To: MTB Distribution

From: Benson I. Margulies

Date: 24 January 1979

Subject: default start_up exec_coms

This MTB proposes enhancements to the process of searching for and executing start_up exec_coms.

The start_up exec_com has a crucial role in the user's environment. It fills the gap between those features of the environment which system programmers have deemed desirable in all processes and those features which may or may not be desirable in a particular process, by allowing the user to declare certain features desirable in all of their processes.

Unfortunately, the current facility is inadequate. While a user can specify the features of the environment that they wish to have in their processes, there is no way for project or system administrators to provide defaults for users who are too naive to create a start_up. Thus these people live in the aforementioned gap between the things that are part of the system-provided environment and those that the user wants temporarily. It is often an unfriendly environment indeed. Those of us who consult with naive users on a regular basis are commonly amazed at how inconvenient their lives are for want of a few simple commands executed at the beginning of their process.

Putting extra stuff in process_overseer_ is not the answer for at least two reasons: First, many of these features have many parameters that the user will probably want to change as they get more experienced. As an example consider accepting interactive messages. A site or project might decide that its users should accept messages by default. Once the user learns more about the facility, however, they will probably want to use the -brief option, perhaps not use it if it was the default, or use even use -call to do something more sophisticated.

Second, it is neither reasonable to expect, nor wise to encourage, every project administrator to open up the code for process_overseer_ and write pl1 to set up defaults for the project. The most obvious reason to avoid this would be that other changes to process_overseer_ would take a long time to percolate down to most users.

Thus it is neccessary to provide a mechanism with which project and system administrators can use command language to specify defaults for their users' environment.

The other observed lack in the facility is that is no way for project administrators to specify mandatory features of their users' environment without a private initial procedure.

Therefore this MTB proposes changes to the start_up exec_com facility to attain two goals:

- Allowing administrators (system and project) to set up the default environment of users who lack start_up exec_coms.

- Allowing those administrators to use start_up exec_coms to set up mandatory environments for their users without having to code, debug, and install a nonstandard process_overseer_.

The first goal is easily motivated. There are several features of the users' environment for which defaults can be useful at either a project or site level. A project might want to set extra search rules, or activate auditing, or abbrev, or accept messages. A site might decide to do similar things, especially if many of their users were on undelegated projects and wished issues like these to be dealt with for them.

A straightforward design that would meet this goal would be for the standard process_overseer_ to search as now for a start_up.ec in the homedir, but if it fails to find one it would try the projectdir and finally >sc1. This would give both project and site administrators the desired ability to provide a default environment without dealing in process overseers.

The second goal is more complicated. There are several ways in which a start_up exec_com could serve the function of a special process overseer. The most obvious example is the limited service subsystem, commonly referred to as the lss. A user in an lss has a normal Multics environment, but is restricted to a specified subset of the command language. There are two ways to enter an lss. The most commonly used method now is for the project administrator to give the user in question a special initproc. Once this is done, even the commands in the start_up exec_com must be in the list of legal commands.

The alternative method could be to give the enter_lss command, which would cause the lss to be entered on the next call to the listener. An administrator could write a start_up that would attach audit_ to user_i/o and then give the enter_lss command. The lss would then be entered. In general, the idea is to allow the administrator to do anything that can be done with command language with the security of a process overseer.

Part of a facility for accomplishing purpose is already provided in the form of the no_start_up attribute of the PMF. Failure of the project administrator to grant a user this attri-

bute is supposed to guarantee that the user executes their start_up exec by prohibiting the -ns option to the login command. Unfortunately, a user without the no_start_up attribute can always "escape" the start_up by quitting out of it. To make matters worse, the search scheme described above would conflict with this. If the start_up was provided as a project start_up, then the user would have to be denied the ability to create one of their own. If it was in the homedir, then the user could not have access to modify it. This would be a severe limitation, since the most common use would probably be things like lss's for text processing, in which the user has access to all editing tools (suitable for creating start_up exec_coms), and has a normal homedir.

While users at several sites have requested the ability to supply a mandatory start_up exec for their projects, it does not seem advisable to put the neccessary code into the standard process_overseer_. For one thing, most projects probably would not need it. For another, putting it in the default initproc would require the addition of keywords and PMF attributes, in addition to login control arguments, to control it. Therefore it seems reasonable to supply an alternate initproc as part of the standard system that would implement the mandatory project start_up. Then both this initproc and process_overseer_ could use the original searching scheme to find the personal start_up. The no_start_up keyword, useless though it might be, need not be changed at all. This is the proposed design.

two_start_up_overseer_

This initproc is used to provide a mandatory start_up exec_com in the project directory to be executed before the normal start_up exec_com. If this procedure is specified for a user's initial procedure it will do the following: 1) Set the user's search rules to the normal defaults. 2) Look for project_start_up.ec in the project directory (>udd>[user project]). If it is there, create an unclaimed signal handler that will print any associated error message and then terminate the process. If not, skip the next step. 3) Call the command processor to execute the project start_up. 4) Set up a normal unclaimed signal handler.(1) 5) Look for a user start_up called start_up.ec; first try the homedir, then the project dir, and then >sc1. If there isn't a start_up.ec in any of these places, print the motd. 6) Call the listener.

Both project and personal start_up's are called with two arguments: the first is either "login" or "new_proc", and the second is either interactive or absentee.

To give the user this procedure as an initial procedure, use the initproc keyword of the PMF like:

```
initproc: two_start_up_overseer_;
```

To make it the default for the project, use:

```
Initproc: two_start_up_overseer_;
```

To test a project start_up it is advisable to login using a user id with the v_init_proc attribute and specify the overseer using a login command like:

```
login Satan BlackMagic -po two_start_up_overseer_
```

Now if the start_up has an error which causes a fault, it will still be possible to log in and fix it by not supplying the -po control argument.

---

(1) Note that this means that any fault taken in the project start_up will terminate the user's process.