Why can't I have variadic COM methods?

devblogs.microsoft.com/oldnewthing/20150424-00

April 24, 2015

Raymond Chen

COM methods cannot be variadic. Why not?

Answer: Because the marshaler doesn't know when to stop.

Suppose variadic COM methods were possible. And then you wrote this code:

```
interface IVariadic
{
 HRESULT Mystery([in] int code, ...);
};
IVariadic *variadic = something;
uint32_t ipaddr;
HRESULT hr = variadic->Mystery(9, 192, 168, 1, 1, &ipaddr);
```

How would COM know how to marshal this function call? In other words, suppose that `variadic` is a pointer to a proxy that refers to an object in another process. The COM marshaler needs to take all the parameters to `IVariadic::Mystery`, package them up, send them to the other process, then unpack the parameters, and pass them to the implementation. And then when the implementation returns, it needs to take the return value and any output parameters, package them up, send them back to the originating process, where they are unpacked and applied to the original parameters.

Consider, for example,

```
interface IDyadic
{
 HRESULT Enigma([in] int a, [out] int *b);
};
IDyadic *dyadic = something;
int b;
HRESULT hr = dyadic->Enigma(1, &b);
```

If `dyadic` refers to an object in another process, the marshaler does this:

- Allocate a block of memory containing the following information:
  - Information to identify the `dyadic` object in the other process,
  - the integer 1.
- Transmit that block of memory to the other process.

The other process receives the block of memory and does the following:

- Use the information in the memory block to identify the `dyadic` object.
- Extract the parameter `1` from the memory block.
- Allocate a local integer variable, call it `x`.
- Call `dyadic->Enigma(1, &x)`. Let's say that the function stores 42 into `x`, and it returns `E_PENDING`.
- Allocate a block of memory containing the following information:
  - The value `E_PENDING` (the `HRESULT` returned by `dyadic->Enigma`),
  - The integer 42 (the value that `dyadic->Enigma` stored in the local variable `x`).
- Transmit that block of memory to the originating process.

The originating process receives the block of memory and does the following:

- Extracts the `HRESULT` `E_PENDING`.
- Extracts the value 42.
- Stores the value 42 into `b`.
- Returns the value `E_PENDING` to the caller.

Note that in order for the marshaler to do its job, it needs to know every parameter to the method, whether that parameter is an input parameter (which is sent from the originating process to the remote process), an output parameter (which is sent from the remote process to the originating process), and how to send that parameter. In our case, the parameter is just an integer, so sending it is just copying the bits, but in the more general case, the parameter could be a more complicated data structure.

Now let's look at that variadic method again. How is the marshaler supposed to know what to do with the `...`? It doesn't know how many parameters it needs to transfer. It doesn't know what types those parameters are. It doesn't know which ones are input parameters and which ones are output parameters.

In order to know that, it would have to reverse-engineer the implementation of the `IVariadic::Mystery` function and figure out that the first parameter, the number 9, is a code that means that the method takes four 8-bit integers as input and outputs a 32-bit integer.

This is a rather tall order for the client side of the marshaler, since it has to do its work without access to the other process. It would have to use its psychic powers to figure out how to package up the parameters, as well as how to unpack them afterward.

Therefore, COM says, “Sorry, you can’t do that.”

But what you can do is encode the parameters in a form that the marshaler understands. For example, you might use a counted array of `VARIANT`s or a `SAFEARRAY`. The COM folks already did the work to teach the marshaler how to, for example, decode the `vt` member of the `VARIANT` and understand that, “Oh, if the value is `VT_I4`, then the `VARIANT` contains a 32-bit signed integer.”

Bonus chatter: But wait, there is a MIDL attribute called `[vararg]`. You said that COM doesn’t support variadic methods, but there is a MIDL keyword that says variadic right on the tin!

Ah, but that `[varargs]` attribute is just a sleight of hand trick. Bceause when you say `[varargs]`, what you’re saying is, “The last parameter of this method is a `SAFEARRAY` of `VARIANT`s. A scripting language can expose this method to scripts as variadic, but what it actually does is take all the variadic parameters and store them into a `SAFEARRAY`, and then pass the `SAFEARRAY`.”

In other words, it indicates that the last parameter of the method acts like the C# `params` keyword.

Raymond Chen

Follow