TO: Distribution

FROM: Gary C. Dixon

DATE: May 7, 1973

SUBJECT: A proposed extension to the Multics star and equals conventions

This bulletin proposes an extension to the current Multics storage system star and equals conventions. Most users and system programmers feel that some sort of extension is desirable, but few have been able to agree on what the extension should be. This memo proposes an extension which has been under consideration for several months, and which has gained initial approval.

The extension may be summarized as follows:

1) Allow at most one double asterisk (**) as any component of a star name.
2) Interpret a question mark (?) used as any letter of a star name component to mean "matches any character in this letter and component position".
3) Allow at most one double equal sign (==) as any a component of an equal name.
4) Interpret a percent character (%) used as any letter of a component in the second entry name of a pair to mean "represents the character in the first entry name which has the corresponding letter and component position".

Appendix I of this bulletin is a replacement for the MPM Reference Guide Section 1.5, which deals with "Constructing and Interpreting Names". The extension summarized above is described in more detail by the definitions and examples in this Reference Guide replacement section.

The extension will be implemented by rewriting the match_star_, check_star_, and equal_ subroutines. As a part of this implementation, the rules defining the acceptable formats for star and equal names will be changed slightly. The purpose of this change is to minimize the amount of processing performed by the subroutines mentioned above, to remove ambiguities which would otherwise arise in the extended conventions, and to further enforce system naming conventions.

The following changes to star name format will result from the extension:

A) Question marks (?) will be interpreted as special characters, according to the extension summarized above.
B) Less than (<) and greater than (>) characters will not be permitted in star names.
C) No more than two asterisks will be allowed in any component of a star name.
D) Asterisks will not be allowed in any component of a star name which also contains non-asterisk characters.
E) Spaces will be allowed in star names. They will be interpreted as normal characters.

Currently, some users employ the undocumented feature in which the star convention accepts star names with: 1) a component containing three or more asterisks; or 2) a component containing asterisks and non-asterisk characters. The asterisks in these components are currently interpreted as question marks will be under the extended star convention. We will attempt to minimize the affect of changes C and D on such users by disabling these changes for the first few months of extended star convention operation, and interpreting these stars as if they were question marks. After the users have had several months to learn the extended convention, we can then enable Changes C and D.

The following changes to the format of equal names will result from the extension:

F) Percent characters (%) will be interpreted as special characters, according to the extension summarized above.
G) Less than (<) and greater than (>) characters will not be permitted in equal names.
H) No more than two equal signs will be permitted in any component of an equal name.
I) Equal signs will not be permitted in any component of an equal name which also contains non-equal sign characters.

In addition, the interpretation of one particular class of equal names will differ under the extension. Currently, the MPM states

> A double equal sign [used] as the rightmost component of the second entry name of a pair is equivalent to the corresponding component in the first entry name and any components following it;

and

> Any components appearing after the double equal [sign] are ignored.

As a result, the command

```
rename  a.b.c  d.==.f
```

is now interpreted as

```
rename  a.b.c  d.b.c
```

The extension will change this interpretation to be

```
rename  a.b.c  d.b.c.f
```

By not discarding the f which follows the double equal sign, the interpretation under the extended convention is more compatible with a similar use of the double asterisk under the star convention. This change in interpretation should not affect many users. Ample notification of all of the above changes will be given to users before the extension is installed.

Note that equal names cannot include spaces now, and spaces will not be permitted in equal names under the extension either. The reason is that equal_ receives only pointers to its arguments. It must compute the length of the first entry name and the equal name, according to the algorithm.

```
length = index (substr (name, 1, 32), " ");
if length = 0 then
     length = 32;
```

The calling sequence of equal_ would have to be changed to permit blanks in the entry name or equal name. Such a change is difficult to perform because equal_ is called in an unknown number of system and user programs.

Appendix II of this bulletin shows the changes proposed for match_star_'s calling sequence to allow spaces within star names. This change can easily be made because match_star_ is currently an internal interface available only in Ring 0, and called only by star_ (which implements hcs_$star_). As part of the extension, I plan to make match_star_ available to system and user programs which operate in Rings 1 through 5 (eg, archive).

The calling sequence of check_star_ does not have to be altered. Fortunately, this sequence includes the length of the star name to be checked, as well as a pointer to the name.

I would appreciate receiving your comments on this proposal. Comments may be forwarded by any of the following methods:

PHONE:
MIT ext 3-3224 or 253-3224

IPC COURIER:
GDixon's bin, Bldg 39

INTERDEPARTMENTAL MAIL:
G. Dixon
Rm 39-584

6180 MULTICS MAIL:
mail comment GDixon Multics

Appendix I: A Replacement for the MPM Reference Guide Section 1.5

CONSTRUCTING AND INTERPRETING NAMES

The various types of names used on Multics are constructed and interpreted according to certain conventions. The names in question are user names, segment names, command names, subroutine names, I/O stream names and condition names.

User names are discussed in the MPM Reference Guide section, Access Control, since they are primarily used to specify access control information.

A segment may be named in two ways. Its location in the storage system hierarchy is specified by its path name. The name by which it is known in a process is its reference name. The star convention and equals convention provide short hand methods of specifying segment names. Offset names allow specification of externally known locations in a segment.

Path Names

As described in the MPM Introduction Chapter 3, Beginner's Guide to the Use of Multics, each segment (or directory or link) in the Multics storage system has an entry in a superior directory. Any segment (or directory or link) may be found by following the appropriate entries from a designated directory through inferior directories until the desired segment (or directory or link) entry is reached. An absolute path name is just such a sequence of entry names starting from the root directory. A relative path name is a sequence relative to the current working directory. Path names, whether relative or absolute, are typically used as arguments to commands and subroutines.

An entry name is a string of 32 or fewer ASCII characters. Only the greater-than (>) and less-than (<) characters are not allowed in entry names, since they are used to form path names as described below. Several other characters are not recommended for entry names -- asterisk (*), question mark (?), equal sign (=), percent (%) and dollar sign ($) -- because standard commands attach special meanings to them. Each is explained below.

In general, entry names will consist of the upper- and lower-case alphabetic characters, the digits, the underscore (_) and the period (.), and must have at least one nonblank character. The underscore is used to simulate a space for

readability; e.g., a segment might be named new_seg. (Including a space in an entry name is permitted, but is cumbersome since the command language uses spaces to delimit command names and arguments.) The period is used to separate components of an entry name, where a component is a logical part of the name. Several system conventions depend on components. For example, compilers on Multics expect the language name to be the last component (or suffix) of the name of a source segment to be compiled; e.g., square_root.pl1 for a PL/I source segment.

An absolute path name is formed from a sequence of entry names, each preceded by a greater-than character. The initial greater-than indicates that the entry name following it designates an entry in the root directory. Thus, an absolute path name has the form >first_dir>second_dir>third_dir>my_seg.

The directory first_dir is immediately inferior to the root, second_dir is an entry in first_dir, etc. A maximum of 16 levels of directories is allowed from the root to the final entry name. The number of characters in the path name may not exceed 168. Each intermediate entry in the chain may be either a directory or a link to a directory. The final entry may be a directory, a segment or a link.

A relative path name looks like an absolute path name except that it does not contain a leading greater-than character, and may begin with less-than characters as explained below. It is interpreted by various commands to be a path name relative to the user's working directory. The simplest form of relative path name is the single name of an entry in the user's working directory. For example, the relative path name alpha refers to the entry alpha in the user's working directory. On a slightly more complex level, the relative path name sub_dir>beta refers to the entry beta in the directory sub_dir which is immediately inferior to the user's working directory.

The less-than character may be used at the front (left end) of a relative path name to indicate that the directory immediately superior to the working directory is where the following entry name is to be found. This principle may be extended so that several less-than characters cause the superior directory several levels higher than the working directory to be searched for the first entry name in the relative path name.

In the following examples, the user's working directory is

>dir1>dir2>dir3>dir4

A relative path name of

new_seg

would designate the segment with the absolute path name

>dir1>dir2>dir3>dir4>new_seg

A relative path name of

dir5>old_seg

would designate the segment

>dir1>dir2>dir3>dir4>dir5>old_seg

A relative path name of

<dir0>newer

would designate the segment

>dir1>dir2>dir3>dir0>newer

A relative path name of

<<<sample_dir>game_dir>chess

would designate the segment

>dir1>sample_dir>game_dir>chess

The star convention

The asterisk character (loosely called a star) and the question mark are used to designate groups of entries (in a single directory) which have similar names. This convention is applicable only in the final entry name of a path name. An asterisk used as any component of an entry name matches (i.e.,

designates) any character string in that component position. Thus, a set of entries is specified. For example, the entry name

```
*.pl1
```

designates all two-component entries in the user's working directory which have pl1 as the second component;

```
sub_dir>my_prog.new.*
```

designates all three-component entries in the directory sub_dir (which is immediately inferior to the working directory) which have my_prog.new as the first and second components; and

```
*
```

and

```
*.*
```

designate, respectively, all one-component and two-component entries in the working directory.

A double asterisk used as any component of an entry name matches any number of components (including zero) in that component position; however, only one double asterisk component is permitted in each name. For example,

```
my_prog.**
```

designates all segments with my_prog as the first (and possibly only) component;

```
*.my_seg.**
```

designates all segments with two or more components of which the second is my_seg; and

```
**.pl1
```

designates all segments with pl1 as the last (and possibly only) component.

The entry name ** designates all entries in the specified directory.

A question mark used as any letter of a component matches any character in that letter position and component position. For example, the entry name

ad?

designates all three-character one-component entries in the user's working directory which begin with ad;

!???????????????

designates all 15-character one-component entries in the user's working directory beginning with ! (names of this form are called unique names, and are created by the unique_chars_ subroutine); and

sub_dir>prog?.**.pl1

designates all entries in the directory sub_dir (which is immediately inferior to the user's working directory) with two or more components, the first of which has 5 characters and begins with prog, and the last of which is pl1.

The main use for the star convention is to perform commands on a set of entries with similar names; e.g., delete all segments with a first component of square_root or list all PL/I source segments.

Although names containing two or more consecutive periods are permitted as entry names, their interpretation under the star convention has no meaning. Commands which invoke the star convention will reject such names. (If an entry with such a name is accidentally created, the fs_chname command may be used to rename the entry.)

The Equals Convention

Some commands (eg, rename) deal with pairs of entry names as arguments. The equals convention allows an equal sign or a percent character to be used in the second entry name of a pair to represent one or more characters from the first entry name. An equal sign used as a component of the second entry name means that the character string which forms the corresponding component of the first entry name is to be substituted for the equal sign. For example,

```
rename  random.data_base  ordered.=
```

is equivalent to

```
rename  random.data_base  ordered.data_base
```

and

```
rename  random.data.base  =.=
```

is equivalent to

```
rename  random.data.base  random.data
```

The command

```
rename  *.data_base  =.data
```

renames all two-component entry names with data_base as the second component to have, instead, the second component data.

If an equal sign appears in a component for which there is no corresponding component in the first entry name, then the equal sign in the second name is discarded. That is,

```
rename  alpha  beta.=.gamma
```

is equivalent to

```
rename  alpha  beta.gamma
```

A double equal sign used as any component of the second entry name of a pair represents any number of components (including zero) in the corresponding component position of the first entry name; however only one double equal sign is permitted per name. For example,

```
rename  one.two.three  1.==
```

is equivalent to

```
rename  one.two.three  1.two.three
```

and

```
rename  one.two.three.four  1.==.4
```

is equivalent to

```
rename  one.two.three.four  1.two.three.4
```

The command

```
addname  **.ec  ==.absin
```

adds one name to each entry with a name whose last component is ec. The last component of this new name is absin, and the first components (if any) are the same as those of the name ending in ec.

If the first entry name of a pair does not contain any components which correspond to a double equal sign in the second entry name, then the double equal sign is discarded. For example,

```
rename  able  ==.baker
```

is equivalent to

```
rename  able  baker
```

A percent character used as any letter of a component of the second entry name of a pair represents the character of the first entry name which is in the corresponding letter and component position. For example,

```
rename  *.data  %%%.data
```

will rename all two-component entry names with a last component of data to have a first component which has been truncated to three (or fewer) letters.

Although names containing two or more periods are permitted as entry names, their interpretation under the equals convention has no meaning. Commands which invoke the equals convention will reject such names. (If an entry with such a name is accidentally created, the fs_chname command may be used to rename it.)

Reference Names

Procedures executing in a process need to refer by name to other segments known in that process. Such a name is a reference name. A reference name may be the same as an entry name of the segment, or may be different. For example, when a dynamic linkage fault occurs for a reference name, the linker searches (using search rules) for a segment which has an entry name identical to that reference name. A procedure call, an invocation of a command through the command processor, or a reference to an external data segment is of this type, as is a segment made known by the hcs_$make_ptr subroutine. Search rules (telling which directories to search for the entry name) may be specified by the user or may be system defaults. The default search rules are described in the MPM Reference Guide section, The System Libraries and Search Rules. Alternatively, the user may explicitly designate the reference name to be associated with a specified segment. The initiate command and the hcs_$initiate and hcs_$initiate_count subroutines perform this function. In this case, the reference name need not have any similarity to any entry name of the segment.

Since a reference name is associated only with segments made known in a process, the same reference name may be used in two different processes to refer to two different segments. Also, a reference name/segment binding exists only for the duration of the process in which it is specified. It is possible to break

that binding by terminating the segment, thus causing all links to that segment to be unsnapped and causing the segment to no longer be known in the process (by any reference name). The reference names of a terminated segment may be used again in the process to refer to a different segment. (See the write-up for the terminate command and the term_ subroutine.)

Individual reference names may be unbound in a process without terminating the segment unless the reference name removed was the only one on the segment. Note that no links are unsnapped so that previous connections made to a segment using that reference name remain in force.

Offset Names

Procedures frequently have more than one entry point, and data segments frequently have internal locations which are known externally by symbolic name. The names of the entry points and the internal locations are called offset names. Both designate symbolically an offset within the segment. The location specified may be referenced by the construction ref_name$offset_name where the dollar sign separates the reference name and offset name.

In many cases the entry point to a procedure has the same name as the segment itself (or the segment has several entry names corresponding to the names of its entry points). A shorthand notation allows the offset name to be assumed to be the same as the reference name. For example,

```
call square_root (n);
```

is interpreted to mean

```
call square_root$square_root (n);
```

and the command line

```
rename a b
```

is equivalent to

rename$rename a b

It is worthwhile to remember that if the user has renamed one of his procedure segments (perhaps to preserve an old copy) or has linked to a segment using a different name, he must thereafter use the full reference name/offset name construction when referencing that segment as a procedure or external data segment. It is also important to note that if a reference name/segment binding has been established in a process, then merely renaming the segment will not break the association in that process. To do this, the segment must be terminated.

Command, Subroutine, Condition and I/O Stream Names

These names all have some conventions in common.

1) Each is permitted to be not more than 32 characters in length.

2) All ASCII characters are legal in any position except as noted in points 3 and 4 below.

3) System subroutine names will end in an underscore to prevent conflicts with subroutine names given by users. (I.e., the user may easily avoid conflicts by refraining from having an underscore as the last character of his subroutine names.)

4) Condition and I/O stream names which are part of the system should end in an underscore to help prevent conflicts with names given by users. A glance at the MPM Reference Guide sections, List of System Conditions and Default Handlers, and List of Names with Special Meanings, reveals many system condition and I/O stream names which do not observe this convention. These names were incorporated into the system before this convention was established, and changing them would be difficult.

 (End)

Appendix II: Proposed calling sequence for match_star_

match_star_

Subroutine Call
05/07/73

Name: match_star_

This procedure implements the Multics storage system star convention by comparing an entry name with a name containing stars or question marks (called a star name). Refer to the MPM Reference Guide Section 1.5, "Constructing and Interpreting Names", for a description of the star convention and a definition of acceptable star name formats.

Usage

```
dcl  match_star_ entry (char(*), char(*), fixed bin(35));

call match_star_ (star_name, entry_name, code);
```

1) star_name is a star name. (Input)

2) entry_name is the entry name to be compared with the star name. (Input)

3) code is a status code which may be:

error_table_$match
the entry name matches the star name.

error_table_$nomatch
the entry name does not match the star name.

error_table_$badstar
the star name does not have an acceptable format. (Output)

 (End)