Published: 01/26/67
(Supersedes: BX.12.01, 02/01/66)

Identification

The representation of options in storage
C. Marceau

Purpose

This section describes the representation of options in the options stack and the permanent options list. The user does not have to read this section which is concerned solely with implementation, in order to use the options facility as it is described in BX.12.00.

The representation of options described here is designed to enable ready reference of options by name and by frame, and to minimize storage requirements.

Options in the file system hierarchy

Options are recorded for each user, since each user has his own options. For each user there is a permanent options list (the perm_op_list segment) in his user profile (see BQ.4.03). Perm_op_list is created by a procedure called create_op_list (not yet documented). Each time that the user logs in, a copy of his permanent options list is made. The copy, called the op_stack segment, is an entry in the process profile directory which contains the profile of the interactive Working Process (for a description of logging in, see BQ.3.02). Op_stack is where the actual stacking of options takes place. The options stack in op_stack is pushed and popped as described in BX.12.00. The permanent options list is not pushed or popped. It records the permanent values of options (values in frame one). Thus if a user logs in twice, he has two option stacks. His permanent options list determines the initial form of each stack. Thereafter the stacks are independent of each other.

Review of terminology

An option is a binary switch which may be set on or off. An option may also have a specification, a character string which provides additional information when the switch is on.

The combination of an on|off value and a specification is the value of the option.

An option is set if some value for the option is recorded in either the permanent options list or the options stack. An option is unset if neither the permanent options list nor the options stack records a value for the option.

The stacking of options

A frame of the options stack is created when the stack is pushed and deleted when the stack is popped. Frames are numbered 1, 2, 3, ... Frames 1 and 2 have special significance: frame 1 is called the permanent frame and frame 2 the session frame. Values recorded in frame 1 of the options stack are recorded in the perm_op_list segment as well as in the op_stack segment; these values are valid from one session to another. Values recorded in frame 2 are valid for the duration of a console session (or until logout); the pop_opt procedure (see BY.9.02) does not pop the options stack (op_stack) beyond frame 2.

An option is said to be set in frame n if it has a value in frame n. An option is unset in frame n if it has no value in frame n, i.e., if it has not been set in any frame up to and including frame n.

When the stack is pushed, a new frame is created, and all options which were set in the previous frame are now set in the newly-created frame as well. Clearly it would be inefficient to represent, for every frame, all of the options which are set in the frame. For example, the user may set no new options in frame n, yet the frame might have to record the values of 100 options set in frame n-1.

To avoid such inefficiency, not every option which has a value in frame n is represented in frame n. In general, an option is represented only in those frames in which the user explicitly set it.

When the user sets an option, he creates an explicit setting of the option in the options stack, in the permanent options list, or both. A setting is an explicit record of the value of an option in a frame.

After an option has been set in a frame, its value in that frame remains valid in all later frames, until it is set again. Thus the option has a value in all later

frames, but it may not, and probably will not, have a setting in all later frames. The value of an option in frame n is the value of the option in frame m, where m is the largest number ≤n such that the option has a setting in frame m.

Accessing the permanent options list and the options stack

When any option handling procedure accesses perm_op_list or op_stack, it treats the segment as internal data; namely, a PL/I controlled structure called option_seg. The statement

```
call generate_ptr (pathname, ptr);
```

returns a pointer (ptr) to the origin of the segment pathname. The pathname of perm_op_list is

```
>user_profile_dir > name.projid > perm_op_list
```

The pathname of op_stack is

```
pdirdir > p > pprofile > op_stack,
```

where name and projid are the name and project ID of the user, and p is the name of the process directory for this process.

Structure of the options stack

The options stack and the permanent options list each have the same structure. They consist of

1) a frame table to record frames of the stack,

2) a hash table for hashing option names,

3) headers for options which the user has set,

4) settings for options.

Each of these items is discussed in this section. Very briefly, options are hashed by name. When a user sets an option for the first time, he creates an entry for it in the hash table. The hash table entry points to an option header which records the name of the option and a setting. The header is located in a frame. When the user sets the option later, in another frame, a new setting of the option is created in that frame.

When a option handling procedure (see BY.9.01 - BY.9.05) accesses either op_stack or perm_op_list, it treats the segment as a controlled PL/I structure:

```
dcl  1 option_seg ctl (ptr);
     2 nopt fixed,
     2 fno fixed,
     2 htsize fixed,
     2 ftsize fixed,
     2 htptr bit (18),
     2 ftptr bit (18),
     2 space area ((K));
```

option seg- the segment referred to may be either op_stack or perm_op_list. ptr is obtained by generate_ptr (see above).

fno- current frame number. In perm_op_list, fno = 2. This is so that when perm_op_list is copied as op_stack, the current frame will be the session frame (i.e., frame 2), containing settings for the current session.

htsize- length of the hash table. The hash table is created with length 30, but may be expanded when necessary. (See BY.9.05)

ftsize- length of the frame table. The frame table is created with length 20, but may be expanded or contracted (See BY.9.02.)

htptr- relative pointer to the origin of the hash table.

ftptr- relative pointer to the origin of the frame table.

space- a PL/I area, in which option handling procedures allocate storage. The contents of space are explained later in this section. K must be determined by create_op_list when perm_op_list is created.

Representation of a frame

The frame table is used to access frames of the options stack. For each frame in the stack, there is an entry in the frame table which contains a relative pointer to the most recent setting in the frame. Each setting in a frame points to the previous (i.e., next most recent) setting in that frame. The settings are double-chained so that any setting may be deleted easily by rechaining.

The frame table is used primarily for popping the options stack. Suppose the current frame is frame k. The kth entry in the frame table points to a setting in frame k. This setting points back to another, and so on, until the back frame pointer for some setting is null. When all these settings are deleted, and the kth entry of the frame table set null, then the stack has been popped. Now frame k-1 is the current frame.

The frame table is allocated in option_seg.space when the segment (perm_op_list) is first created. The frame table is declared by:

```
dcl ft (p→option_seg.ftsize) bit (18) ctl (ftptr);
```

Ft(i) is a relative pointer to the most recent setting in frame i. If there are no settings in frame i, ft(i) is null.

In perm_op_list only ft (1) is meaningful.

Figure 1 shows the chaining of settings within a frame. The diagram shows only the relations of settings within frame. It is not intended to represent the actual structure of a setting (see below, option settings).

The Hash Table

Options can be accessed by name or by frame. To pop the stack one must access options by frame but usually options will be referenced by name. Since there is no practical limit to the number of options which the user may set, hash coding the names of options seems advisable. Further, it must be possible to discover quickly whether or not an option is set, and hashing can provide quick verification that a given name is not among the names of options which are set.

The hash table is initially allocated (by create_op_list) in the area space:

```
allocate ht in (ptr→option_seg.space) set (p);
ptr→option_seg.htptr = p;
```

The length of the hash table (htsize) is 30 when perm_opt_list is created. Such a hash table will accommodate 20 options (the ratio of the number of filled entries to the length of a hash table should not exceed 2/3). The hash table

can be expanded to accommodate a larger number of options (this is done by addopt when the number of options grows large - see BY.9.05).

The hash table is declared by:

```
dcl  1 ht (p→option_seg.htsize) ctl (htptr),
       2 d bit (1),
       2 v bit (1),
       2 pointer bit (18);
```

d - deletion bit. If an option was represented in this entry and is then deleted, normally the vacant bit is set to 1 and the d bit left unchanged (=0). However, if the next entry in the hash table is not vacant, the d bit in this entry is set to 1.

v - vacant bit, = 1 if the entry is vacant, = 0 if the entry is deleted or if the entry points to an option header.

pointer - relative pointer to an option header. If either d or v = 1, pointer is meaningless. The option header contains the name of the option (see below).

Representation of a single option

An option is represented in the options stack by a setting in each frame of the stack in which the user has set the option. The first setting of the option (i.e., the setting in the frame with lowest frame number) is called the option header. The header also contains the name of the option.

All settings of an option are double-chained to the header of the option by relative pointers. The header, in frame n_1, points to a setting in frame n_2, which points to a setting in frame n_3, and so on, where $n_1 < n_2 < n_3$. The last setting in this series is known as the current setting. The value of the current setting is the current value. Note that the current setting does not necessarily lie in the current frame. However, it does determine the current value, which is the value in the current frame.

Figure 2 represents a single option in the options stack. The diagram shows only the relations of settings within a single option. It is not intended to represent the actual structure of a setting (see below, Option settings).

To find the current value of the option, it is necessary to find the current setting. The header points directly to the current setting. Suppose the header is in frame 1, the next setting in frame 2, and the third (current) setting in frame 4. Suppose further that the current frame is 5. To find the value of the option in frame 3: the header points to the current setting (frame 4). The current setting points back to the setting before it (frame 2). Since the option is not set in frame 3, the value in frame 2 holds in frame 3.

Option Settings

A setting of an option (except for the header) has the following form:

```
dcl  1 setting ctl (setp),
     . 2 lset bit (18),
       2 nset bit (18),
       2 lopt bit (18),
       2 nopt bit (18),
       2 fno fixed,
       2 set,
       3 sw bit (1),
       3 spec bit (18);
```

lset - relative pointer to the previous setting of this option.

nset - relative pointer to the next setting for this option. In the current setting, nset is null (i.e., all zeros).

lopt - relative pointer to the previous setting (of another option) in the same frame. lopt is null in the first setting in a frame.

nopt - relative pointer to the next setting (of another option) in the same frame. nopt is null for the last setting in the frame.

fno - number of the frame in which this setting lies.

set - this substructure records the value of the option for this setting.

sw - switch for the option.

spec - relative pointer to a specification for the option.

The specification for an option is an adjustable character string:

```
dcl  1 spec ctl (specptr),
     2 nochar fixed binary (9),
     2 string char (specptr→spec.nochar);
```

nochar - the number of characters in the specification.

string - the character string representing the specification.

The header for an option has the following form:

```
dcl  1 header ctl (headp),
     2 nochar fixed bin (9),
     2 name char (headp→header.nochar),
     2 cset bit (18),
     2 nset bit (18),
     2 lopt bit (18),
     2 nopt bit (18),
     2 fno fixed,
     2 set,
     3 sw bit (1),
     3 spec bit (18);
```

nochar - number of characters in the name of the option

name - name of the option

cset - relative pointer to the current setting of this option. nset is null if the header contains the only setting of the option.

nset - relative pointer to the next setting of this option, nset is null if the header contains the only setting.

lopt - relative pointer to previous setting in this frame, lopt is null for the first setting in the frame.

nopt - relative pointer to next setting in this frame. nopt is null for the last (most recently set) setting in the frame.

Figure 1: Representation of a Frame

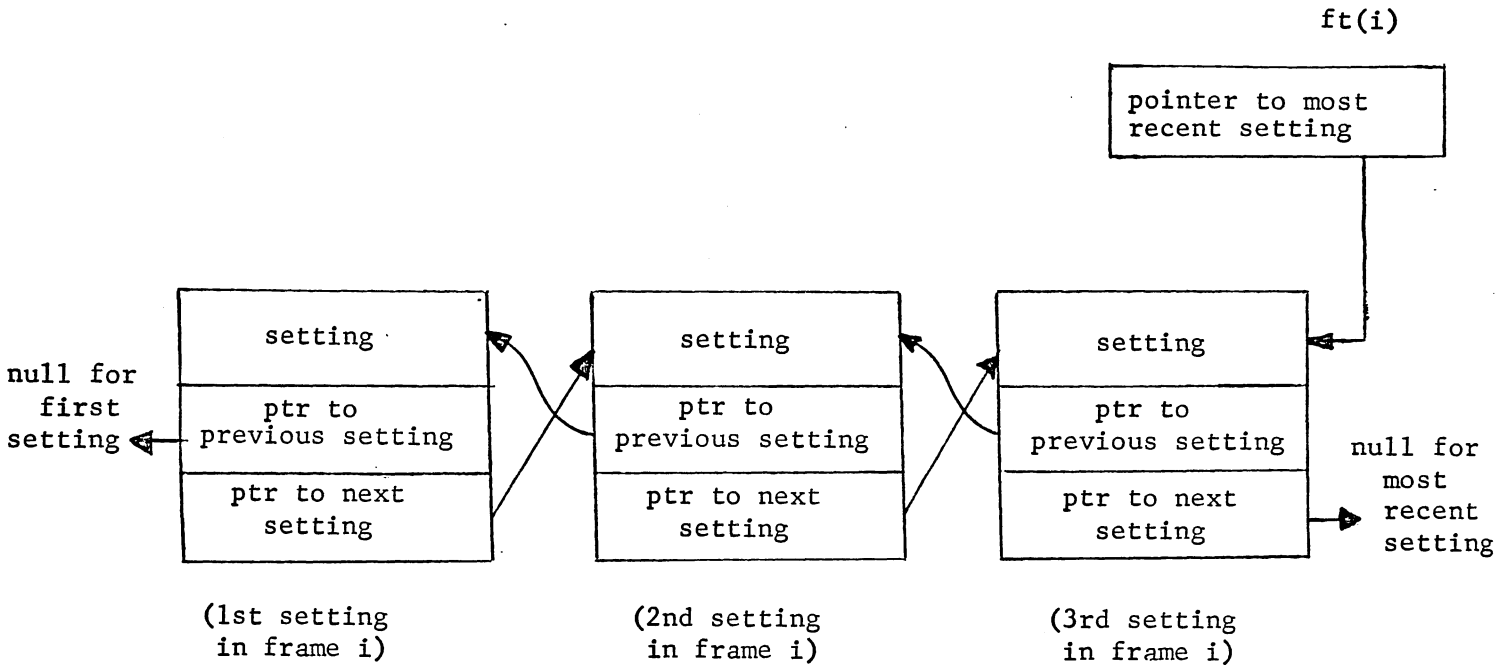

Figure 2: A Single Option