SAL Annotations: Don’t Hate Me Because I’m Beautiful

23 February 2015 **Categories:** GENERAL, WDK **Tags:** code analysis, sal, static analysis by SNoone

OK, well, I don’t think anyone is going argue for the beauty of SAL annotations. However, just because they’re hideous to look at doesn’t mean that you shouldn’t bother with them. In fact, here at OSR one might say we’ve become obsessed with SAL annotations. We find ourselves annotating functions more and more, especially when it comes to functions that we expect someone else to use (i.e. callbacks and APIs).

In case you’ve passed on SAL annotations up to this point, or find yourself stuck limited to simply **_In_** and **_Out_** annotations, I thought I’d break down a function that I just finished annotating and am pretty pleased with. If you’re not familiar AT ALL with SAL annotations, then I suggest you stop now and read the excellent MSDN documentation: Using SAL Annotations to Reduce C/C++ Code Defects.

The function in question is **PolicyGetKeyNewFile**, which is a customer supplied callback defined as part of the Policy DLL interface to our next generation file encryption Solution Kit. The goal of this callback is for the customer to return four pieces of information for a new encrypted file:

1. A customer defined Policy Header Data blob to be stored within the file, along with its size
2. The encryption algorithm to use to encrypt the data for the file
3. Key material for the encryption, along with its size
4. An optional cookie value that will be passed to a separate callback when the last handle to the newly created file is closed (note that this is a seldom used feature for a specific use case)

The callback is provided path information for the newly created file as well as the thread identifier of the thread creating the file. This function is allowed to fail if the necessary information cannot be retrieved and a successful call must return valid values for all of the above (i.e. none of these are optional).

A trivially annotated prototype for the callback is as follows:

```
bool
PolicyGetKeyNewFile(
    _In_ FE_POLICY_PATH_INFORMATION *PolicyPathInfo,
    _In_ DWORD ThreadId,
    _Out_ PVOID *PolHeaderData,
    _Out_ DWORD *PolHeaderDataSize,
    _Out_ LPCWSTR *PolUniqueAlgorithmId,
    _Out_ PVOID *PolKey,
    _Out_ DWORD *PolKeySize,
    _Out_ PVOID *PolCleanupInfo
);
```

The **_In_** and **_Out_** are certainly better than nothing as they make the direction of the parameters clear. *PolicyPathInfo* and *ThreadId* are provided by us when calling the callback, everything else must be provided by the callback itself before returning. However, with the exception of the two input parameters, there is SO much more we can do with SAL to make the intent of these parameters clear that it’s really a waste to not bring this further. Note that the annotations are not simply cosmetic, each of these annotations is used by Visual Studio’s Code Analysis feature to find defects in the usage of the parameters. What’s even better is that we can find defects in both the implementation of the callback as well as in the code that calls the callback. More about that as we look further into this.

The first refinement that we can make is to use **_Outptr_** instead of **_Out_** when expecting a pointer return. While these parameters all have data types that indicate their usage, we can be explicit about this in our SAL and give Code Analysis a bit more information about what we’re doing. An updated revision of the function might be:

```
bool
PolicyGetKeyNewFile(
    _In_ FE_POLICY_PATH_INFORMATION *PolicyPathInfo,
```

What’s New at OSR

Working on NVMe?
So are we! In fact, we've built a driver that's proven to be faster, more reliable, and use less CPU time than other available options. Use it off the shelf, or customize it to your needs.
Read More..

OSR's New File Encryption Solution Framework
Build your encryption product faster, better, and with great perf, starting with our base code.
Read More..

Join the OSR Hints and Tips Email List
Want to receive regular technical hints, tips, and notifications of new tech blog postings from OSR? Join our Hints and Tips list!
To join the list just send a blank email to join-osrhints@lists.osr.com.

Job Opening at OSR
We're hiring in engineering!
Read More..

Upcoming Seminars

Developing File Systems for Windows
12-15 May 2015, Boston/Waltham, MA
Course Outline..

Windows Internals and Software Driver Development
18-22 May 2015, Dulles/Sterling, VA
Course Outline..

Writing WDF Drivers: Core Concepts
8-12 June 2015, Boston/Waltham, MA
Course Outline..

More Seminars

OSR’s NTDEV List

- Storport PDO I/O thread performance question (4 msgs) 2 April 2015
- class implementation in driver development (5 msgs) 2 April 2015
- Windows XP debug COM port still visible after botting in Debug mode (7 msgs) 2 April 2015
- =?utf-8?Q?Re:_[...question?= 2 April 2015
- Driver pairs and Driver triples? 2 April 2015

```
        _In_ DWORD ThreadId,
        _Outptr_ PVOID *PolHeaderData,
        _Out_ DWORD *PolHeaderDataSize,
        _Outptr_ LPCWSTR *PolUniqueAlgorithmId,
        _Outptr_ PVOID *PolKey,
        _Out_ DWORD *PolKeySize,
        _Outptr_ PVOID *PolCleanupInfo
    );
```

But we can bring this much further. The pointer returned in *PolHeaderData* is sized by the DWORD returned in *PolHeaderDataSize*. Same deal with *PolKey* and *PolKeySize*. SAL actually gives us a way to express this by putting some modifiers on our **_Outptr_** annotations, indicating that the resulting pointer is actually a buffer of *Size* bytes. The annotation for this is **_Outptr_result_bytebuffer_(*Size*)**, which we can see in our next update to the function:

```
    bool
    PolicyGetKeyNewFile(
        _In_ FE_POLICY_PATH_INFORMATION *PolicyPathInfo,
        _In_ DWORD ThreadId,
        _Outptr_result_bytebuffer_(*PolHeaderDataSize) PVOID *PolHeaderData,
        _Out_ DWORD *PolHeaderDataSize,
        _Outptr_ LPCWSTR *PolUniqueAlgorithmId,
        _Outptr_result_bytebuffer_(*PolKeySize) PVOID *PolKey,
        _Out_ DWORD *PolKeySize,
        _Outptr_ PVOID *PolCleanupInfo
    );
```

Granted, we're getting a big ugly here, but trust me this is worth it! Code Analysis will now perform bounds checking on both the implementation of this function as well as the caller of this function. If the callback supplies a length that is larger than the buffer allocation, the callback will receive a Code Analysis warning. If the caller tries to read more bytes from the buffer than specified by the length, the caller gets a Code Analysis warning. Win, win!

But, wait, there's more! We don't want or allow the callback to return a length of zero for these buffers, so *PolHeaderDataSize* and *PolKeySize* must be greater than zero. We can actually express this in the SAL using the **_Deref_out_range_** annotation:

```
    bool
    PolicyGetKeyNewFile(
        _In_ FE_POLICY_PATH_INFORMATION *PolicyPathInfo,
        _In_ DWORD ThreadId,
        _Outptr_result_bytebuffer_(*PolHeaderDataSize) PVOID *PolHeaderData,
        _Out_ _Deref_out_range_(>, 0) DWORD *PolHeaderDataSize,
        _Outptr_ LPCWSTR *PolUniqueAlgorithmId,
        _Outptr_result_bytebuffer_(*PolKeySize) PVOID *PolKey,
        _Out_ _Deref_out_range_(>, 0) DWORD *PolKeySize,
        _Outptr_ PVOID *PolCleanupInfo
    );
```

Now the callback must specify a size greater than zero for each of these results or, you guessed it, they'll get a Code Analysis warning.

There are two more refinements we can make to the remaining out parameters. *PolUniqueAlgorithmId* is used to return an LPCWSTR, thus we know it's a constant wide character string. Using SAL, we can refine slightly by indicating that the resulting string pointer must be NULL terminated using the **_Outptr_result_z_** annotation.

Lastly, *PolCleanupInfo* must be set by the callback but it may be set to NULL, thus we can indicate that NULL is a valid return value for this parameter by using the **_Outptr_result_maybenull_** annotation. This makes the (almost complete) annotation:

```
    bool
    PolicyGetKeyNewFile(
        _In_ FE_POLICY_PATH_INFORMATION *PolicyPathInfo,
        _In_ DWORD ThreadId,
        _Outptr_result_bytebuffer_(*PolHeaderDataSize) PVOID *PolHeaderData,
        _Out_ _Deref_out_range_(>, 0) DWORD *PolHeaderDataSize,
        _Outptr_result_z_ LPCWSTR *PolUniqueAlgorithmId,
        _Outptr_result_bytebuffer_(*PolKeySize) PVOID *PolKey,
        _Out_ _Deref_out_range_(>, 0) DWORD *PolKeySize,
        _Outptr_result_maybenull_ PVOID *PolCleanupInfo
    );
```

Now for one final improvement…Clearly our intention is that these output parameters must be specified *when the callback is successful*. On error, we don't care if they are set or not because the caller will not inspect them. Enter the **_Success_** annotation, which does two things:

1. Indicates what return values from this function indicate success
2. Makes any parameter annotated as output optional in the failure case(s). In other words, if the return value of the callback is a failure, there is no need for the callback to set the output of these parameters.

If you're familiar with SAL, you might rightfully ask how the second point above is different from marking the parameters as some variant of **_Out_opt_**. The key is that by annotating the return value, we are saying that these parameters must be set if the function is successful. However, on failure, they can effectively be treated as optional. An **_Out_opt_** is truly optional, meaning that it need not be set for either success of failure, which is not what we want.

In our case, we want to indicate that a return value of **true** means the callback was successful. Thus, we finally have the full prototype of this function in all of its SAL ~~gory~~ glory:

```
bool
_Success_(return == true)
PolicyGetKeyNewFile(
    _In_ FE_POLICY_PATH_INFORMATION *PolicyPathInfo,
    _In_ DWORD ThreadId,
    _Outptr_result_bytebuffer_(*PolHeaderDataSize) PVOID *PolHeaderData,
    _Out_ _Deref_out_range_(>, 0) DWORD *PolHeaderDataSize,
    _Outptr_result_z_ LPCWSTR *PolUniqueAlgorithmId,
    _Outptr_result_bytebuffer_(*PolKeySize) PVOID *PolKey,
    _Out_ _Deref_out_range_(>, 0) DWORD *PolKeySize,
    _Outptr_result_maybenull_ PVOID *PolCleanupInfo
);
```

Sure, it's ugly. Sure, it took several tries before the annotations were *exactly* how I wanted them. However, I've now saved Policy DLL developers from any confusion about what our intentions are for these parameters. Even better, if they enable Code Analysis for every build (which we strongly recommend), they'll get an error at compile time for returning an unexpected value.

If I have managed to intrigue you with the above, I highly recommend at scanning the Annotating Function Parameters and Return Values documentation page. The list here is not comprehensive (for that you will need to refer to **sal.h**), but does a nice job pointing out the major annotations.

I'll also leave you with two closing tips as you dive further into the world of SAL:

1. If you're not sure how to use an annotation, search the Microsoft supplied header files (both WDK and SDK). There are multiple examples of just about everything
2. As you're working through your annotations, you must actually **try the code** to ensure that the annotations are catching what you think they're catching. For example, in the above annotations setting *PolKeySize* to zero should result in a Code Analysis warning. You would be foolish to simply believe that you're properly expressing what it is you mean. The annotations are subtle, weird, sometimes buggy, and overall just not necessarily intuitive. Better to know that you have it right than have a false sense of security.

Share this:

Twitter 9 Facebook 10 LinkedIn 21 More

OSR Open Systems Resources, Inc.

105 Route 101A
Suite 19
Amherst, NH 03031
USA
+1.603.595.6500

Recent Posts

23 March 2015
Windows 10 Preview WDK and HLK Now Available for Download

18 March 2015
Updates to Driver Testing — Not Just for Certification

Latest Tweets

The March/April 2015 issue of The NT Insider is available: http://t.co/Qj06J4RkzU - As always, Twitter hears first! Please RT.
9 days ago

Pre-release of #Win10 WDK & WLK (and update of VS 2015) available for download! MSFT download page is here: http://t.co/Q86caYOD2t
10 days ago

Follow @OSRDrivers