Dummy's Guide to Extending GAP 4

Andrew Solomon

April 1999

Trick: Remember to declare the collections category

When creating new elements don't forget to declare the CategoryCollections.

For example, suppose we want to declare a new multiplicative element called a "bongle". You can type the following into a GAP session by way of experiment.

```
DeclareCategory("IsBongle", IsMultiplicativeElement);
DeclareRepresentation("IsBongleRep", IsComponentObjectRep, ["data"]);
BongleFam := NewFamily("FamilyOfAllBongles",IsBongle);
BongleType := NewType(BongleFam, IsBongle and IsBongleRep);

# Now create a bongle
b1 := Objectify(BongleType, rec(data := 1));
```

Before creating any collection of bongles first create the collections category:

```
DeclareCategoryCollections("IsBongle");
lb := [b1];
#gap> CategoryCollections(IsBongle)(lb);
#true
```

The moral of the story is that if you don't create the collections category *before* the list. It won't be put in the CollectionsFamily of b1 and therefore it won't be in the family of the list [b1] and the list won't know it is a collection of bongles. For the same reason, if you have written

```
IsMagmaOfBongles := IsMagma and CategoryCollections(IsBongle);
```

and perform

```
m:= Magma(b1); # the magma generated by b1
```

then

```
IsMagmaOfBongles(m)
```

will be false unless you have declared the collections category before any collection of bongles has been created. Try it!

Rule of Thumb: *Always declare the collections category right after the category.*

Tip: Creating new multiplicative objects from old

Very often it happens that you want to create a new magma which is very like another magma you already have only with a new multiplication, and perhaps some new elements.

The way this can be achieved is to create a new category of elements.

Suppose we want a category of objects which are just like cyclotomics only every product is the left operand:

```
DeclareCategory("IsBozo", IsCyclotomic);
DeclareRepresentation("IsBozoRep", IsComponentObjectRep, ["data"]);
BozoFam := NewFamily("FamilyOfBozos",IsBozo);
BozoType := NewType(BozoFam, IsBozo and IsBozoRep);
MakeBozo := n->Objectify(BozoType, rec(data := n));
```

```
InstallMethod(\+,
"bozo + bozo", IsIdenticalObj,
[IsBozo and IsBozoRep,
```

```
IsBozo and IsBozoRep], 0,
function(x, y)
        return MakeBozo(x!.data + y!.data);
end);

InstallMethod(\*,
"bozo * bozo", IsIdenticalObj,
[IsBozo , IsBozo ], 0,
function(x, y)
        return x;
end);


InstallMethod(PrintObj, "for bozos", true,
[IsBozo and IsBozoRep], 0,
function(x)
        Print(x!.data);
end);
```

```
gap> a := MakeBozo(1);
1
gap> b := MakeBozo(2);
2
gap> a+b;
3
gap> a*b;
1
```

Tip: Making General Mappings correctly

This is an abbreviation of the information in the header of mapping.gd: Every General mapping should be in one of two categories: IsNonSPGeneralMapping or IsSPGeneralMapping where SP stands for *structure preserving*.

The mappings in the first category should have ImagesElm and PreImagesElm installed as primary methods. These methods shouldn't call any of the other standard methods since other methods for IsNonSPGeneralMappings assume that they may call ImagesElm or PreImagesElm to perform any necessary computation. In fact many things are "automatic" when these primary methods are installed.

On the other hand, mappings which are IsSPGeneralMapping don't assume the existance of ImagesElm or PreImagesElm. Therefore the programmer is free to write these and other methods in the "most sensible" way.