Right now, we have the following large union which can be a pointer into any type of raw. There are several problems with it:

- It is expensive to update. Any time it changes, the entire system has to be recompiled (although the pointer shouldn't change size).
- It is cluttered, since it has every possible type in it, and not much room for comments on each specific one.

```
typedef union Raw
{
  int *Raw_int;
  long int *Raw_long;
  caddr_t Raw_ptr;              /* caddr_t is a pointer */
  Char Raw_text;                /* can't we just call this char* ? */
  long int *Raw_bitmap;
  Vector Raw_vector;
  struct data *Raw_data;
  struct zone *Raw_zones;
  struct fragment *Raw_fragments;
  struct region *Raw_region;    /* we mean Region Raw_region, but it *
                                /* isn't defined yet!*/
  struct font_field *Raw_font_field;
} Raw;
```

To solve this problem, we split the typedef into three components:

1. The header:

```
/* <raw/header.h> */
typedef union Raw
{
```

2. The type lines (which vary based on application):

```
/* <raw/ints.raw> */
  int *Raw_int;
  long int *Raw_long;
/* <raw/text.raw> */
  Char Raw_text;
/* <raw/nest.raw> */
  struct data *Raw_data;
  struct zone *Raw_zones;
  ... etc ...
```

3. The footer:

```
} Raw;
/* <raw/footer.h> */
```

We can thus define files containing each component separately. The header and footer are single, global includes, while the type lines can be in multiple local and/or global includes. For example, we could define the above as

```
#include <raw/header.h>
#include <raw/ints.raw>
#include <raw/text.raw>
#include <raw/nest.raw>
#include <raw/footer.h>
```

If we want to define a new type, we would just create a file

```
/* "newtype.raw" */
  struct newtype *Raw_newtype;
```

Then we could make out local Raw type just contain it (assuming we didn't need any of the interesting ones):

```
#include <raw/header.h>
#include "newtype.raw"
#include <raw/footer.h>
```

Allowing an application to use predefined types requires simply adding a line that calls them to the definition:

```
#include <raw/header.h>
#include "newtype.raw"
#include <raw/text.raw>
#include <raw/footer.h>
```

Sanity checking can (and will!) be done on all of the global types. This will be performed by creating a single set of includes from everything in the raw directory with a .raw extension, and checking its sizeof. This will allow us to detect the following conditions:

- making a Raw type which is not a pointer. Deal with this by fixing the .raw file to have a real pointer type.
- adding a pointer type which (on some obscure machine) is bigger than existing ones. This should force the new large type to be added to the header file, so that it always appears in any local raw. Also, that will force anything that uses Raw at all to be recompiled.

This will not catch errors in local types, but they should be made global at some point anyhow. Perhaps replacing local types with a testraw directory that is available to everyone would suffice, so we can sanity check in both places.

The raw type should probably be the first thing created in a given file. It should definitely occur before including data.h, since region is based on Raw. Raw that contains regions is based the fact that struct region * is enough to identify the pointer size, but since Raw is a union, Region can’t guess its size - thus Raw must come first.