

## **A pioneer APL+WebServices Application**

Pierre Windal ([pierre@windal-conseil.com](mailto:pierre@windal-conseil.com))  
Eric Lescasse ([eric@lescasse.com](mailto:eric@lescasse.com))

### **Introduction**

A year ago we sat through Fred Waid and Brian Chizever's training sessions to learn the basics of APL+WebServices. One of my clients wanted his application ported to the web and Eric was confident that it could be done with the new APL 2000's web services. The application looked straightforward enough and the APL+Webservices, like good wines, were getting better year after year. Fred even said it could be done overnight and I still remember him and Jairo working in Eric's hotel room at one o'clock am to set it up as an example for the next training session. Well, it took us a few more nights to iron the wrinkles, but it eventually worked. This is the story of this porting that we would like to share with you.

### **The Renault Application**

#### **Context**

According to a June 2004 press release, the French car manufacturer Renault said it had set up a world training strategy aimed at providing every single employee, wherever he works and whatever his age or function, the formal training he/she needs to perform his/her job. In 2003, for instance, three employees out of four had followed at least one training session, and the objective was to train everybody in a single year.

The quality of the training as perceived by the trainees is assessed via a short structured questionnaire filled out at the end of the training session. This on-the-spot assessment, referred to as "hot evaluation" for short-term satisfaction evaluation, generates a few thousand questionnaires per year which are transmitted to Quadrature for tabulation and reporting. The results of these questionnaires are then fed to the trainers for corrective actions.

In some cases, the "hot" evaluation is followed several months later by a "cold" evaluation to find out how the initial satisfaction level ages over time – long-term satisfaction evaluation. The questionnaires used for these later evaluations may vary from one session to the other and have little in common with the standard hot evaluation questionnaire.

#### **Product overview**

Fancy a one-page satisfaction questionnaire in two sections. The first one tags the training session with six bits of information: date, location, unit in charge, and three hierarchical codes for the type of training dispensed, referred to as "module", "maille (mesh)" and "session". A session is a particular instance of some training described by a maille and a module. Modules are grouped into mailles, and mailles are grouped into domains. There are to-date close to 3000 modules organized into 100 mailles, managed by 7 units and 15 sites. The second section is made of 19 satisfaction items rated on a 4-point scale from "not at all satisfied" to "completely satisfied". These items span three major dimensions of satisfaction:

pertinence, quality and organization. They are transformed into scores ranging from 0 to 20 to mimic the traditional French student scoring system.

The reporting consists of four standardized tables that cover most kinds of needs, from the more general to the more specific. Of course, the cube has to be split according to everybody's need along the six defining keys: date, location, unit in charge, maille, module, session. These keys are treated either as filters or as repetition factors, depending on whether one wants to limit the tables to a subset of the data or to perform the same analysis for all elements of one or several keys. All legitimate users have instant and dynamic access to the data base, and can slice it as they please. They can also filter the tables according to the answers to any of the satisfaction item.

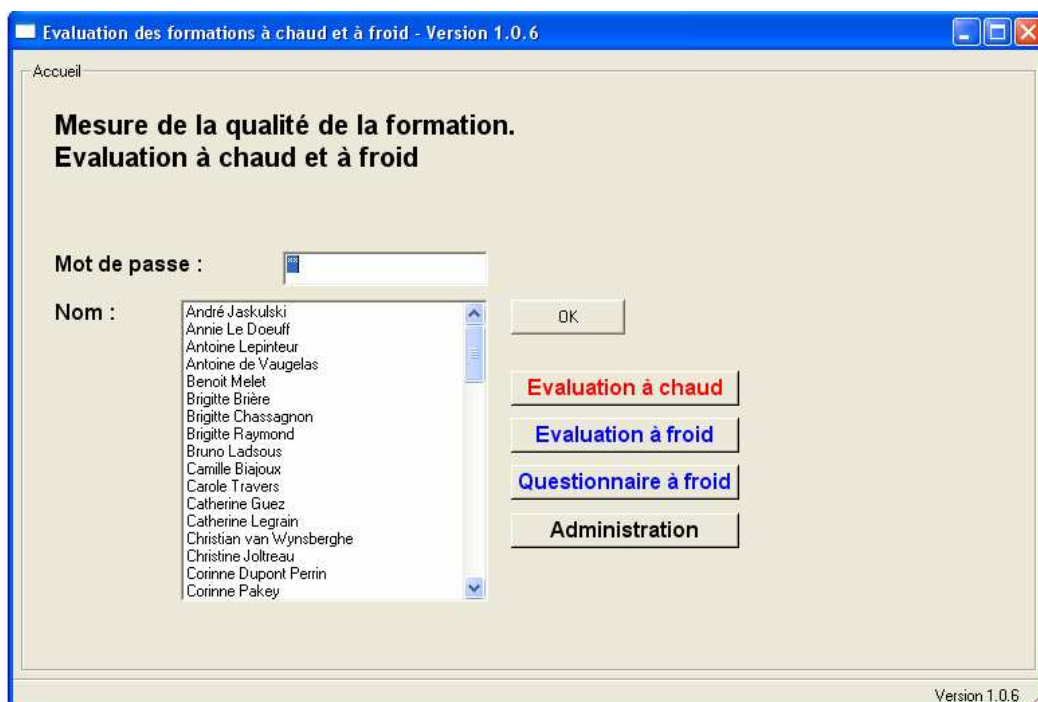
While the demands from our client were moderate – the local IT guys were bypassed – the tables ultimately had to be in Excel and properly groomed (some colour, some kind of graphs).

## Product load

Albeit simple, the user interface is heavy: 6 frames, 146 objects and 250 Ko worth of functions to be published on the client side. There are about 90000 questionnaires to treat per year and three of the four reporting tables may run several thousands lines long which have to be fed to the client. Speed is a problem and bad programming shows up early and dramatically. Tables that took seconds to show up locally suddenly take minutes. Porting an application involves much rewriting and rethinking to avoid bouncing back and forth time-consuming pieces of information.

## A quick look at the product

Before going over the few steps required to get started and summarizing what we have learned from this experience, let us have a look at the product. It sure looks like Windows.

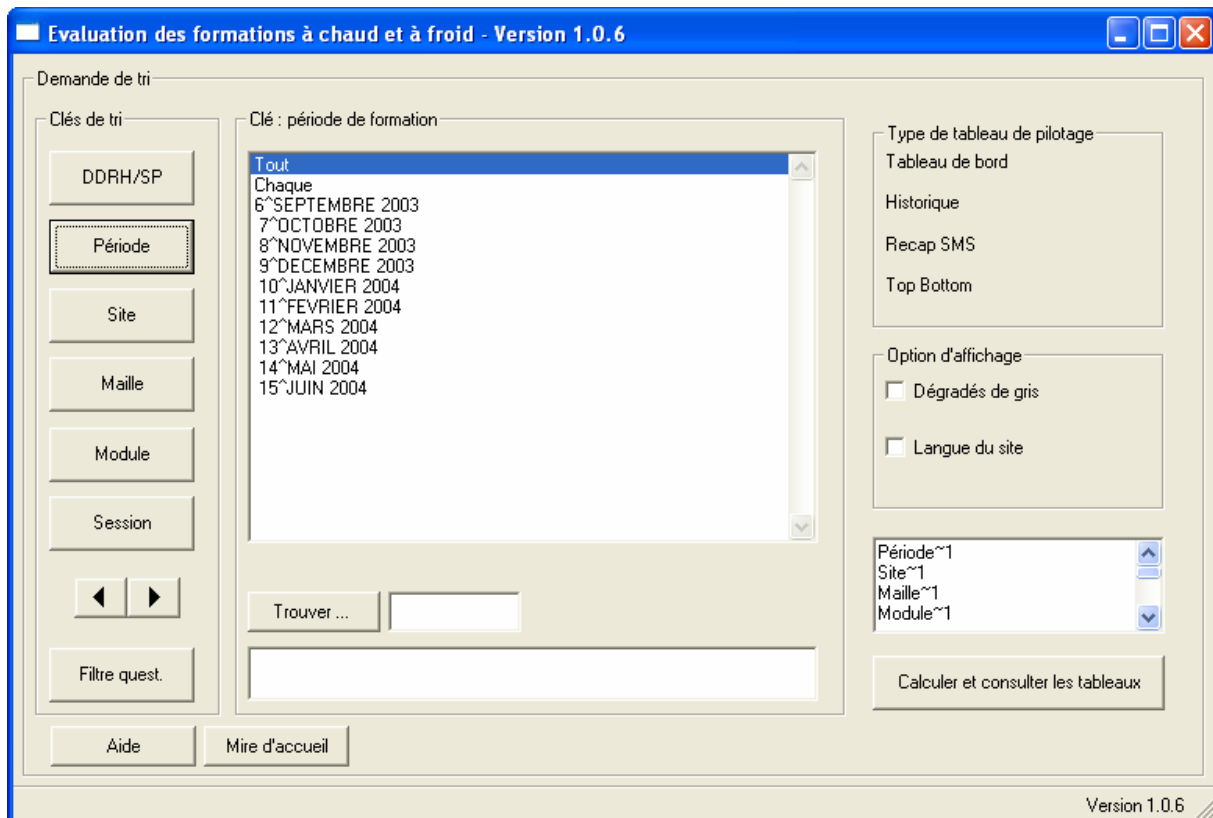


The user identifies itself in the list of legitimate users and provides his password. Passwords are used to select the appropriate subset of data for which the user has clearance. Four options are available:

1. Hot evaluation: to access the reporting of the short-term satisfaction.
2. Cold evaluation: to access the reporting of the long-term satisfaction.
3. Questionnaire generator: to consult the questionnaires' base or build a new questionnaire for long-term satisfaction.
4. Administration: to consult or modify the privileges associated with each password.

*Jairo: how can the form be made to fill all the available browser' space? Users want it maximized at launch time, all the more so that maximizing takes an awful lot of time in the browser.*

Upon clicking on the “hot evaluation” button, the following screen comes into view.



The first six left-hand side buttons, from “DDRH/SP” to “Session” correspond to the six aforementioned keys. Upon clicking on either one, the available options are displayed by its side. They are augmented by two key words – *All* and *Each* – *All* is the default value, *Each* instructs the program to do the reporting for each modality of the active key (for instance: each month). It is possible to combine all keys (each site by each period and so on...).

Since there is a list-box per key, the selections made by the user are persistent. All it takes is to click on the appropriate button to find out which selection was made. The current selection is also shown in the lower-right edit box.

*Comment: initially, the contents of the keys were updated at each selection, to speed up the selection process. However, what took no time locally ended up being too time-consuming on-line, as we had to perform the updating on the server and feeds back the updated lists, some of them possibly long (remember : 3000 modules !). This is one of the many instances where we had to trade off flexibility for speed.*

The spinner allows the users to select their preferred order for the tables where order matters: either site/mesh/module or mesh/module/site.

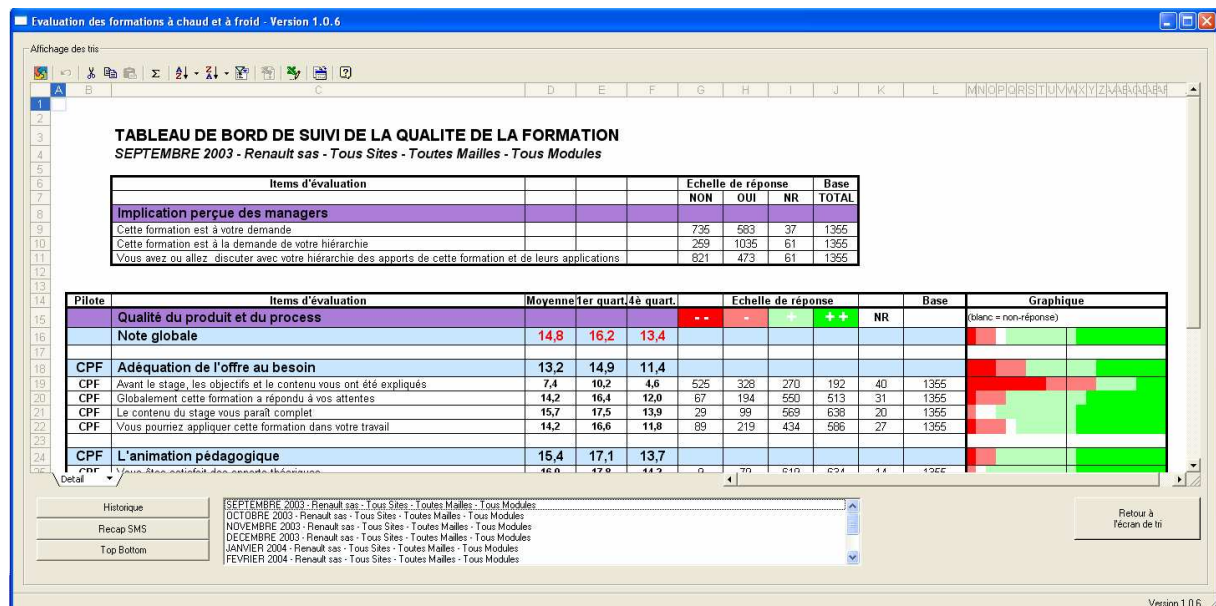
The button “Filter quest.” displays the satisfaction items should the results be filtered according to the answers of either one.

The button “Trouver” (=“Find”) searches for the line corresponding to the adjacent string typed in. This is useful when the list contains thousands of items.

*Comment: we could have used the onChange event, but chose against it for speed's sake. Also, we could not get it to work on-line.*

Finally, the user can speak in various languages (French, for the time being) or ask the graphs to use shades of gray rather than colours. Thinking that everybody had access to a colour printer was a mistake we did not make.

When the user is done with slicing the cube, he clicks on the “Calculer et consulter les Tableaux” (=“Run and see the tables”) button.



This is the first of four standardized types of tables. For each satisfaction item, it provides a concise view of the results for the user selection: mean, quartiles, distribution and a colour-coded text graph. The user can modify the table through the toolbar or save it into an XML or Excel file.

In this particular instance, we asked for a table per month with the option *Each*. To save time, note that we do not provide but the first of these tables together with the list of all requested tables. For the same reason, we do not compute right away the remaining three standardized

tables: history, recap and top/bottom. Once again, we preferred to do a piecewise job and get something on the screen as soon as possible rather than take the time to compute all tables of all types. Waiting ten times ten seconds with a little action in between does not feel as boring as waiting one hundred seconds in one chunk.

*Comment: initially, we started with a sheet per type of table in the same frame. This was a bad idea as what is passed on to the client is not a specific sheet, but the whole set of sheets. The corresponding XML file kept growing and took an inordinate amount of time to load. Better to work one sheet at a time per frame to avoid loading the same sheet time and again.*

## History table

**Historique des notes par site, maille et module**  
Période : de SEPTEMBRE 2003 à JUIN 2004

Indicateur de tendance/période précédente  
 Note en hausse : ↑  
 Note stable : →  
 Note en baisse : ↓

Indicateur de situation relative (couleur de fond)  
 Note appartenant au premier quartile (vert)  
 Note appartenant aux quartiles intermédiaires (jaune)  
 Note appartenant au dernier quartile (rouge)

Site/Maille/Module	SEP. 2003	OCT. 2003	NOV. 2003	DEC. 2003	JAN. 2004	FEV. 2004	MAR. 2004	AVR. 2004	MAI. 2004	JUI. 2004
<b>GUYANCOURT</b>										
Note globale	14,4	14,6 →	14,9 ↑	15,0 →	15,0 →	14,5 ↓	14,8 ↑	15,8 ↑	15,3 ↓	15,2 ↓
Adéquation	13,7	13,3 ↓	13,6 ↑	13,8 →	14,2 ↑	13,6 ↓	13,8 →	14,9 ↑	14,2 ↓	13,5 ↓
Animation pédagogique	14,2	15,0 ↑	15,4 ↑	15,5 →	14,8 ↓	14,8 →	15,2 ↑	16,3 ↑	15,6 ↓	15,6 ↑
Organisation	15,7	15,3 ↓	15,5 ↑	15,6 →	16,3 ↑	15,0 ↓	15,4 ↑	16,3 ↑	16,0 ↓	16,0 ↑
Note globale subjective	14,4	15,1 ↑	15,8 ↑	15,6 ↓	15,4 ↓	15,0 ↓	15,7 ↑	16,6 ↑	16,0 ↓	15,5 ↓
<b>Maille : INGENIERIE MECANIQUE</b>										
Note globale							15,5	15,0		
Adéquation							15,0			
Animation pédagogique							16,2			
Organisation							14,6			
Note globale subjective							13,7			

Here is another example of using colours and symbols to enliven a table without resorting to graphs. This table provides the means of five key dimensions (groups of items) across time for all sites, mesh and modules. The colours are a measure of location with respect to the quartiles, the arrows gives the trend: getting better, staying put, getting worse.

As for the recap and top/bottom tables, the history table keeps growing as more data are fed into the base. Such a table can be 5000 lines long and much more depending on the user selection.

*Comment: for this table, our selection called for 10000 questionnaires which generated an XML file of 1600 Ko bounced back to the client side. Clearly, this table is best suited to handling small requests – a few periods, a few meshes or modules. While it took only 5 seconds to get the first summary table, computing and loading the history table for the same selection took 90 seconds with a reasonably high speed line (1024 Ko max).*

## Top/Bottom table

■ Evaluation des formations à chaud et à froid - Version 1.0.6

Affichage des tris:

**TOP 5 des notes par site, maille, module, session**  
**Période : JUIN 2004**

Site, Maille, Module, Session		Notes de satisfaction					
<b>Les 5 sites aux notes globales calculées les plus hautes</b>		Effectif	Glob. calc.	Adéquation	Animation	Organisation	Glob. decl.
VIVIER COMMUN GROUPE		1	<b>16,3</b>	16,1	17,2	15,0	20,0
SIEGE		877	<b>16,3</b>	15,3	16,5	17,1	16,9
DOUAI		682	<b>16,2</b>	14,1	17,3	16,6	17,5
SANDOUVILLE		791	<b>16,1</b>	14,8	16,9	16,3	17,1
CLEON		812	<b>15,8</b>	14,8	16,1	16,4	16,6
<b>Les 5 mailles aux notes globales calculées les plus hautes</b>		Effectif	Glob. calc.	Adéquation	Animation	Organisation	Glob. decl.
DN7144*CREATIVITE		14	<b>20,0</b>	20,0	20,0	20,0	20,0
DN7143*MAQUETTAGE		23	<b>19,7</b>	19,6	19,8	19,6	20,0
COM7136*ANIMATION RESEAU		9	<b>18,3</b>	17,4	18,7	18,7	19,3
TERT7128*COMMUNICATION		8	<b>17,9</b>	18,7	17,9	16,9	20,0
FAB7155*FABRICATION ASSEMBLAGE MOTEUR & BOITE		7	<b>17,7</b>	15,2	19,3	17,9	20,0
<b>Detail par site : CERGY</b>		Effectif	Glob. calc.	Adéquation	Animation	Organisation	Glob. decl.
<b>Les 5 mailles les plus hautes</b>							
TERT7977*BUREAUTIQUE		7	15,7	13,5	16,8	<b>16,7</b>	17,2
EA7023*CONDITIONS DE TRAVAIL		10	15,2	14,3	15,5	<b>15,9</b>	15,4

Top

Historique: JUIN 2004 - Renault sas - Tous Sites - Toutes Mailles - Tous Modules

Recap SMS

Top Bottom

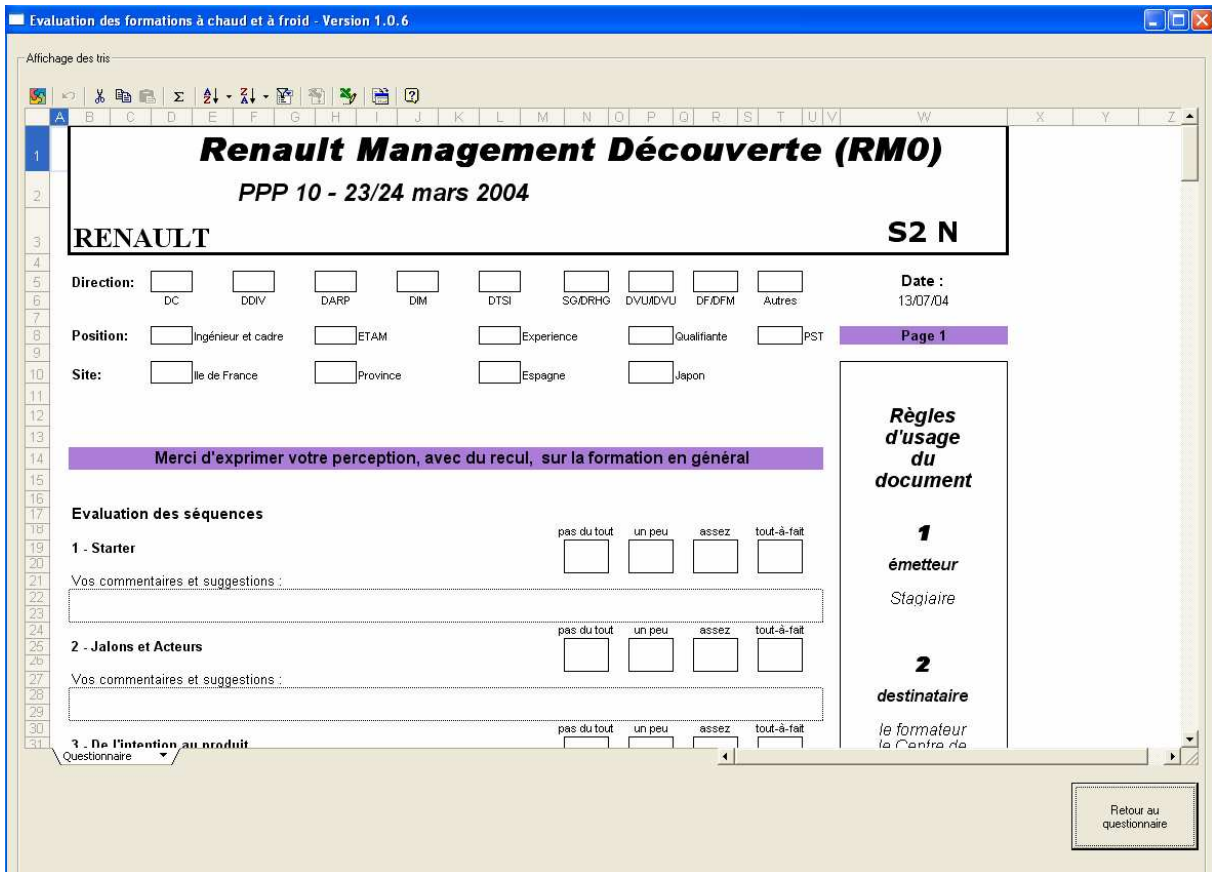
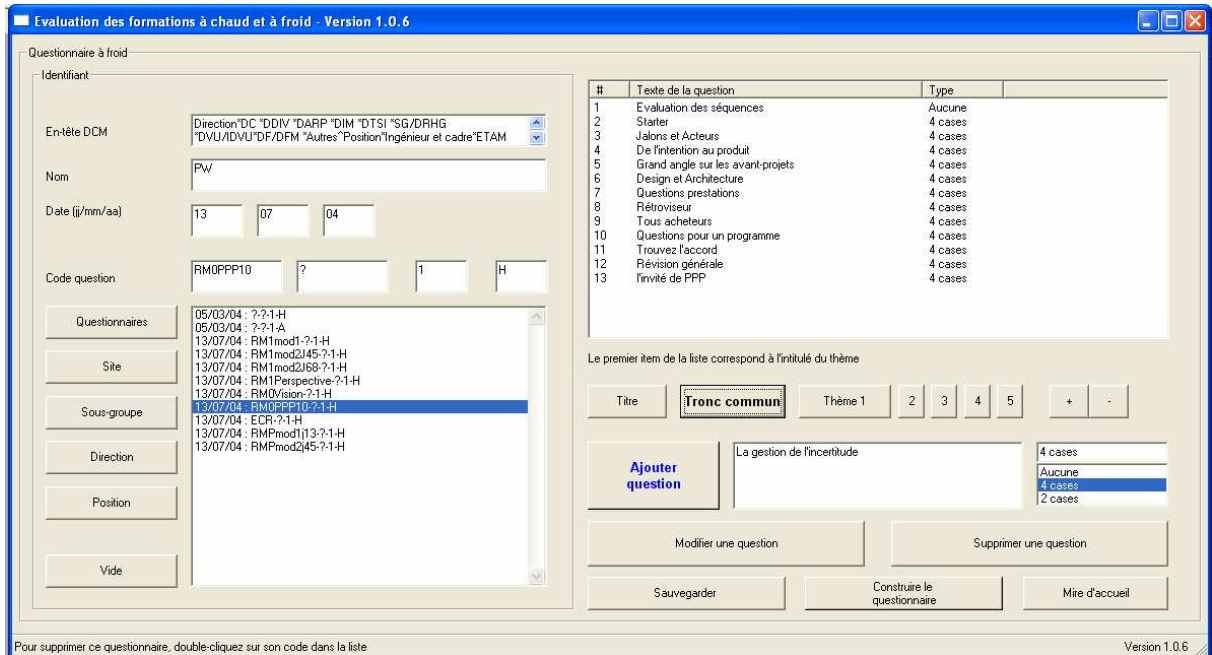
Calcul du Top/Bottom...

This table generates two sheets, one for the best performing Sites/Meshes/Modules/Sessions, one for the least. It allows training managers to identify quickly which modules need attention. While less demanding on size than the history table, it stills grows proportionally to the diversity of modules taught. As time goes by and questionnaires keep coming, the speed at which the tables are shown tends to decrease as they get bigger and bigger. The solution, of course, is to refrain from using all available data needlessly.

## Questionnaire generator

Here is another busy frame! Remember that the long-term satisfaction evaluation is assessed through all kinds of different questionnaires. While their format is identical, some may ask ten questions when other will ask twenty. The answers to the questions are read via OCR (optical character reading), but not the questions themselves. Hence the need to match all “records” in the data file with the appropriate questionnaire. By setting up an on-line questionnaire base, we solve two related problems: 1/ making sure that all questionnaires look the same (format control); 2/ having a sure proof way of matching answers from trainees with questions asked by the trainers.

The application being specific to one company and one subject matter, the questionnaire generator addresses just basic needs: three types of questions: 4-point scale, 2-point scale, open-end; four actions: add, modify, delete, save. The resulting questionnaire is then printed and dispatched. While it is still a paper and pencil questionnaire, it is clear that it will soon end up being filled up on-line.



*Comment:* not all Excel functionalities made it to MS OWC spreadsheet. In particular, line wrapping is not possible, as well as adding images, so it seems.

## What you need to get started

In order to use the APL+WebServices and APL+WebComponents to deploy an APL+Win application to the web, one needs a few pieces of software. This aspect of web development, well documented by APL2000, will only be mentioned in passing. See also Eric Lescasse URL <http://www.lescasse.com/APLWebComponent2.asp> for a step-by-step approach which we borrowed here.

### Required software

- Windows 2000 Pro, XP Pro or Windows 2003.
  - If the application needs not use the OWC components, Windows 98 might be enough. As for us, in spite of MS claim to the contrary, we have not been able to run OWC on a Windows 98 machine, to our client dismay (most were still using Windows 95).
- Internet Explorer 5 or more.
  - The MS OWC will not work on Netscape or Mozilla.
- IIS (Internet Information Services), 5 or 6.
- Microsoft .Net Framework 1.1.
- APL+Win 5.0 or more.
- APL+WebServices : they are part of APL+Win 5.0, but you need to be an APLDN subscriber to download the latest version.
- APL+WebComponent : not part of APL+Win 5.0. Once again, you need to be an APLDN subscriber.
- Microsoft Office Web Components v10 or v11 (if you need a spreadsheet or a charting object).

### Installation

This is the tricky part. Don't forget to ...

- Register APL+Win 5.0 as a COM server with the 2 following console commands :

```
Regsvr32 c:\aplwin50\aplwco.dll  
c:\aplwin50\aplw.exe /regserver
```

- Run the APLServicesSetUp.msi installation

- Prepare the workspace

Create a directory called LC (or whatever) under C:\INETPUB\WWWROOT

Create a subdirectory of LC called C:\INETPUB\WWWROOT\LC\WEBSERVICES (or whatever)

The following files can be downloaded from the APLDN Web Site.

Download and unzip the JSAVESDKFILES.ZIP file into C:\INETPUB\WWWROOT\LC\WEBSERVICES

Download and copy the latest JSAVESDK.W3 file into C:\INETPUB\WWWROOT\LC\WEBSERVICES

Download and copy the latest APLWS.W3 file into C:\INETPUB\WWWROOT\LC\WEBSERVICES

Download and copy the latest JSCRIPT\_FILES.ZIP file into

C:\INETPUB\WWWROOT\LC\WEBSERVICES

And that's it!

## A few lessons learnt

I (Pierre) am here as a testimonial of how easy it is to port an APL+Win application to the web, the rationale being: if I can do it, so can you. The truth, as we all know, is less rosy. Developing an APL+Win application for the web is more demanding than developing the same application for a local machine. You need to understand how APL fits into the .net environment, to know a little bit of this (DHTML), a little bit of that (Visual Basic) and, of course, the full support of experts unless you are one yourself. Beginners will probably encounter problems of the following kind:

- The Jscript compiler does not always behave as expected, even when you think you are doing the right thing. It takes the heat off to assume somebody else is mistaken, but more often than not, hopefully, the mistake will be yours. If not, you can rely on APL2000 to correct the problem in no time. One cannot expect a five years old compiler as Jscript to be as bug-free as the thirty years old APL system.
- What works fine in the APL+Win session may not necessary work in the browser for reasons one learns either the hard way, by trial and error, or by asking somebody who knows. Programmers who like to pack as much code in a line as possible may need to take it easy. For instance, the following code will not work in the browser :

```
:select (↑□wi 'caption')
```

You need to write it in two lines:

```
A←↑□wi 'caption'  
:select A
```

But not always. This will work in the browser:

```
'fm.st' □wi 'SetStatus' 1 'Hello'
```

But not that (it will come out as H.e.l.l.o.):

```
A←'Hello'  
'fm.st' □wi 'SetStatus' 1 A
```

- All gui objects must have a unique name independent of its parenthood. Jscript cannot tell the difference between say `fm.fr1.bn1` and `fm.fr2.bn1`, `fm` being a form and `fr1`, `fr2`, frames. If you are not aware of this detail, the application runs wild.
- Obvious logical principles are easy to overlook: functions running on the server may not call the user interface. Also, when running on the server, `□chdir` points to the `C:\Windows\System32` directory for it acts a COM object. This default directory must be changed explicitly.
- Beware of port as the following message from Eric to Jairo illustrates:

“I have spent half the day trying to make RunAtServer work on my machine. Everything I tried failed and I always get the ‘Server not available’ message. I finally decided to change the port from 2000 to 1000 and this solved the RunAtServer problem. Do you have any idea why ...”

This is what I meant by understanding how APL fits into the .net environment. Unlike Eric, I just had no clue that the port might be a problem.

- In doubt, use parentheses as in ( A[ I ; ] ). Sometimes, the parser has a problem with sparse code.
- Order sometimes matters: it is not a bad idea to set the ‘style’ property of a control before anything else.
- When everything else fails, use RunAtServer. Eric must have programmed five different browser versions of the utility TEXTREPL, to no avail. They all worked fine in APL, though.
- Talking about utilities, Eric has also produced several versions of SStoMAT to please the browser. The one we eventually used worked fine, except that it did not always return a rank 2 matrix, as in APL. In the browser, the instruction `ρSStoMAT ' /Hello '` did not return a 1 by 5 matrix as in APL, but a vector of length 5. Easy to correct, once the problem was detected.
- Because the system locks during a RunAtServer call, use `⌈wgive` to allow the browser to show a frame or an hourglass, for instance. Note that the hourglass pointer is a property that is not inherited in the browser. You must set it on the button that fires the `onClick`.
- Thanks to APL2000 taking care of our funny accented characters, through both HTML and UTF8 encoding in the soap web service.
- If you have to use OWC, it is much faster and easier to populate the spreadsheet on the server than on the client, as you can make full use of its properties and methods. The content of the spreadsheet is transformed in XML with the `xXMLData` property and placed in a temporary native file. Do not forget to create a virtual path in the APL Web Services configuration console to call the function `GetXMLFile`. This function will return the XML file to the browser and the server will then delete it. See the above mentioned URL for details. Many thanks to Jairo for this great improvement.
- While you’re at it, don’t forget to change the string `&amp;` back to `&`. For some reason, the `xXMLData` method turns all `&` into `&amp;`.

```
data←NomForme ⌈wi 'xXMLData'  
data←' /& ; /& 'TEXTREPL data
```

- It is now possible to display a “preview” message while the application loads. Text and any valid HTML code can be used.
- When you are satisfied that the damn thing at long last works, don’t rejoice too soon. Your client might want to access your application via Mozilla or Netscape with an

obsolete Windows 95 machine with tons of firewalls and proxies and a low resolution screen. When there are scores of them, your polite suggestion to upgrade may fall flat.

- Finally, never underestimate your client's information system people capacity for nuisance, all the more so if it has been left out of the development process. At the very end, I was told I had to re-do the whole application along the "Renault" safety guidelines. Basically, that meant no Microsoft products, no ActiveX, no OWC, everything running smoothly on Windows 95. Eventually, they gave up on this thoughtless demand.

## The APL+WebComponent development cycle

Eric recommends the following development cycle:

1. Design your application interface first, leaving aside as much as possible code which will run on the Server; concentrate on the interface first.
2. Write your **AutoStart** function.
3. Write your **Main** function.
4. Go very slowly, i.e. write a couple of lines at a time.
5. Always check that you are using APL constructs which are supported by JSaveSDK.
6. Test it within the APL+Win ActiveX Server workspace, in APL mode (example: `Main`"); correct any APL bug.
7. Save the workspace.
8. Then go to JSaveSDK and Republish your application. Once you have selected the functions you want to publish, you need to click 2 buttons in JSaveSDK in this order each time: **Republish** and **Package All**.
9. Test your application in the local browser (example: `http://localhost:4000/owc.htm`).
10. If everything runs fine, come back to the workspace and write a couple more lines of code and then loop at step 5.
11. If a problem occurred while testing in the browser, you need to debug the APL+WebComponent version of your application.  
(this is a little hard, and that's the reason why you really need to write one or 2 lines of code before testing again in APL and then in the browser)
12. Once the whole interface is running fine, i.e. all your published functions are running fine in the Browser, you can start writing code running on the Server.
13. Remember to use `RunAtServer` to call any subroutine running on the Server.
14. Remember that these subroutines must be monadic and return a result.
15. Remember that you should NOT do any interface stuff within the subroutines running on the Server: this is reserved to the Client side. Do this interface stuff on the Client instead and pass the resulting necessary data as arguments to the subroutines running on the Server.
16. Remember that Server subroutines arguments and result are passed from the Client to the Server and vice versa: as much as possible keep these arguments and results variables as small as possible.
17. In all cases, go very slowly, republish any time you change anything to a published function and test in the browser.

## What next?

It is fair to say that the client is quite happy with the application as it stands, but being a client, it keeps asking for more. Here is a sample of his demands.

- **Speed up the loading time.**
  - Actually, loading the application takes 90 seconds. It creates a LOT of controls and most of the time before display is used up in the browser to build the interface, not to download content from the server. Reducing the load time means making the UI smaller or more efficient (our problem, not APL 2000's).
  - As for what exactly happens, here is a description :
    - The *renault.htm* file is requested by the browser.
    - All the jsavefiles (*apl\_win.js*, *apl\_lang.js*, *blankinit.htm*, etc.) are downloaded if no cached version exists on the client or if the server has a newer version. This is done using the *Date* and *Last-Modified* headers of the HTTP protocol.
    - During the loading of *renault.htm* a request is made to the */jsaveservice/service1.asmx* virtual path. If a new copy of *renault.w3* is needed, it will be created.
- **Maximize the form to start with.**
  - To-date, the user has to maximize the form as soon as it appears on the screen. While maximizing the Renault form takes no time in APL, it needs five seconds to do so in the browser. On top of the initial ninety seconds, these additional five seconds seem like an eternity.
- **Remote updating of files.**
  - The program files (*renault.w3*, *renault.htm*) and the three data files reside on Eric's server. Each time I upload updated files, Eric must stop and restart the server. If it could be done remotely, that would be great. Jairo gave us a hint – create a virtual path in the server to upload the files – but we have not made any progress toward this goal yet.
- **Display a tif file with a Chart Object.**
  - The questionnaires are scanned and saved in tif files. We are still trying to figure out how to efficiently display them in the browser.
- **A more stable localhost server**
  - At times, the localhost server does not react, i.e. it does not respond to the *RunAtServer* calls. To get it on its feet again, one has to reboot the machine.

## Conclusion

Surf on APL 2000 web site and you'll read that tagline: *"To err is human, but when the eraser wears out ahead of the pencil, you're overdoing it"*. We used a lot of erasers in the process of setting up the Renault application. We also got first-class assistance from Jairo. The client is happy and asks for more. We cannot wait to oblige him and share our experience with whoever asks for it.

A solution you gave us for the Renault application to pass the data from the server to the client via an xml file, reaches its limit when the data to be sent is large. XML is so verbose.

For instance, what ends up being saved in an Excel file of size 400Ko with 1700 lines, requires an XML file of size 2250Ko. I looked into the xml file, it had 65000 lines of xml code.

Clearly, the instruction `[]wi 'xXMLData'` does the trick, but raises some problem at the client size where big chunks of data are not allowed in by the local IT cops.

Would you know of a way to compress the data at the server, send it and then decompress it on the client size ?

Thank you for your help.