Subject: Case Insensitive Hash but still preserve cases of original keys
Posted by SonicKenking on Mon, 14 Jul 2014 06:24:08 GMT
View Forum Message <> Reply to Message

Hello Community,

I tried to implement a special Hash object with following features
1. Keys are of String type
2. Keys should be case insensitive
3. The original cases of keys should be preserved and reported when the keys() method is called.

Let's say it is called SpeicalHash. It should support following operations:

```
h = SpecialHash()
h['X'] = 42
print, h.keys() ; output 'X'
print, h['x'], h['X'] ; output 42, 42
print, h.haskey('x'), h.haskey('X')  ; output 1, 1

h['x'] = 1337
print, h['x'], h['X'] ; output 1337, 1337
```

I came up with an implementation (attached at the end of this post) and it meets all of the above requirements. However, when I tried to print the variable h, it reports error of "Key does not exist"

```
print, h ; Output % Key does not exist: "X"
```

I found out the PRINT command calls Hash::_overloadPrint to do the job and it in turn calls Keys() method to get the keys. I guess that is where the key "X" comes in. However, I don't understand how PRINT gets the value of a key because none of the overload Bracket methods are called (the cases of keys are taken care of in SpecialHash's bracket methods). I can only think of two possibilities:

1. The Hash::_overloadPrint method calls Hash's own bracket methods to get the value of a key. Therefore the SpecialHash's bracket methods did not get called. If this is true, it seems to be a bug in IDL's Hash implementation and should be fixed.

2. The Hash::_overloadPrint method calls some other hidden method to get the value of a key. If this is the case, can someone show me how it is done?

Thanks!

Yang

```
; ============== SpecialHash Implementation Starts Here ==============


function SpecialHash::keys
```

```
    return, list(self.keylist, /extract)
end

function SpecialHash::hasKey, _keys
    keys = strlowcase(_keys)
    return, self->Hash::hasKey(keys)
end

function SpecialHash::_overloadBracketsRightSide, isRange, $
    sub1, sub2, sub3, sub4, sub5, sub6, sub7, sub8

    sub1 = strlowcase(sub1)
    return, self->hash::_overloadBracketsRightSide(isRange, sub1, sub2, sub3, sub4, sub5, sub6,
sub7, sub8)

end

pro SpecialHash::_overloadBracketsLeftSide, objref, value, isrange, $
    _sub1, sub2, sub3, sub4, sub5, sub6, sub7, sub8

    sub1 = strlowcase(_sub1)
    self->Hash::_overloadBracketsLeftSide, objref, value, isrange, sub1, sub2, sub3, sub4, sub5,
sub6, sub7, sub8
    if (self->Hash::keys()).count() eq self.keylist.count() + 1 then self.keylist.add, _sub1

end

function SpecialHash::init
    if ~self->Hash::init() then return, 0
    self.keylist = list()
    return, 1
end

pro SpecialHash__define
    class = {SpecialHash, inherits hash, $
        keylist: list() }
end


; ============== End of Code ==============
```

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by SonicKenking on Mon, 14 Jul 2014 06:41:17 GMT
View Forum Message <> Reply to Message

On Monday, July 14, 2014 4:24:08 PM UTC+10, SonicKenking wrote:
> Hello Community,

>
>
>
> I tried to implement a special Hash object with following features
>
> 1. Keys are of String type
>
> 2. Keys should be case insensitive
>
> 3. The original cases of keys should be preserved and reported when the keys() method is called.
>
>
>
> Let's say it is called SpeicalHash. It should support following operations:
>
>
>
> h = SpecialHash()
>
> h['X'] = 42
>
> print, h.keys() ; output 'X'
>
> print, h['x'], h['X'] ; output 42, 42
>
> print, h.haskey('x'), h.haskey('X') ; output 1, 1
>
>
>
> h['x'] = 1337
>
> print, h['x'], h['X'] ; output 1337, 1337
>
>
>
> I came up with an implementation (attached at the end of this post) and it meets all of the above requirements. However, when I tried to print the variable h, it reports error of "Key does not exist"
>
>
>
> print, h ; Output % Key does not exist: "X"
>
>
>
> I found out the PRINT command calls Hash::_overloadPrint to do the job and it in turn calls Keys() method to get the keys. I guess that is where the key "X" comes in. However, I don't understand how PRINT gets the value of a key because none of the overload Bracket methods

are called (the cases of keys are taken care of in SpecialHash's bracket methods). I can only think of two possibilities:
>
>
>
> 1. The Hash::_overloadPrint method calls Hash's own bracket methods to get the value of a key. Therefore the SpecialHash's bracket methods did not get called. If this is true, it seems to be a bug in IDL's Hash implementation and should be fixed.
>
>
>
> 2. The Hash::_overloadPrint method calls some other hidden method to get the value of a key. If this is the case, can someone show me how it is done?
>
>
>
> Thanks!
>
>
>
> Yang
>
>
>
> ; ================ SpecialHash Implementation Starts Here ==============
>
>
>
>
>
> function SpecialHash::keys
>
> return, list(self.keylist, /extract)
>
> end
>
>
>
> function SpecialHash::hasKey, _keys
>
> keys = strlowcase(_keys)
>
> return, self->Hash::hasKey(keys)
>
> end
>
>
>

```
>  function SpecialHash::_overloadBracketsRightSide, isRange, $
>
>     sub1, sub2, sub3, sub4, sub5, sub6, sub7, sub8
>
>
>
>     sub1 = strlowcase(sub1)
>
>     return, self->hash::_overloadBracketsRightSide(isRange, sub1, sub2, sub3, sub4, sub5,
sub6, sub7, sub8)
>
>
>
>  end
>
>
>
>  pro SpecialHash::_overloadBracketsLeftSide, objref, value, isrange, $
>
>     _sub1, sub2, sub3, sub4, sub5, sub6, sub7, sub8
>
>
>
>     sub1 = strlowcase(_sub1)
>
>     self->Hash::_overloadBracketsLeftSide, objref, value, isrange, sub1, sub2, sub3, sub4, sub5,
sub6, sub7, sub8
>
>     if (self->Hash::keys()).count() eq self.keylist.count() + 1 then self.keylist.add, _sub1
>
>
>
>  end
>
>
>
>  function SpecialHash::init
>
>     if ~self->Hash::init() then return, 0
>
>     self.keylist = list()
>
>     return, 1
>
>  end
>
>
>
```

```
>  pro SpecialHash__define
>
>     class = {SpecialHash, inherits hash, $
>
>         keylist: list() }
>
>  end
>
>
>
>
>
>  ; =============== End of Code ==============
```

Forgot to attach my version info

```
IDL> help, !version
** Structure !VERSION, 8 tags, length=104, data length=100:
   ARCH            STRING    'x86_64'
   OS              STRING    'Win32'
   OS_FAMILY       STRING    'Windows'
   OS_NAME         STRING    'Microsoft Windows'
   RELEASE         STRING    '8.3'
   BUILD_DATE      STRING    'Nov 15 2013'
   MEMORY_BITS     INT             64
   FILE_OFFSET_BITS
                   INT             64
```

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by Fabzi on Mon, 14 Jul 2014 06:53:53 GMT
View Forum Message <> Reply to Message

Hi,

On 14.07.2014 08:24, SonicKenking wrote:
> I tried to implement a special Hash object with following features
> 1. Keys are of String type
> 2. Keys should be case insensitive
> 3. The original cases of keys should be preserved and reported when the keys() method is called.

this sounds like a dictionary to me:

```
IDL> h = dictionary()
IDL> h['X'] = 42
IDL> print, h.keys()
X
```

```
IDL> print, h['x'], h['X']
      42      42
IDL> print, h.haskey('x'), h.haskey('X')
           1           1
IDL> h['x'] = 1337
IDL> print, h['x'], h['X'] ; output 1337, 1337
    1337    1337
IDL> print, h.keys()
x
```

Cheers,

Fabien

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by SonicKenking on Mon, 14 Jul 2014 06:57:47 GMT
View Forum Message <> Reply to Message

On Monday, July 14, 2014 4:24:08 PM UTC+10, SonicKenking wrote:
> Hello Community,
>
>
>
> I tried to implement a special Hash object with following features
>
> 1. Keys are of String type
>
> 2. Keys should be case insensitive
>
> 3. The original cases of keys should be preserved and reported when the keys() method is called.
>
>
>
> Let's say it is called SpeicalHash. It should support following operations:
>
>
>
> h = SpecialHash()
>
> h['X'] = 42
>
> print, h.keys() ; output 'X'
>
> print, h['x'], h['X'] ; output 42, 42
>
> print, h.haskey('x'), h.haskey('X') ; output 1, 1

>
>
>
> h['x'] = 1337
>
> print, h['x'], h['X'] ; output 1337, 1337
>
>
>
> I came up with an implementation (attached at the end of this post) and it meets all of the above requirements. However, when I tried to print the variable h, it reports error of "Key does not exist"
>
>
>
> print, h ; Output % Key does not exist: "X"
>
>
>
> I found out the PRINT command calls Hash::_overloadPrint to do the job and it in turn calls Keys() method to get the keys. I guess that is where the key "X" comes in. However, I don't understand how PRINT gets the value of a key because none of the overload Bracket methods are called (the cases of keys are taken care of in SpecialHash's bracket methods). I can only think of two possibilities:
>
>
>
> 1. The Hash::_overloadPrint method calls Hash's own bracket methods to get the value of a key. Therefore the SpecialHash's bracket methods did not get called. If this is true, it seems to be a bug in IDL's Hash implementation and should be fixed.
>
>
>
> 2. The Hash::_overloadPrint method calls some other hidden method to get the value of a key. If this is the case, can someone show me how it is done?
>
>
>
> Thanks!
>
>
>
> Yang
>
>
>
> ; ================ SpecialHash Implementation Starts Here ==============
>
>

```
>
>
>
>  function SpecialHash::keys
>
>     return, list(self.keylist, /extract)
>
>  end
>
>
>
>  function SpecialHash::hasKey, _keys
>
>     keys = strlowcase(_keys)
>
>     return, self->Hash::hasKey(keys)
>
>  end
>
>
>
>  function SpecialHash::_overloadBracketsRightSide, isRange, $
>
>     sub1, sub2, sub3, sub4, sub5, sub6, sub7, sub8
>
>
>
>     sub1 = strlowcase(sub1)
>
>     return, self->hash::_overloadBracketsRightSide(isRange, sub1, sub2, sub3, sub4, sub5,
sub6, sub7, sub8)
>
>
>
>  end
>
>
>
>  pro SpecialHash::_overloadBracketsLeftSide, objref, value, isrange, $
>
>     _sub1, sub2, sub3, sub4, sub5, sub6, sub7, sub8
>
>
>
>     sub1 = strlowcase(_sub1)
>
>     self->Hash::_overloadBracketsLeftSide, objref, value, isrange, sub1, sub2, sub3, sub4, sub5,
sub6, sub7, sub8
```

```
>
>     if (self->Hash::keys()).count() eq self.keylist.count() + 1 then self.keylist.add, _sub1
>
>
>
>  end
>
>
>
>  function SpecialHash::init
>
>     if ~self->Hash::init() then return, 0
>
>     self.keylist = list()
>
>     return, 1
>
>  end
>
>
>
>  pro SpecialHash__define
>
>     class = {SpecialHash, inherits hash, $
>
>         keylist: list() }
>
>  end
>
>
>
>
>
>  ; =============== End of Code ==============
```

A bit more findings:

This is most likely a bug in 8.3

I tried with 8.2.3 and I didn't get any error with "PRINT, h". As 8.3 added implied print, it is likely where the bug is introduced?

However, one more problem with both 8.2.3 and 8.3 is that the keys shown by PRINT is in wrong case:

```
IDL> h = SpecialHash()
IDL> h['X'] = 42
IDL> print, h
```

x: 42

NOTE the x displayed by PRINT is in lowercase while it really should be in uppercase.
The case is also wrong with FOREACH loop

IDL> foreach v,h,k do print, k,v
x 42

Scratching my head ...

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by SonicKenking on Mon, 14 Jul 2014 07:02:56 GMT
View Forum Message <> Reply to Message

On Monday, July 14, 2014 4:53:53 PM UTC+10, Fabien wrote:
> Hi,
>
>
>
> On 14.07.2014 08:24, SonicKenking wrote:
>
>> I tried to implement a special Hash object with following features
>
>> 1. Keys are of String type
>
>> 2. Keys should be case insensitive
>
>> 3. The original cases of keys should be preserved and reported when the keys() method is called.
>
>
>
> this sounds like a dictionary to me:
>
>
>
> IDL> h = dictionary()
>
> IDL> h['X'] = 42
>
> IDL> print, h.keys()
>
> X
>
> IDL> print, h['x'], h['X']
>
> 42 42

>
> IDL> print, h.haskey('x'), h.haskey('X')
>
>   1   1
>
> IDL> h['x'] = 1337
>
> IDL> print, h['x'], h['X'] ; output 1337, 1337
>
>   1337   1337
>
> IDL> print, h.keys()
>
> x
>
>
>
> Cheers,
>
>
>
> Fabien

Hi Fabien,

Thanks for the reply. Unfortunately, Dictionary is not the solution in this case. Because the keys do not necessarily conform the naming rules of IDL variables, e.g. a key with white spaces "Some Key " or non-alphanumeric letter "<Key>".

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by Fabzi on Mon, 14 Jul 2014 09:17:58 GMT
View Forum Message <> Reply to Message

On 14.07.2014 08:24, SonicKenking wrote:
> print, h ; Output % Key does not exist: "X"

Without looking at your code: maybe you should overload Print, too?

SpecialHash::_overloadPrint()

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by SonicKenking on Tue, 15 Jul 2014 00:22:23 GMT
View Forum Message <> Reply to Message

On Monday, July 14, 2014 7:17:58 PM UTC+10, Fabien wrote:

> On 14.07.2014 08:24, SonicKenking wrote:
>
>> print, h ; Output % Key does not exist: "X"
>
>
>
> Without looking at your code: maybe you should overload Print, too?
>
>
>
> SpecialHash::_overloadPrint()

Yes the issue can be solved by writing my own PRINT and FOREACH function. However the inherited PRINT should just work if there is no bug or hidden mechanisms in IDL's HASH implementation.

I am wondering exactly how HASH's PRINT method obtains the key and value information of an object. The overloaded KEYS method gets called but not the overloaded BRACKET methods.

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by Fabzi on Tue, 15 Jul 2014 08:36:22 GMT
View Forum Message <> Reply to Message

Hi,

On 15.07.2014 02:22, SonicKenking wrote:
> However the inherited PRINT should just work if there is no
> bug or hidden mechanisms

The "hidden mechanism" in "print, h" will very likely be a call to
self.key(), which you overrided with your own keylist.

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by Fabzi on Tue, 15 Jul 2014 08:38:02 GMT
View Forum Message <> Reply to Message

On 15.07.2014 10:36, Fabien wrote:
> Hi,
>
> On 15.07.2014 02:22, SonicKenking wrote:
>> However the inherited PRINT should just work if there is no
>> bug or hidden mechanisms
>
> The "hidden mechanism" in "print, h" will very likely be a call to
> self.key(), which you overrided with your own keylist.

>
>

... this can be verified by setting a breakpoint in your
SpecialHash::keys() function and then call: print, h

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by SonicKenking on Tue, 15 Jul 2014 11:11:06 GMT
View Forum Message <> Reply to Message

Yes the PRINT does call SpecialHash::keys(). I noticed that and that is where the key "X" is obtained.

What I am confused is how PRINT gets the value for the key "X"? It seems to me that it should call something like self["X"], i.e. SpecialHash::_overloadBracketsRightSide. However it is not the case and this can be verified by setting a breakpoint as well.

So in summary, only one of two overriden methods, SpecialHash::keys and SpecialHash::_overloadBracketsRightSide, gets called, while in theory both of them should be called. One for getting the key and one for getting the value associated to the key.

That is what I meant by "hidden mechanism" (or bug maybe) in PRINT. How does it get the value without calling SpecialHash::_overloadBracketsRightSide?

Hopefully someone from Exelis could shed some light on this discussion.

Subject: Re: Case Insensitive Hash but still preserve cases of original keys
Posted by Lajos Foldy on Tue, 15 Jul 2014 12:32:30 GMT
View Forum Message <> Reply to Message

On Tuesday, July 15, 2014 1:11:06 PM UTC+2, SonicKenking wrote:

> What I am confused is how PRINT gets the value for the key "X"? It seems to me that it should call something like self["X"], i.e. SpecialHash::_overloadBracketsRightSide. However it is not the case and this can be verified by setting a breakpoint as well.

There is an undocumented function hash::get:

```
IDL> print, (hash(0,0))[0]
       0
IDL> .comp hash_get
% Compiled module: HASH::GET.
IDL> print, (hash(0,0))[0]
HASH::GET called:       0
       0
```

IDL>

hash_get.pro is:

```
function hash::get, key, _extra=ext
print, 'HASH::GET called: ', key
return, 0
end
```

PRINT probably calls self.get(...), so you can try to add a SpecialHash::Get function.

regards,
Lajos
