PX0102

Erik Stevenson

From: Steven Sinofsky
To: billg; bradsi; davidcol; dennisad; jimall; jonl; paulma
Subject: FW: Novell AppWare -- first impressions
Date: Friday, November 05, 1993 10:52AM

This is very interesting. I was looking over the books and stuff and was on the one hand impressed (they got all the right stuff for MIS people--style guides, methodology, etc.) and on the other hand it is scary since it is just another windowing API, and a fairly complete one. This is direct competition to Windows, not VB or VC++. Not what I was expecting based on their white paper (which seemed to focus on using the recently acquired Serius Developer development tool, with some C code for customization).

We're not sure what Borland is going to do with this, but they are making a lot of noise about it.

From: Scott Randell
To: SYS Denis Gilbert's direct reports; Garth Hitchens; Luis Talavera; Richard Tong; Steven Sinofsky
Cc: Scott Randell
Subject: Novell AppWare -- first impressions
Date: Thursday, November 04, 1993 9:04PM

re: Novell AppWare Foundation SDK

==
Well I've taken a quick look at the "NAWF" SDK. Here is my overview and my impressions->

What is it ?

It is just the SDK for the NAWF API. It does not include any visual tools or compilers or anything else of interest.

It is / it contains:

A set of headers for the NAWF API
A set of libraries and DLLs for the NAWF layer on Windows.
Some sample code, an online API help file.
A few funky tools for dealing with their strange resource scripts and other AWKfull things.

It is not / it does not contain:

No visual tools
No compiler
No extensive examples (ok they have Draw and Clock)

How done is it ?

It is a very complete product for an API -- with a lot of documentation. It comes with many books covering the API reference, overviews, a study guide, coding conventions, and so on. They also have a training course for people to learn about it.

They've done a good job, as SteveSi put it, targetting what MIS people like -- thick books, spelled out conventions, very structured taxonomies, courses for learning, hand holding and customer support.
The documentation is well written -- but still a little raw

MS7096165
CONFIDENTIAL

(including review comments to the writers).

The next release of NAWF will be available in February.
The SDK is only for evaluation purposes -- in the future
you will need to get a licence to incorporate the libraries.

What is the NAWF API:

It is a Windows-like C API that abstracts out platform specific differences. They claim they currently run under Windows (Win 3.x, not NT), the Mac and Unix (X Windows). There is no OS/2 version yet. The SDK Kit I got from LuisT is just for Windows development.

The API is split into three major categories:

- Operating System (base features)
- Network Connectivity
- User Interface (gui)

It is really just a C API that looks a heck of a lot like Windows.
It is not really object oriented in any way beyond what WndProcs and switch statements in DialogProcs give you.
The flavor of the APIs are quite different with different conventions (naming conventions in things like passing access objects around) and especially the way objects are instantiated,
There are lots of public and unprotected data structures -- not unlike many C API or portability toolkits I have seen in the past.

There are a ton of APIs -- they revel in their complexity.
Over 750 new APIs (in addition to the platform specific APIs that you may choose to use). Over 150 data structures (with all public data fields, remember this is C), and over 3500 #defines for constants (and I thought Win32 was complicated !-).

They mention a C++ framework (NAWFFW Class Library) layered on top of the NAWF API. It was not part of the SDK.

How do you use it.

For Windows:
You develop and compile code using your favorite PC development tools (MS C, VC++, BC), edit your resources using your favorite tools (DlgEdit, App Studio, Resource Workshop) and just compile and link with the NAWF headers and libs.

They have a bunch of funky command line tools to process resource files, not unlike Bedrock.
The resulting application has the proper native look and feel.

For the Mac, you use MPW (not enough stuff was provided with the Windows SDK to do that of course).

= =

MORE DETAIL:

The Operating System "Series":

This consists of general purpose functionality, abstraction of the operating system and hardware. This is very similar to the services provided by Windows KERNEL, SYSTEM and GDI.
Includes the following "components" or services:

- Character (including locale)
- Data (including collections, formatting, currency, ...)
- Error (for exceptions -- but using GetLastError and

error hooks)
File (including aliases, directories, ...)
Font (platform independent font selection)
Graphics (platform independent graphics -- hmm, looks rather GDI-like)
Instances (this is the unique notion of an object created in a IMHO strange way, usually from some resource information (i.e. a super LoadResource -- not unlike MacApp).
Keyboard (access to key state)
Memory (abstracted memory management -- yes with moveable blocks)
Messages (callback procs, postable messages -- when used it looks a lot like Windows MSG)
Module (abstraction of HMODULE and other app global things)
Pointer (cursor on the screen)
Preferences (== preferences)
Print (device independent printing, GDI like)
Records (for storing in collections, simple data structure with a hell of a lot of complexity)
Resource (resource management ala Windows or Mac)
System (misc system services and general machine info)

The Network Connectivity Series:
This is one of the unique things they have that they play up a lot -- they have a model where they are more network aware than say Windows.
Includes the following "components" or services:

Clipboard (similar to Windows clipboard)
Pipes (a simple pipe model with a ton of options I don't understand -- with network capabilities of course, this is Novell remember)
Tasks (platform independent API to get at multi-tasking info -- doesn't do anything to the 2/3 of the platforms currently supported).

The User Interface Series:
Thiis is the main USER level user interface functionality. They have no "high-level abstractions" (save perhaps Table) like toolbars or status bars. They are designed around a very simple UI, so there aren't a lot of UI Widgets (like spin buttons).

Includes the following "components" or services:
App (misc global app data structure, also provides a Windows specific one too)
Box (~=group box)
Button (~=Windows BUTTON control)
Dialog (souped up dialog manager)
Display (~=Windows STATIC control)
Edit Text (EDIT control, but abstracted to cover Mac TextEdit)
Help (simple help hook, but like many components they have a lot of optional bits -- some of which may actually do something).
Item (screen drawable object -- good for static graphics).
List (LISTBOX more or less)
Menu (~= menu)
Slider (a slider control)
Standard Package (~= COMMDLG)
Table (two dimensional grid control, really ugly)
Void (terrible name, but this is a place holder

for custom output to be drawn on a dialog.
If you can't layout a dialog with the canned
Buttons (BUTTON), Display (STATIC),
List (LISTBOX) or Items (static graphics),
then you plop down your own custom "void"
component.
Window (very HWND-like as you probably could have
guessed).

For all the above when I say "similar to Windows XXX" that means it looks functionally very similar, and even the flavor of the API is similar. Of course the worlds are very very different when it comes to looking at what code you write. It would require a complete rewrite to convert a Windows app to NAWF or vice-versa because there are so many basic differences.

For your amusement, here is the C code to their Hello World sample application ->
<<File Attachment: SAMPLE.C>>

IMHO->
I personally wouldn't want to program to this API, so if it turns out to be popular, then something like OWL or the NAWFFW class library on top of it would be really valuable to programmers who want the portability feature of NAWF without having to deal with the crazy C API.
Windows programmers won't like it since it is not Windows, Mac programmers won't like it since it is too Windows-like (too un-Mac-like). Don't ask me about Unix programmers.

==========
==========
What this means to us (both VC++ and MS as a whole) ?

I'm not really sure.
This is a competitive API, competitive to the Windows and Win32 APIs. It is positioned as a portability toolkit.

Our Mac/Unix story is very similar -- instead of the NAWF API, we have the Win32x API.

This isn't a Bedrock or a Taligent -- it isn't an object oriented system API, it is a strange portability toolkit that isn't functionally all that much better than Windows. And IMHO it is pretty bad from an API point of view. This won't get acceptance as a better API than Windows -- except that it has a "portability story".
Bottom line NAWF is not a direct competitor to MFC.

OWL on AppWare would be a direct competitor to MFC on Windows. This of course depends a lot on what Borland does with its OWL2 and OWL2/Appware products.
The mapping from OWL2 to NAWF would be very weird regardless.

I personally think we can turn this to our advantage, something like:
"Geez Borland just screwed over OWL 1 users with OWL 2 on Windows, now they are abandoning Windows for that AppWare platform. What's AppWare? -- it's a platform/API that Novell is in the process of designing that looks a lot like Windows, but isn't Windows -- isn't object oriented -- and by the way requires a runtime licence you will have to pay to Novell..."

Since AppWare is _not_ vaporware (at least with this Early

Adopter version) -- we can turn this to our advantage.
OWL/AppWare is vaporware (even a TeamB guy
on CIS said that) so we should play up the fact that
OWL/Appware is a vaporware product on a very rough
infrastructure. The only danger I see is if "OWL/AppWare"
remains vapor or pseudo-vapor like Bedrock, Taligent,
OpenDoc,... -- you know how bad we are at competing
with vapor.
Since AppWare is concrete - if developers look at it I don't
think those developers will want it.

==

all for now....

...scott