```
int     prim2flag (pe)
PE      pe;
```

The routine `bool2prim` is a macro which performs the inverse operation. It behaves as if it was defined as:

```
PE      bool2prim (b)
int     b;
```

In actuality, `bool2prim` calls the routine `flag2prim` which builds a presentation element containing the boolean value given and sets the `pe_class` and `pe_id` fields of the presentation element to the desired values.

```
PE      flag2prim (b, class, id)
int     b;
PElementClass class;
PElementID id;
```

5.4.2 Integer

An *integer* is a signed-quantity, whose precision is specific to a particular host. The routine `prim2num` takes a pointer to a presentation element and returns the integer value encoded therein (if the value is `NOTOK`, check the `pe_errno` element of the `PElement` structure to see if there really was an error).

```
int     prim2num (pe)
PE      pe;
```

The routine `int2prim` is a macro which performs the inverse operation. It behaves as if it was defined as:

```
PE      int2prim (i)
int     i;
```

In actuality, `int2prim` calls the routine `num2prim` which builds a presentation element containing the numeric value given and sets the `pe_class` and `pe_id` fields of the presentation element to the desired values.

```
PE      num2prim (i, class, id)
int     i;
PElementClass class;
PElementID id;
```

5.4.3 Octetstring

An *octetstring* is a pair of values: a character pointer and an integer length. The pointer addresses the first octet in the string (which need not be null-terminated), the length indicates how many octets are in the string. The routine `prim2str` takes a pointer to a presentation element and allocates a new string (using the *malloc*(3) routine) containing the value encoded therein.

```
char   *prim2str (pe, len)
PE      pe;
int    *len;
```

The routine `oct2prim` is a macro which performs the inverse operation, copying the original string (and not de-allocating it). It behaves as if it was defined as:

```
PE      oct2prim (s, len)
char   *s;
int     len;
```

In actuality, `oct2prim` calls the routine `str2prim` which builds a presentation element containing the string value given and sets the `pe_class` and `pe_id` fields of the presentation element to the desired values.

```
PE      str2prim (s, len, class, id)
char   *s;
int     len;
PElementClass class;
PElementID id;
```

In addition to `oct2prim`, there are several macros which manipulate octet-strings, setting the class and id of the presentation element to the appropriate

value:

Macro	String
ia5s2prim	IA5 string
nums2prim	Numeric string
prts2prim	Printable string
t61s2prim	T.61 string
vtxs2prim	Videotex string
gfxs2prim	Graphics string
viss2prim	Visible string
gens2prim	General string
chrs2prim	Character string
ode2prim	Object descriptor

Each of these macros are defined in a similar manner to the `oct2prim` macro.

5.4.4 Octetstrings revisited

For efficiency reasons, it is often desirable to represent an octetstring using a `qbuf` structure. Although the format of a `qbuf` is described in Section 4.3.2 on page 119 of *Volume Two*, the *libpsap*(3) library provides several routines for ease of manipulation.

The routine `qb2prim` takes a linked-list of `qbuf`s and builds a presentation element with the `pe_class` and `pe_id` fields of the presentation element set to the desired values:

```
PE      qb2prim (qb, class, id)
struct qbuf *qb;
PElementClass class;
PElementID id;
```

> **NOTE:** The presentation element returned by `qb2prim` contains pointers into the linked-list of `qbuf`s. Therefore care must be taken not to delete any of the `qbuf`s in the linked-list prior to the final use of the presentation element. Note however, than the presentation element may be de-allocated at anytime without affecting the `qbuf`s in the linked-list.

The routine `prim2qb` performs the inverse operation:

```
struct qbuf *prim2qb (pe)
PE      pe;
```

When examining the contents of a `qbuf`, it is most efficient to examine each `qbuf` in the linked list, as in:

```
register struct qbuf *qp;

for (qp = qb -> qb_forw; qp != qb; qp = qp -> qb_forw)
    /* examine qp -> qb_data, qp -> qb_len here */ ;
```

However, some applications may wish to simply convert the `qbuf` into one string and examine it without worrying about traversing the linked list. The routine `qb2str` performs this function:

```
char   *qb2str (qb)
struct qbuf *qb;
```

The routine `str2qb` performs the inverse operation, converting a string to a `qbuf`.

```
struct qbuf *str2qb (s, len, head)
char   *s;
int     len,
head;
```

The `head` parameter indicates whether `str2qb` should allocate a `qbuf` containing both the head of the linked list and one element (`head` is non-zero), or just the element itself (`head` is zero).

The routine `qb_pullup` can be used to compact a linked list of `qbuf`s into a list containing the `qbuf` head and one element:

```
int     qb_pullup (qb)
struct qbuf *qb;
```

This routine returns the manifest constant `NOTOK` on failure, and `OK` otherwise.

The routine `qbuf2pe` is used to set-up a presentation stream which reads from a linked list of `qbuf`s and returns the presentation element encoded therein:

```
PE      qbuf2pe (qb, len, result)
struct qbuf *qb;
int     len,
       *result;
```

This routine returns a presentation element on success, and `NULLPE` on failure. In the latter case, the parameter `result` is updated to reflect the presentation stream error.

Finally, the routine `qb_free` will deallocate a linked list of `qbuf`s:

```
void    qb_free (qb)
```

5.4.5 Bitvector

A *bitvector* is an arbitrarily long string of bits (starting with bit `0`) with three operations:

`bit_on:` which turns the indicated bit on;

```
int     bit_on (pe, bitno)
PE      pe;
int     bitno;
```

`bit_off:` which turns the indicated bit off;

```
int     bit_off (pe, bitno)
PE      pe;
int     bitno;
```

and,

`bit_test:` which returns a boolean value telling if the indicated bit was on.

```
int     bit_test (pe, bitno)
PE      pe;
int     bitno;
```

The routine `prim2bit` takes a pointer to a presentation element and builds such an abstraction containing the value encoded therein.

```
PE      prim2bit (pe)
PE      pe;
```

The routine `bit2prim` performs the inverse operation.

```
PE      bit2prim (pe)
PE      pe;
```

There are also some support routines useful mainly with *pepy*. The routine `strb2bitstr` takes a pointer to a character string and an integer containing the number of bits and constructs a bit vector as described above containing the bits. You need to call `bit2prim` to convert it into a presentation element.

```
PE      strb2bitstr (cp, length, class, id)
char   *cp;
int     length;
PElementClass class;
PElementID id;
```

The inverse operation is handled by the function `bitstr2strb`. This takes bit vector containing a bit string and produces a new character string with the appropriate bits set. The second parameter will contain the number of valid bits on return.

```
char   *bitstr2strb (pe, length)
PE      pe;
int    *length;
```

Finally, the routines `int2strb` and `strb2int` are provided to convert an integer containing a bit list into a string of bits suitable for input to `strb2bitstr`, and to perform the inverse operation

```
char   *int2strb (n, length)
int     n;
int     length;

int     strb2int (cp, length)
char   *cp;
int     length;
```

5.4.6 Object Identifier

An *object identifier* represents an ordered set of authoritative designations used for identification. Internally, the `OIDentifier` is used to represent this notion:

```
typedef struct OIDentifier {
    int     oid_nelem;

    unsigned int *oid_elements;
}       OIDentifier, *OID;
#define NULLOID ((OID) 0)
```

This structure contains two elements:

`oid_elements/oid_nelem:` the (ordered) list of sub-identifiers (and the number of elements in the list).

Two object identifiers can be compared with `oid_cmp`, which takes two pointers to `OIDentifier` structures as arguments, and returns `0` if the two structures are identical, `1` otherwise.

```
int     oid_cmp (p, q)
OID     p,
        q;
```

Further, an object identifier can be duplicated with `oid_cpy`. This routine takes a pointer to an `OIDentifier` structure as an argument, and returns a pointer to a new `OIDentifier` structure, or `NULLOID` on error.

```
OID     oid_cpy (oid)
OID     oid;
```

Simlarly, the routine `oid_free` can be used to free an object identifier which has been allocated.

```
void    oid_free (oid)
OID     oid;
```

The routine `sprintoid` can be used to return a null-terminated string describing the object identifier.

```
char   *sprintoid (oid)
OID     oid;
```

The default routine `oid2ode` is identical to the `sprintoid`, but is intended for user customization. For example, whilst `sprintoid` should always be used when an object identifier should be expressed in "dot notation", the user may define a new `oid2ode` which returns symbolic names, if so desired.

The routine `str2oid` can be used as the inverse of `sprintoid`:

```
OID     str2oid (s)
char   *s;
```

Finally, the routine `ode2oid` can be used to fetch an object identifier from the *isobjects*(5) database described in Chapter 9.

```
OID     ode2oid (descriptor)
char   *descriptor;
```

This routine performs a lookup using the `getisobjectbyname` routine and then returns a pointer to a static `OIDentifier` structure containing the desired information. The routine returns `NULLOID` on failure.

The routines `prim2oid` and `oid2prim` are used to translate between a machine-specific internal form and the machine-independent form.

```
OID     prim2oid (pe)
PE      pe;

PE      oid2prim (o)
OID     o;
```

In actuality, `oid2prim` is really a macro which calls the routine `obj2prim` which builds a presentation element containing the object identifier given and sets the `pe_class` an `pe_id` fields of the presentation element to the desired values.

```
PE      obj2prim (o, class, id)
OID     o;
PElementClass class;
PElementID id;
```

5.4.7 Timestring

A *timestring* represents a date/time in many forms. Currently, two forms are supported: *universal time* and *generalized time*. Internally, the `UTCtime` structure is used to represent both notions:

```
typedef struct UTCtime {
    int     ut_year;
    int     ut_mon;
    int     ut_mday;
    int     ut_hour;
    int     ut_min;
    int     ut_sec;

    int     ut_msec;

    int     ut_zone;

    int     ut_flags;
}       UTCtime, *UTC;
#define NULLUTC ((UTC) 0)
```

This structure contains several elements:

`ut_year`: the year, either since the current century (e.g., `86`), or absolute (e.g., `1986`);

`ut_mon`: the month of the year, in the range `1..12`;

`ut_mday`: the day of the month, in the range `1..31`;

`ut_hour`: the hour of the day, in the range `0..23`;

`ut_min`: the minute of the hour, in the range `0..59`;

`ut_sec`: (optionally) the second of the minute, in the range `0..59`;

`ut_usec`: (optionally) fractions of a second, in microseconds;

`ut_zone`: (optionally) the timezone, expressed as minutes from UT; and,

`ut_flags`: various flags describing the timestring:

`UT_ZONE`: the `ut_zone` element is present;

`UT_SEC`: the `ut_sec` element is present; and,

`UT_USEC`: the `ut_usec` element is present.

The routine `prim2utct` takes a pointer to a presentation element and returns the universal time encoded therein.

```
UTC     prim2utct (pe)
PE      pe;
```

The routine `utct2prim` is a macro which performs the inverse operatoin. It behaves as if it was defined as:

```
PE      utct2prim (u)
UTC     u;
```

The routine `prim2gent` takes a pointer to a presentation element and returns the universal time encoded therein.

```
UTC     prim2gent (pe)
PE      pe;
```

The routine `gent2prim` is a macro which performs the inverse operatoin. It behaves as if it was defined as:

```
PE      gent2prim (u)
UTC     u;
```

In actuality, both `utct2prim` and `gent2prim` call the routine `time2prim` which builds a presentation element containing the universal or general time given and sets the `pe_class` an `pe_id` fields of the presentation element to the desired values.

```
PE      time2prim (ut, generalized, class, id)
UTC     u;
int generalized;
PElementClass class;
PElementID id;
```

To get a null-terminated string representation of a `UTCtime` structure, the macros `utct2str` and `gent2str` can be used. Each take a single argument, a pointer to a `UTCtime` structure. These call the routine `time2str`: