOV10:16:07:37	8	33
WORK.D_EX	16NOV10:16:07:36	16NOV10:16:07:36	8	33

Variables Summary

Number of Variables in Common: 8.
Number of Variables with Conflicting Types: 1.
Number of Variables with Differing Attributes: 7.
Number of ID Variables: 1.

Listing of Common Variables with Conflicting Types

Variable	Dataset	Type	Length	Label
LENGTH	WORK.MYEX	Num	8	LENGTH
	WORK.D_EX	Char	5	Length

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Label
DATASET	WORK.MYEX	Char	8	DATASET
	WORK.D_EX	Char	4	Dataset
VARIABLE	WORK.MYEX	Char	8	VARIABLE
	WORK.D_EX	Char	8	Variable
LABEL	WORK.MYEX	Char	40	LABEL
	WORK.D_EX	Char	41	Label
TYPE	WORK.MYEX	Char	8	TYPE
	WORK.D_EX	Char	7	Type
FORMAT	WORK.MYEX	Char	8	FORMAT
	WORK.D_EX	Char	8	Format
ORIGIN	WORK.MYEX	Char	2000	ORIGIN
	WORK.D_EX	Char	7	Origin
COMMENTS	WORK.MYEX	Char	2000	COMMENTS

(2) Without any variable attributes to 'steer' import validation, the SAS 'excelxp.map' method doesn't give any kind of note or warning about the artificial difference we created in variable type (recall the 'ADSF' entered into one of the numeric cells mentioned previously). Additionally, variables intended to be a specific length (e.g., SAS label fields for a define.xml) are allowed to exceed this requirement without qualms using SAS's method. Field validation, as built in to Access's import is preferable in these instances.

(3) Although all special characters that we tested (e.g., tab and carriage returns) were rendered appropriately by both methods (which certainly isn't the case for some import methods – e.g., CSV), some formatted text values (e.g., strikethoughs and unusual font types or colors, as demonstrated below) ended up being set to missing by the excelxp.map approach. Our method using Access as an intermediate to XML file kept these values present as they appeared in the original Excel workbook:

COMPL	Computed Compliance (%)	Integer	8	5.1	*Derived*	~~= 100*TAKEN/(NBDAY*6)~~ Merge with D_DOSE on VISITNUM. Identify the number of tablets not taken during each visit as Sum[D_DOSE.DSTBNTK*(D_DOSE.DSNDRC)] Planned number of tablets (denominator) = (6*(NBDAY)) - number of tablets not taken during visit. Compliance = [TAKEN / planned number of tablets] * 100
OCOMPL	Overall computed compliance (DB + OL) (%)	Integer	8	5.1	*Derived*	Merge with D_DOSE on VISITNUM. Identify the number of tablets not taken overall as Sum[D_DOSE.DSTBNTK*(D_DOSE.DSNDRC)] Planned number of tablets (denominator) = (6*(NBDAY)) - number of tablets not taken overall. Compliance = [sum(TAKEN) / planned number of tablets] * 100 where period = 1 or 2

The comment fields for the 2 records above ended up as blank when using the excelxp.map import method.

The COMPARE Procedure
Comparison of WORK.MYEX with WORK.D_EX
(Method=EXACT)

Variables with Unequal Values

Variable	Type	Len1	Len2	Label	Compare Label	Ndif	MaxDif	MissDif
COMMENTS	CHAR	2000	380	COMMENTS	Comments	2		2

Value Comparison Results for Variables

	COMMENTS Comments Base Value	Compare Value
VARIABLE	COMMENTS	Comments
COMPL	= 100*TAKEN/(NBDAY*6	
OCOMPL	Merge with D_DOSE on	

CONCLUSION

Access is a true database, and as such lends itself well to performing the tasks as an intermediary between Excel and SAS. As a SAS-server-user using lacking any desktop SAS installation, having a method that allows one to import Excel files while disconnected from SAS itself (e.g., using a laptop in the airport or other area lacking an internet connection) can be advantageous. Additionally, reliability is enhanced while allowing for some degree of attribute validation without requiring SAS to participate directly in the import. Since Access is a scriptable application, a secondary benefit of the method described in this paper can be derived by automating the entire process. Applying a schema, importing a multi-sheet workbook, writing out XML from Access and creating a SAS map file can all be scripted. Such a VBScript has been developed by the author and can be provided upon email request (please see contact information below).

REFERENCE

Fairfield-Carter, Brian, Sherman, Tracy, and Hunt, Stephen (2005), 'Instant SAS® Applications With VBScript, Jscript, and dHTML', Proceedings of the 2005 SAS Users Group International Conference.

Vincent DelGobbo (2004), 'From SAS® to Excel via XML', Proceedings of the 2004 SAS Users Group International Conference.

ACKNOWLEDGMENTS

I'd like to thank Brian Fairfield-Carter for his technical expertise as well as my friends in Biostatistics and Programming at ICON Clinical Research for their consistent inspiration, encouragement and support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Stephen Hunt
ICON Clinical Research
Email: stephen.hunt@iconplc.com