If you say that you don't care about something, you shouldn't be upset that it contains garbage

devblogs.microsoft.com/oldnewthing/20080611-00

June 11, 2008

Raymond Chen

There are many situations where you pass a structure to a function, and the function fills in the structure with information you request. In some cases, the function always fills in the entire structure (example: `GlobalMemoryStatus`). In other cases, you tell the function which bits of information you care about, to save the function the effort of computing something you weren't interested in anyway (example: `TreeView_GetItem`).

In the latter case, if you say that you aren't interested in certain parts of the structure, and then you change your mind and start paying attention to them, don't be surprised if you find that there's nothing interesting there. After all, you said you didn't care.

For example, if you call `TreeView_GetItem` and set the mask to `TVIF_IMAGE | TVIF_PARAM`, this means that you want the function to set the `iImage` and `lParam` members of the `TVITEM` structure and that you don't care about the rest. After the call returns, the values of those two members are defined, since you said that's what you wanted, and the remainder of the output fields are *undefined*. They might contain useful information, they might contain garbage, you're not supposed to care since you said that you didn't.

Why might fields you said you didn't care about still contain information (correct or incorrect)? It might be that the value is so easy to compute that checking whether the value should be set takes more work than actually setting it! In such a case, the function might choose to set the value even if you didn't say that you needed it.

On the other hand, the value might be an artifact of a translation layer: You pass a structure saying, "I'm interested in two out of the four members." The function in turn calls a lower lever function with a different structure, saying, "I'm interested in two out of the five members of this different structure." After the call returns, the middle-man function converts the lower-level structure to the higher-level structure. Sure, it may also "convert" stuff that was never asked for, but you said you weren't interested, so they just get garbage. In other words, the function you're calling might be defined like this:

```
// The pinfo parameter points to this structure
struct FOOINFO {
 DWORD dwInUse;
 DWORD dwAvailable;
 DWORD dwRequested;
 DWORD dwDenied;
};
// The dwMask parameter can be a combination of these values
#define FOOINFO_INUSE     0x0001
#define FOOINFO_AVAILABLE 0x0002
#define FOOINFO_REQUESTED 0x0004
#define FOOINFO_DENIED    0x0008
BOOL GetFooInfo(FOOINFO *pinfo, DWORD dwMask);
```

Now, the `GetFooInfo` function might just be a middle man that talks to another component to do the real work.

```
// lowlevel.h
struct LOWLEVELSTATS {
 DWORD dwUnitSize;
 DWORD dwUnitsInUse;
 DWORD dwUnitsAvailable;
 DWORD dwUnitsRequested;
 DWORD dwUnitsGranted;
 DWORD dwTotalRequests;
};
// The dwMask parameter can be a combination of these values
#define LLSTATS_UNITSIZE  0x0001
#define LLSTATS_INUSE     0x0002
#define LLSTATS_AVAILABLE 0x0004
#define LLSTATS_REQUESTED 0x0008
#define LLSTATS_GRANTED   0x0020
#define LLSTATS_REQUESTS  0x0040
BOOL GetLowLevelStatistics(LOWLEVELSTATS *pstats, DWORD dwMask);
```

The resulting `GetFooInfo` function merely translates the call from the application into a call to the `GetLowLevelStatistics` function:

```
BOOL GetFooInfo(FOOINFO *pinfo, DWORD dwMask)
{
 LOWLEVELSTATS stats;
 DWORD dwLowLevelMask = LLINFO_UNITSIZE;
 if (dwMask & FOOINFO_INUSE)
  dwLowLevelMask |= LLSTATS_INUSE;
 if (dwMask & FOOINFO_AVAILABLE)
  dwLowLevelMask |= LLSTATS_AVAILABLE;
 if (dwMask & FOOINFO_REQUESTED)
  dwLowLevelMask |= LLSTATS_REQUESTED;
 if (dwMask & FOOINFO_DENIED)
  dwLowLevelMask |= LLSTATS_REQUESTED | LLSTATS_GRANTED;
 if (!GetLowLevelStats(&info;stats, dwLowLevelMask))
  return FALSE;
 // Convert the LOWLEVELSTATS into a FOOINFO
 pinfo->dwInUse = stats.dwUnitSize * stats.dwUnitsInUse;
 pinfo->dwAvailable = stats.dwUnitSize * stats.dwUnitsAvailable;
 pinfo->dwRequested = stats.dwUnitSize * stats.dwUnitsRequested;
 pinfo->dwDenied = stats.dwUnitSize *
                  (stats.dwUnitsRequested - stats.dwUnitsGranted);
 return TRUE;
}
```

Notice that if you ask for just `FOOINFO_DENIED`, you still get the `dwRequested` as a side effect, since computing the number of requests that were denied entails obtaining the total number of requests. On the other hand, you also get garbage for `dwInUse` since the call to `GetLowLevelStats` didn't ask for `LLSTATS_INUSE`, but the code that converts the `LOWLEVELSTATS` to a `FOOINFO` doesn't know that and converts the uninitialized garbage. But since you said that you didn't care about the `dwInUse` member, you shouldn't be upset that it contains garbage.

You now know enough to answer this person's question.

(Note of course that I'm assuming we are not returning uninitialized garbage across a security boundary.)

Raymond Chen

Follow