How can I implement SAFEARRAY.ToString() without going insane?

devblogs.microsoft.com/oldnewthing/20120921-00

September 21, 2012

Raymond Chen

A colleague needed some help with manipulating `SAFEARRAY`s.

> I have some generic code to execute WMI queries and store the result as strings. Normally, `VariantChangeType(VT_BSTR)` does the work, but `VariantChangeType` doesn't know how to convert arrays (e.g. `VT_ARRAY | VT_INT`). And there doesn't seem to be an easy way to convert the array element-by-element because `SafeArrayGetElement` expects a pointer to an object of the underlying type, so I'd have to write a switch statement for each variant type. Surely there's an easier way?

One suggestion was to use the ATL `CComSafeArray` template, but since it's a template, the underlying type of the array needs to be known at compile time, but we don't know the underlying type until run time, which is exactly the problem.

Let's start with the big switch statement and then do some optimization. All before we start typing, because after all the goal of this exercise is to avoid having to type out the massive switch statement. (Except that I have to actually type it so you have something to read.)

Here's the version we're trying to avoid having to type:

```
HRESULT SafeArrayGetElementAsString(
    SAFEARRAY *psa,
    long *rgIndices,
    LCID lcid, // controls conversion to string
    unsigned short wFlags, // controls conversion to string
    BSTR *pbstrOut)
{
  *pbstrOut = nullptr;
  VARTYPE vt;
  HRESULT hr = SafeArrayGetVartype(psa, &vt);
  if (SUCCEEDED(hr)) {
    switch (vt) {
    case VT_I2:
      {
        SHORT iVal;
        hr = SafeArrayGetElement(psa, rgIndices, &iVal);
        if (SUCCEEDED(hr)) {
          hr = VarBstrFromI2(iVal, lcid, wFlags, pbstrOut);
        }
      }
      break;
    case VT_I4:
      {
        LONG lVal;
        hr = SafeArrayGetElement(psa, rgIndices, &lVal);
        if (SUCCEEDED(hr)) {
          hr = VarBstrFromI4(lVal, lcid, wFlags, pbstrOut);
        }
      }
      break;
    ... etc for another dozen or so cases ...
    ... and then special cases for things that need special handling ...
    case VT_VARIANT:
      {
        VARIANT varVal;
        hr = SafeArrayGetElement(psa, rgIndices, &varVal);
        if (SUCCEEDED(hr)) {
          hr = VariantChangeTypeEx(&varVal, &varVal,
                                   lcid, wFlags, VT_BSTR);
          if (SUCCEEDED(hr)) {
            *pbstrOut = varVal.bstrVal;
          } else {
            VariantClear(&varVal);
          }
        }
      }
      break;
    case VT_UNKNOWN:
    case VT_DISPATCH:
    case VT_BSTR: // other cases where we need to release the object
      ... more special cases ...
    }
```

```
  }
  return hr;
}
```

The first observation is that you can make `VariantChangeType` do the heavy lifting. Just read everything (whatever it is) into a variant, and then let `VariantChangeType` do the string conversion.

```
HRESULT SafeArrayGetElementAsString(
    SAFEARRAY *psa,
    long *rgIndices,
    LCID lcid, // controls conversion to string
    unsigned short wFlags, // controls conversion to string
    BSTR *pbstrOut)
{
  *pbstrOut = nullptr;
  VARTYPE vt;
  HRESULT hr = SafeArrayGetVartype(psa, &vt);
  if (SUCCEEDED(hr)) {
    VARIANT var;
    switch (vt) {
    case VT_I2:
      hr = SafeArrayGetElement(psa, rgIndices, &var.iVal);
      if (SUCCEEDED(hr)) {
        var.vt = vt;
      }
      break;
    case VT_I4:
      hr = SafeArrayGetElement(psa, rgIndices, &var.lVal);
      if (SUCCEEDED(hr)) {
        var.vt = vt;
      }
      break;
    case VT_R4:
      hr = SafeArrayGetElement(psa, rgIndices, &var.fltVal);
      if (SUCCEEDED(hr)) {
        var.vt = vt;
      }
      break;
    ... etc for another dozen or so cases ...
    ... there is just one special case now ...
    case VT_VARIANT:
      hr = SafeArrayGetElement(psa, rgIndices, &var);
      break;
    default:
      // an invalid array base type somehow snuck through
      hr = E_INVALIDARG;
      break;
    }
    if (SUCCEEDED(hr)) {
      hr = VariantChangeTypeEx(&var, &var,
                               lcid, wFlags, VT_BSTR);
      if (SUCCEEDED(hr)) {
        *pbstrOut = var.bstrVal;
      } else {
        VariantClear(&var);
      }
    }
  }
```

```
  return hr;
}
```

We can get rid of the special cases for `VT_UNKNOWN`, `VT_DISPATCH`, `VT_RECORDINFO`, and `VT_BSTR`, since `VariantClear` will do the appropriate cleanup for us.

You can actually stop there, since the compiler will perform the next optimization for us. But since the goal is to save typing, we can perform the optimization manually to save us from having to write out all those `SafeArrayGetElement` calls.

Observe that all the `var.iVal`, `var.lVal`, `var.fltVal`, etc., members are all unioned on top of each other. In other words, the address of all the members is the same. We can therefore merge all the cases together. (As noted, this is something the compiler will already do, so the goal here is not to create more efficient code but just to reduce typing.)

```
HRESULT SafeArrayGetElementAsString(
    SAFEARRAY *psa,
    long *rgIndices,
    LCID lcid, // controls conversion to string
    unsigned short wFlags, // controls conversion to string
    BSTR *pbstrOut)
{
  *pbstrOut = nullptr;
  VARTYPE vt;
  HRESULT hr = SafeArrayGetVartype(psa, &vt);
  if (SUCCEEDED(hr)) {
    VARIANT var;
    switch (vt) {
    case VT_I2:
    case VT_I4:
    case VT_R4:
    case ... etc ...:
      // All of the above cases store their data in the same place
      hr = SafeArrayGetElement(psa, rgIndices, &var.iVal);
      if (SUCCEEDED(hr)) {
        var.vt = vt;
      }
      break;
    case VT_DECIMAL:
      // Decimals are stored in a funny place.
      hr = SafeArrayGetElement(psa, rgIndices, &var.decVal);
      if (SUCCEEDED(hr)) {
        var.vt = vt;
      }
      break;
    case VT_VARIANT:
      // Variants too, because it obvious isn't a member of itself.
      hr = SafeArrayGetElement(psa, rgIndices, &var);
      break;
    default:
      // an invalid array base type somehow snuck through
      hr = E_INVALIDARG;
      break;
    }
    if (SUCCEEDED(hr)) {
      hr = VariantChangeTypeEx(&var, &var,
                               lcid, wFlags, VT_BSTR);
      if (SUCCEEDED(hr)) {
        *pbstrOut = var.bstrVal;
      } else {
        VariantClear(&var);
      }
    }
  }
  return hr;
}
```

And then you can generalize this function so it returns a `VARIANT`, so that it becomes the caller's responsibility to do the `VariantChangeType(VT_BSTR)`. This also allows the caller to figure out how to deal with things like `VT_UNKNOWN`, which `VariantChangeType` doesn't know how to handle. (Perhaps it should be converted to the string `"[object]"`.) Or maybe the caller might want to use this function to convert all `SAFEARRAY`s to `VT_ARRAY | VT_FIXEDBASETYPE`.

```
HRESULT SafeArrayGetElementAsVariant(
    SAFEARRAY *psa,
    long *rgIndices,
    VARIANT *pvarOut)
{
  VariantInit(pvarOut);
  VARTYPE vt;
  HRESULT hr = SafeArrayGetVartype(psa, &vt);
  if (SUCCEEDED(hr)) {
    switch (vt) {
    case VT_I2:
    case VT_I4:
    case VT_R4:
    case ...:
      hr = SafeArrayGetElement(psa, rgIndices, &pvarOut->iVal);
      if (SUCCEEDED(hr)) {
        pvarOut->vt = vt;
      }
      break;
    case VT_DECIMAL:
      // Decimals are stored in a funny place.
      hr = SafeArrayGetElement(psa, rgIndices, &pvarOut->decVal);
      if (SUCCEEDED(hr)) {
        pvarOut->vt = vt;
      }
      break;
    case VT_VARIANT:
      // Variants too, because it obvious isn't a member of itself.
      hr = SafeArrayGetElement(psa, rgIndices, pvarOut);
      break;
    default:
      // an invalid array base type somehow snuck through
      hr = E_INVALIDARG;
      break;
    }
  }
  return hr;
}
```

Exercise: Since `decVal` is unioned against the `tagVARIANT`, can we also collapse the `VT_DECIMAL` and `VT_VARIANT` cases together?

Exercise: Why is the final typing-saver (collapsing the case statements) valid? Don't we have to worry about the possibility that the `VARIANT` type may change in the future?

Exercise: What defensive actions could be taken to protect against that possibility raised by the previous exercise?

Raymond Chen

Follow