Subject: Re: Value: why not float support?
Posted by rylek on Mon, 20 Dec 2010 22:08:52 GMT
View Forum Message <> Reply to Message

Hi there!

I'm afraid this is a difficult nut to crack. At first glance it seems that by adding the support of a new type (like float) to Value we are just enriching the U++ environment and not losing anything. Yet in reality I'm afraid we are losing some of Value's clarity. I, for instance, on numerous occasions make a switch over a Value's GetType(). Now, for instance, when we agreed some time ago to add BOOL_V and INT64_V, I had to manually adjust them to be able to accept a new value subtype. Some code broke in a very tricky way back then.

All right, my switches might not be the cleanest programming technique; at least I should finally ask Mirek to agree to add a series of inline functions to Core/Value.h to check Value types for these convertible groups of datatypes; like

```
inline bool IsNumberType(int value_type)
{ return value_type == BOOL_V || value_type == INT_V
|| value_type == INT64_V || value_type == DOUBLE_V; }
```

- equivalents of .IsNumber() etc. Value member functions, just operating on the type constants; but imagine what the above is going to look like when, after FLOAT_V, we continue to add LONG_DOUBLE_V and [U]INT8/16/32/64_V.

I myself, when working with ActiveX, for instance, sometimes find myself longing for better type distinction in Value in order to be able to provide a better mapping between Value and VARIANT. But then I ask myself: do I want U++ to become such pile of mess as the COM and Value such monster as VARIANT?

And yet, it's evident that on numerous previous occassions we didn't adhere to the position we now hold. We have already extended the numeric type set from the original INT_V / DOUBLE_V pair (not mentioning that even INT_V is in fact superfluous) to INT64_V and BOOL_V (see? we didn't upgrade INT_V to 64-bit, we added a different value type). We have STRING_V and WSTRING_V (here there is some justification because conversion of long binary blocks transferred through Strings, which was always seen as a sound option under U++, is time and memory consuming and potentially even lossy), DATE_V and TIME_V.

This whole discussion would be much easier if the value type was two-dimensional; the above type families (reflected in the member functions IsNumber(), IsString() and IsDateTime()) could then represent a 'principal' value type which would have a 'biggest' or 'most general' representant (double, WString and Time in the above case) and a (perhaps extensible) family of derived subtypes which would be able to convert themselves to and from the type family representant.

But it's not and, as I see it, it would cost all of us many a hair to seamlessly rework it like this. Even so, you still have situations like serializing Values where it's extremely unpleasant to have the type family growing all the time.

Regards

Tomas