PicOS
EEPROM Object Loader

Version 0.5
January 2009

© Copyright 2008-2009, Olsonet Communications Corporation.
All Rights Reserved.

Preamble

The role of the EEPROM object loader is to make it possible to store in EEPROM images of C structures that can be loaded from there as ready memory-resident objects usable by the praxis. The tricky part is to allow those structures to contain pointers to other objects that are stored in the same chunk of EEPROM.

The operation

The loader consists of a single function described below. To make this function available to the praxis, you should put this:

```
#include "ee_ol.h"
```

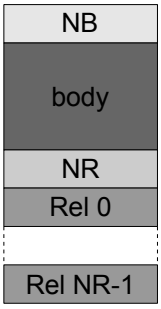
somewhere in the program's header. Here is the function:

```
byte ee_load (lword addr, void **op);
```

The first argument specifies the EEPROM address from where the object is to be loaded. The second argument points to the pointer which will be set with the object's memory address. On success, the function returns zero and sets the pointer. On failure, the function can return **`EE_LOADER_NOMEM`** (17), which means that **`umalloc`** has failed, or **`EE_LOADER_GARBAGE`** (33), which indicates a formal problem with the object's layout (its relocation). This is explained below.

Object layout in EEPROM

The object need not be aligned in EEPROM at any particular boundary. Its layout looks like this:



where NB is a word (2 consecutive bytes in little endian format) storing the number of bytes occupied by the object's body. The loader first reads this word and tries to **`umalloc`** NB bytes for the object. If this operation fails, the loader exits returning **`EE_LOADER_NOMEM`**. Otherwise, the loader reads into the allocated memory area NB subsequent bytes from the EEPROM. Then it reads one more word (NR) interpreting it as the number of relocation words that follow. In particular, if the object requires no relocation, that number will be zero.

For each relocation word, the loader performs the following actions:

1. The most significant bit of the word is interpreted as a string flag. The lower 15 bits are interpreted as a word offset into the object's body area.

2. If the offset points beyond the object's body area, the loader frees the object and returns **EE_LOADER_GARBAGE**.

3. Otherwise, the indicated word of the object is modified by adding to its contents the address of the object's origin in memory. If the new value of the word (interpreted as an address) lies outside the boundaries of the object's body, the loader frees the object and returns **EE_LOADER_GARBAGE**.

4. If the string flag in the relocation word was set, the loader assumes that the word points to a strings and verifies whether the entire string is present within the object's body. If this isn't the case, the loader frees the object and returns **EE_LOADER_GARBAGE**.