

Rich, Handsome & Successful: The Power of SAS® to the Scientists & Statisticians in Research

David Shannon, Amadeus Software, UK

ABSTRACT

This paper presents how a successful SAS/AF® application used by scientists and researchers was presented to users through a Windows application for usability and speed whilst retaining the robustness of the SAS® 9.

SAS Integration Technologies was used to interface the tried and trusted algorithms to a bespoke application developed using Microsoft Visual Basic .NET. This created an interoperable and scalable environment for the current and future of the business.

A discussion and presentation is made of the techniques used to integrate SAS via the integrated object model (IOM) of Integration Technologies, through the use of workspaces, stored processes and access to SAS metadata and data via OLEDB.

A presentation will be given of the final application demonstrating the ability to harness the power of SAS/Stat and SAS/Graph to users who are focussed on the rapid delivery of answers within Unilever's Research and Development Division.

INTRODUCTION

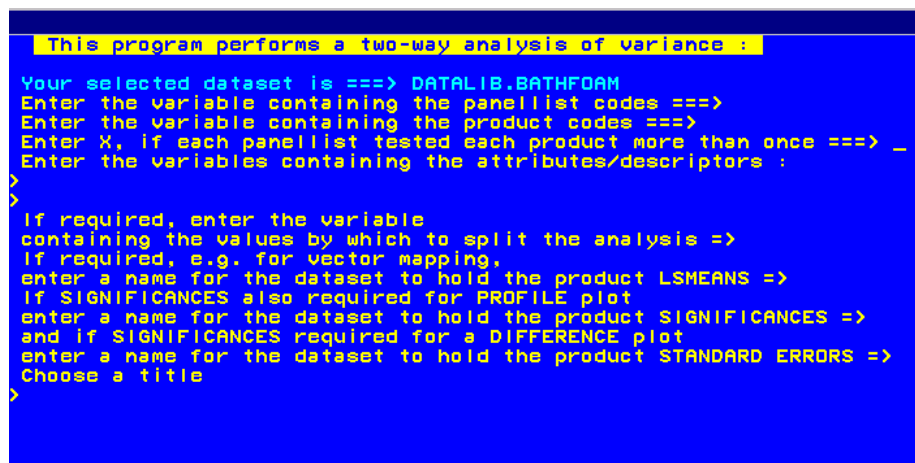
Unilever's consumer researchers analyse various aspects of consumer research data for household brands such as Dove, Lynx and many more using SAS/AF applications allowing the capture of results, statistical analysis and visualisation of these data for interpretation and reporting.

THE PROBLEM

After 20 years use a solution was sought that enabled Unilever's consumer research and development to remain competitive and keep abreast of the latest data capture and analysis methods.

A task based SAS/AF application Research Guidance System (RGSYS) was the first application to be redeveloped.

By modern standards the interface was unintuitive and slow to use. Figure 1 shows a SAS/FSP screen which was a typical method of retrieving parameters for the process from the user:

A screenshot of a SAS/FSP (File Specification Panel) screen. The background is blue with yellow text. At the top, a yellow banner reads "This program performs a two-way analysis of variance :". Below this, the text "Your selected dataset is ==> DATALIB.BATHFOAM" is displayed. A series of prompts follow: "Enter the variable containing the panellist codes ==>", "Enter the variable containing the product codes ==>", "Enter X, if each panellist tested each product more than once ==> _", and "Enter the variables containing the attributes/descriptors :". There are two right-angle brackets ">" on separate lines. Then, "If required, enter the variable containing the values by which to split the analysis =>" is shown, followed by "If required, e.g. for vector mapping, enter a name for the dataset to hold the product LSMEANS =>". Next, "If SIGNIFICANCES also required for PROFILE plot enter a name for the dataset to hold the product SIGNIFICANCES =>" is displayed, followed by "and if SIGNIFICANCES required for a DIFFERENCE plot enter a name for the dataset to hold the product STANDARD ERRORS =>". Finally, "Choose a title" is shown, followed by another right-angle bracket ">".

```
This program performs a two-way analysis of variance :
Your selected dataset is ==> DATALIB.BATHFOAM
Enter the variable containing the panellist codes ==>
Enter the variable containing the product codes ==>
Enter X, if each panellist tested each product more than once ==> _
Enter the variables containing the attributes/descriptors :
>
>
If required, enter the variable
containing the values by which to split the analysis =>
If required, e.g. for vector mapping,
enter a name for the dataset to hold the product LSMEANS =>
If SIGNIFICANCES also required for PROFILE plot
enter a name for the dataset to hold the product SIGNIFICANCES =>
and if SIGNIFICANCES required for a DIFFERENCE plot
enter a name for the dataset to hold the product STANDARD ERRORS =>
Choose a title
>
```

Figure 1: A typical user interface in the original application

There was no on-line help facility and the graphics were considered outdated relative to modern capabilities.

From an administrative perspective the system was also difficult to roll-out and maintain.

OBJECTIVES

The key objectives of the project were:

- Provide a handsome user interface similarly to Microsoft packages that users are familiar with;
- Build on the robustness of the behind the scenes code by extracting existing analysis algorithms from SAS SCL and convert into SAS Stored Processes;
- Develop the application in such a way as to support the future integration of additional functionality by adding to RGSYS with functionality from other SAS/AF applications without needing to redeploy the whole application;
- Deliver full on-line help facilities;
- Ensure the application is easily deployable and easy to upgrade across all Unilever sites internationally.

THE SOLUTION

Several possible technologies were considered. For reasons including existing skills sets, rapid development and source code ownership Visual Basic .NET was selected to develop the Windows interface and the user input validation.

An n-tier design was constructed that facilitated a plug-in based architecture, see Figure 2. Around 50 units of functionality in the original application, such as data handing tasks, panel monitoring, sensory analysis and repeated measures analysis became a plug-in in the new application.

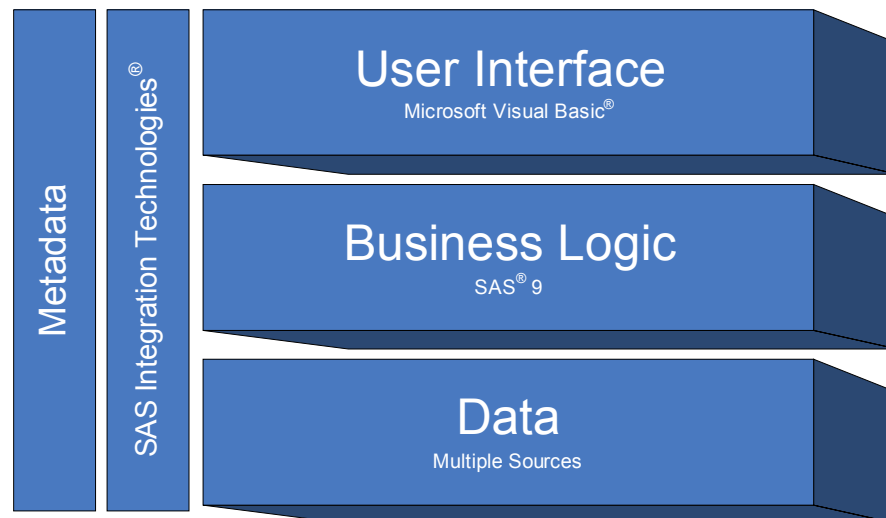


Figure 2: Three Tier Design

To support seamless integration and common use of the SAS Integrated Object Model (IOM) a framework was constructed that enables plug-ins to reside in assemblies (literally DLL files) that are dynamically loaded when the application is launched.

Such an approach enables one class to perform all the calls to SAS on behalf of all plug-ins implemented. The benefits to this approach provide consistency for users, scalability and manageability which overall leads to robustness.

PLUG-IN ARCHITECTURE

A base class was derived that acts like a template to all plug-ins created. The base class itself is literally a windows form that acts as a dialog form when called by the parent application.

All plug-ins are loaded via a technique called reflection. Reflection examines the attributes of classes such as its interfaces, methods, properties and events. Plug-ins are implemented with a specific function enabling the application to know whether each class is actually a plug-in belonging to RGSYS.

This technique also means it is very simple for the application to be scalable. Adding the new plug-in simply has to be located in the appropriate directory for the main application to perform reflection on the DLL at start-up; thus loading any plug-ins within.

Figure 3 below, shows a high level view of the relationship between the physical assemblies and presents how the interface to the main framework of the application is accessed by a plug-in.

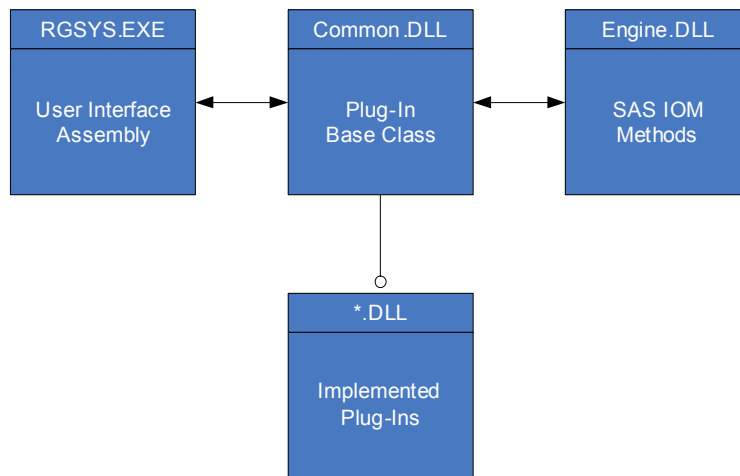


Figure 3: High Level Class Design

The top row of assemblies collectively makes up the RGSYS framework.

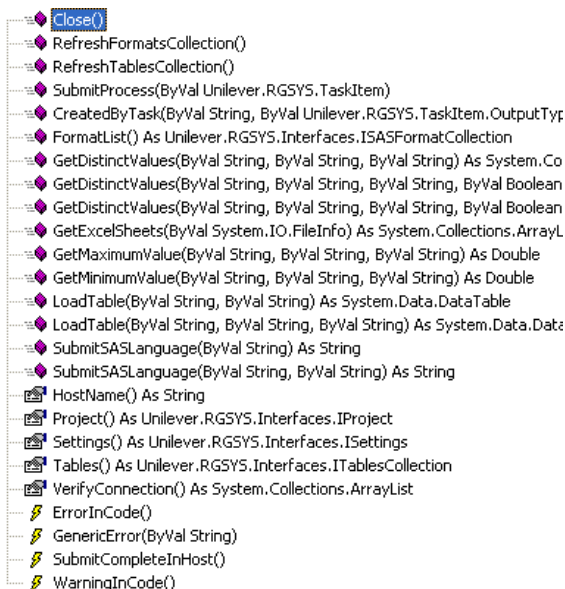


Figure 4: Visual Studio Class View of Engine Interface

In practice a single DLL contains several individual plug-ins for organisation and efficiency.

Developers of plug-ins are exposed to only the methods, properties and events needed to implement plug-ins. The methods, properties and events providing SAS functionality to the plug-in is shown in Figure 4 (left).

The RGSYS framework needs to know certain information about each plug-in. This serves to inform the framework that this is a genuine plug-in and that provide the information about the plug-in used to display it to the user through the interface.

An example of this function is shown in Figure 5 below:

```
Protected Overrides Function GetTaskInfo() As PluginInfo
    With GetTaskInfo
        .Category = "Reports"
        .Label = "Listing Report"
        .Provider = Provider.SAS
        .Routine = "r_listing"
        .Name = "Listing"
        .Category = "Reporting"
        .Version = New Version(1, 0)
        .ApplicationName = "RGSYS"
    End With
End Function
```

Figure 5: Implementing the Plug-In Signature

Before exploring the technical work of integrating SAS with .NET, at this stage it is appropriate to share the additional activity that helped make the project a success.

PLANNING

With such a large scale application development project, planning and design were essential in managing the development lifecycle.

A solution design document detailed the overall structure of the application detailing what functionality would be implemented, with details of constraints, assumptions and an overall design. Its benefits were two fold. Firstly allowing the customer and developers to see clearly what is to be delivered and secondly enabling software testers to develop a test plan whilst development was taking place.

Further low level designs were authored for each of the 48 plug-ins created, again providing benefit to Unilever, the developers and the testers.

SOFTWARE TESTING

Testing was performed by team members who were independent of anyone programming code.

The lead tester took responsibility for designing test cases and test scripts ensuring not only that the application was robust and bug free, but also that all intended features functioned as per design.

This approach achieved an unprecedented level of quality at the point of delivery.

USING SAS INTEGRATION TECHNOLOGIES

INTEGRATED OBJECT MODEL (IOM)

The real strength of the application is that SAS is used to perform all the data manipulation, analysis and reporting. The functionality of SAS is driven through the IOM. This is a set of interfaces that enable a .NET (via COM) or Java programmer to treat a SAS session as an object.

The IOM object model contains a series methods, properties and events that expose the functionality of a SAS session. The interfaces employed by RGSYS are shown below.

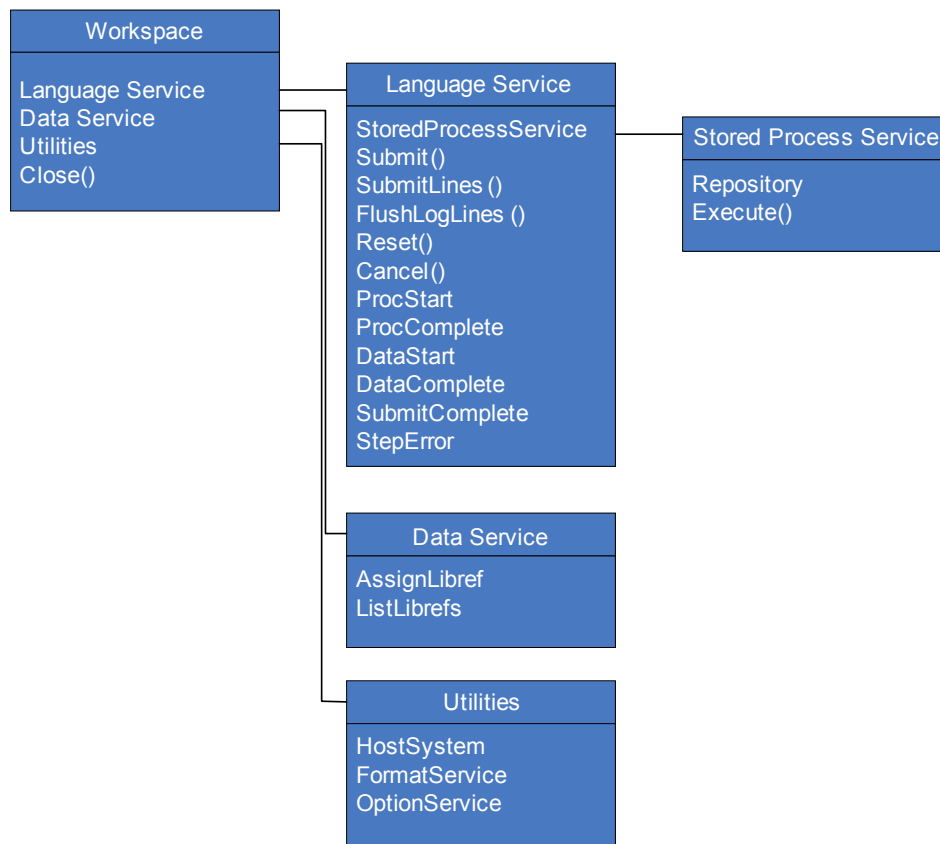


Figure 6: IOM Interfaces used by RGSYS

One single class was implemented that allows all plug-ins to run its code, access data, parse log files or any other functionality that SAS is used for.

STORED PROCESSES

What are traditionally SAS programs or SCL entries were re-written to be SAS Stored Processes.

A stored process is simply a program with an in-line comment of `PROCESSBODY`. When the SAS session encounters this comment it replace the comment with one `%LET` statement for each parameter passed into the macro. Otherwise the code runs as any other batch SAS session.

In addition to simply taking the original SCL code, enhancements were made to take advantage of SAS developments made over the last few years, including the output delivery system, new procedure options and functions.

Figure 7 shows a section from one of the more simple stored processes:

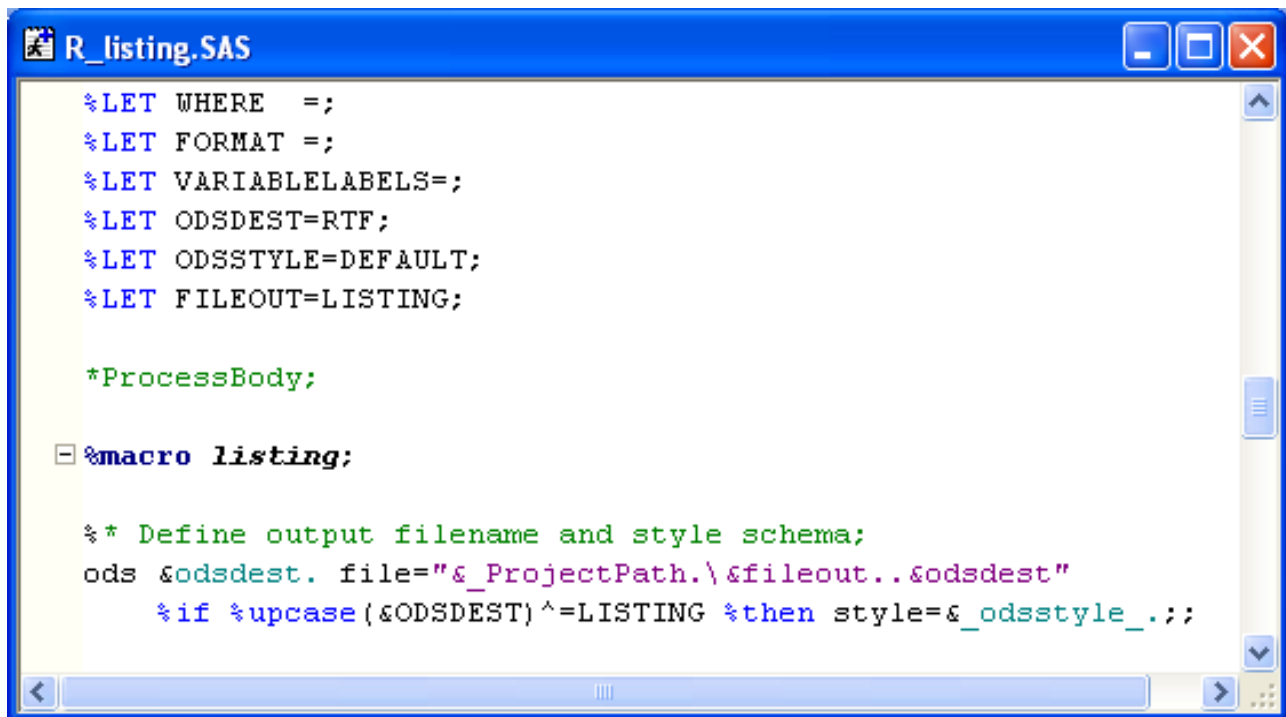


Figure 7: A Simple SAS Stored Process

Notice the PROCESSBODY comment. Above this line any macro variables that are used in the code are declared as either blank or to a default value.

As all stored processes run in the same SAS session all macro variables exist in the same global symbol table. It is important to ensure that any parameters used in the code are initialised correctly to avoid potential error by carrying over a macro variable with the same name from a previously executed stored process.

So how is a stored process executed and how are parameters passed from the Visual Basic environment into the SAS session?

As per the IOM shown in Figure 6, a stored process object is created from the language service object. A repository is defined, which identifies the physical location where the stored processes are located. Stored processes are then executed via the Execute method, where name-value pairs are passed.

```

'Run the code
oSP.Execute(Routine, mNP.ToString)

```

Figure 8: Executing a Stored Process

Where:

oSP is the stored process object;

Routine is a string object containing the name of the program without the .SAS file extension;

mNP is a StringBuilder object, containing name-value pairs such as "PARAM1=VALUE1 PARAM2=VALUE2" etc.

Therefore PROCESSBODY is replaced with:

```

%LET PARAM1=VALUE1;
%LET PARAM2=VALUE2;

```

This works both *successfully* and robustly.

The single area for attention is that of macro quoting functions. If the values of macro variables are special macro symbols syntax errors can result in the stored process. To circumvent this possibility RGSYS scans the contents of each value and places the appropriate quoting function around the value.

CAPTURING THE SAS LOG

Key to validating the any SAS program is its log. This is no different with the tasks of RGSYS. Furthermore the log is automatically parsed to ensure that no inappropriate messages were generated.

The COM events raised by the IOM language service enable the class to capture whenever any of the following events occur:

- Proc step begins
- Proc step ends
- Data step begins
- Data step ends
- An error occurred in the code
- The code submitted has completed

In each of the above events a method is called to flush the SAS log from the language service. A much simplified version of the actual code used to do this is shown below:

```
'Declare an array with 20 elements to contain the current 'flush'
'of the log
Dim LogLines(20) As String
'Spin until the flush has fewer than 20 lines (i.e. no more log available)
Do
    obLS.FlushLogLines(20, cc, lt, LogLines)
    For Each s As String In LogLines
        If s.StartsWith("ERROR") Then
            RaiseEvent ErrorInCode()
        End If
        LogFile.Append(s) 'Append the log to the string builder
        LogFile.Append(vbCrLf) 'Append a carriage return after each line
    Next
    'Check to see if there are more log lines available
Loop Until LogLines.Length < 20
```

Figure 9: Capturing the Log

Here an array (LogLines) is declared that holds 20 string elements. The FLUSHLOGLINES method calls 20 lines at a time from the language service. If on a flush the array is populated with less than 20 elements there are no further lines left in the log.

Now the log is available in the array any possible logic can be applied to examine its contents. In Figure 9, above, the first word in each log line is examined, if it matches "ERROR" then an event is raised to the rest the system indicating which can be consumed by plug-ins or the application. Further checks take place but are removed in the sample above for simplicity.

To retain the log, the contents of the arrays are appended to a StringBuilder object (LogFile) which is serialised to disk and associated with the project and task. This enables the software to display a link to the file in the results window.

DATA ACCESS

A critical function of the application is to read data from many sources.

All SAS data sets available to the user through the application are interrogated for their metadata whenever a user selects a data set such that information beyond its name is required. Microsoft's ADO .NET GETTABLESCHEMA and GETCOLUMNSHEMA methods are used to retrieve the information through the IOM's OLEDB provider. This is essentially the metadata retrieved from PROC CONTENTS for example.

A critical function within several areas of the software is displaying data to the user. The IOM's OLEDB provider is utilised again, however support is required for presenting data in both its internal and formatted values. The ADO (rather than ADO .NET) classes can be used for accessing formatted.

When accessing any data source through ADO, the following key objects are required:

- Connection: This defines what OLEDB provider will be used to read the source, where the source physically resides amongst several other possible options;
- Record set: This object represents the collection of the attributes and the data retrieved by the query.

The IOM OLEDB provider has a custom property that must be set to enable the reading of data with any permanently associated formats applied. The property name belongs to the record set object and is called "SAS Formats". Its property should be set to "+_ALL_" to return all columns both in their internal and formatted values. Furthermore the steps should be performed in the following order:

1. Define the connection object and set its provider to be a SAS provider, e.g. `SAS.LocalProvider.1`. The connection can now be opened.
2. Define the record set object and associate the connection with the record set.
3. Set the SAS Format property to take the value "+_ALL_" (or specific column names to return only specific columns with both internal and formatted values). Figure 10, below, shows how the properties are looped over to determine which the "SAS Format" property is before setting its value.

```
'After associating the connections with the recordset (and before opening!)
'we can set the SAS custom OLE/DB properties...
For Each prop As ADODB.Property In obRecordset.Properties
    'Debug.WriteLine(prop.Name)
    If prop.Name = "SAS Formats" Then prop.Value = "+_ALL_"
Next
```

Figure 10: Setting the SAS Format property

4. Finally, the record set can be opened and looped over to return the data.

Note that for each column in the SAS data set, two columns are returned – firstly the internal and then the formatted columns. This was handled by creating two .NET data tables to store the data.

SAS/Access products are used to convert the raw data sources (bottom tier in Figure 2). These are converted into native SAS data sets before being used by the interface of the application.

OTHER DESIGN FEATURES

There are several features about the application which made it a *rich* and productive experience for the statistician's and researchers. These are paid for in this section.

PROJECT BASED WORKFLOW

RGSYS was designed to treat a plug-in as a template for performing a given task, e.g. ANOVA, data reporting etc. A user can add as many tasks into a project as they need, either of the same or different plug-ins.

Each task is stored in the project then all tasks are joined into a workflow for sequential execution. The workflow is presented graphically along side the list of available tasks and the current project's results as seen in Figure 11, below:

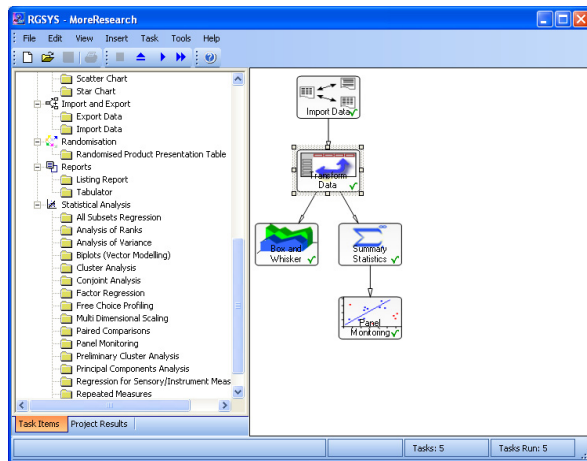


Figure 11: RGSYS User Interface with Task List

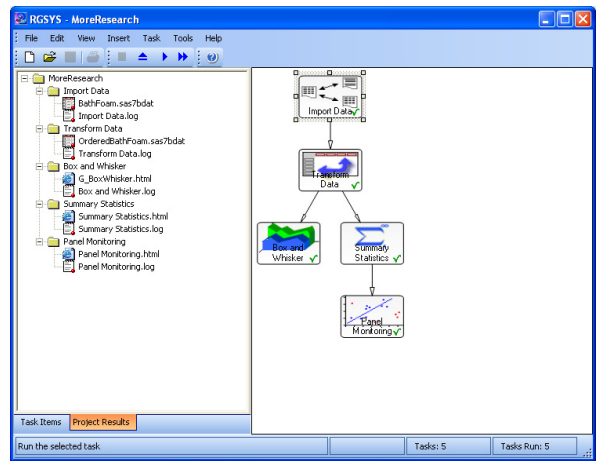


Figure 12: RGSYS User Interface with Project Results

For each task a log file and outputs generated a link is displayed in a tree as shown in Figure 12, above.

As discussed, there are several plug-ins available in the application. These were implemented with a great degree of consistency for ease of use. Figure 1 showed the ANOVA screen from the original RGSYS application.

The original application sometimes required several screens had to request all of the analyses options and associated diagnostic graphics. Figure 13 and Figure 14 show how this functionality is now presented on a single tabbed form:

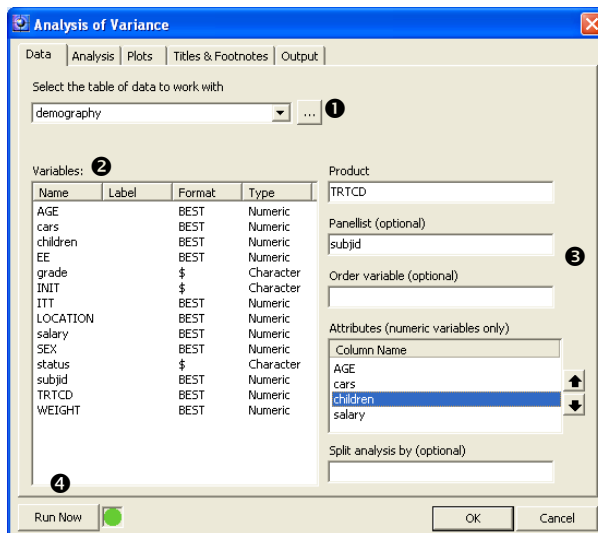


Figure 13: Data Selection in the ANOVA Task

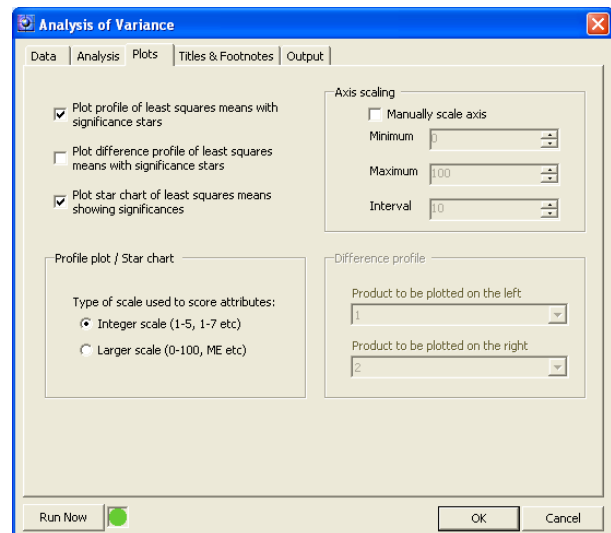


Figure 14: Requesting plots in the ANOVA task

Figure 13 shows a table of data to work with; (1) which exists inside the project libraries. The selected tables columns and associated metadata (2) are automatically displayed for the user to work with. Columns are then assigned to task roles (3). Noting that the OK button and also the Run Now (4) button do not become available until the minimum required roles have been assigned within the task. The traffic light next to the run now button indicates the status of the task, ready to run, more roles require defining and running.

Figure 14, shows how several options were incorporated into a single form that formally were spread between two frames.

LOCALIZATION

Localization is the task of presenting an application in the "locale" of the current users PC. This was considered in

the design and implementation of RGSYS, as there are many worldwide users.

Figure 15 shows a form running on a PC with the language set to Spanish. The translation of text is not something that is built in with Visual Basic or Visual Studio! A translator must be employed to create the resource file for the languages desired.

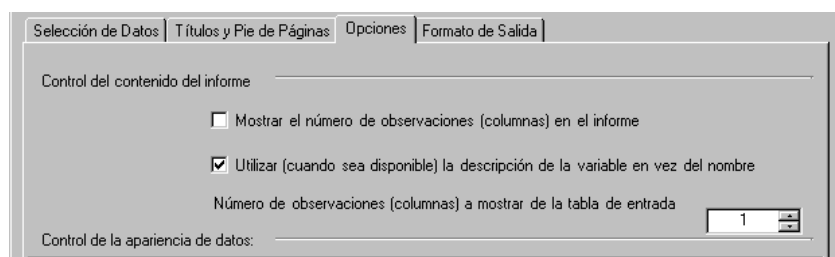


Figure 15: A Task Displayed in Spanish

It was decided at the design stage to make the functionality available to translate any give part of the application. To achieve this any strings displayed are stored in separate files called resource files. One resource file is created per language.

The software automatically determines what the appropriate resource file is for the current language and displays the strings accordingly.

DEPLOYMENT

An original requirement was to enable the ease of deployment not only of the initial roll-out, but also of the future functionality to be added to the project.

This was achieved through the implementation of an auto-updater which is aided by the very nature of Microsoft's .NET Framework.

Microsoft .NET assemblies (i.e. *.EXE and *.DLL's in the case of RGSYS) do not have to be registered onto a PC in the same way traditional ActiveX or COM assemblies do. This means that .NET applications can literally be copied and pasted into place.

To facilitate auto-updating a server location was defined that hosts the most up-to-date copy of the application (SAS Stored Processes included). When RGSYS is launched it checks various attributes about all the files in its installation against the server copy. If differences are determined such that newer or new files exist on the server they are downloaded onto the users PC where a fail safe installation is performed.

The deployment is touch free, requiring only that a developer or administrator deploys one copy of the latest files onto the server.

WHAT ABOUT SAS ENTERPRISE GUIDE?

A comment which might validly be raised is of the similarities to SAS Enterprise Guide amongst other Business Intelligence suits; therefore why go to the effort and expenditure of building a custom solution?

Firstly, whilst there are similarities to Enterprise Guide (these are genuinely coincidence!) there are also a number of fundamental differences in how the packages operate with respect to projects and data.

The simplest single reason for this approach is that when the project was originally discussed and designed in 2003, SAS Enterprise Guide did not support the level of extensibility that would be required to host Unilever's custom functionality.

Finally, Unilever desired to own and maintain the source code within the business enabling rapid and nimble response to user needs and requirements.

CONCLUSION

Unilever have shown that it is possible to present the complex tasks of experimental design, data capture and analysis in an intuitive user interface using the language and terminology familiar to the statistician's and scientists in their research divisions.

By leveraging the macros and analysis programs accumulated over several years and that have proved accurate and trustworthy the new interface provides the results that gain the trust of its users.

SAS Integration Technologies was the key to interoperability between the application and the power of SAS.

RGSYS now presents Rich functionality, a handsome user interface and has been a successful deployed and utilised application.

ACKNOWLEDGEMENTS

Kate Browning and Phil Helme of Unilever Research.
Andy Beggs of Amadeus.

CONTACT INFORMATION

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

David Shannon
Amadeus Software Ltd.
Orchard Farm
Witney Lane
Leafield
OX29 9PG
Work Phone: +44 (0)1993 878287
Email: david.shannon@amadeus.co.uk
Web: www.amadeus.co.uk

Copyright (©) 2001 – 2005 Amadeus Software Ltd.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.