Subject: Re: Copying a hash
Posted by Paul Van Delst[1] on Mon, 06 Aug 2012 22:56:43 GMT
View Forum Message <> Reply to Message

With apologies for replying to my own email, but I had to try it out....

On 08/06/12 17:44, Paul van Delst wrote:
> On 08/06/12 16:54, Matt wrote:
>> Hi All,
>>
>> Does anyone know if there's a simple way that I can make a copy of a hash, which I can then edit independently of the
>> original? For example, it seems that, like a pointer, changes that I make to the copy are also applied to the
>> original:
>>
>> IDL> original=hash('A', [1, 2]) IDL> copy=original IDL> copy['A', 1]=10 IDL> print, copy A: 1 10 IDL>
>> print, original A: 1 10
>>
>> I can copy to a new hash key-by-key:
>>
>> copy=hash() foreach variable, original, key do copy[key]=original[key]
>>
>> Which works fine, unless one of the elements in the hash is itself a hash, then I end up with the same problem one
>> level down.
>>
>> Is there something simple I'm missing here?
>
> This is what the documentation says:
>
> -----%<-----
> To create a new hash variable whose elements are copies of the values in the original hash, you could use the following:
>
>   newHash = HASH(origHash.Keys(), origHash.Values())
>
> Another method to copy a hash is to use array syntax to copy all of the elements:
>
>   newHash = origHash[*]
>
> This is equivalent to using HASH(origHash.Keys(), origHash.Values()) and is provided as a programming shortcut.
>
> For example:
>
>   hash1 = HASH('key1', 1, 'key2', 2)

```
>   hash2 = hash1[*]
>   hash2['key1'] = 'hello'
>   HELP, hash1['key1'], hash2['key2']
>
>  IDL Prints:
>
>   <Expression> STRING = 1
>   <Expression> STRING = 'hello'
>
>  Note that the value in hash1 remains unchanged.
>  -----%<-----
>
>  There's no mention of what happens if a hash value is itself a hash though.
```

```
IDL> o=hash('a',[1,2],7,'a string',5.0,hash('b',indgen(10)))

IDL> print, o
5.00000: <ObjHeapVar1(HASH)>
a:           1           2
7: a string

IDL> c=o[*]

IDL> print, c
5.00000: <ObjHeapVar1(HASH)>
a:           1           2
7: a string

IDL> c['a',1]=10

IDL> print, c
5.00000: <ObjHeapVar1(HASH)>
a:           1          10
7: a string

IDL> print, o
5.00000: <ObjHeapVar1(HASH)>
a:           1           2
7: a string
```

But, as you can see, the hash in each is the same object reference.

```
IDL> print, c[5.0,'b']
       0       1       2       3       4       5       6       7       8       9

IDL> print, o[5.0,'b']
       0       1       2       3       4       5       6       7       8       9
```

```
IDL> c[5.0,'b',4]=100

IDL> print, o[5.0,'b']
       0       1       2       3     100       5       6       7       8       9
```

Bummer. To be honest, I'm not sure what the correct behaviour should be. Recursively copy all the components? I guess if
we think of the numbers and strings as objects also, then the answer should probably be yes.... ? Why duplicate one type
of object (int, float, or string) but not another (hash or list)? Still... it just doesn't seem right.

cheers,

paulv