Published: 08/19/69

Identification

Standard Service System Command Standards
V.L. Voydock

The standard service system consists of a collection of carefully coded, well-maintained commands which are intended to be fast, reliable and easy to use. Potential candidates for the standard service system will be carefully examined. They must conform to the following standards:

1. All standards expected of any Multics command. (See MSPM Section BX.0.01.)

2. No varying character strings may be used, either explicitly or implicitly. Implicit uses of varying strings may be found by checking in the linkage section portion of the listing segment for calls to condition, reversion, varst_ and free_.

3. All uses of the segment free_ must be cleared in advance with the Programming Manager. Its use will be permitted only in exceptional cases. (The star convention handler and the translator interface routine ti_ have been cleared to use free_).

4. Creation of a separate data segment to circumvent restriction 3. Above (i.e., make your own "free" segment) must also be cleared in advance.

5. Only approved library subroutines may be called. (See MSPM Section BY.0.01 for a list of approved subroutines.) Note that names of approved subroutines end in "_".

6. Calls to the EPL run-time string package must be avoided whenever possible. Look for calls to stgop_ in the linkage section portion of the listing segment.

7. The working set of a command must be made as small as possible. To this end the command writer must be aware of the consequences of his actions. He should try to avoid EPL constructs which generate horrible code or calls to EPL run-time routines. The linkage section portion of the listing segment should be checked for unexpected references.

8. The routine ioa_ should be used whenever it is necessary to format an output line (e.g., "call ioa_ ("Segment not found: ^a", segname") where segname is a variable containing the name of a segment). See MSPM Section BS.13.45.

9. Any command for which the star convention makes sense must use the star convention. Any command for which the equals convention makes sense must use the equals convention. See MSPM Section BX.8.00. If the command accepts the star convention, the routine check_star_ (See BS.13.52) must be called to check the entry name for stars so that the full star-convention machinery will be invoked only when necessary. Commands which use the star convention handler should have an entry point which does not invoke the star convention machinery. This allows one command to call another without invoking un-needed machinery (i.e., it knows the arguments it is passing do not contain stars). It also allows the user to manipulate names with imbedded stars properly. For consistency this entry point name should be the command name concatenated with "_ns", e.g., rename$rename_ns.

10. Commands whenever possible must accept pathnames (not just entry names) as arguments. The routine expand_path_ (see BS.13.50) should be called to convert a relative pathname into an absolute pathname.

11. Any command for which multiple arguments make sense must accept multiple arguments.

12. All commands should have an abbreviated name consisting of the first letter of the first two or three syllables of its name (e.g., rename - rn, delname - dn, unlink - ul).

13. All command names and abbreviations must be cleared in advance.

14. When it is necessary to avoid ambiguity, control arguments must be preceded by a leading minus sign,

    a. fortran alpha list
    b. dp HARRY file1 file2 -del file3 ...

15. As much as possible, command arguments should be order independent.

16. A global options facility exists and should be used when appropriate. See MSPM Section BS.1.12.

17. Commands should check for error codes which have special meaning to them and print appropriate error messages or allow for user intervention if the error is recoverable. The subroutine com_err_ should be called if the command does not have a special interpretation for an error code. In all cases, error messages must contain the name of the command which generated them.

18. Commands should not be "too powerful", that is, typing errors should not cause disastous results. For example with the current remove command, "remove a>b" will delete segment b in directory a whereas "remove a> b" will remove the directory a. To remedy this, there should be two commands "delete", which deletes only non-directory branches and "delte_dir" which deletes only directory branches.

19. All commands which accept a variable number of arguments should obtain these arguments using the procedure ms_$fetch_arg. That is, procedure declarations of the form "command: proc (arg1,...,argn)" should not be used.

20. The routine cv_bin_ should be used instead of bin_dec and bin_oct to avoid use of string package.

And finally the great intangible:

21. "Commands should be designed with the user in mind." Calling sequences should be logical (e.g., the user should not have to remember that a "q" as a third argument to command "farfal" causes all files with second component name "fred" to be deleted, whereas a "?" in the same position suppresses this feature). Commands should allow user intervention when appropriate. For example, the delete command should allow the user to decide whether an RE file should be deleted, rather than forcing him to change the file's access

mode and re-submit the delete request (or worse, delete the file without warning). Judicious use of red console output is encouraged. It should be used to call attention to important or unusual occurrences. Remember, over-use destroys the whole purpose of red output -- a command which outputs everything in red may as well output everything in black. "Canned" messages printed by commands should not contain characters which come out as escape characters on IBM model 1050 and model 2741 consoles and on model 37 teletypes (e.g., "¢<segment¢> not found"is not an acceptable message).