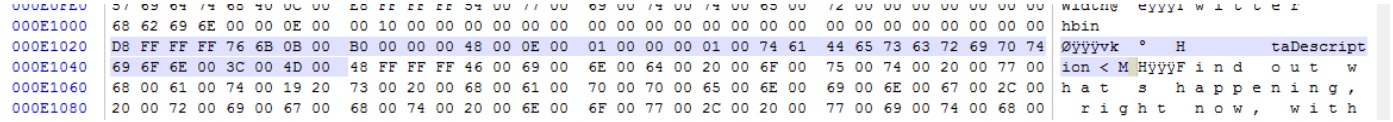
Registry hive basics part 3: VK records

In part 2 [http://binaryforay.blogspot.com/2015/01/registry-hive-basics-part-2-nk-records.html] , we covered NK records. Next up is an NK record's best friend, the VK record!

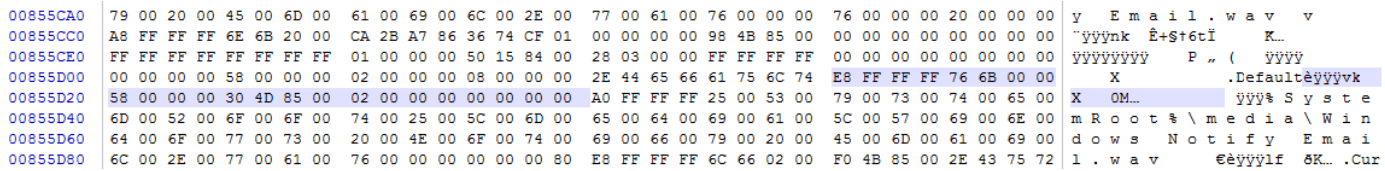
VK record

A VK (value key) record contains the information necessary to define a value that is associated with an NK record. An example of a VK record as it exists on disk is shown below.



[http://4.bp.blogspot.com/-jounXjv3DEI/VMkbK1pXosI/AAAAAAAAAMg/EG5QmMDynT4/s1600/VKWithName.png]

Here is another example of a VK record.



[http://4.bp.blogspot.com/-hmdVyHYvZc8/VMkb_ATQHmI/AAAAAAAAAMo/S5iXCbeQm0E/s1600/vkWithoutName.png]

Can you spot the difference from the first one? If not, don't worry! Keep reading and all will become clear.

In total, there are up to 10 pieces of information in a VK record. Almost all of them are important. These include:

- Size
- Signature
- Name length
- Data length
- Data
- Type
- Flags
- Name

Notice there is no last write time for VK records. Because of this, it is not possible to tell when a given key has been updated (unless perhaps there was only one value in a given key).

Size

The size starts at offset 0x00 and is stored as a signed 32-bit integer. In the first screenshot above, the hex value is 0xD8FFFFFF (stored in little endian format). Converting the value to decimal results in -40 (negative 40).

The size works the same way in VK records as it did in NK records, so our actual record size is 40 bytes.

Signature

The signature for a VK record is found at offset 0x04. The signature for a VK record is the ASCII string "vk" which is 0x766B in hex.

Name length

The Name length starts at offset 0x06 and is stored as a 16-but unsigned integer (2 bytes long).

In the first example, the value is 0x0B00, or 11 in decimal. In the second example, the value is 0x0000, or 0 in decimal.

How can a value have a name with zero length? How can we talk about a value with no name?

Have you ever wondered why so many values you view in the Registry with regedit.exe have a name of **(Default)**? That is because when regedit sees a name length of zero , it uses (Default) for the name. This is done because it is much more efficient to not store the same string over and over and over for so many values in a typical Registry hive.

This is the difference in the screenshots I was alluding to above.

When name length is zero, the Name is not present in the VK record.

When the length is greater than zero, the Name is present in the VK record. In the first example above, the length is 11. While we haven't talked about where the Name lives, if you look at the first example above you may be able to spot it. The value name in the first example is 'Description' which is 11 characters long.

Data length

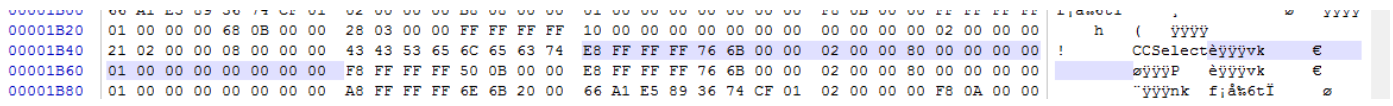
The data length starts at offset 0x08 and is stored as a 32-but unsigned integer.

In the first example, the data length is 0xB0000000, or 176 in decimal. In the second example, the data length is 0x58000000, or 88 in decimal.

These lengths seem strange though. How can a VK record that is 40 bytes in size hold a value that is 176 bytes? This mystery will be solved when we get to the next section on Data.

There is one other critical thing to understand when looking at data length and that is the concept of *resident data*. Much like MFT records, the Registry can store data that is small enough directly in the VK record itself.

To make this easier to understand, we have to look at yet another VK record.



[http://2.bp.blogspot.com/-nb0BAUWt7i4/VMkjKRxAa5I/AAAAAAAAAM4/QyYgQtc7UmE/s1600/vkResident.png]

In this example, the data length is 0x02000080, or 2,147,483,650 in decimal. Now either this is the biggest value in the history of the Registry or there is some trickery going on.

The key to determining whether or not data is resident to a VK record is to convert the Data length to binary form and inspect the high order bit. Recall that Data length is a 32-bit number, so when we convert to binary form, we

want to make sure we pad the left side with zeros until we get to an overall length of 32 characters.

If we take our first two sizes and convert them to binary, we get the following:

176 --> 00000000000000000000000010110000
88 --> 00000000000000000000000001011000

But watch what happens when we do the same with the last example:

2,147,483,650 --> 10000000000000000000000000000010

Do you see the magic? In the last example, the high bit is one and when this is set, the data is resident. Once we know the data is resident, we can subtract 0x80000000 from the data length (0x80000002 - 0x80000000 == 0x02) to get to the actual Data length.

Based on this, can you take a guess as to the maximum number of bytes a VK record can hold when the data is resident?

Keep reading to find out!

Data

Data starts at offset 0x0C and is 4 bytes long.

In the first example at the top, Data is 0x48000E00. In the second it is 0x304D8500. In our example discussing resident data, it is 0x00000000.

An important thing to understand when it comes to Data is that at this point all we can tell is the bytes that make up the VK record's value. In other words, all we will have once we figure out the Data is a sequence of bytes. Until we get to the Type discussion we will not know how to interpret the Data into something more meaningful. We will talk about Type in the next section.

There are two cases we have to look at when it comes to Data: resident and non-resident.

Since we just concluded a riveting discussion on how to determine whether or not data is resident, let's talk about the resident case first.

When data is resident to a VK record, this means the Data bytes (0x00000000 as we saw in our resident example above) **is the data** for the VK record. Now that sounds strange I know, but when we get to the non-resident discussion in a moment it will make more sense.

Now that you know this, the answer to my last question should be a bit more clear. Since Data can only be four bytes long, the maximum number of bytes that can be stored for resident data would be four. In our example above, the resident data had a length of two, but there are other possibilities which changes which bytes are used:

- A data size of 4 uses all 4 bytes of the data offset
- A data size of 2 uses the last 2 bytes of the data offset (on a little-endian system)
- A data size of 1 uses the last byte (on a little-endian system)
- A data size of 0 represents that the value is not set (or NULL)

Since our resident data has a length of two, the Data bytes for that record are 0x0000.

Resident data is typically used to store data related to 32-bit (or smaller) numbers and possibly very small strings.

Now that the resident case is out of the way, we can talk about the non-resident case.

In our first two examples, Data has values of 0x48000E00 and 0x304D8500, but recall our data length was 176 and 88 respectfully.

How can the data length be 176 when we only have four bytes to work with?

The answer is that when data is non-resident, the Data bytes are a **relative offset** to where the actual data lives.

Before we proceed, it would be prudent to talk about another common structure found in Registry hives, a data node.

A data node is a very simple structure that looks like this:

- Size: A 32-bit unsigned integer starting at offset 0x00.
- Data: Starting at offset 0x04.
- Padding: Optional to ensure Size is a multiple of 8 bytes

A data node is simply used to store runs of bytes used by other, higher level, record types. There is at least one data node that has a known signature ("db" lists) which we will cover when we talk about all of the different list formats used in the Registry.

Before we continue, what I would like you to take a moment to think about is this: What is the relationship, if any, between the Size of a data node and the Data length in a VK record? (Cue Jeopardy music...) Your answer does not need to be in the format of a question.

Now that we know what a data node is, how to we get to the one that corresponds to the Data for our VK record? As mentioned above, the bytes stored in Data are the relative offset to the where the actual data lives.

To get to the **absolute offset**, we need to add 0x1000, or 4096, bytes to our relative offset. If we do that we get 0x0E0048 + 0x1000 == E1048, so our data node will be found at absolute offset 0xE1048. The data node is shown below.

```
000E1020  D8 FF FF FF 76 6B 0B 00  B0 00 00 00 48 00 0E 00  01 00 00 00 01 00 74 61  44 65 73 63 72 69 70 74  Øÿÿÿvk  °   H        taDescript
000E1040  69 6F 6E 00 3C 00 4D 00  48 FF FF FF 46 00 69 00  6E 00 64 00 20 00 6F 00  75 00 74 00 20 00 77 00  ion < M HÿÿÿF i n d   o u t   w
000E1060  68 00 61 00 74 00 19 20  73 00 20 00 68 00 61 00  70 00 70 00 65 00 6E 00  69 00 6E 00 67 00 2C 00  h a t   s   h a p p e n i n g ,
000E1080  20 00 72 00 69 00 67 00  68 00 74 00 20 00 6E 00  6F 00 77 00 2C 00 20 00  77 00 69 00 74 00 68 00   r i g h t   n o w ,   w i t h
000E10A0  20 00 74 00 68 00 65 00  20 00 70 00 65 00 6F 00  70 00 6C 00 65 00 20 00  61 00 6E 00 64 00 20 00   t h e   p e o p l e   a n d
000E10C0  6F 00 72 00 67 00 61 00  6E 00 69 00 7A 00 61 00  74 00 69 00 6F 00 6E 00  73 00 20 00 79 00 6F 00  o r g a n i z a t i o n s   y o
000E10E0  75 00 20 00 63 00 61 00  72 00 65 00 20 00 61 00  62 00 6F 00 75 00 74 00  2E 00 00 00 65 00 50 00  u   c a r e   a b o u t .   e P
000E1100  E0 FF FF FF 76 6B 04 00  10 00 00 00 E8 FF 0D 00  01 00 00 00 01 00 0C 00  4E 61 6D 65 57 4C 5F 4C  àÿÿÿvk      èÿ          NameWL_L
000E1120  D8 FF FF FF 76 6B 09 00  12 00 00 00 48 01 0E 00  01 00 00 00 01 00 0C 00  53 65 72 76 69 63 65 49  Øÿÿÿvk      H           ServiceI
```

[http://3.bp.blogspot.com/-BB2AkkQR-30/VMkrgzLfy1I/AAAAAAAAANI/MI2zbOoYR9w/s1600/vkDataRecord.png]

In the case we are discussing, the data node just happens to follow the VK record we have been looking at. This isn't always the case, so just follow the offset and you will get to your data node.

As is the case with most Registry structures, the first four bytes is the length of the node. In this case its 0x48FFFFFF, or -184. This includes the size of the node, so if we account for the size, the remaining part of the node, 180 bytes, is where we can expect to find our Data.

As with the case with resident data, at this point all we know how to do is gather the bytes that make up the value for this VK record. Recall from above the data length was 176 bytes. If we start at offset 0x04 of our data node and take 176 bytes, we get this:

46 00 69 00 6E 00 64 00 20 00 6F 00 75 00 74 00 20 00 77 00 68 00 61 00 74 00 19 20 73 00 20 00 68 00 61 00 70 00 70 00 65 00 6E 00 69 00 6E 00 67 00 2C 00 20 00 72 00 69 00 67 00 68 00 74 00 20 00 6E 00 6F 00 77 00 2C 00 20 00 77 00 69 00 74 00 68 00 20 00 74 00 68 00 65 00 20 00 70 00 65 00 6F 00 70 00 6C 00 65 00 20 00 61 00 6E 00 64 00 20 00 6F 00 72 00 67 00 61 00 6E 00 69 00 7A 00 61 00 74 00 69 00 6F 00 6E 00 73 00 20 00 79 00 6F 00 75 00 20 00 63 00 61 00 72 00 65 00 20 00 61 00 62 00 6F 00 75 00 74 00 2E 00 00 00

Now that we have our bytes, we can interpret it appropriately after we learn about Types. Some of you may have an idea as to what kind of Data the above represents already (Hint: NULL terminated Unicode string).

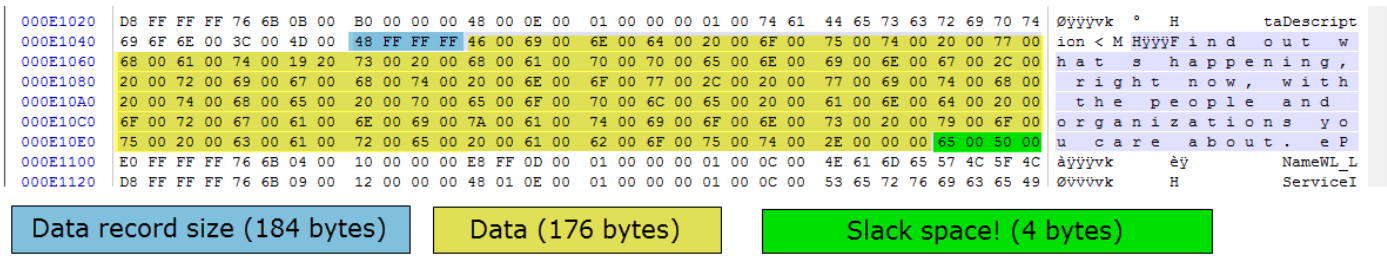
Now is the time for you to recall what your answer was as it relates to the Size of the data node and the Data length of the VK record. Did you figure it out?

As we saw above, after accounting for the data node size we are left with 180 bytes. Our Data has a Data length of 176 bytes, which leaves us four bytes. So what are these four bytes?

Value slack space!

The takeaway here is that it is critical to read in the entire data node vs just jumping to an offset and reading Data length bytes. If we did that, we would miss being able to look at Value slack.

So now we can wrap up data nodes with a nice visual:



[http://1.bp.blogspot.com/-ni2_EOu_XdY/VMksqtqA9GI/AAAAAAAAANU/luKNThE02BQ/s1600/vkDataRecordBreakDown.png]

There is one other technique used to get to data, the big data case. Since this post is already getting long, the big data case will be deferred until we talk about all of the different lists.

Remember at this point we have no idea HOW to interpret what we found for our Data value. We need the VK record Type for that.

Type

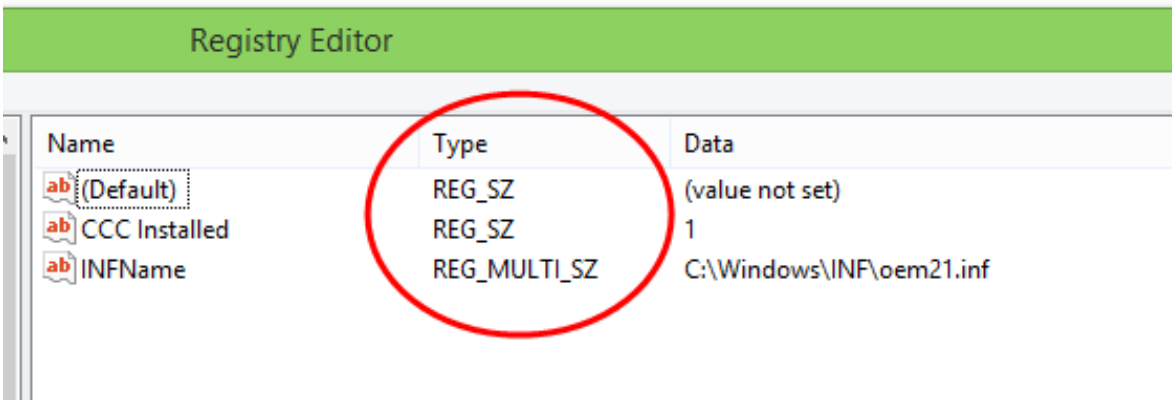
The Type is found at offset 0x10 and is stored as a 32-bit unsigned integer. In our first example, the Type is 0x01000000, or 1 in decimal. There are over a dozen standard types used in the Registry, as seen below:

- RegNone = 0x0000: No type (the stored value, if any)
- RegSz = 0x0001: A string value, normally stored and exposed in UTF-16LE

- RegExpandSz = 0x0002: An 'expandable' string value that can contain environment variables, normally stored and exposed in UTF-16LE
- RegBinary = 0x0003: Binary data (any arbitrary data)
- RegDword = 0x0004: A DWORD value, a 32-bit unsigned integer (little-endian)
- RegDwordBigEndian = 0x0005: A DWORD value, a 32-bit unsigned integer (big endian)
- RegLink = 0x0006: A symbolic link (UNICODE) to another Registry key, specifying a root key and the path to the target key
- RegMultiSz = 0x0007: A multi-string value, which is an ordered list of non-empty strings, normally stored and exposed in UTF-16LE, each one terminated by a NUL character
- RegResourceList = 0x0008: A resource list (used by the Plug-n-Play hardware enumeration and configuration)
- RegFullResourceDescription = 0x0009: A resource descriptor (used by the Plug-n-Play hardware enumeration and configuration
- RegResourceRequirementsList = 0x000A: A resource requirements list (used by the Plug-n-Play hardware enumeration and configuration)
- RegQword = 0x000B: A QWORD value, a 64-bit integer (either big- or little-endian, or unspecified)
- RegFileTime = 0x0010: FILETIME data

One thing to notice is that for the RegNone Type, it says the data has no type, not that there is no data.

You have seen these before if you have ever used regedit.exe to view the Registry.



[http://3.bp.blogspot.com/-XnxDql7eaS8/VMlPCouJL3I/AAAAAAAAANs/sUCjUcy0cVg/s1600/RegEditTypes.png]

There are also Types that do not have a value that corresponds to anything in the list above. These are typically seen in the SAM Registry hives and often correspond to part of a users SID (the right most part, 1005 for example).

Once you know the Type, it is now possible to decode the bytes we have in Data to something more meaningful.

In the case of our example above, we had the following bytes:

46 00 69 00 6E 00 64 00 20 00 6F 00 75 00 74 00 20 00 77 00 68 00 61 00 74 00 19 20 73 00 20 00 68 00 61 00 70 00 70 00 65 00 6E 00 69 00 6E 00 67 00 2C 00 20 00 72 00 69 00 67 00 68 00 74 00 20 00 6E 00 6F 00 77 00 2C 00 20 00 77 00 69 00 74 00 68 00 20 00 74 00 68 00 65 00 20 00 70 00 65 00 6F 00 70 00 6C 00 65 00 20 00 61 00 6E 00 64 00 20 00 6F 00 72 00 67 00 61 00 6E 00 69 00 7A 00 61 00 74 00 69 00 6F 00 6E 00 73 00 20 00 79 00 6F 00 75 00 20 00 63 00 61 00 72 00 65 00 20 00 61 00 62 00 6F 00 75 00 74 00 2E 00 00 00

Since our Type == 0x1, we now know this Data should be interpreted as a Unicode string. Using a converter

[http://rishida.net/tools/conversion/] , we can see these bytes are the string:

Find out what 's happening, right now, with the people and organizations you care about.

At the end of the bytes we can see 00 00 which is used to terminate a Unicode string.

Flags

At offset 0x14, a 16-bit unsigned integer is used to determine whether or not the VK record's Name is stored in ASCII or Unicode.

In the first example, above, the Flags value is 0x0100, or 1 in decimal. If this flag is greater than zero, the name is stored in ASCII. If it is equal to zero, it is stored in Unicode. It is somewhat rare for value names to be stored in Unicode, but it does happen, particularly when dealing with Registry hives on Windows systems that do not use English as the default language. If your Registry parser of choice is unaware of this distinction you will not get what you expect for Value names (or NK record names for that matter).

Name

At offset 0x18, the Name of the VK record begins. After looking at the Flags value, we know how to interpret the VK record's name. We know from earlier that our Name length is 11 bytes, so if we look at 11 bytes starting at offset 0x18, we see:

44 65 73 63 72 69 70 74 69 6F 6E

Since Flags is greater than one, we interpret this as an ASCII string and when we do this, we get:

Description

If you take a look back at the initial hex view of this record you can see the VK record name.

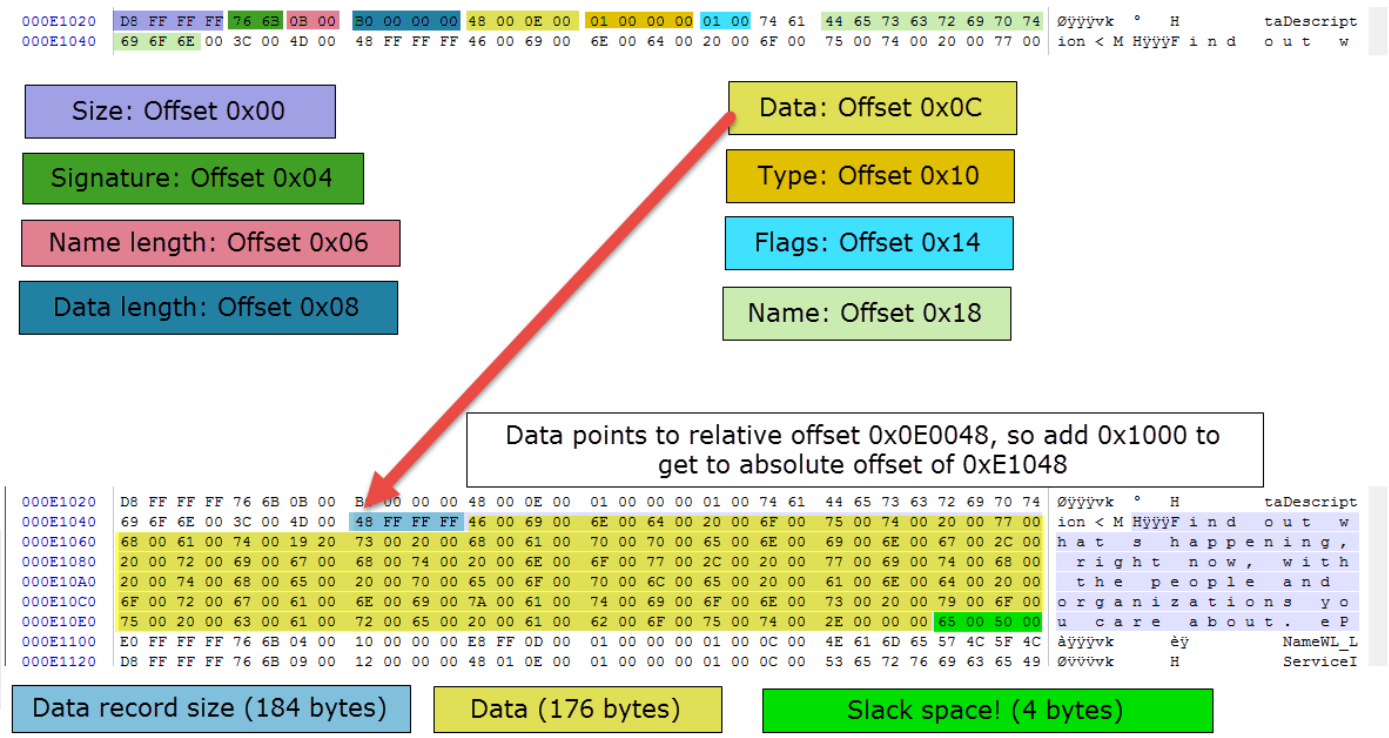
Padding

As with NK records, VK records must be a multiple of 8 bytes long. In the first example, the total length of the VK record is 40 bytes (0x28) long. Since the Name starts at offset 0x18 and is 11 (0x0B) bytes long, we end up using 0x18 + 0x0B == 0x23 bytes (35 in decimal). The remaining five bytes of the record (00 3C 00 4D 00) are padding and in some cases may be part of the previous name used by a VK record.

If we then take these five bytes and add them to the 35 bytes from the VK record, we get 40 bytes, which is precisely what we were expecting.

Putting it all (back) together

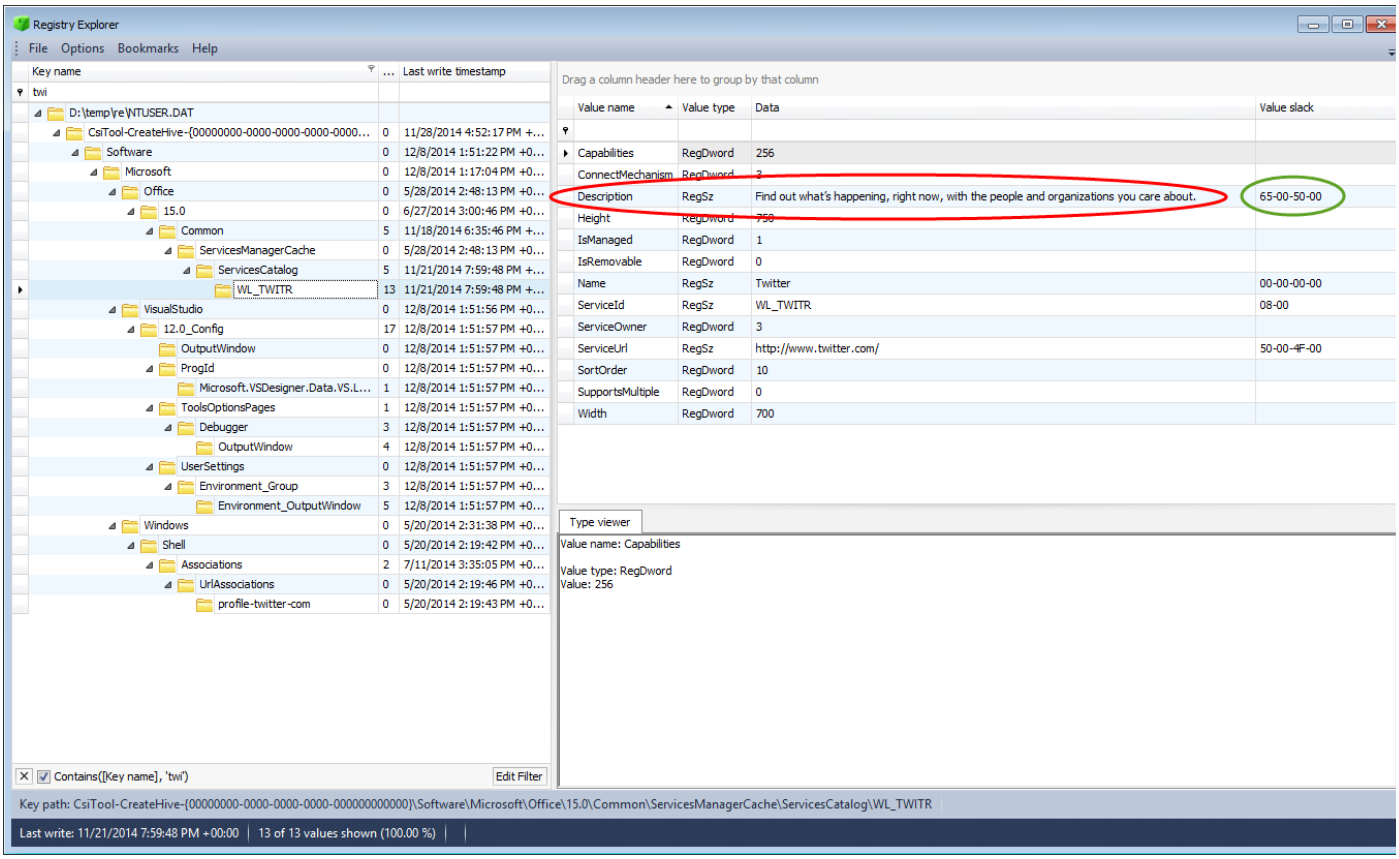
Let's take our first example and break it down into the most important parts.



[http://3.bp.blogspot.com/-vvcgv7u4BRI/VZ58Uzs3O6I/AAAAAAAAAtU/6Eci8OCbRbE/s1600/vkColorBreakdown.png]

For a full breakdown of the VK record layout, review the documents posted earlier.

If we look at this VK record in Registry Explorer, it looks like this:



[http://2.bp.blogspot.com/-4vNhpnISRhg/VMlW_aestnI/AAAAAAAAAN8/5FcFGxQVjp8/s1600/VKRE.png]

Here you can see the key (WL_TWITR) the VK lives in and the values under that key, including its Value name, Value type, Data, and Value slack.

Registry hive basics part 4 will cover the SK [http://binaryforay.blogspot.com/2015/02/registry-hive-basics-part-4-sk-records.html] , or security key record. Part 5 will cover lists.

Posted 29th January by ERZ

Labels: Registry

3 View comments

Ashraf H July 9, 2015 at 6:36 AM

thanks for the great explanation. Minor correction, in break down picture, signature should be at offset 0x04 not 0x02. Are you still planning to post about lists?

Reply

Replies

ERZ August 22, 2015 at 6:37 PM

List post is up

Reply

ERZ July 9, 2015 at 7:52 AM

Thanks for catching that! its updated. Yes i plan on doing lists (and hopefully sooner than later!)

Reply

Enter your comment...

Comment as: ggyy (Google)

Sign out

Publish Preview Notify me