Subject: Re: _extra and call_method
Posted by btt on Tue, 01 Feb 2005 15:18:25 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Ben Tupper writes:
>
>
>> I'm running on just one cup of coffee this morning so maybe this is a fuzzy
>> question: could you explain the circumstances in which this is useful? If you
>> return dummy named structure - well, what about all the work that goes into
>> populating its properties it via the INIT function? Or is this for simple data
>> structures (ala widget event structures, etc. ?)
>
>
> Since objects are implemented as named structures in IDL,
> I seem to find a number of instances where it would be
> helpful to know what the names of the fields in that object
> structure are. For example, one of the hugely time-consuming
> tasks in object writing is creating the GetProperty and SetProperty
> methods that allow you to manipulate and set/get values in the
> object structure. Wouldn't it be nice to automate those tasks
> and be able to get and set any property (field) in the object
> without necessarily knowing ahead of time what those properties
> might be? For example, I might like to respond to this:
>
> anObject -> SetProperty, Foo=5
>
> Without specifically having to define the FOO keyword for the object.
>
> If FOO were a field of this object, I could write a generic SetProperty
> method like this (I'm leaving out a couple of important details, but I
> plan an article soon):
>
> PRO myObject::SetProperty, _Extra=extra
>
> ; What keywords are you looking for?
> keywords = Tag_Names(_extra)
>
> ; What properties (fields) can be changed?
> Call_Procedure, Obj_Class(self) + '__define', struct ;**********
> properties = Tag_Names(struct)
>
> ; Set the value of each field according to the keyword value.
> FOR j=0,N_Elements(keywords)-1 DO
> propertyIndex = Where(StrPos(properties, keywords[j]) EQ 0, match)
> IF match EQ 1 THEN self.(propertyIndex) = _extra.(j)
> ENDFOR

>
> END
>
> I can do something similar for a GetProperty method. Adding (copying,
> really) these two generic methods to every object I create, is MUCH
> less time consuming than defining each and every keyword for each
> and every property I hope to change.
>

Hi again,

Just got to thinking on this some more. Here's one item I thought you might be willing to share your thoughts on: in the case of multiple inheritances does each keyword get checked in for each generation of inheritance and if they do, does it matter?

For example...

```
PRO APPENDAGE__DEFINE, class
class = {APPENDAGE, isJointed:0}
END

PRO LEG__DEFINE, class
class = {ARM, INHERITS APPENDAGE, hasOpposingThumb: 0}
END


So, if I use the generic keyword checking for LEG does "isJointed" get checked
twice?  Propbably it doesn't matter a hoot if they do get checked twice.



PRO LEG::SetProperty, _EXTRA = extra

... do that neat keyword checking thing here ...

self->APPENDAGE::SetProperty, _Extra = extra
END
```

Cheers,
Ben