TO: MSPM Distribution
FROM: C. Marceau
SUBJ: Options
DATE: January 13, 1967

The following MSPM Sections, BX.12.00 - BX.12.02, represent a complete redesign of options in Multics. Previous BX.12 sections are obsolete and should be discarded.

Published: 01/18/67
(Supersedes: BX.12.00, 2/3/66)

Identification

The use of options in Multics - an overview
C. Marceau

Purpose

An option is a sort of argument to a procedure. It is like an argument in that it enables the user to exert some measure of control over procedures in execution. Unlike an argument, however, it need not be explicitly stated each time the procedure is invoked. This has 2 major consequences:

1) it is an obvious convenience to the user who invokes the procedure very often;

2) it enables the user to affect even those procedures which he does not call directly, viz. system procedures called by other system procedures called by other system procedures....

Section BX.12.00 presents an overview of options, what they can do and how to use them. It is divided into 3 parts:

A) what options are and what they do

B) the stacking of options

C) how to set and read options.

Section BX.12.01 discusses the representation of options in storage. It is very detailed and not of interest to the general user of options. BX.12.02 discusses usage and implementation of the option commands: option, delopt, and printopt. BY.9.01 - BY.9.05 discuss usage and implementation of the option procedures in the Multics library.

A. What options are and what they do

Definitions

In Multics, an option is a binary switch which the user may set on or off as he wishes. Some options also have a specification, a character string which provides additional information about the user's wishes. The combination of an on/off switch and a specification (if any) is called

the value of the option.

Since an option is only a switch with a specification, it has no meaning in itself. An option has meaning only when it can affect some procedure which checks the option and modifies its actions according to the value of the option. What an option means is defined by the procedure(s) which check it, not by the user who sets it. Thus if a user sets his "zilch" option on, but no procedure ever checks "zilch", then "zilch" has no meaning. This is an important point to remember, for it can happen that some procedure checks the option "glitch" even though the user has never set "glitch". Then "glitch" has a meaning-namely, the meaning given to it by the procedure which checks it.

An option like glitch is said to be unset. By convention, all procedures should act as though an unset option is off. (A subtle distinction between unset and off will appear later in this section.) In Multics, a record is kept only of options which are set, so that options overhead is kept to a minimum.

A user may set an option and sometime later cease to need the option-for example, he no longer uses the procedure which checks that option. He can then delete the option. Deleting an option is the opposite of setting it: when a user sets an option, he begins to keep a record of the value - on or off - of the option; when he deletes the option, he ceases to record any value for the option.

Note that deleting an option does not keep any procedure from checking the option; a procedure which tries to check the option will find it is unset, and should act as though the option were set off.

Thus an option may be set or unset. If set, it may be on or off. If an option is unset, no record is kept of it, and an off value is assumed.

Further, an option may be either local or global. A local option is an option checked by only one procedure. It has the meaning given to it by that procedure alone. It is uniquely defined.

A global option is an option checked by more than one procedure. It is therefore defined by more than one procedure. Needless to say, it is very desirable that all procedures which check a single option assign to it approximately the same meaning. Nonetheless, the detailed effect of the option on each procedure which checks it is bound to be different.

A global option is not useful if the procedures which check it assign to it wildly differing meanings. It is useful in defining, for example, a common mode of activity for a number of procedures.

Local value of a global option

Suppose that zilch is a global option, checked by procedures alpha, beta, and gamma. The user can set zilch on globally so that all three procedures consider it on. It is also possible for him to set it locally for procedure alpha, so that alpha considers the option on, but beta and gamma consider it off.

He refers to the local setting by concatenating the name of the procedure with a period and the option name:

```
alpha.zilch
```

When he sets "alpha.zilch" on, only alpha will consider zilch to be on.

Now suppose that the user wants zilch to be on for every procedure except alpha. Then he sets "zilch" on, and sets "alpha.zilch" off. Even though zilch is on, the local setting takes precedence over the global setting, and alpha considers zilch to be off.

Note that a global setting will hold for all procedures which check the option, unless a local setting specifically overrides it.

Thus if zilch is set on and is not set locally for alpha (i.e. alpha.zilch is unset), then alpha.zilch is not considered to be off. Alpha.zilch must be set in order to override the global setting of zilch. This is the exception, noted above, to the general rule that an unset option should be considered off.

A global option zilch is spoken of as one option, just as we speak of one mode in which several procedures operate. But the option is defined by several procedures, and hence we can speak of "the value of alpha.zilch" or say "beta.zilch" is unset. Zilch should be thought of as one option, but one which can have local settings and local values as well as its global setting and global value.

System Options

System options are options which commands and system procedures check. Hence they are defined by the system. They enable the user to exert some control over the way system procedures

operate. The user can set or delete system options (recall that deleting an option means that the procedure will consider it off.)

The options facility does not differentiate between system options and other options. The only distinction of system options is that system procedures define them.

System options, like any options, can be local or global. The user is free to check a global system option in any procedure of his own, but he should make sure that his procedure assigns a meaning to the option which is similar to its meaning in system procedures.

For example, "brief" is a global system option which means, if it is on, that only essential messages are to be printed at the (interactive) user's console. Brief curtails, for example, long chatty messages which many users do not want to read. Brief does not stop important messages, such as news of a fatal error in execution. Any user can have his procedures check "brief" and modify their actions according to the value of the "brief" option. The user can also set brief locally for his procedure.

Note that options are not magic. The user cannot set brief on and expect his procedure beta to operate in brief mode. Beta will operate in brief mode only if it is written so that it checks the brief option, and curtails printout if brief is on. All system procedures respect system options and follow the conventions described in this section.

Names of system options

Since system options are defined by system procedures, system option names are reserved. Suppose a system procedure checks the "sysglop" option and a careless user sets sysglop on without knowning what it means or even that the system procedure checks it. He is courting horrible disaster. MSPM section BB.4.05 lists the names of all local and global options defined by the system. The user must check this list before blithely setting an option.

B. The stacking of options

A record is kept of all options which are set (remember that an unset option is considered to be off and need not be recorded). We call the record of set options the options list.

In Multics the options list is stacked. That is, there is a push-down stack (the options stack), each frame of

which records the values of options which are to hold for some length of time. The options stack operates in a way similar to the temporary storage stack, but is independent of the temporary storage stack or any other stack in Multics.

The first frame of the options stack records "permanent" values of options. When the user wants an option to have a certain value each time he logs in, he sets the option in the first, or permanent, frame. Permanent frame values remain the same from one console session to another.

If the user wants to set an option for the duration of a single console session, he sets it in the second, or console session, frame. Values in this frame are valid only until logout.

The user can override a "permanent" value by setting the option in frame two. Thus if he usually wants "zilch" on, he sets it on in frame one. But one day he wishes to operate, for that day only, with "zilch" off. So he sets zilch off in frame two. The off value for zilch holds until he logs out. The next time he logs in, zilch is on. Values of options set in one frame hold in all later frames until explicitly changed.

Stacking options through the Shell

All commands which the user issues pass through the Shell, the command language interpreter (see BX.2.00 on the Shell). Each time the Shell calls a command, it pushes down the options stack. The user has the opportunity, when issuing a command, to set certain options for the duration of the command. If the user has just logged in, values which he gives to the options in a command line are set in frame three of the options stack. When the command is finished, the Shell pops up the stack to frame two.

For example, suppose the user writes

```
alpha arg [glitch]
```

to issue the command alpha with an argument arg. In square brackets he puts the name of the option he wants to set. In the Multics command language (see BX.1.00) [glitch] is an interjected command which sets the option glitch. For every system option there is a command by the same name which sets the option. Of course the user can also write simple commands which set options for his procedures. In the example above, "glitch" could be a user-defined

option, as well as the name of a procedure which the user wrote to turn "glitch" on. Or "glitch" could be a system option, set by a system-provided command.

The shell also handles stacking for immediate-value commands (see BX.1.00 for a discussion of immediate-value commands). Suppose the current frame of the options stack is 5, and the user issues the command line:

```
alpha arg  {beta [glitch]}
```

The Shell pushes down the options stack once for alpha and once for beta. The "glitch" option is not set in frame 6 (it retains whatever value it had before), but is set in frame 7. The values in frame 7 hold for the duration of command beta. When beta is completed, the shell pops the stack to frame 6 and calls alpha.

Suppose the user (again at stack frame 5) types

```
alpha arg [brief]  {beta [glitch]}
```

and assume that "brief" is a global option. In the Multics command language (see BX.1.00), this command line has the effect of setting "brief" on for both alpha and beta. The interjected command also sets "alpha.brief" on. (This is done because the user might have set alpha.brief off previously, and the value of alpha.brief if set takes precedence over the global value of brief, even if brief is set in a later frame.)

To avoid any possible confusion, please note that "global" does not mean "holding in all frames". A global option, like a local option, may be set in one frame, or many, or none. A global option is an option defined by more than one procedure.

Note: In the above example, the interjected command does not set "beta.brief". If the user has set beta.brief off, then beta will remain under control of the beta.brief option and will not operate in brief mode.

Further, the three following command lines are possible in the Multics command language:

1) [brief] alpha arg1

In this example, brief is set on, but alpha.brief is not set.

2) {get_command} [brief] arg1

In this example, get_command returns a character string, for example, "alpha", which is interpreted as a command name. Then brief and alpha.brief are set on, but get_command.brief is not set. Get_command is executed before the interjected command "[brief]" is executed.

3) [brief] {get_command} arg1

Only brief is set.

Interjected commands which set system options on can also be used to set them off. Thus if the user normally operates with brief on, but wants to execute command alpha with brief off, he types

alpha arg [brief off]

Normally the user need never concern himself with frames of the options stack. The Shell and other system procedures push the options stack down and pop it up at appropriate times. However, the more sophisticated user may wish to set options for very short periods of time, viz. within one group of procedures. When he does this he must push down the stack before setting the option(s), and pop up the stack later. System procedures are provided for pushing and popping the stack (see below, p. 8).

Note: options may be stacked to a virtually unlimited depth. To avoid catastrophe, the option procedures set "unlimited"=40. Cancerous stacking will result in an error condition should that depth be exceeded. The user can change the value of "unlimited" by setting the "option_stack" option on, with a specification which sets the maximum depth to which options may be stacked, e.g. "10" or "500".

MSPM section BX.12.01 discusses the representation of options in storage. It tells where the options stack is located in the file system hierarchy, and describes how the stacking of the options list takes place.

C. How to set and read options

The Multics system provides commands and other procedures to handle the setting and reading of options. The user does not have to manipulate any data bases. He calls a command or procedure to set or read his options.

The options procedures do not distinguish between user-defined options and system-defined options. They do distinguish between local and global options when reading options.

There are three commands to set and read options.

The option command is used to set options. It can change the value of an option permanently (i.e. in frame one) or for the duration of the console session (in frame two).

The delopt command is used to delete options. The option is deleted from all frames. After deletion an option is unset and procedures consider it to be off.

The printopt command prints the values of all (set) options in all frames of the options stack.

Section BX.12.02 tells how to use these commands and discusses their implementation.

Option handling procedures

To check and to set options a procedure calls one of the option-handling procedures. Read_opt is the procedure to check local options. Given the option name and a frame number, read_opt returns the value (switch and specification) of the option and tells whether the option is set. If the option is unset, read_opt returns an off value for the switch and no specification.

The procedure which checks global options is read_global. Read_global first checks for a local value of the global option. If the option is locally unset, read_global returns the global value for the option. Section BY.9.01 describes read_opt and read_global.

When the user wishes to set options from his own procedures, he calls on options procedures. Push_opt pushes down the options stack, pop_opt pops it up. These procedures are described in BY.9.02.

To set an option in a given frame, the user calls modset. Another primitive, modopt, sets the option in the given frame, and in all subsequent frames. If 5 is the current frame, and the user wants to change the value of zilch in frame 2, he calls modset. To change zilch in frames 2, 3, 4, and 5, he calls modopt. If zilch was previously unset, or if it is being set in the current frame, modset and modopt have the same effect. But if zilch was on in frame four, and the user calls modopt to set zilch off in frame 2, then zilch will now be off in frame 4. Modset and modopt are described in BY.9.03.

Three procedures give information about the options list:

option_frameno, which returns the number of the current frame of the options stack;

option_names, which returns the names of options which are set;

option_values, which returns values of an option (the values in all stack frames).

These three procedures are described in BY.9.04.

Other primitives are addopt and delete_opt, used primarily by other options procedures. The user will probably not need them. Addopt is used to set an option for the first time; delete_opt deletes an option from the options list (the user can do this by calling the delopt command). Addopt and delete_opt are described in BY.9.05.

Example

The user wishes to set certain options for the duration of his group of procedures phi, psi, and chi. All three check the option "help". Furthermore, chi should run with the "alone" option on only when it runs alone, hence should find "alone" off whenever it runs with phi and psi. In execution, phi calls psi and chi as closed subroutines.

Phi begins by pushing down the options stack through a call to push_opt. Phi then calls modopt to set "alone" on. (In this example, phi assumes that "help" already has the appropriate value.)

Chi checks the "alone" option by calling read_opt. All three procedures call read_global to check the "help" option.

Before returning to its caller, phi pops up the options stack. Section BY.9.03 shows how the above example might be coded, with calls to the appropriate option-handling procedures.