Paper 43-26

Successful Web Enablement of a SAS® Product

Ian Sutton, Pioneer Software Limited, Wellington, New Zealand

ABSTRACT

This paper discuses the trials, tribulations and rewards of the conversion of our SAS OLAP Product (Futrix) to fully enable its functionality via the web.

Various topics are discussed including what web technologies/languages we found most appropriate and the relevant enhancements that SAS V8 brings. We discuss the research we carried out and the amount of work that was involved, how to maximize your existing skills and investment in development, and the problems we encountered and how we solved them or worked around them.

Also included is a demonstration of the final result: the web enabled Futrix product and what was achieved with approximately 1000 hours work.

BACKGROUND AND INTRODUCTION

We had a very extensive application, written entirely in SAS for use by Windows PC clients running SAS. Our clients told us that they also wanted this same functionality available via their web browsers, and hence our need to web enable the Futrix end user interface.

Futrix consists of approximately 140 screens, but fortunately only about 35 of these are used by the end user interface which is the portion we were web enabling. There were about 120,000 lines of code in total, of which about half related to the end user, but most of this was within class methods, and not actually associated directly with a particular screen.

We undertook extensive research before we started this process to ensure that we made the best decisions about how we go about it with the best result for the clients and also involving the least effort from our part with the minimum on-going maintenance.

THE NATURE OF THE "FUTRIX®" APPLICATION

The following is some more background to help you understand the nature of the application we were web enabling.

Futrix is a reporting and analysis application of the on-line analytical processing (OLAP) type. There is a main menu entry point to select from various reports, and then once a report is selected everything revolves around one central reporting window.

Futrix comprises of two major aspects:

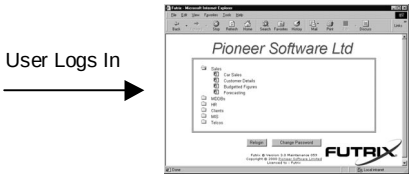
- the end user interface/environment
- the administration interface/environment

The end user interface is used by lots of users, however the administration interface is only used by a small number of administrators. For this reason, we decided to initially only web-enable the end user interface, as the administrators typically had SAS installed on their PCs anyway, and even if they didn't it was an easy task to set them up. Therefore this paper is essentially just the web-enabling of the end user aspects of the application.

The following two steps/diagrams show the nature of the way a user would typically traverse the screens within the application:

STEP 1: LOGIN AND SELECT REPORT

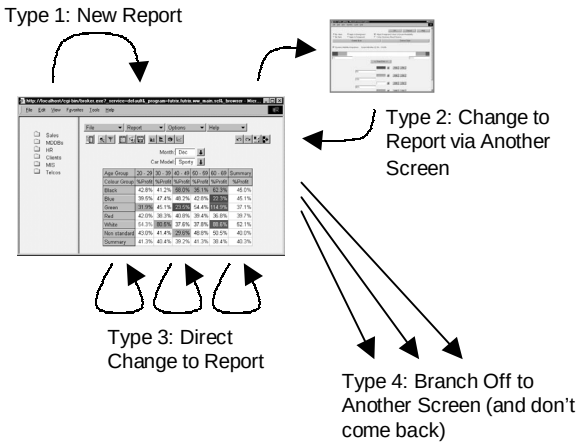
The first thing a user must do is to login, and then select a report.



STEP 2: REPORT INTERACTION

Then once viewing a report there are four types of user actions that will traverse the web pages generated by SAS:

- Type 1 – a new report selected from the side menu explorer
- Type 2 – change to report via an intermediate screen
- Type 3 – change to report directly via:
  - pull down menus
  - toolbars
  - popup menus
  - double-clicking
- Type 4 – branch off to another screen (and don't pass control back) for actions such as:
  - exporting reports
  - extracting detail level data
  - further statistical analysis
  - branching off to a site's custom applications



The fact that there was one central screen (the reporting window) made our process of web enabling the application a lot simpler.

Despite the fact that we needed to web enable the user interface, we also still needed to support the users who used Futrix via SAS on their PC. One site may have a mixture of users, some using a SAS client and some using a web client, and some users that may even use both. Therefore essentially they are using exactly the same system, but via a different delivery medium – for this reason we wanted to keep the interface layer as thin as possible so as much as possible was handled by the server, which was independent of the interface used to display results and return actions from the user. This design feature also made the task of web-enabling Futrix significantly easier.

PROJECT PLAN

Our project plan involved the following Steps 1 to 4. However, I have also included a "Step 0" as this is what we had done just prior to this project, and it proved to be very relevant to the success, speed and efficiency with which we were able to complete the project, t hough we did not fully appreciate just how significant it would be at the time we started this project.

STEP 0: DATA DRIVEN AND OBJECT ORIENTED DESIGN/IMPLEMENTATION

Prior to this entire web-enabling project, we had re-written Futrix entirely from the previous version (V2.01), to be almost entirely data driven and implemented in an object oriented fashion. This did slow down development significantly for about 6 to 9 months, as we were basically re-writing it from scratch. This process of taking several steps backwards was most certainly worth it so that we could run forwards.

The reason we decided to do this was initially purely to enable us to simplify maintenance and to allow us to very quickly be able to add new functionality, change the way things worked, and allow clients to be able to customize their environment much more so to their own needs and requirements. However, what we didn't fully realize at the time, and why I am mentioning this here, is that the whole process of web enabling Futrix was made "orders of magnitude" easier because this was done in advance. I cannot emphasis this point strongly enough, as it underlies everything in this paper. Our task was made so much easier because of the dynamic and object oriented nature of the application. Some examples of this are:

- a separate messaging module for all user messages (both storage of the message text and the delivery of them)
- separate and data driven user actions for what the user can do (pull down menus, toolbars, popup menus, double-clicking)
- separate actioning of these commands via methods and classes
- separate and layered approach to:
  - extracting data for queries
  - converting data from query into a report model
  - using a viewer to display that report model (table, graph, sas client, web client)

The following three concepts we continuously strived for, and from all the experience we have gained from this entire project, we estimate that if your application adheres fully to these then you are already more than 75% of the way to web enabling it without writing a single line of web enabled code. These are:

- Keeping Everything Generic
- Data Driven
- Object Oriented

This entire process was certainly non-trivial and involved about 1500 hours work to design, write and convert to fully utilize these concepts.

STEP 1: RESEARCH

The research that we undertook was designed for us to answer the following questions:

- What web languages/technology should we use
- What expectations should we realistically have for the way the final screens will look
- What expectations should we realistically have about how long this project will take
- What new skills will we need to learn and where can we get these skills from

This step involved about 90 hours work, however this was just the pure research, whereas the following "Step 2: Prototype" can also be considered a large part of the research process as we were still clarifying and refining the answers to the above questions throughout Step 2 as well.

STEP 2: PROTOTYPE

For a project of this size we always start with a worki ng/functional prototype, however in this case it proved to not only be desirable but absolutely vital.

For the prototype we selected a core set of functionality and screens which were primary to the system, and would in their own right make for a very useful environment. We settled on the following main areas of functionality for out prototype (which made up approximately 25% of the major areas of end user functionality):

- Main Menu Selection of Reports
- Drill Down
- Tabular and Graphical Reporting
- Report Layout Control
- Subsetting Filters Control
- Reach-through to Detail level Records

To get to this point took about 150 hours of work, which was actually considerably less than we expected.

In the process of doing this we learnt a lot and we then had some fine tuning, some maintenance and some trying out of new ways of doing things with this prototype. This additional work amounted to approximately 50 hours.

Therefore the total amount of work on the prototype was about 200 hours.

STEP 3: FULL IMPLEME NTATION (WITH SAS V6.12)

The previous stage of developing a working, functional prototype proved to be more valuable than we dreamed when we first started this project. This wasn't so much from the perspective of what we learnt how to do, but what we learnt **not** to do.

As any application developer knows, once you have finished a project, you always know much better "what not to do", and also how various things could have been done differently if you were to start the entire project again. We often end up throwing something away and starting again, as it then ends up simpler, more flexible and significantly easier to maintain. But this does require that one does not get "emotionally attached" to the work they have done previously.

This is exactly what happened wit h this project. We threw away the prototype entirely. We started from scratch again as we had learnt new ways of doing things which would benefit us significantly as the complexity and size of the application we were developing was a lot bigger than our prototype. The specific lessons we learnt are reasonably specific to our application and are really beyond the scope of this paper, so I won't go into them in detail here. They mainly related to things like how information, commands and user actions were communicated between SAS and the web browser.

This step involved approximately 700 hours work to get to a production level of the application with all of the functionality that we wanted in there.

STEP 4: CHANGES & ENHANCEMENTS (WITH SAS V8)

We are now embarking upon this last stage of the project. Unfortunately at the time of writing this paper (ie the written version), we had not completed this process, but we should have by the time this paper is presented orally (approximately three months later). Therefore the paper presented at the conference shall have much more detail about this aspect.

SAS Version 8 won't help us much with the functionality it allows us

to provide to the end users, as we have already achieved everything we wanted to with SAS Version 6.12. However SAS Version 8 does bring quite a few server side enhancements, which should help mainly with performance and simplified maintenance.

We estimate that we will spend approximately 200 hours researching this, then implementing various changes and enhancements to enable clients who are using SAS Version 8 as their sas web server, to take full advantage of the enhancements that SAS has made.

DIFFERENT LANGUAGES/TECHNOLOGIES

The very first question we needed to answer was: What SAS technologies are we going to use to implement this solution. It was fairly obvious that we would use SAS/IntrNet but even with that there were many options of what we could do. This decision process was not complex because the only real solution which offered what we required was to use the Application Dispatcher from within SAS/IntrNet. This is because we required:

- the SAS server to do most of the processing and provide finished results to the web browser
- the web browser to be able to communicate back to the SAS server

One of the most significant parts of our research was choosing the most appropriate web language to write the web portions of our application in. Like many situations with computing there are many ways to go about it. We initially looked at the following languages to assess the pros and cons of each:

- HTML
- DHTML (Dynamic HTML)
- JavaScript
- Java

We also looked at XML and ASP, but at this stage and for the type of application we were creating, it was soon obvious that the above would do just fine and with the ability to keep things simple.

We soon found that the line is often blurred between DHTML and JavaScript, and also that HTML was such a fundamental on the web that we would be using this for certain aspects anyway. The decision process ended up coming down to a choice between:

- A combination of HTML, DHTML and JavaScript; or
- Java

The conclusion to this decision was not that hard either, because of the following reasons:

- DHTML and JavaScript turned out to be extremely powerful and fast enough for our requirements
- Java required the downloading of the applets which some users were not keen on (compared to no additional download for the other option, other than plain text scripts)
- Some of our clients did not want any Java applets to be run on the web clients
- The learning curve required for Java was significantly higher
- More processing would need to be done on the client, taking away from our server side processing (for both SAS and web clients), and hence we would have had to duplicate code and logic that was already written and working, where DHTML and JavaScript could directly make use of this.

Hence we decided to go down the HTML, DHTML and JavaScript route, however on the downside there were also some points worth mentioning here:

- Java would provide a “slicker” user interface
- certain functions/user actions would be faster with Java
- there were some functions/user actions that would actually be a lot easier to implement with Java (despite most things being easier with DHTML and JavaScript)

WHICH WEB BROWSERS TO SUPPORT?

Initially we just assumed that we would support both of the two main web browsers:

- Internet Explorer; and
- Netscape Navigator

However the more we learnt about what was possible with Internet Explorer the more we realized that Netscape Navigator had a lot of catching up to do when it came to the power it provided to applications developers.

Even though the syntax varied slightly between the two browsers for many of the things we wanted to achieve, we were happy to write support for both (via a generic JavaScript layer to remove the specifics of one browser or the other). However it soon became obvious that by still supporting Netscape Navigator users we were holding back the Internet Explorer users, and not ending up with the best possible solution. For this reason we took the decision to only support Internet Explorer as the web browser.

We used Internet Explorer Version 4 (IE4) as the minimum requirement, however there are many enhancements between IE4 and IE5, and so we have a lot of specially written code for IE5 users to take advantage of many of these new features if the user’s browser can make use of these features.

One of the main features we wanted to be able to use was the ability to click the right hand mouse button and for the user to get a popup menu of items applicable to what they clicked on, or to be able to double-click with the left hand mouse button to perform the default action. This control to developers was only provided with IE5. Therefore we again had to write specific code to provide this for IE5 users and if they were using IE4, then to provide all the actions via a single left mouse button click.

OPTIMISING PERFORMANCE & SAS VERSION 8

We have been very pleasantly surprised with the performance of the end result we achieved using SAS6.12 as the SAS server behind the web server. It also works equally well with SAS V8 in this role, however we know that there is always room for improvement, and there are also quite a few enhancements that SAS/IntrNet in SAS V8 brings. At the time of writing this paper this is currently what we are working on with regards to:

- Network traffic
- Load balancing
- Utilizing shared memory
- Concurrent users and shared query cache

These topics will be covered in more detail in the paper presented at the conference (as we should have finished these enhancements in the intervening 3 months since writing this).

SKILLS

Our team of developers had very extensive skills in:

- SAS Application Development (SAS/AF)
- Client/Server applications
- Databases (and large volumes)
- Dealing with large numbers of concurrent users

However we had next to no real experience in:

- web browsers (other than the usual "surfing")
- SAS/IntrNet
- HTML, DHTML and JavaScript

To maximize the existing skills we had and to minimize the learning curve required to get to where we needed to be, we hired a top web developer who could help us with the web browser and web languages, though he knew nothing of SAS or SAS/IntrNet so it was up to us to be the "glue" between the server application and the web based screens.

The approach we took which worked exceedingly well, was that we described:

- what we wanted the web screens to do
- how we wanted these screens to look
- what we wanted to provide to them and what we wanted returned from them

He then whipped up these screens very easily using HTML, DHTML and JavaScript as we requested, documenting and explaining what he was doing as he went and thereby passing the knowledge onto us.

We then used these examples he had created, and made them generic for our purposes and incorporated them directly into Futrix on the server as embedded HTML, DHTML and JavaScript.

The basic concept of execution at run-time by the end user goes like this:

1) The Futrix SAS server runs its SAS application
2) Based on the current state and action received from the user's web browser a web page is generated which is purely a text script made up of HTML, DHTML and JavaScript
3) This entire page is then sent via the Application Dispatcher to the user's web browser
4) The user then actions something else which then sends a string of parameters back to SAS (and re-start again from step 1)

This all worked like a dream, with very little maintenance. A simple, yet powerful solution.

The web developer we hired was also a whiz with Java, and so we also gave him some similar exercises using Java, to ensure we had made the correct decision earlier (ie to not use it). These did take longer to develop to our requirements, and as we expected the final result looked slick and worked well, but we had a nightmare trying to integrate it into our server side processing (never totally succeeding actually), and so were pleased we had taken the route we had. Note: We were very impressed with what we saw though from a clients' perspective, and so will soon be canvassing some existing clients to assess whether they would like us to move more in that direction.

VISUAL COSMETICS

One other bonus of the web development process, and of getting in an expert, was to learn about all the other wonderful things that were possible. We learnt all about Cascading Style Sheets (CSS), and soon saw the power that these provided.

We ensured that every element we wrote to the web browser had its own CSS class. This then allowed us to totally change the cosmetics of the environment by simply altering the Cascading Style Sheet that the user was using.

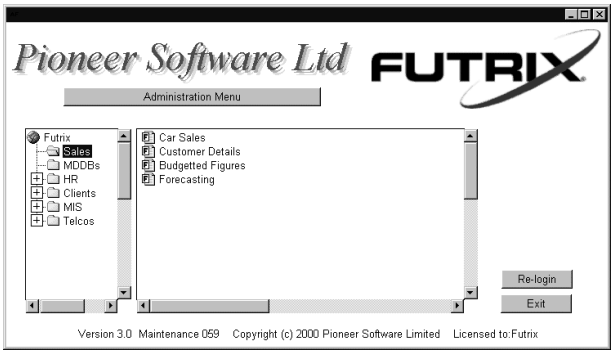
One down side of Cascading Style Sheets are that they are not hierarchical, and hence there is now inheritance from one class to another. This means, for example, that it is not possible to change the text font in a single place and then for every class that uses text to also change to use that font. For this reason we ended up creating a SAS application to create the Cascading Style Sheets for us, where we could control the hierarchical nature of all of our visual classes. This approach worked very well but was not trivial to implement.

FINAL RESULT

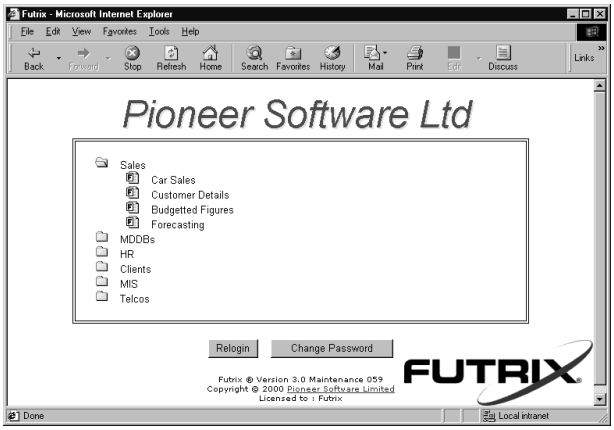
We ended up with almost all of the functionality that was available on the SAS client also available via the web client. There were a few more advanced features which we left out of the previous release so that we could get it out there into the real world as soon as possible and see how it performed and was received by clients. Results and feedback so far have been excellent. We are now adding the additional advanced functionality which we didn't have in the first web release, which will be made available in the upcoming release.

The following screen shots are some examples of how the SAS client windows compare to the web client windows:

MAIN MENU EXAMPLE



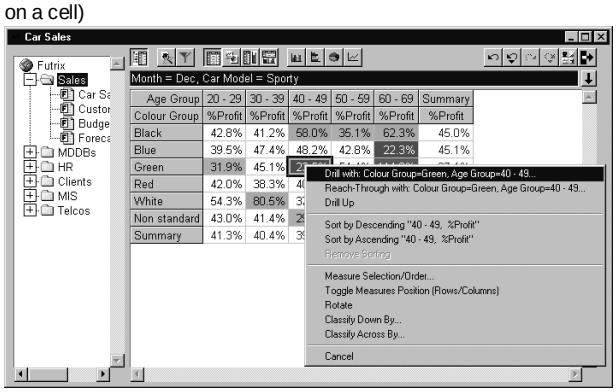
Example of SAS client window



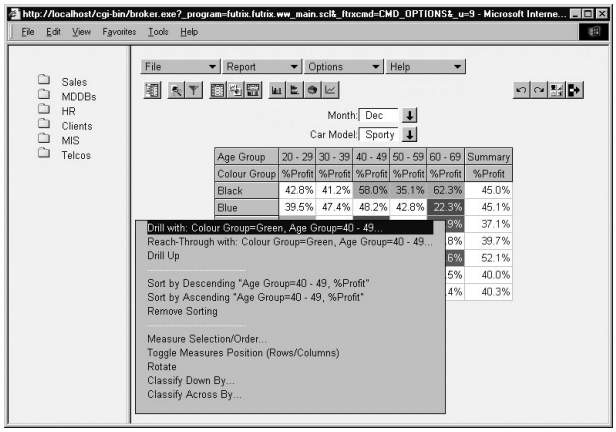
Example of web client window

MAIN REPORTING WINDOW EXAMPLE

(with the popup menu you get when clicking the right mouse button on a cell)

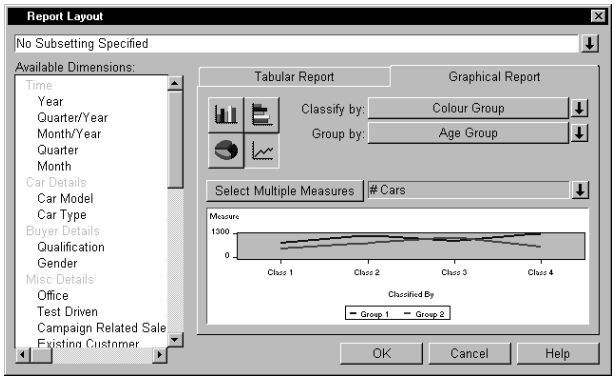


Example of SAS client window

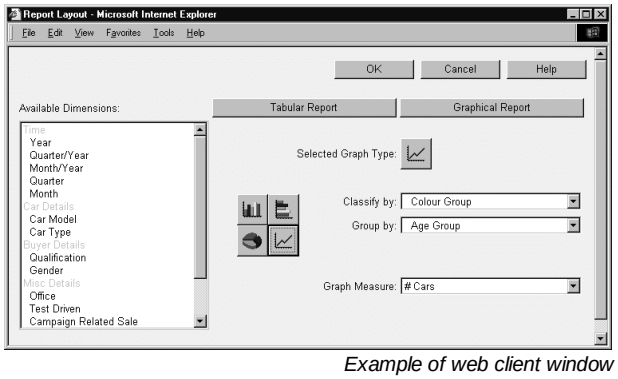


Example of web client window

REPORT LAYOUT EXAMPLE

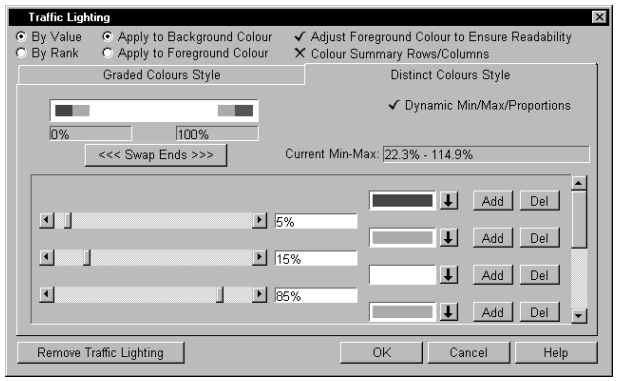


Example of SAS client window

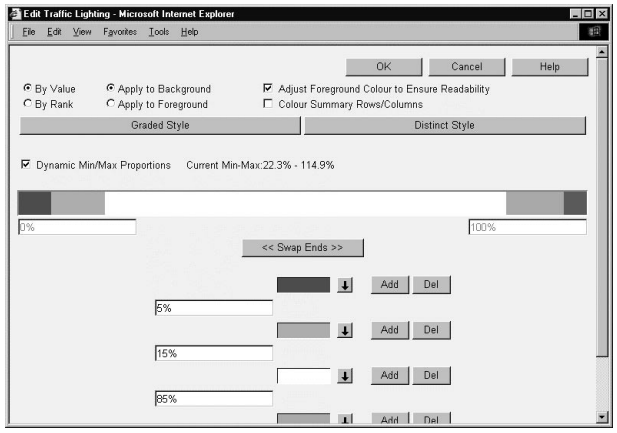


Example of web client window

ASSIGNING/EDITING TRAFFIC LIGHTING EXAMPLE



Example of SAS client window



Example of web client window

A TOTALLY DIFFERENT PARADIGM

There is an entirely different way to web enable your SAS applications, without having to write a single line of web specific code, or even make a single modification to your existing SAS application.

This is done by using a third party product such as Tarantella, which allows you to use pretty much any application via web browsers. I have run SAS via this method and also run the SAS Client version of our Futrix product via this method, and I was pretty amazed at the results. It was possible to do pretty much everything with zero additional development work, and nothing specific to the web. The performance was also very good (though this was just with a single concurrent user).

If you need to web enable an application which has already been written for SAS clients, then this method may well be worth investigating further. As a downside it does mean having to license a third party piece of software, and the other significant issue would be in the lack of load balancing if you have lots of users and multiple servers.

CONCLUSIONS

Once the web-enabled version of Futrix was completed and in production, we looked back and thought “that was a lot easier than we expected”. Though we could then see clearly with hind-sight that our task would have been very significantly more difficult if it hadn’t been for the following:

- the initial application was generic, data driven and object oriented before we started web enabling it
- the interface layer was very thin, with most of the work being done by the server which was independent of any particular

interface
- there weren't a massive number of screens that needed to be displayed to the user

We were also very pleased with some of the decisions we made along the way. Especially the following:
- to use HTML, DHTML and JavaScript, which were incredibly powerful and pretty easy to use
- to get in expert help/consulting from an experienced web developer (even though they had no knowledge of SAS at all)

We now just need to make more use of the specific enhancements within SAS Version 8, which we shall discuss further in the oral presentation of this paper at the SUGI conference.

BIOGRAPHY

After using SAS for 3 years for applications development, Ian joined the SAS Institute and worked for the SAS New Zealand office for 3 years, and then spent a year working in the SAS UK office. During this time he gained considerable experience in efficient database design and population, and also in applications development especially of the EIS/OLAP variety. Since leaving SAS about 4 year ago, Ian created the Futrix business intelligence product. He now does a mixture of product development/support for Futrix and general SAS consulting.

CONTACT INFORMATION

If you have any questions about any aspect of this paper or its contents then please feel free to contact me directly:

Author:	Ian Sutton
Company:	Pioneer Software Limited
Web:	www.futrix.com
Email:	ian@futrix.com