PAPER AD02

Analysis software in .NET and SAS®

Jules van der Zalm, OCS Consulting, 's-Hertogenbosch, the Netherlands

ABSTRACT

For one of its clients, OCS Consulting has developed an application which offers special statistical functions to the users. The application allows for the entering of test results and all descriptive information belonging to a test. From this information, the application can produce several graphs and calculations that allow for the analysis of the test results. This is a client server application that consists of a familiar looking user interface developed with Microsoft .NET technology using SAS® 9.1.3 server on the back-end taking care of all user interactions and the necessary statistical procedures. The data is stored in an Oracle database to allow multiple users to perform data entry.

This papers details the technology used. Both the paper and the presentation will present visual representations of an example user interface and graphical interpretations of the interaction between the interface and the SAS back-end.

INTRODUCTION

As a SAS software company, OCS is often requested by clients to develop applications in SAS that allow for the use of the powerful SAS procedures and functions but one the other hand have a familiar interface that allows for intuitive interaction with the user. Unfortunately current versions of SAS do not give you the option to develop an advanced user interface around your SAS programs.

The subject of this paper is a methodology that combines the advanced calculations and graphics of SAS procedures with the strength, flexibility and familiar looks of a graphical user interface. The user interface is an application on its own that communicates with SAS on the background. It can be created in any 3rd generation programming language. The latest application that OCS Consulting developed using this methodology was built in C#.

The methodology consists of five major components which are outlined in the diagram below and explained further in this paper.

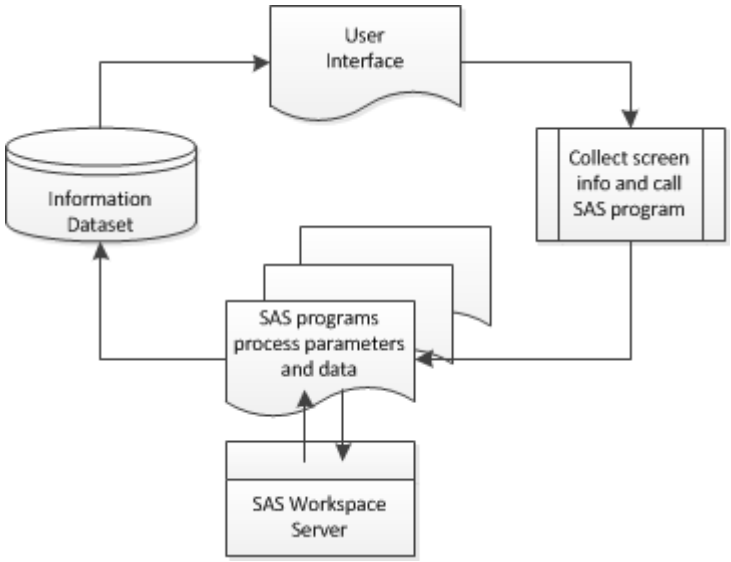

Image 1: A schematic overview of the five major components.

THE USER INTERFACE

The user interface is developed separately from the SAS backend. Its design is based on the functional and technical specifications of your application. In its final stage it should contain all the possible elements that should be presented on the screen. Such elements may be text labels, text boxes, dropdown menus, sliders, spinners, tab controls, list boxes and many more. Basically, any type of element that you're used to from applications that you may already use. These elements may not always be visible to the user, or their visibility may depend on the user's permissions or on the stage the application is in. Either way, any possible element that should be presented to the user at any time should already be in the user interface. The processes on the SAS backend decide whether or not each element is displayed, and in which way.

Elements can have an initial value set by the user interface. This is especially useful for labels, as these don't tend to change often. Some elements on the user interface trigger a certain event. This event can be an action performed by the user interface itself, like disabling a certain interface element, or it may be a call to a SAS program. In which case communication with SAS is initiated, the name of the program that is to be called must be defined in the interface.

Since the user interface is completely independent from SAS, it does not matter in which language it is programmed. Moreover, the user interface can be completely replaced with a user interface written in a different language based on the same technical design and the SAS backend doesn't even require any changes. The same goes for the database: by using the proper techniques and SAS procedures, the database can be changed from one system to another (e.g. MS SQL to Oracle) with only the slightest change to the SAS programs.

The next paragraphs will go into a bit more detail on the complete process of communication between the user interface and SAS.

A SIMPLE EXAMPLE

Before this paper goes in-depth on the matter, this paragraph will present to you a simple example representing the actual topic of the paper. Have a look at the mockup below.

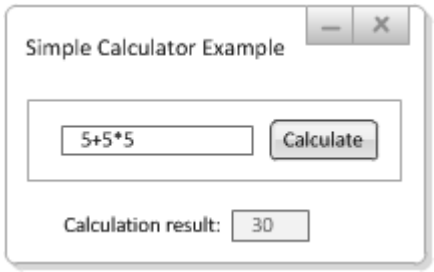

Image 2: A simple representation of a user interface.

This very simple application form represents a user interface. The user is expected to enter a formula in the text field and then hit the 'Calculate'-button. The interface will then come back with the outcome of the formula. To you as a user it looks nothing like SAS, but on the background SAS is actually involved.

The user interface is connected to a SAS service either on the local host or on a centralised server, depending on the working environment. Upon clicking the 'Calculate'-button, the user interface sends SAS code over to the SAS server, which executes it. Such code may look like the following.

```
%let textbox1=5+5*5;
%include "calculate.sas";
```

The SAS program that is included performs the calculation using the entered value as input and sends back the result to the interface and the interface presents it to the user. Of course, in this example the formula entered in the textbox must be one that is compatible with the SAS syntax. The communication to SAS and sending back the results are the key to the methodology and are detailed in the rest of this paper.

This very simple example can be extended to complex applications with many different SAS programs being called performing many kinds of data processing. It's even possible to have SAS create its graphs and have those presented in the interface.

THE SAS WORKSPACE SERVER

The SAS Workspace Server is a service that is provided by SAS. It comes as a stand-alone service, but you may also know it from the background service that Enterprise Guide uses. The Workspace Server is a service that holds your SAS session, runs your SAS programs and stores your datasets. Multiple connections may be made to the service at the same time. Each connection gets its own session, so that users do not interfere with each other's session.

When the user interface requires SAS interaction, it communicates with the SAS Workspace Server. A session is initialised upon starting up the user interface. The SAS Workspace Server receives the code that is sent by the user interface. It interprets the code, executes it and stores the result in the (work) library. At the point where it's finished processing, the workspace server sends a SAS return code to the user interface to tell it that the result is ready for processing. What this result exactly is will be explained further in this paper.

THE INTERFACE TALKING TO SAS

The user interface application, programmed in for instance C#, makes use of Integration Technologies to connect the SAS Workspace Server. This allows for the submission of SAS code to the workspace server. The SAS code is generated by the user interface based on the fields on the screen. Those fields, or moreover the information therein as entered by the user, are the parameters to the SAS program being processed. In the simple example presented before, there is only one field that we call 'textbox1'. Suppose the user filled in the value '5+5*5' as a formula, then the application generates SAS code that generates a macro variable textbox1 with value 5+5*5: %let textbox1=5*5+5.
In a bit more sophisticated interface you could imagine two textbox and a dropdown box. In each textbox you would enter a single number and in the dropdown box you would select an operator. The code presented below is what the application would then generate and send to the SAS workspace server when the 'Calculate' button is clicked.

```
%let textbox1=5;
%let textbox2=10;
%let operator=+;
%include "calculate.sas";
```

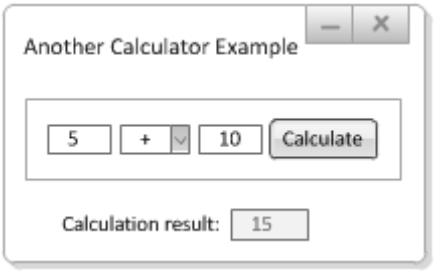

Image 3: Another calculator example

As you can see the generated code does not only create macro variables based on the fields on the form, but is also includes a SAS program. This is the SAS program that performs the calculation and reports the result back to the interface. The name of this program is defined in the C# interface code.

SAS RESPONDING TO THE USER INTERFACE

Once the SAS program is finished, what remains is a (work) library with perhaps several datasets and maybe some additional macro variables. Programs may even put graphs or PDF files in a certain

location on the hard drive. All in all it's probably not really consistent between different SAS programs and therefore not something that the interface can work from.

Therefore, what's required is a uniform and structural approach. This is achieved by creating a so called information dataset. This dataset with a fixed structure describes the results of the SAS program by translating it to each field on the user interface that should be filled. In the example of the calculation program the information dataset may look like the following:

FIELD	TYPE	VALUE	CHARNUM	ENABLED
CalcResult	TextBox	15	NUMERIC	0

The variable FIELD represents the name of the field which is a TextBox, as described in TYPE. The VALUE that should be presented in this field is 15, which´ TYPE is numeric. The field is not ENABLED, making the element greyed out on the screen as this is not a user-editable field.

The creation of this information dataset is the end result of a SAS program. After the program has completed processing a SAS return code is passed to the interface by which it knows that it can start reading the information dataset. The interface reads the dataset observation by observation, and when the complete dataset is processed the user interface elements are populated with the values as calculated by the SAS program.

The columns in the information dataset are not fixed. They can be any number of columns, depending on the requirements for your application. You may want to add certain columns to the information dataset to describe more metadata for your user interface elements, e.g. for colours or 'visible or not'.

INCORPORATING SAS GRAPHS AND DOCUMENTS

Although the information dataset is an efficient way of communicating to the user interface it only allows for the sending over of textual and numeric values. You simply cannot put a graph or a document in a dataset, even though the user interface could be prepared for the representation of any of those. Fear not, there is a way to overcome this shortcoming.

Every SAS procedure that creates a graphical image or a document (PDF, RTF, HTML, et cetera) has an option to save the file to a specified location. This location may be any place on the local hard drive or somewhere on the network. What is sent to the interface is the location to which the file is saved. The interface can read from that location (if, of course, that location is shared with the local computer and user) and present the image or document on the screen.

COMPLEXER PROGRAMS

Simple examples are nice to show and explain functionality, but in the real world applications would never be as simple as the examples shown before. More complex applications provide more functionality but are also more difficult to develop and maintain. Therefore it's important to apply an efficient structure to your programs. Separating them by purpose, placing them in distinct folders and labelling them makes life easier.

For separating, we chose to define three different purposes which we call layers. More experienced programmers may find that these layers sound familiar. Not by coincidence, because the layers are both used in the interface (developed in C#.NET) and on the SAS side. This paper will only go into detail on the SAS side.

Each layer is basically a set of SAS programs placed in a designated directory. Each program has a specific function to the application and is, preferably, a SAS macro.

On the top there is the Presentation Layer. This contains all the programs with which the user interface communicates. They can accept interface parameters and communicate back to the interface by creating information datasets. Ideally, programs in the Presentation Layer don't perform any kind of logic other than talking to the layer below and the user interface. The Presentation Layer is the communicator between the Business Logic Layer and the user interface.

The Business Logic Layer is the layer where the magic happens. Any processing, calculating, image rendering, document creation et cetera is performed in this layer. None of these programs are ever called by the interface directly, nor does it generate information datasets for the user interface to interpret. All of this is done by the Presentation Layer.

The Data Layer is the layer that's responsible for communication with a database. Databases are the essence of practically all application as everything, especially in pharmaceuticals, thrives on data. Being an Oracle or MS SQL database or just SAS datasets, communicating with the data is done through the Data Layer. Some (transactional) database systems may even allow multiple users to read and write from the tables at the same time, allowing for example multi-user data entry.

The communication between the layers is presented visually in the image below.

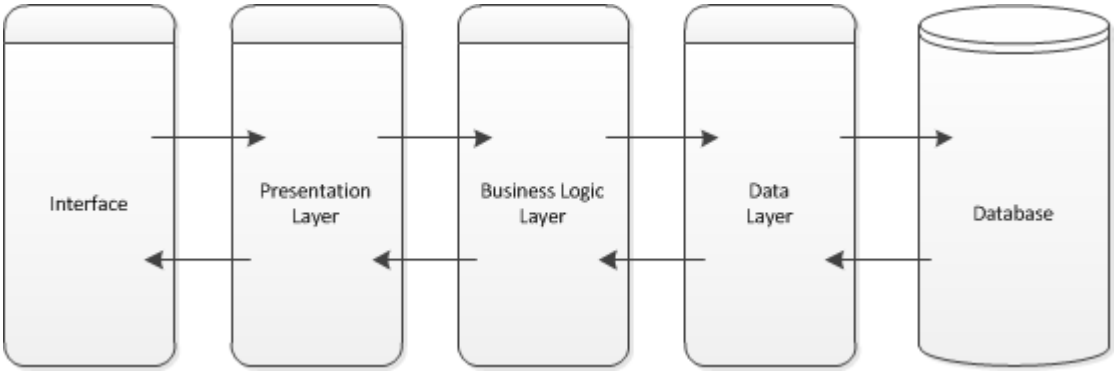

Image 4: The interface, the layers of the SAS programs and the database.

As mentioned before, this architecture allows for modifying or replacing any layer with minimal to no changes to the other elements in the methodology. Therefore changing for instance a database, requires minimal changes in the data layer leaving the rest of the application unaffected.

Since the C# interface has similar layered architecture the application can be migrated to a web interface changing only the presentation layer and none of the Business Layer or Data Layer in the interface, and none of the SAS programs.

A COMPLEX EXAMPLE

Now that the theory has been discussed, it's time for a bigger example. This example is based on an application that allows for the entry of laboratory measurements and the visual representation of those measurements. A mock-up of the user interface is presented below. For the example some elements had already been filled in before and are loaded back upon loading the patient data.

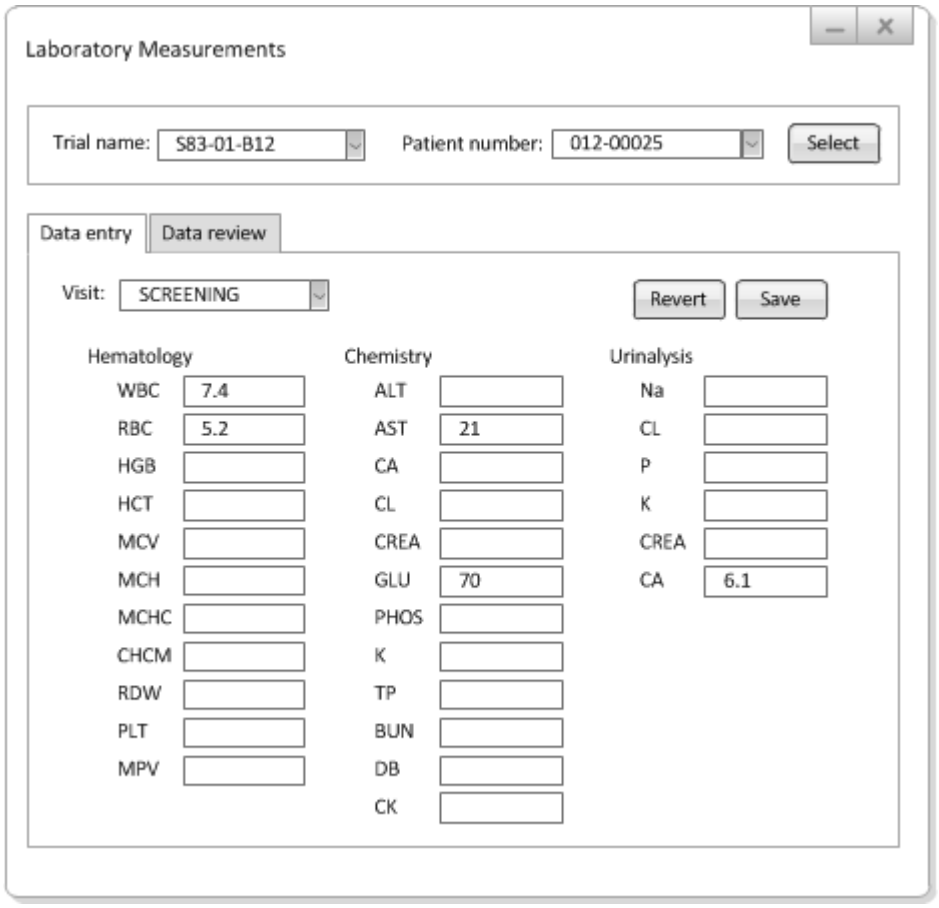

Image 5: A more complex visualization of a user interface.

The interface consists of a selection area and two tab pages. The left tab entitled 'Data entry' is meant for the entry or correction of laboratory measurements for subjects that have entered the selected trial. The tab entitled 'Data review' is meant for the display of a visual representation of the data. It allows the user to see the measurements of certain lab parameters over time.

An application like this is completely data driven. The list of trials in the dropdown on the top is derived from the trial data. The dropdown with list of patient numbers next to it lists all patients that have had a screening visit in the selected trial. As such, you cannot select the patient number before the trial is selected.

Once the patient number is selected and the Select-button is clicked, the available visits for that patient will be populated in the dropdown box Visit. If the visit is selected, the data driven entry fields to be presented are retrieved from the trial specifications containing the laboratory parameters (WBC, RBC, HGB, et cetera). These entry fields are then populated with already available values, if any.

POPULATING THE USER INTERFACE

Without going into detail about the actual SAS code that is used to generate the information dataset, this is what the information dataset would look like when sent to the interface after a visit is selected. The information dataset describes the elements to be displayed on the screen, and their initial values, as seen in the screenshot.

#	FIELD	TYPE	VALUE	CHARNUM	VISIBLE	ENABLED	DEFAULT
1	LblCol1Header	Label	Hematology	CHAR	1		
2	LblCol1_1	Label	WBC	CHAR	1		
3	LblCol1_2	Label	RBC	CHAR	1		
…	…	…	…	…	…		
13	LblCol1_12	Label		CHAR	0		
14	LblCol2Header	Label	Chemistry	CHAR	1		
…	…	…	…	…	…		
27	LblCol3Header	Label	Urinalysis	CHAR	1		
…	…	…	…	…	…		
33	LblCol3_6	Label	CA	CHAR	1		
…	…	…	…	…	…		
39	LblCol3_12	Label		CHAR	0		
40	TbCol1_1	Textbox	7.4	NUM	1	1	
41	TbCol1_2	Textbox	5.2	NUM	1	1	
42	TbCol1_3	Textbox		NUM	1	1	
…	…	…	…	…	…		
52	TbCol2_2	Textbox	21	NUM	1	1	
…	…	…	…	…	…		
43	TbCol2_6	Textbox	70	NUM	1	1	
…	…	…	…	…	…		
68	TbCol3_6	Textbox	6.1	NUM	1	1	
69	BtnRevert	Button	Revert		1	1	
70	BtnSave	Button	Save		1	1	
71	CbVisit	Combobox	SCREENING		1	1	1
72	CbVisit	Combobox	BASELINE		1	1	0
73	CbVisit	Combobox	VISIT 1		1	1	0

NOTE: Where a row says '…', interpret the whole list of elements. It would take up a large chunk of space in this paper to put all the element definitions in there.

What does this tell the interface? Let's go through the information dataset record by record.

Record #1 states that the interface element titled LblCol1Header (that is the name chosen by the interface programmer) should have the value 'Hematology'. The 1 under 'Visible' means that the label should actually be shown on the interface. Records #2 through #39 populate the rest of the labels. They are the labels that show the laboratory parameters and the group headers ('Chemistry' and 'Urinalysis'). Their text is based upon what's in the trial's lab dataset. The interface allows for a list of

12 parameters per column (in this example). If a group of parameters has less than 12 items, then the remaining labels should not be shown on the interface. In such cases the information dataset columns 'Visible' is used; it is set to 0.

Records #40 through #68 take care of the actual laboratory values, the textboxes in which values can be entered. In the example, some textboxes already had value when the visit 'SCREENING' was selected. Records #40 and #41 populate such a textbox by stating the value. Record #42 'populates' an empty textbox; the value is missing in the information dataset. Again, if there are less than 12 laboratory parameters to be entered, remaining textboxes are hidden by setting Visible to 0.

Records #69 and #70 define the buttons on the interface. The value represents the text that is displayed on the button. If required the buttons can be disabled so they can't be clicked. In this example that would be useful, for example, when the user is not allowed to save values but only see them.

Comboboxes are very special elements. A combobox is the kind of item that is used on this interface to select the visit. A single combobox can (and usually will) contain multiple items, of which one should be selected by the user. All the items in the combobox must be mentioned in the information dataset. One of the elements can be set as the default, telling the user interface to select the value when the interface loads. In the example, the records #71, #72 and #73 represent the items in the combobox for the visits. They all have the columns 'Default' filled, but only the one that is to be selected by default has a 1 in there. This will therefore be the selected value on the screen.

OTHER INTERFACE ELEMENTS

The list of possible user interface elements is not limited to those mentioned in the examples above. Checkboxes, radiobuttons, spinners, listboxes, sliders, datagrids and graphs, amongst others, belong to the list of elements that can be used to populate the user interface. The handling of these via the information dataset is pretty straight forward for most of these. For datagrids and graphs that doesn't hold true.

Datagrids are the elements that display tabular data on the screen. They are basically one single element, but that element displays a range of cells, like an Excel spreadsheet does. Those cells can be populated through SAS programs, but they can also be used for data entry.

To populate a datagrid through SAS, the information dataset is not used in the regular way. The grid is populated based on a set of two datasets: one containing the actual data to be presented, and one that describes the columns that are presented, the metadata. The metadata describes for example the width of the columns, whether they can be edited or not, the header label and their value type (numeric or character). The information dataset only holds one single record that mentions the names of the two datasets. The datagrid is then created based on the 'metadata'-dataset and then populated based on the 'actual data'-dataset.

Something similar goes for graphs. The information dataset simply cannot hold an actual image. The graph element on the interface receives only the path and filename of an image through the information dataset. The referred image is at that time already created by, for instance, the PROC SGPLOT procedure in the SAS program that created the information dataset. You could even generate multiple graphs (e.g. of different time windows) and use a slider to show a different picture for each slider position, like in Image 6. That way the user can have the feeling of scrolling through the graph while they are actually static images.

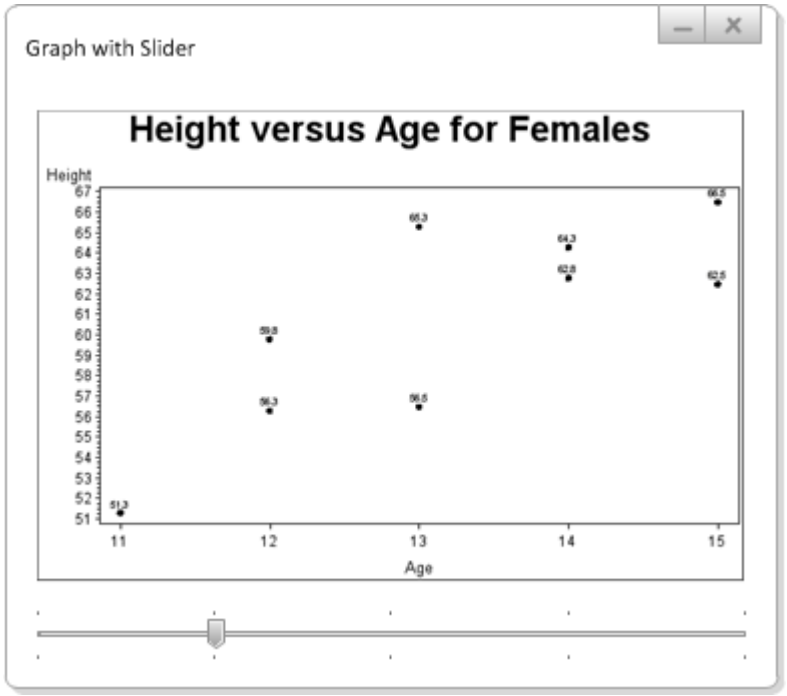

Image 6: A slider presenting multiple graphs simulating a scrolling experience.

THE POTENTIAL / CONCLUSION

It may be very well clear that the methodology presented provides great potential to the user experience of SAS applications. Data entry, data review, graphical presentation and document integration tailored to your specific needs are only a tip of the iceberg.

Having the user interface separated from the data handling and processing makes the application an easily controlled environment. Moreover, the users cannot read or edit the application or the SAS programs. As long as the user interface and the SAS programs maintain their validated status (which would normally be until a new version is released) the output that is generated does not require additional validation.

When designing an application that makes use of the methods presented here it's very important to come up with proper functional and technical design documents. Carefully describe what the application should and should not do, decide which functionality is to be coded in SAS and which in your interface programming language and document your interface elements well. This will all makes it easier to develop and maintain your future application.

ACKNOWLEDGMENTS

I'd like to thank my colleague Raymond Ebben for reviewing this paper and my colleagues Fred Junier, Chris Gibby and Tao Cheng for working with me on the development of an application using this methodology.

CONTACT INFORMATION

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

Jules van der Zalm
OCS Consulting
Ruwekampweg 2g
5222 AT 's-Hertogenbosch
The Netherlands

+31 (0)73 526 6000
sasquestions@ocs-consulting.com
www.ocs-consulting.com/nl