

SAS Standard Reporting Application For e-publication Of MS Office Documents

Simon Pickles, Fidelity Investments Ltd

Abstract

A important component of the Data Warehouse is the production and distribution of high quality Static Reports that can be read by any member of the company with a standard desktop set up. This paper describes a SAS application that schedules the production of sophisticated reports in MS Office format and distributes them via email or on company Intranet to Business community. This is a mainly technical paper concentrating on the application design and programming techniques used. The techniques discussed will be mainly SAS BASE, AF and Excel OLE Automation although other areas will be mentioned (DDE, SAS ODS, JAVA and ASP Pages). SAS Platforms are Windows NT and Unix in Client Server mode.

Introduction

One of the first requirements to be encountered when developing a Data Warehouse solution is the ability to automate the delivery of regular static reports to business user's desktops in a high quality, readable, printable format. In my experience, until recently this realistically meant MS Office documents although HTML and PDF are now becoming acceptable. With increasing user sophistication the challenge is to keep up with expectations.

The Standard Reporting Application described in this paper has functionality beyond that described above but it is the satisfaction of that requirement that will be addressed in most detail. The full list of functional requirements would include:

- Users can request and run reports interactively with output returned to the desktop.
- Users can request and schedule reports for regular run.
- Output should be in format readable by all Business users with Standard Desktop set up (i.e. MS Office).
- Output from scheduled reports should be delivered by email or on a company Intranet Site.
- Output should be printable where appropriate.
- Reports should be recorded and archived.
- The interface should allow parameter entry for filtering and adjusting summarisation level.
- Ideally all this should be available without SAS on the users desktop.

Application Structure And Components

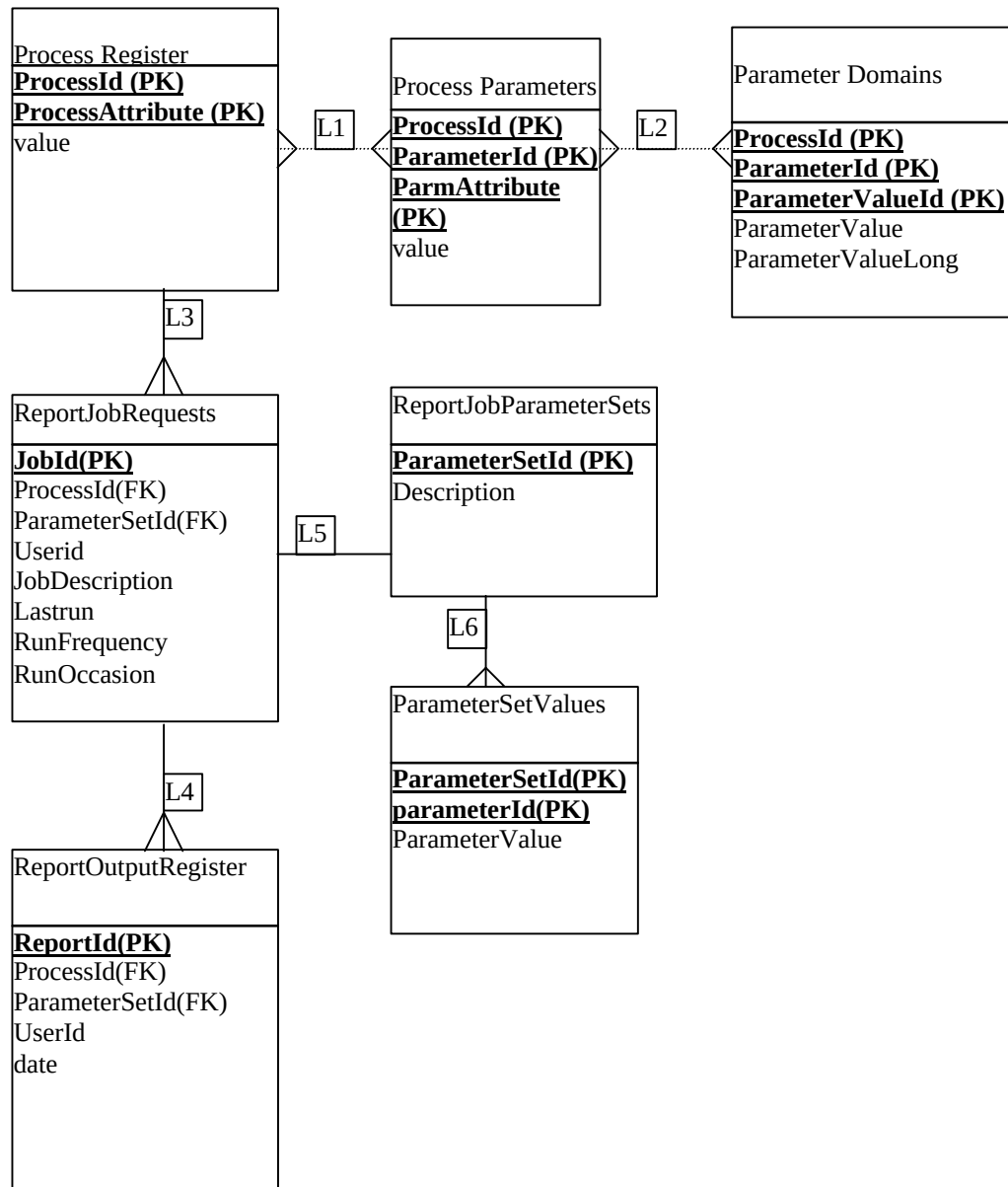
The Standard Reporting Application can be divided into the components described in the table below. Identification of these separate components is important when future enhancements and changes of architecture are considered.

Component	Platform(s) + Software	Description
Application Control Data / Metadata	Unix Server SAS Data Sets	
Business Data Base	Unix Server SAS Data Sets	Client, Account, Sales data etc.
Report Request Interface	Windows SAS/AF	Interface for requesting reports, entering parameters and viewing reports.
Report Scheduling	Unix Server BASE SAS Code	Code to control the running of requested reports at requested frequency.
Report Query	Unix Server BASE SAS Code	Code to query Business Database and create result set(s).
Result Sets	Unix Server SAS Data Sets	Output from the Report Query. These are named in a way that they are linked to the ReportId recorded during the run.
Report Formatting	Unix Server SAS BASE. Windows SAS BASE + AF/SCL.	Formatting of Result Sets into Reports for end users. Output formats include Excel, HTML and PDF.
Publication	1. Email from Unix Server BASE SAS. 2. Intranet web server site populated via ftp from Unix BASE SAS	Delivery of reports to users desktops via email or Intranet Site.

Application Control Data

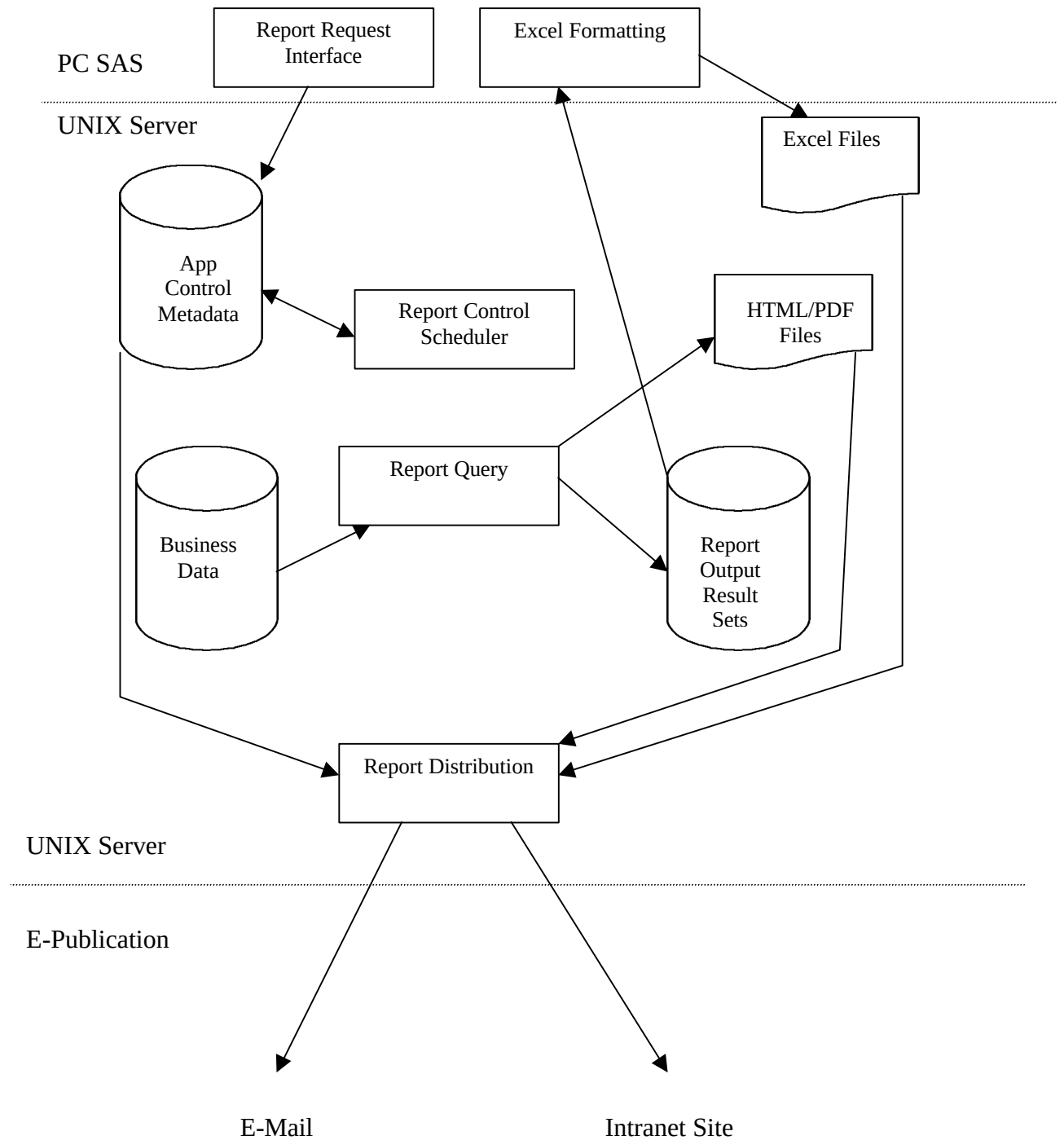
Table	Contents
Process Register	Each Report is registered in the Data Warehouse Process Register with a special Attribute identifying it as a Report. Other Special attributes include: - Report Excel Formatting Routine - SCL Entry - Report Format Type - Excel, HTML, PDF - Report Subject Area - Assets, Sales, Exceptions - Report Department - Direct Marketing, Finance - Excel Template File
Process Parameters	Each Report process has an associated set of Parameters each with Parameter Attributes including: - Default value - Description For Users - As it appears On the Screen - Edit method - SCL/FAME Entry for entering Values.
Parameter Domains	The list of values a particular parameter can take.
Report Job Requests	Each Request of a report Process generates a Report Job request.
ReportJobParameterSets	The Set of values given to parameters for a run of that report.
Report Output Register	Each run Of a Report Job generates Report output occurrence.

Application Control Database Physical ER Diagram



- L1 - Each Report Process may or may not have one or more Parameters associated with it. The link is via Processid and is many to many as both tables have many rows per ProcessId.
- L2 - Each Process Parameter may or may not have a defined set of possible values.
- L3 - Each Report Process is requested one or more times.
- L4 - Each Report Request can be run one or more times creating one or more Report Outputs.
- L5 - Each report Request has an associated Parameter Set.
- L6 - This Parameter Set identifies a set of Parameter Values. These are the values that parameters take for a Report Job request.

Application Architecture Diagram



Report Request Interface

This is the user interface for requesting reports and is currently available only as a SAS AF application. Report processes are selected within the Report Department and Report Subject Area under which they are registered. The appropriate parameters for that report are then selected from the ProcessParameters Table and presented to the user for edit. When edited the registered Edit method for the parameter is run. This could be a Date entry, Drop down List, Free Text Entry or a Bespoke Frame for that parameter.

Once the parameters have been defined the report can be run immediately or scheduled for regular run. The ReportJobsRequest table is updated and if run immediately a call is generated to the Report Control MACRO defining the Report Jobid to be run. The Report Control MACRO then picks up the Report Process and Parameter information from the ReportJobsRequest Table.

Users can also subscribe to Scheduled reports so that they receive e-mail notification when the report has been run.

Report Query

This is a BASE SAS program accepting parameters defined in the ReportJobsRequest table.

Result Sets

Output is set of SAS Tables or HTML/PDF Files named with a unique id linking them together as one set. This unique id is recorded on the ReportOutputRegister Table.

Formatting Function.

When the output is not Excel this function is incorporated within the Report Query. For Example PROC TABULATE Output is converted to HTML Using ODS. See detailed section below on Excel Formatting.

Report Scheduling

The Business Data of the Data Warehouse is refreshed daily overnight. Each Load is either a Daily, Weekly(Fridays) or Monthly(end of month) run. At the end of the data base build scheduled reports are run. The logic for running a report is based on 2 variables.

Run Frequency - 3 Byte Field indicating if the report is run on Daily/Weekly/Monthly Runs.
Run Occasion - For special cases such as '4th business day of month'.

Each Report Output is recorded with Unique Id and Date in the ReportOutputRegister.

Report Distribution

2 methods of distribution are used.

- **Company Intranet Site** - All newly generated Excel Reports are Formatted into Excel and uploaded to the server. HTML Menus are generated based on the Application Control Data. These Excel Files, HTML Menus along with the HTML Reports and PDF Reports are then sent from the Unix Server via ftp to populate the intranet site. As the whole menu system is data generated there is no such Site Maintenance required.

- **Email** - Where users subscribe to a certain report they are informed by email when this report is ready. This sends the appropriate url on the Intranet Site. This use the FILENAME EMAIL technique to send the email from the Unix Server.

Excel Formatting

In my recent experience 90% of Static Report output has been required in Excel Format. For that reason I will restrict this section to the use of SCL OLE Automation to transform SAS data set format Result Sets into Excel reports. The actual formatting process takes place necessarily on a Windows PC and the PC is the client in our current architecture. However, once created the Excel Files can be uploaded in binary format to the Unix Server from where they can be distributed to users without SAS installed on the desktop.

Transforming Excel VBA Into SAS SCL

SCL OLE Excel Automation code is very closely related to Excel VBA Code which is the code generated when MACROS are recorded in Excel. When writing the SCL to perform a certain Excel task it is very common to record that task in excel and then write the 'equivalent' SCL code. This is not always as straight forward as it may seem and there is a definite learning curve involved. Once the developer becomes more familiar with the translation process the Excel VBA documentation becomes more and more useful. An especially useful tool is the Object Browser in Excel VBA. This is a good way of understanding the Excel Object Model and finding the syntax of appropriate methods.

SAS Methods For The OLE Automation Class

The art of translating Excel VBA into SAS SCL lies largely in knowing which SAS Method to choose. The examples in the table below show the main SAS methods available and how they combine to perform a small amount of VBA code. The following VBA Code freezes the panes on the active window such that column headers always appear at the top of the window when scrolling down.

```
Range("A2").Select
ActiveWindow.FreezePanes = True
```

The following table shows how this would be coded in SAS SCL.

Method	Use	Examples
'_new'	To Start An Excel Automation	hostcl = loadclass('sasHELP.fsp.hauto'); call send(hostcl, '_NEW_', excelobj, 0, 'Excel.Application');
'_compute'	Generate a SAS Object Id (prange) for an Excel Object Attribute which needs arguments to define it. This is very much my definition of the method's use and not the true definition.	call send(excelobj, '_COMPUTE', 'Range', 'A2', prange);
'_do'	Perform A Method on an object giving no return value	call send(prange, '_DO', 'Select');
'_getproperty'	Get an Attribute for an Excel Object. This could represent another Excel Object or something like an Option Setting.	call send(excelobj, '_GETPROPERTY', 'ActiveWindow', pawind);
'_Setproperty'	Set an Attribute Value for an Excel Object	call send(pawind, '_SET_PROPERTY_', 'FreezePanes', ' True');
'_getreference id'	Get an Excel Object Id value which you then need to pass as an argument to a method.	

The AF Application to perform the Excel formatting function in fact uses a variety of SAS techniques including SCL OLE Automation. These will be described by examples of the type of Formatting possible.

Simple Data Export Without Special Excel Formatting

This is the function of exporting the data from SAS to Excel. In this example it is done via an Export to csv then the automation of Excel opening the csv file and pasting the contents into a selected worksheet. The EXPTOCSV MACRO is user written with functionality to ensure correct formatting of output variables and the use of Column labels. One reason for saving to csv then importing is the slower performance of sending cells of data individually via more direct OLE Methods. I have not come across a way of sending large arrays of data in one OLE method call as the arguments to such VBA methods are Array Variants which as far as I know cannot be sent from SAS. One popular alternative is to use DDE to send the data and I would not try to argue against it.

The code below is an SCL entry which can be run to demonstrate the methods. To test it save as OLETEST1.SCL in any SAS catalog.

```
init:

csvload='sashelp.class';
tfile='c:\temp\class.csv';
outfile='c:\temp\oletestout.xls';
template='c:\temp\oletest1.xls';

call method('OLETEST1.SCL','dataload',csvload,tfile);
call method('OLETEST1.SCL','olestart',excelobj,template);
call method('OLETEST1.SCL','olepaste',excelobj,template,1,tfile,'NEWSHEET','');
call method('OLETEST1.SCL','oleclose',excelobj,outfile);

return;

DATALOAD:
method dbload $17 optional=tempfile $200;
submit continue;

%EXPTOCSV(sasds=&dbload,ofile=&tempfile,colhead=name);

endsubmit;

endmethod;
Return;

* Presume template already exists**;

OLESTART:
method excelobj 8 template $200 ;
hostcl = loadclass('sashelp.fsp.hauto');
call send(hostcl,'_NEW_',excelobj, 0,'Excel.Application');
call send(excelobj,'_SET_PROPERTY_', 'Visible', 'True');
call send(excelobj,'_GET_PROPERTY_', 'Workbooks',wbsobj);
call send(wbsobj,'_DO_', 'Open', template);
call send(wbsobj,'_TERM_');
endmethod;

return;
```

OLEPASTE:

```
method excelobj 8 template $200 i 8 tempfile $200 optional=
    sheetnam $30 rangepos $10;

    * BY Default send data to position A1 in Excel **;
    * In this example only A1 will work **;
    if rangepos='' then rangepos='A1';

    ** Tempfile is csv file name, template is Excel file where data is pasted **;
    winname=reverse(scan(reverse(tempfile),1,'\'));
    outwin=reverse(scan(reverse(template),1,'\'));

    * Open the csv File **;
    call send(excelobj,'_GET_PROPERTY_', 'Workbooks',wbsobj);
    call send(wbsobj,'_DO_', 'Open',tempfile);

    * Get the active work sheet**;
    call send(excelobj,'_GET_PROPERTY_', 'ActiveSheet',wsobj);
    * Select Cell Area in csv File *;
    call send(wsobj,'_GET_PROPERTY_', 'Cells',pCells);
    call send(pCells,'_DO_', 'Select');
    call send(excelobj,'_GET_PROPERTY_', 'Selection',pselect);
    call send(pselect,'_DO_', 'Copy');

    * Activate the required output workbook **;
    call send(excelobj,'_COMPUTE_', 'Windows',outwin,pwinbk1);
    call send(pwinbk1,'_DO_', 'Activate');

    * Add The Required OUTPUT Sheet Name **;
    call send(excelobj,'_GET_PROPERTY_', 'Sheets',psheets);
    call send(psheets,'_DO_', 'Add');
    call send(excelobj,'_COMPUTE_', 'Sheets', 'Sheet' !! left(put(i,2.)),psheet);
    call send(psheet,'_DO_', 'Select');
    if sheetnam ne '' then call send(psheet,'_SET_PROPERTY_', 'Name',sheetnam);
    call send(psheet,'_GET_PROPERTY_', 'Name',sheetnam);

    ** Paste The Data **;
    call send(excelobj,'_COMPUTE_', 'Range',rangepos,prange);
    call send(prange,'_DO_', 'Select');
    call send(excelobj,'_GET_PROPERTY_', 'ActiveSheet',wsobj);
    call send(wsobj,'_DO_', 'Paste');
    call send(excelobj,'_SET_PROPERTY_', 'CutCopyMode', 'False');

    ** Close The csv file **;
    call send(excelobj,'_COMPUTE_', 'Windows',winname,pwinusrs);
    call send(pwinusrs,'_DO_', 'Activate');
    call send(excelobj,'_GET_PROPERTY_', 'ActiveWorkbook',pawkbk);
    call send(pawkbk,'_DO_', 'Close');

    ** Terminate the Automation Objects **;
    call send(psheets,'_TERM_');
    call send(psheet,'_TERM_');
    call send(pawkbk,'_TERM_');
    call send(prange,'_TERM_');
    call send(pwinbk1,'_TERM_');
    call send(pwinusrs,'_TERM_');
    call send(wsobj,'_TERM_');

endmethod;
return;
```

```

OLECLOSE:
    method excelobj 8 outfile $200;
    call send(excelobj, '_GET_PROPERTY_', 'ActiveWorkbook', pawkbook);
    call send(excelobj, '_COMPUTE_', 'Range', 'A2', prange);
    call send(prange, '_DO_', 'Select');

    * if outfile exists then delete to avoid Save As overwrite warning *;
    if fileexist(outfile) then do;
        delfile=outfile;
        link delfile;
    end;

    call send(pawkbook, '_DO_', 'SaveAs', outfile);

    * Clear up;
    call send(prange, '_TERM_ ');
    call send(pawkbook, '_TERM_ ');
    call send(excelobj, '_TERM_ ');
endmethod;

return;

delfile:
fname=' ';
rc=filename(fname, delfile);
if (rc=0) and (fexist(fname)) then rc=fdelete(fname);
rc=filename(fname, '');
return;

```

Introducing Excel Formatting

As can be seen coding can be quite laborious and it is important to develop a set of methods which can be called to perform typical formatting requirements. I list just 2 examples, again because of the volume of code involved.

Autofit Columns And Wrap the Text on The Column Headings

```

AUTOFIT:
    method excelobj 8 ;
    * Autofit the columns for visual effect;
    call send(excelobj, '_GET_PROPERTY_', 'ActiveSheet', wsobj);
    call send(wsobj, '_GET_PROPERTY_', 'Columns', pCols);
    call send(pCols, '_DO_', 'Autofit');

    call send(excelobj, '_COMPUTE_', 'Rows', '1:1', prows);
    call send(prows, '_DO_', 'Select');
    call send(excelobj, '_GET_PROPERTY_', 'Selection', pselect);
    call send(pselect, '_SET_PROPERTY_', 'Wraptext', 'True');

    call send(pselect, '_TERM_ ');
    call send(pCols, '_TERM_ ');
    call send(wsobj, '_TERM_ ');
    call send(prows, '_TERM_ ');
endmethod;
return;

```

Grid A Range Of Data

* Grid the data , numbers 7,8,9,10,11,12 represent attributes such as *;
* xinsidevertical , xlEdgeLeft, xlEdgebottom etc *

GRID:

```
method excelobj 8 topleft $4 botright $4;
range=compress(topleft) !! ':' !! compress(botright);
call send(excelobj, '_COMPUTE_', 'Range', range, prange);
call send(prange, '_DO_', 'Select');
call send(excelobj, '_GET_PROPERTY_', 'Selection', pselect);

call send(pselect, '_COMPUTE_', 'Borders', 9, xledge);
call send(xledge, '_SET_PROPERTY_', 'Weight', 2);
call send(xledge, '_TERM_');

call send(pselect, '_COMPUTE_', 'Borders', 7, xledge);
call send(xledge, '_SET_PROPERTY_', 'Weight', 2);
call send(xledge, '_TERM_');

call send(pselect, '_COMPUTE_', 'Borders', 10, xledge);
call send(xledge, '_SET_PROPERTY_', 'Weight', 2);
call send(xledge, '_TERM_');

call send(pselect, '_COMPUTE_', 'Borders', 8, xledge);
call send(xledge, '_SET_PROPERTY_', 'Weight', 2);
call send(xledge, '_TERM_');

call send(pselect, '_COMPUTE_', 'Borders', 12, xledge);
call send(xledge, '_SET_PROPERTY_', 'Weight', 2);
call send(xledge, '_TERM_');

call send(pselect, '_COMPUTE_', 'Borders', 11, xledge);
call send(xledge, '_SET_PROPERTY_', 'Weight', 2);
call send(xledge, '_TERM_');

call send(pselect, '_TERM_');
call send(prange, '_TERM_');
endmethod;
```

return;

Using Row Attributes In The Result Set

By developing SCL Methods which perform conditional formatting on rows in the result sets based on values of variables in that result set then we can introduce standardised formatting without having to write a new SCL Excel Formatting routine for every report process. Three such variables are currently used.

Variable Name	Function
PRNTATTR - Print Attributes	Different Values Indicate Print Functionality LB - Line Break PB - Page Break
FORMATTR - Formatting Attributes	Different Values Indicate Formatting Functionality ST - Sub Total (Bold/Italic) GT - Grand Total (Bold with Underline + Overline) BD - Bold IT - Italic
WHSEET - Output Worksheet Name	Allows sub-setting of a result set into multiple worksheets on the same output Excel File. This is performed by multiple calls to the standard Exporting Methods with varying where clauses and target sheet names.

Using Template Target Files

This is where users can define the look of a required report by using test data in multiple sheets and developing tables and Charts that reference that data. Where regular complex reports are produced in this way enormous time savings are made on the previous method of users running their own data extracts to populate the spreadsheets and then distributing the reports to other staff.

The development of such reports is again quite an art with many different ways available to achieve a similar result. There have been 2 particularly useful techniques I have come across both using named ranges. The code below is the method developed for naming ranges. Calls to this method pass the Range Name (rangename) and the Cell Area Definition (rangearea) for the Named Range (rangearea).

```
NAMERNGE:
    method  excelobj 8 rangename $20 rangearea $50;

/*****
VBA Example
ActiveWorkbook.Names.Add Name:="Sales", RefersToR1C1:="=Sheet1!R2C4:R30C4"
*****/

    call send(excelobj, '_GET_PROPERTY_', 'ActiveWorkbook', actwbk);
    call send(actwbk, '_GET_PROPERTY_', 'Names', names);

    call send(names, '_DO_', 'Add', rangename, rangearea);

    call send(actwbk, '_TERM_');
    call send(names, '_TERM_');

endmethod;
return;
```

Excel Named Ranges For Tabulations

Each column of data exported to Excel can be given a unique range name based on the range of cells it covers. This column and row information needed to define the range is of course known in the SAS Application as it can be derived from the variable number and count of observations in the relevant SAS Data Set. In this way the different sheets of data exported can be treated as tables in the database with columns in the tables represented by named ranges. By then using conditional sum formulae in Excel tabulations with sophisticated formatting can dynamically work off the exported data sheets.

The below formula queries returns a result equivalent to the SQL of

```
Select sum(sales)
From table
Where year='2000' and month='12'

={SUM(IF(MONTH1='12',IF(YEAR1='2000',SALES1,0),0))}
```

In order to identify the worksheet when the same variable is in 2 worksheets I have used a sequential suffix on the names for the named ranges. This represents the worksheet number.

NB: when entering a conditional sum formula in Excel you must press CNTL/Shift/Enter instead of Enter.

Named Ranges For Charts

If you need a data series of a Chart to be the last 15 rows of data from a column in a worksheet this is a varying range formula (as number of rows changes) and cannot be hard coded in the chart definition. However, if you enter a named range as the series and change the definition of the range programmatically from SAS then the series will change as the number of rows changes.

Printing

2 Techniques are generally used to give Printable output

- Page Break Automation And 'Fit To Page' Setting - These are options that allow multiple page tabular output to print correctly. Once a report structure is well known then hard coded % zoom settings can be applied to ensure a good output.
- Print Area Definition - By defining a Print Area on a Template file very sophisticated print output can be generated.

Running Excel VBA Macros

In some cases it is easier to record an Excel Macro and to run that macro from SAS SCL. I maintain an Excel file on the Unix Server that contains a set of Excel Macros. If I need to run one the file is downloaded from the server, opened and the required Excel Macro Run. The below code runs a Macro called Pagesetup1 for setting specific print Attributes.

```
send(excelobj, '_DO_', 'Run', "'C:\FDOBJPC\XLMACROS.xls'!Pagesetup1");
```

Alternatively Excel Macros can be saved into template files then run after data has been exported to the template file.

What's Next ?

Separation of the components of the application make it a good candidate for replacing individual components with different technology. The next challenge is to provide a thin client interface for requesting reports to be run and an Excel Formatting component which is not on the PC.

The Report Request Interface is largely data driven already so any thin client application which can read the application control data , present appropriate parameter entry objects, and call SAS processes on the Unix would suffice. So far at Fidelity we have developed a JAVA SWING application which interfaces with SAS via SAS/IntrNet connect classes, and a browser based ASP application for querying SAS Metadata documenting the Business Database contents.

One solution to the Excel formatting would be to have an NT machine running a SAS spawner and the Unix server sending result sets to be formatted into Excel , returned in binary and presented to the users via a Browser.

Acknowledgements

I would like to thank Allan Reading of ARC Consulting Ltd for his initial development of the Intranet Distribution Component and pre ODS HTML Rendering work. Other important contributors to the Application include Phil Cumberland and John Stimpson of Fidelity Investments Ltd.

Contact email: Simon.Pickles@uk.fid-inl.com