Subject: HASH with case insensitive keys versus a new DICT class
Posted by chris_torrence@NOSPAM on Tue, 11 Dec 2012 20:14:56 GMT
View Forum Message <> Reply to Message

Hi all,

I've been toying around with adding support for the "." operator to HASH. So, for example, you could do the following:

```
h = HASH()
h.field1 = "my data"  ; adds the key "FIELD1" with data "my data"
print, h.field1
```

So, in essence, this would be a "dynamic" structure.

The only problem with this approach is that the keys become all uppercase, because of the case insensitivity of IDL variables. Also, if you add keys during initialization (or using the square brackets), then they would need to be valid IDL variable names, otherwise you couldn't access them using the ".". So no spaces or special characters.

I see three possible solutions:

1. Add a keyword to HASH() that forces it into "valid IDL variable" mode. Keys can only be strings. The keys could all be stored as the user provided them, but internally the actual hash would be done with uppercase versions of the keys. The Hash could throw errors if the key wasn't a valid IDL variable name.
Advantage: still uses the HASH interface, the case of keys can be preserved & returned to the user
Disadvantage: confusing - you could have 2 hashes in your program that behave differently, depending upon a creation keyword that you might not even know was set.

2. Change the HASH behavior, so if a key is a string, then internally it constructs its hash using an "IDL_Validname()" version of the key. Again, we would store the original keys, so they could be returned intact. Numeric keys would be unchanged.
Advantage: No weird keyword to have to explain - just a single hash class.
Disadvantage: Backwards compatibility issues - could no longer have 2 keys that differed only in their case. Would need to explain that if you want to use "." then you have to be "careful" with your key names.

3. Add a new "DICT" class, that behaves differently than HASH. Keys can only be strings. Keys could still be stored (and returned) with mixed case, errors are thrown if a key isn't a valid IDL variable name. Internally, the actual hash would be done with uppercase versions of the keys.
Advantage: No backwards compatibility issues. Documentation is very clear.
Disadvantage: Yet another class.

Just to reiterate, in all 3 cases, the original "mixed" case keys would be stored, and could be returned with the hash.Keys() method.

Thoughts? Is anyone storing 2 keys that differ only in their case? Are you using HASH, and if so, for what purpose?

Thanks!
-Chris
ExelisVIS
