

# Dandelions, VCR Clocks, and Last Logon Times: These are a Few of Our Least Favorite Things

## But At Least Now We Can Do Something About Calculating the Last Logon Time

Sometimes the things that should be so easy turn out to be incredibly difficult. For example, have you ever tried to get rid of dandelions? (One of the Scripting Guys once found a dandelion growing on the *roof of his house*.) How about getting your VCR clock to stop flashing **12:00** over and over again? (It's been estimated that as many as 25% of all the VCRs in use are even now flashing **12:00**.) And as one Scripting Guy discovered, it's actually easier to build a new Volkswagen Passat from scratch than it is to change the headlight.

The same thing has always been true of a seemingly-innocuous Active Directory task: determining the last time a user logged on to the domain. Determining the last logon time ought to be pretty easy; after all, Active Directory includes an attribute – **lastLogon** – that tells you the last time a user or computer logged on. How hard could it be simply to retrieve and report the value of that one attribute?

Well, as it turns out, surprisingly hard. For one thing, there's some difficulty regarding the way the last logon date and time are stored in Active Directory. But that can be solved with some clever coding and mathematics. A much bigger stumbling block is this: the lastLogon attribute is not replicated from one domain controller to another. Suppose a new user logs on to domain controller A. You now write a script that requests the last logon time for our new user, and the script happens to connect to domain controller B. Oddly enough, the script will tell you that the user has *never* logged on, even though you know for a fact that the user is logged on right now.



## Hey, What's Going on Here?

So why is Active Directory lying to you? Well, it's not really lying, it's just that domain controller B doesn't know that the user logged on to domain controller A. Because the lastLogon attribute is not replicated throughout the domain, if our new user has never logged on to domain controller B then domain controller B will have no knowledge of the user's last logon time. In fact, to determine the last logon time for a user you have to retrieve the lastLogon attribute from every domain controller in the domain and then compare all those values to determine the true last logon time.

Yuck.

The Scripting Guys, who are asked several times a day how to determine the last time a user logged on to a domain (Mom, please, quit asking: you *don't* want to know!), have just one thing to say: thank goodness for Windows Server 2003. The lastLogon attribute is still present in the Active Directory schema for Windows 2003 and this attribute still isn't replicated from one domain controller to another. But that's OK, because there's a brand-new attribute in the schema: **lastLogonTimestamp**. This attribute also keeps track of the last time a user logged on to the domain, but – wonder of wonders – this new attribute *is* replicated from one domain controller to another. Want to know the last time a user logged on? Then just write a script and connect to *any* domain controller; the value will be the same on each one.

It *is* like a miracle, isn't it?

In this article we'll show you a script that can return the last logon time for a user in a Windows Server 2003 domain. Before we do this, however, bear in mind that we still face a few complicating factors. For one thing, it's important to note that the last logon timestamp will typically *not* report the user's true last logon time. Why not? Well, imagine a group of users who log on and log off several times a day. Each time one of these users logs on that information would have to be replicated throughout the entire domain. That could generate a large amount of replication traffic, and for little purpose: typically you care about only the so-called "stale" accounts, users who haven't logged on in the last few weeks. For the most part, you don't need an up-to-the-minute report on each user's last logon status. Because of that, the lastLogonTimestamp is replicated only once every 14 days. This helps limit replication traffic, although it also means that the lastLogonTimestamp for any given user could be off by as much as 14 days.

**Note.** If that actually *is* a problem then you can simply connect to each domain controller and retrieve the value of the lastLogon attribute for the user. The lastLogon attribute isn't replicated throughout the domain, but it *is* updated on the authenticating domain controller each time a user logs on. But if you're trying to answer a question like "Do we have any users who haven't logged on in the past two weeks?" then the lastLogonTimestamp will more than suffice.

Believe it or not, the fact that the lastLogonTimestamp isn't 100% accurate actually makes our script a little *easier* to write. As you'll see, we have to go through some mathematical gyrations in order to convert the lastLogonTimestamp to a date-time value we can make sense of. If we had to adjust for possible time zone differences between our computer and the domain controllers that would make our math even *more* complicated. But we don't really have to worry about that. After all, we already know – in advance – that our last logon time could be off by as much as 14 days. Based on that, there's no reason to worry about a few hours' worth of time zone differences.

The other complicating factor, as we hinted at, is this: the lastLogonTimestamp is stored as a 64-bit integer. When you query the lastLogonTimestamp you don't get back a date-time like May 15, 2005 8:05 AM. Instead, you get back the number of 100-nanosecond intervals that passed between January 1, 1601 and the time the user last logged on. (Come on: we're not clever enough to make up something like *that!*) Consequently most of our code will be involved in taking that weird 64-bit integer value and converting it to a date and time.

**Note.** In case you're wondering, a number of years ago the American National Standards Institute (ANSI) adopted a system of counting days; this system began with December 31, 1600 as Day 0. In turn, that made January 1, 1601 the first "official" day in history, with all subsequent dates and times being based on the number of nanoseconds elapsed since the 0 hour on January 1, 1601. (That day was a Monday, by the way.) These so-called ANSI decimal dates were originally designed for use with the COBOL programming language and have continued to be used by Windows and other operating systems.

Incidentally, did we mention the fact that VBScript can't actually handle the 64-bit integer returned by lastLogonTimestamp? Well, we should have: 64-bit integers are not supported in VBScript. But at least there is a workaround for this: ADSI's IADsLargeInteger interface can break this into a pair of 32-bit integers for us, and VBScript can handle those two integers just fine. That means we can still work with the lastLogonTimestamp attribute: we just need to use an additional step, one in which we add the two 32-bit integers to get a single value that VBScript is comfortable with.



[↑ Top of page](#)

## Don't Worry: Everything's Going to Be Just Fine

Still with us? (We were afraid we'd scared everyone off.) Despite all those dire warnings the script that returns the last logon time for the user really isn't all that bad. Here, see for yourself:

```
Set objUser = GetObject("LDAP://cn=Ken Myer, ou=Finance, dc=fabrikam, dc=com")
Set objLastLogon = objUser.Get("lastLogonTimestamp")

intLastLogonTime = objLastLogon.HighPart * (2^32) + objLastLogon.LowPart
intLastLogonTime = intLastLogonTime / (60 * 1000000)
intLastLogonTime = intLastLogonTime / 1440

wscript.Echo "Last logon time: " & intLastLogonTime + #1/1/1601#
```

The script starts off easy enough: we simply bind to the user account in Active Directory and then use the **Get** method to retrieve the lastLogonTimestamp, storing that value in an IADsLargeInteger object with the object reference objLastLogon.

**Note.** One of the nice things about ADSI is that, in general, we don't have to tell it which interface to use; you might notice

that we never create an instance of the `IADsLargeInteger` object. Instead ADSI typically figures that sort of thing out for itself. You can see we also never explicitly tell it that we're working with a user object. ADSI is smart enough to determine that without any help.

This is where things get a tad bit hairy. The `IADsLargeInteger` object has two properties: **HighPart**, which stores the upper 32 bits of our 64-bit integer; and **LowPart**, which stores the lower 32 bits of the integer. To combine those into a single value we use this line of code:

```
intLastLogonTime = objLastLogon.HighPart * (2^32) + objLastLogon.LowPart
```

Don't worry too much about the math; we're just taking the `HighPart` times two to the 32<sup>nd</sup> power, and then adding the `LowPart`. Unless you're a glutton for mathematical punishment just take it on faith that this formula is correct.

Believe it or not, that one line of code actually gives us the last logon time for the user; the only problem is that the last logon time comes back as the number of 100-nanosecond intervals that expired between January 1, 1601 and the user's last logon. That's going to be a value similar to this:

```
1.27588712492538E+17
```

How...nice...

What we need to do, obviously, is convert that value to something a little easier to deal with. As everyone knows, there are 1,000,000,000 nanoseconds in a second; therefore, there are 10,000,000 100-nanosecond intervals in a single second (10,000,000 x 100 = 1,000,000,000). We need to know that because – as we noted earlier – the `lastLogonTimestamp` measures time in 100-nanosecond intervals. If we carry out the math one step further that also means there are 600,000,000 of these 100-nanosecond intervals in each minute.

Don't worry about it; we'll wait until your head stops spinning. Better? OK, let's proceed. Where are we going with this? Well, armed with this knowledge we can now use *this* line of code to tell us how many *minutes* elapsed between January 1, 1601 and the time the user last logged on (note that we're taking 60 seconds times the 10,000,000 100-nanosecond intervals in each of those seconds):

```
intLastLogonTime = intLastLogonTime / (60 * 1000000)
```

And because there are 1,440 minutes in every 24-hour day, this line of code tells us how many days have elapsed:

```
intLastLogonTime = intLastLogonTime / 1440
```

And, yes, we could have used just one equation rather than two. We just thought that breaking it into two pieces would make it a little easier for you to follow.

As soon as we know the number of days that passed we can add that number to the date January 1, 1601 and generate a date-time value that makes some sense. (For example, suppose we determined 3 days had passed. We'd add 3 to January 1, 1601 and come up with a last logon time of January 4, 1601.) Here's the code that does this addition:

```
Wscript.Echo "Last logon time: " & intLastLogonTime + #1/1/1601#
```

And here's an example of the output we get:

Last logon time: 4/25/2005 2:54:09 PM

Definitely crazy. But, on the bright side, you *can* connect to any domain controller in the domain and retrieve this information. Like we said, in Windows 2000 you still have to do all this high-falutin' mathematics; on top of that, though, you also have to connect to each and every domain controller, retrieve the value of the lastLogon attribute, and then compare all those values to determine the last time the user logged on. Compared to that, lastLogonTimestamp represents a huge leap forward.

Now if Microsoft could just do something about those flashing **12:00s** we'd be in business.