The Old New Thing

What the various registry data types mean is different from how they are handled

5 Feb 2009 10:00 AM | 55

Although you can tag your registry data with any of a variety of types, such as REG_DWORD or REG_BINARY or REG_EXPAND_SZ. What do these mean, really?

Well, that depends on what you mean by *mean*, specifically, who is doing the interpreting.

At the bottom, the data stored in the registry are opaque chunks of data. The registry itself doesn't care if you lie and write two bytes of data to something you tagged as REG_DWORD. (Try it!) The type is just another user-defined piece of metadata. The registry dutifully remembers the two bytes you stored, and when the next person comes by asking for the data, those two bytes come out, along with the type REG_DWORD. Garbage in, garbage out. The registry doesn't care that what you wrote doesn't many any sense any more than the NTFS file system driver doesn't care that you wrote an invalid XML document to the file `config.xml`. Its job is just to remember what you wrote and produce it later upon request.

There is one place where the registry does pay attention to the type, and that's when you use one of the types that involve strings. If you use the `RegQueryValueA` function to read data which is tagged with one of the string types (such as REG_SZ), then the registry code will read the raw data from its database, and then call `WideCharToMultiByte` to convert it to ANSI. But that's the extent of its assistance.

Just as the registry doesn't care whether you really wrote four bytes when you claimed to be writing a REG_DWORD, is also doesn't care whether the various string types actually are of the form they claim to be. If you forget to include the null terminator in your byte count when you write the data to the registry, then the null terminator will not be stored to the registry, and the next person to read from it will not read back a null terminator.

This simplicity in design pushes the responsibility onto the code that uses the registry. If you read a registry value and the data is tagged with the REG_EXPAND_SZ type, then it's up to you to expand it if that's what you want to do. The REG_EXPAND_SZ value is just part of the secret handshake between the code that wrote the data and the code that is reading it, a secret handshake which is well-understood *by convention*. After all, if `RegQueryValueEx` automatically expanded the value, then how could you read the original unexpanded value?

Windows Vista added a new function `RegGetValue` which tries to take care of most of the cumbersome parts of reading registry values. You can tell it what data types you are expecting (and it will fail if the data is of an incompatible type), and it coerces the data to match its putative type. For example, it auto-expands REG_EXPAND_SZ data, and if a blob of registry data marked REG_SZ is missing a null terminator, `RegGetValue` will add one for you. Better late than never.

Blog - Comment List MSDN TechNet

Comments

John

5 Feb 2009 10:10 AM
#

This is the main thing about the registry I have never liked. It seems like the only point of having a type associated with the value is so that things like RegEdit can work. Generally you are reading from and writing to a specific registry value for which you know the type beforehand.

Andrew
5 Feb 2009 10:14 AM
#

@John

But what about areas like the Registry Editor, where it uses that type to show a different editor?

SmartyPants
5 Feb 2009 11:15 AM
#

Its comforting to know you can still create hidden reg keys using internal NT api's and regedit (atleast on WS 2003) cant open them.

quarterlife
5 Feb 2009 11:22 AM
#

"The registry doesn't care that what you wrote doesn't many any sense any more than the NTFS file system"

That doesn't make any sense at all (which is funny considering what you were trying to say)

SmartyPants
5 Feb 2009 11:29 AM
#

@quarterlife

No, it makes sense. Are you a non-native speaker?

Mark Sowul
5 Feb 2009 11:36 AM
#

No, it doesn't make sense, because it says "many" instead of "make."

[*Holy cow, I read it twice and didn't see the typo until you pointed it out. -Raymond*]

Koro
5 Feb 2009 12:12 PM
#

So that's what RegGetValue does!

I was still wondering from the last time.

Anonymous Coward
5 Feb 2009 12:19 PM
#

Since NTFS is a file system, and the registry is like a filesystem, will the Windows team ever merge the two into one that performs well for both kinds of data? So that the registry hyves can then be simply folders in your profile that you can browse to, make shortcuts to keys, open them in Notepad, &c?

ton
5 Feb 2009 1:04 PM
#

Raymond, I'm curious are you familiar with the history behind why the registry was designed as a hierarchical database instead of a relational database?

Mark
5 Feb 2009 2:31 PM
#

ton: why would you ever need more than one way to get a setting, or to retrieve a large subset according to a common property? The decision between HKCU and HKLM is taken wrongly enough of the time already.

And how would you design it? A large proportion of settings in the registry are optional. You'd either end up with 3 tables, or so many that it would effectively be a filesystem. That comes with enormous memory and maintenance difficulties, not to mention the likelihood of unimplemented classes (thinking WMI here).

I'm not trying to bash your idea here - I'm interested in whether you have some existing system in mind.

ton

5 Feb 2009 3:26 PM
#

@Mark it's not really about having multiple ways to retrieve a setting. It's that relational databases have proven to be a better storage model over the years than hierarchical database systems have. That's what the registry really is; a hierarchical database that stores configuration settings for how the windows operating system and installed applications will behave.

A common problem is that programs and sometimes people will corrupt the registry by deleting a setting or supplying a nonsense value. If the registry had been designed as a relational database instead then deletions that destabilize the system could be prevented thru referential integrity and value constraints. It just would have made for a more stable system overall as opposed to what Microsoft has to support with the registry now. However, I fully acknowledge that it is much too late to change this now because of backwards compatibility issues and its also why I only asked Raymond about the history.

Bryan
5 Feb 2009 5:29 PM
#

I don't understand how the registry being a relational database would protect against someone corrupting the registry by deleting a setting or supply a nonsense value.

As far as I'm aware, the major advantage of a relational database is the fact that you can model more relationships than can be modeled in a hierarchical database. I don't understand how that modeling technique (which would've been more expensive right?) would prevent misuse of the registry.

Cooney
5 Feb 2009 5:50 PM
#

> Since NTFS is a file system, and the registry is like a filesystem, will the Windows team ever merge the two into one that performs well for both kinds of data? So that the registry hyves can then be simply folders in your profile that you can browse to, make shortcuts to keys, open them in Notepad, &c?

I doubt it; reg data is far more granular than FS data, so different rules are required to make each perform well.

ton
5 Feb 2009 6:04 PM
#

@Bryan

An application would have a table or group of tables in relational style registry. A group of settings would be a row. A single setting would be a column. Is it starting to become clearer now how SQL like statements could now be used to constrain what gets deleted

and added? How good is your understanding of SQL and DBMS?

Bryan
5 Feb 2009 8:35 PM
#

That makes some level of sense, but the prospect of enforcing constraints seems like it defeats the whole prospect of the registry being lightweight. I would only say I know only basic RDBMS knowledge, but I know table data can be heavy (com structured storage being a case-in-point).

agrirmSmadcaf
5 Feb 2009 8:39 PM
#

dgdfgds fgf gdsf gssd f gfsd df dsfdgdfgds fgf gdsf gssd f gfsd df dgdfgds fgf gdsf gssd f gfsd df

ton
5 Feb 2009 9:00 PM
#

@Bryan

You must understand the registry is storing data that determines whether or not your computer can boot! Instead of being lightweight the registry needs to be robust and durable.

Anonymous Coward
5 Feb 2009 9:44 PM
#

>I doubt it; reg data is far more granular than FS data, so different rules are required to make each perform well.

But there is no reason the merged filesystem couldn't contain both rulesets. In fact, it already does in a sense, since the hives are already saved in regular files in the filesystem.

Mark
5 Feb 2009 9:59 PM
#

ton: it seems there's two changes with your suggestion, viz.

1) The isolation of program data, which is currently done through cooperation of programs and permissions on individual keys. Would you allow programs to access each other's tables? If so, how do you make sure they don't do that accidentally?

An alternative is to have different tables for each class of data (Paths, Dwords, UninstallInfos), but you'd then need huge indexes, and a sensible way for adding new classes.

2) The formation of settings into rows and columns. I don't think this will help at all: nearly all registry data is individual settings, and doesn't fit into a grid at all.

How good is your understanding of SQL and DBMS? Good enough to know that forgetting the WHERE clause in a DELETE is about as easy as stomping someone else's registry key. I also feel that the benefits of RDBMS only emerge with indexing and normalisation.

Perhaps if the Windows 3.1 OLE registry had been designed to use tables (like MSIs) it would be more efficient now. But your average program just wants to store window sizes or the last 5 opened documents: the registry was doomed as soon as someone decided to store config in it.

As for why that happened, my guess (which may or may not be as good as Raymond's) is that it was economy. Imagine someone in the NT team looking around for some way to manage the rapid proliferation of .ini files - a hierarchical database that was already coded would have been too tempting not to use. (For some context, see the History part of http://home.eunet.no/~pnordahl/ntpasswd/WinReg.txt and bear in mind developers were used to GetPrivateProfileString, etc.)

Mark
5 Feb 2009 10:08 PM
#

Anonymous Coward: the trouble isn't in the granularity below the filesystem layer, but above. How would Windows tell a program that it can open a value like a text file, but can't get its last modified date? Or that cmd's current directory is longer than MAX_PATH characters long? Better to let programs treat them separately, since they'll have to anyway.

ton
5 Feb 2009 10:56 PM
#

@Mark

You are over complicating my brilliant design :-)

1)All access to the database,isolation, and "accidents" can be controlled by only allowing each program to have permissions to its own database and tables.
(e.g.GRANT,REVOKE,DENY)

2)Even if there is only one column in the table it's still better than using a hierarchical design especially when it comes to critical data. Also, most programs I've seen usually have *multiple* settings for each program if they are non trivial which is almost always for most commercial programs that software vendors sell. SQL and RDMS are fully

capable of fulfilling all requirements you brought up.

http://en.wikipedia.org/wiki/SQL#Data_control

http://en.wikipedia.org/wiki/DBMS#DBMS_Features_and_capabilities

Read both links carefully and you'll start to see the possibilities.

Anonymous Coward
5 Feb 2009 11:36 PM
#

It is already perfectly possible to create paths longer than MAX_PATH so while merging might increase the chance of hitting that, it doesn't really create a problem that doesn't already exist. The correct solution would be to remove the whole MAX_PATH restriction. This would probably require moving over to a new API, and saying 'no sorry you can't' to ancient applications. We already do that by the way when an application using the ANSI API tries to open セーラー服と機関銃.mp3 for example. Similar applies to the time, ACL's or lack thereof, and so on. For old applications using the old API everything will appear as it was as long as they access the registry through the registry functions, but for migrated applications like the shell things would be much simpler because you'd have one API for doing one thing, and as I said it would enable a lot of new features. Features which we should have been able to take for granted, given that the registry is a file system. And even unmigrated applications would benefit most of the time, seeing as when I concatenated the most ridiculously long keyname I could find (somewhere in the Windows part of the registry) with the place where ntuser.dat is now, I only got about half of MAX_PATH.

Dave
6 Feb 2009 5:02 AM
#

>That makes some level of sense, but the prospect of enforcing constraints seems like it defeats

>the whole prospect of the registry being lightweight.

Exactly. As the current registry shows it was perfectly possible to make it a bloated mass of cruft without having to resort to implementing constraints.

A Crazy Person
6 Feb 2009 6:55 AM
#

Wow!

Armed with this information, can I now hide extra configuration information in the bytes after a DWORD value, storing a whole pile of hidden settings, masquerading as a zero DWORD value in the UI?

(ducks)

Anonymous
6 Feb 2009 8:37 AM
#

And, unfortunately, we will not be able to actually use RegGetValue for several years more, since there are still a lot of people who use XP or earlier systems.

I recently heard from a developer from another company that they cannot use a more recent version of their chosen development environment, which creates programs that require newer API functions, because several of their clients are still on Windows 9x.

DrkMatter
6 Feb 2009 9:09 AM
#

"All access to the database,isolation, and "accidents" can be controlled by only allowing each program to have permissions to its own database and tables."

Except that this does not only involve a change to the registry, but to the whole Win32 security model which, as far as I know, has no concept whatsoever of application identity. Security constraints are always applied on a per user basis.

Thom
6 Feb 2009 9:13 AM
#

@ton

What is the mechanism by which you grant each program it's own permissions? How do you protect against collisions, impersonations, etc.? What about "suites" where several programs work together and share registry data? What about all the registry data that is written by programs but largely used by windows itself (or other programs... interface stuff,etc.)?

Anonymous Coward
6 Feb 2009 10:18 AM
#

>not be able to actually use RegGetValue for several years

Can't you just use RegGetValue from Wine?

http://source.winehq.org/git/wine.git/?f=dlls/advapi32/registry.c;hb=HEAD

A cursory glance seems to indicate that it's implemented in terms of other API's so you can add it in a helper DLL if you need it.

>each program it's own permissions

And that isn't even what you really want... you really want different instances of programs to possibly have different sets of permissions too. I think the only way to solve that problem would be to make Windows (or Linux depending on what's easier to do) more object oriented, eventually turning the old Win32 and POSIX API's into an emulation layer.

Bryan
6 Feb 2009 10:35 AM
#

I entirely disagree with the Registry being robust and durable. No, the last thing we need is another slow mechanism that makes it difficult and unhelpful to store information.

Further, all that permissions work would be a bear. I don't really feel like having to develop or use a registry management library just for storing application data.

In the end, your solution seems overcomplicated and restrictive. Windows can certainly boot without all of the registry data intact as well as long as it can load the hives themselves.

ton
6 Feb 2009 10:54 AM
#

@DrkMatter

Remember basic object-oriented design. The registry would have application identity knowledge and enforce security access to the registry the win32 security model would go unchanged.

@Thom

The registry would grant permissions to each installed program. The registry could store a sha-256 hash to verify application identity. In the suite case it would simply be a collection of tables for each component in the suite. The last case only changes in one way because of a move to the relational model from a hierarchical model, and that is the way the data is accessed. What I have presented is pretty simple; if you are willing to completely forget about backwards compatibility :-)

Thom
6 Feb 2009 11:39 AM
#

@ton

But the question becomes how would the registry know when and to which program to grant permissions. The hash would have to be updated each and every time the application was updated, else all settings would be lost. What if I have 2,3,4 or 5 versions of a program on my computer for some reason? All have different hashes, what data is shared and what isn't, and who decides?

In these instances do you ask the user and trust their answer? Do you decide when a program is installed or updated? Do you let the program tell you, which only works for things it currently knows of like updates or new add-ons, or lets some malware fool you?

Same for the suite of applications, who tells you what is part of the suite? The user? The application(s) - which may not be installed all at once but separately over time? What if some are updated and others aren't? What about third party add ons, especially competitive ones that a program might wish to block?

Even tossing out backwards compatibility it grows very complex very quickly, so much so that it becomes an unworkable solution. I wish it didn't.

DrkMatter
6 Feb 2009 11:50 AM
#

@ton

Even if you disregard the concerns about determining application identity, which Thom explained, there is still the question of ressource ownership. The registry and the data it contains belong to the user, not the application. Much like how the files that make up an application's executable data also belongs to the user. If the user wants to delete those files, or replace them with others, it is his choice: much like it is the user's choice to overwrite or delete any registry configuration for any application.

Anonymous
6 Feb 2009 12:29 PM
#

@Anonymous Coward:

> Can't you just use RegGetValue from Wine?

Only if your code's license is LGPL-compatible. I suppose you could create a LGPL-licensed helper DLL and use it in your application, however. You would then only have to redistribute the DLL's source code together with your application.

But you still would be reimplementing RegGetValue instead of simply using it.

ton
6 Feb 2009 12:46 PM
#

>Even if you disregard the concerns about determining application identity, which Thom explained, there is still the question of ressource ownership.

Both of your objections about application identity stem from a lack of understanding of how hash algorithms work. If there is a different version of the same program then it would create a different hash value. It's just that simple. As for resource ownership the user can override the constraints if they want but they should be ready to deal with the

consequences and they would be prompted with such information. You can't save everyone from winning the Darwin award.

I have only proposed a change from a hierarchical model to a relational one for data storage for the registry. ALL THE OTHER CAPABILITIES OF THE REGISTRY ARE COMPLETELY UNAFFECTED BY HOW IT ACTUALLY STORES DATA.

It just adds additional benefits and tools that a hierarchical system can't provide. geez...

Bryan
6 Feb 2009 12:58 PM
#

"Both of your objections about application identity stem from a lack of understanding of how hash algorithms work. If there is a different version of the same program then it would create a different hash value."

That's the point: sometimes, we don't want it to, other times we do. The application I'm working on has 4 different versions that can be installed at the same time for feature-related reasons. In addition, each of those versions has 1 - 3 minor versions that have to share registry data with the other versions; however, must also be independently identified as minor versions.

Users aren't going to care about the above situation. UAC will look like a walk in the park compared to trying to help a user understand how to deal with the above scenario.

Your system also adds additional complexity, management requirements, and critical design issues that a hierarchical system doesn't have.

night
6 Feb 2009 1:22 PM
#

@Coward

> Since NTFS is a file system, and the registry is like a filesystem, will the Windows team ever merge the two into one that performs well for both kinds of data?

Yes, I would favor that, too, having lost data due to corrution of Outlook Express databases and OLE structured storage many times in the past. Keep things simple and have everything in a single, reliable, transparent storage system. Modern file systems should be strong enough for that.

Thom
6 Feb 2009 1:36 PM
#

@ton

Sorry ton, but I think the lack of understanding is on your part. All those hash values have to be associated with data in the registry and it is *impossible* for the OS, the

software, or the user to give a definitive answer to what data should be associated with what hashes and available to what programs.

The OS has no way of knowing what program(s) should or should not be able to read or write particular data aside from the original hashed executable that wrote it. Allow all - in other words you've just tagged every bit of data with the hash of the executable that wrote it - then what have you gained but a lot of overhead to tell you who (originally?, last? journalled?) wrote the data. Deny all others but that one hashed executable, then you limit or break everything but that exact one. You can't do *anything* based on those hashes without breaking something, so why bother with them to begin with.

You can't rely on allowing the original executable to tell the OS what programs can read or write the data because the original executable only knows about (at best) programs that currently exist, and probably only those that are already installed. It has no knowledge of future updates or expansions to itself, updates to suite programs, new add-ons (including third parties), new OS components, etc.

You can't rely on the user to tell you or even aid you, because even the most technically savy user has no idea what data is being written/read, when, and why.

Any change in OS components, OS features, installed software, etc., can require that those relations be changed *if they are actually used for anything* - but there is NO way for the OS, the original software, the user, or even the new software to determine how to do so with any accuracy.

Markus
6 Feb 2009 1:52 PM
#

ton. relational model offer only additional problems as means of access control. relational model offer only overhead as diff means of storage only.

Anonymous Coward
6 Feb 2009 6:21 PM
#

>corruption of Outlook Express databases and OLE structured storage

Yes, what's with all these subfilesystems in the first place? They just make things more difficult for the end user (editing a CHM file is significantly more involved than editing a folder of HTML files) and tend to be implemented badly. CHM is a case in point, someone once remarked that it looked like it was cooked up by an intern.

>could create a LGPL-licensed helper DLL

That was what I said yes. On Vista you could simply not install the DLL and use the system provided function, if you're desperate to shave a few kB of your working set. But if you're that desperate you shouldn't be running Vista.

The whole hashes idea sounds bonkers to me. It's fragile, probably more annoying than UAC, provides less security than the current system could provide, and worse, it sounds complicated. Which means that most people will not bother. It cannot properly differentiate between different instances of the same program without doing trickery,

and it requires a lot of configuring that sounds like it would be hard to do dynamically. All in all, it sounds complex, contrived, fragile, and non-intuitive.

I'd prefer an object-oriented approach. Take things like Java or .NET as examples. If you pass an object to a function, you can call methods on its interfaces. Now imagine that you can port that concept... like an application would be the function that you could pass objects to. Over the past decades we've come a long way in understanding object-oriented design, we now know a lot about it, how to reason about it, and I think applying that knowledge to operating system design and security would be a very fruitful endeavour.

The Imp
7 Feb 2009 6:24 AM
#

@ton

A good reason for not designing (or redesigning) the registry as a relational database, is that traditionally, they are designed such that getting multiple chunks of data at once is as simple as getting single chunks. But with the registry, it's generally implicit that you're going to be asking for solitary chunks of data only. Hmm. Well, I guess that's less true than it used to be… And the fact that the original design of the registry never expected to have very much of anything in it, and certainly not almost every setting for every program on the system.

Why do you assume that you cannot add referential integrity checks on a hierarchical (or spatial or otherwise non-relational) database?

@Anonymous Coward

NTFS and the registry cannot be merged as you suggest (however good that idea might otherwise be). They both support the same ACL system, but there is no guarantee that the filesystem where the registry settings will live, will be an NTFS filesystem. If it isn't, there will be NO way to enforce ACLs on registry entries. I do love the idea, though.

Anonymous Coward
7 Feb 2009 7:50 AM
#

>no guarantee ... will be an NTFS filesystem

But in the case that it is, it could be merged. And if it isn't... well, we're using a subfilesystem already, so we might as well make this subfilesystem (in the case that say ntsuser.dat is saved on a FAT volume) this NTFS/registry merger. It'd be pretty much like working with an image file. I think it could be done, but then, I'm an optimist.

eth0
8 Feb 2009 12:53 PM
#

> They both support the same ACL system, but there is no guarantee that the filesystem

where the registry settings will live, will be an NTFS filesystem. If it isn't, there will be NO way to enforce ACLs on registry entries.

Registry ACL permissions are not the same as NTFS ACL permissions.

They're both ACLs but not interchangeable.

Remember, as far as the NTFS filesystem is concerned, the separate Hives are just like any other file.

And Registry ACLs are available on FAT32 systems as well.

Anonymous Coward
8 Feb 2009 3:05 PM
#

The registry and NTFS permissions are not entirely identical, but the basic ACL architecture appears to be the same. They could certainly be merged. On FAT32 however you only have the DOS attributes, no ACLs. So to make it work there you either need subfilesystem trickery (which we're already doing) or you would need to store the required additional meta-data in files in the filesystem, which you hide from the API.

steveg
8 Feb 2009 5:40 PM
#

My guess the Registry is not an RDMS is because it would've made Win3.x/95 run even slower (I think the Registry came first on 3.x). Theoretically a bespoke registry implementation would be faster than a more generic RDMS.

I do, however, like the sounds of making the Registry visible in the file system -- ala /proc on linux (for those who don't know most anything can show up in /proc as, apparently, a file even though they're not really... eg /proc/cpu might be a text file that contains info about your cpu).

grep "malware" /proc/registry/HKCU | delreg

Miral
8 Feb 2009 7:38 PM
#

@A Crazy Person:

That's the first thing that occurred to me as well. I wonder how regedit displays such things?

Anonymous Coward
8 Feb 2009 7:55 PM

#

@Miral: there's only one way to find out...

Stefan Kanthak
9 Feb 2009 2:55 PM
#

@steveg

<http://www.codeplex.com/RegNamespace>

<http://www.regxplor.com/>

And don't forget to see the REALLY old sample code on MSDN too!

Anonymous Coward
9 Feb 2009 11:58 PM
#

It's certainly cool that things like that are possible, after all, that's why ZIP files don't suck anymore, but given that shell namespace extensions are not visible in the actual filesystem, these things unfortunately certainly have their limitations, as is also apparent when you're working with ZIP files.

ceapseAcank
10 Feb 2009 7:59 AM
#

dgdfgds fgf gdsf gssd f gfsd df dsfdgdfgds fgf gdsf gssd f gfsd df dgdfgds fgf gdsf gssd f gfsd df

Stefan Kanthak
10 Feb 2009 1:12 PM
#

@Anonymous Coward

Nobody keeps you from implementing ZIP or registry access as extension to the Windows filesystem: junctions exist!

Anonymous Coward
10 Feb 2009 7:08 PM
#

>junctions exist

Then why didn't Microsoft implement ZIP folders that way? I suspect that junctions don't do what you say they do. Certainly all information that turns up in a web search suggests that what you say is impossible. If you didn't pull it out of your arse, please point me to some example code demonstrating the possibility.

Stefan Kanthak
11 Feb 2009 10:27 AM
#

Forgive me that my suggestion is above your mental capabilities! Junctions are an instantiation of the more general "reparse points". If you dont understand their possibilities then choose another way to implement the desired functionality, for example file system filter drivers. THINK!

Stefan Kanthak
11 Feb 2009 10:28 AM
#

Forgive me that my suggestion goes beyond your mental capabilities! Junctions are an instantiation of the more general "reparse points". If you dont understand their possibilities then choose another way to implement the desired functionality, for example file system filter drivers. THINK!

eth0
11 Feb 2009 4:25 PM
#

> On FAT32 however you only have the DOS attributes, no ACLs.

So: Registry and NTFS permissions are not AT ALL identical.

They're both ACLs, that's it.

Just like you are a mammal and a possum is.

But you're no possum - at least I hope not for your sake ;)

Anonymous Coward
11 Feb 2009 5:24 PM
#

>THINK

In the meantime I've read up on junction points, and they can't be used the way you say they can. In other words, I don't see much evidence that you're doing what you tell me to do: thinking. If you aren't lying, provide links and example code.

>not AT ALL identical

They are sufficiently homologous and a common API could certainly be written. The old API's would simply show the best fit of what's actually there in terms of the new API.