TO: Distribution

FROM: Gary C. Dixon

DATE: June 27, 1973

SUBJECT: The Star/Equals Extension Revisited

MSB-99 proposed an extension to the current Multics storage system Star and Equals Conventions. The comments I received on the proposal pointed out a few mistakes in the MSB, highlighted some ambiguities in the MSB and in my own thinking, and suggested several further extensions to the conventions. The attached IPC publication attempts to correct the mistakes, clarify the ambiguities, and to present an additional extension to the conventions. It represents the final specifications for the implementation of the extension.

I have also attached tentative descriptions of the new entry points which will implement the extensions. These descriptions are subject to change before the extension is implemented, but will provide an idea of the facilities which will be available as a part of the extension.

The gate entries, hcs_$star_ and hcs_$star_list_, will still be used to list directory entries matching a particular star name. These gate entries will not change. A new procedure, match_star_name_, will compare entry names with a star name. This procedure will be available in Rings 0 through 5, so that it may be called by user commands and subsystems which wish to implement their own processing of star names (eg, the archive command could list table entries for the components which match a star name). It is a replacement for the match_star_ procedure, which is called by hcs_$star_ and is currently accessible only from Ring 0. The procedure check_star_name_ will validate star names, and it will indicate whether or not they contain stars or question marks. This procedure is a replacement for check_star_, which will become an obsolete interface, but which will be modified to accept extended star names so that programs which call check_star_ can be converted gradually. The procedure get_equal_name_ will implement the Equals Convention. It will be available in Rings 1 through 5, and will replace equal_. equal_ will become an obsolete interface, but will be modified to accept extended equal names (as long as they do not contain spaces) so that programs which call equal_ can be converted gradually.

I would appreciate receiving your comments on this extension. Written comments may be forwarded to me as follows:

By IPC Courier:
GDixon's bin, Bldg 39

By MIT Interdepartmental Mail:
G. Dixon
Room 39-584

By 6180 Multics Mail:
mail comment GDixon Multics

Star/Equals Convention Extension
Temporary Operating System Series
TOS-P

MSB-103

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
INFORMATION PROCESSING CENTER

June 27, 1973

CHANGES TO THE 6180 MULTICS STAR AND EQUALS CONVENTIONS

In the near future, the 6180 Multics Star and Equals Conventions will be changed to extend the flexibility and usefulness of star and equal names. This memo describes the extensions by presenting a set of rules for constructing and interpreting extended star and equal names, by summarizing how the extensions differ from the current Star and Equals Conventions, and by giving examples of how the extended star and equal names may be used. The memo also outlines our plan for installing the extension.

The portion of the Star and Equals Conventions which is currently installed and available to users is described in the Multics Programmers' Manual Reference Guide Section 1.5. We begin with a description of extended star names.

Extended Star Names

Many commands which accept path names as input allow the final entry name in the path to be a star name. A star name is an entry name which identifies a group of entries in a single directory. Commands which accept star names perform their function for each directory entry identified by the star name.

A star name identifies all entries in a directory having an entry name which matches the star name. A special type of matching is performed in which some character strings of the star name are compared with corresponding strings of the entry name, while other character strings of the entry name are ignored. If the star name strings match the entry name strings, then the entry name matches the star name. Therefore, the entries identified by a star name all have similar names.

Under the extended Star Convention, the matching will be performed according to the rules for constructing and interpreting star names listed below:

1) A star name is an entry name. Therefore, it is composed of a string of 32 or fewer ASCII printing graphics or spaces, none of which may be the less-than (<) or greater-than (>) character.

2) A star name is composed of one or more non-null comoonents. This means that a star name may not begin or end with a period (.), and may not contain two or more consecutive periods.

3) Each question mark (?) character appearing in a star name component is treated as a special character. The question mark matches any character which appears in the corresponding component and letter positions of the entry name.

4) Each asterisk (*) character appearing in a star name component is treated as a special character. (1) The asterisk matches any number of characters (including zero) appearing in the corresponding component and letter positions of the entry name. Only one asterisk may appear in each star name component, except for a double star component as noted in the next rule.

5) A star name component consisting only of a double star (**) is treated as a special component. The double star component matches any number of components (including zero) in the corresponding component position of the entry name. Only one double star component may appear in a star name.

Note that the rules above do not require that star names contain asterisks or question marks. Therefore, an entry name which does not contain either of these special characters can be used as a star name, as long as it does not contain any null components. Note too that the rules above impose no restrictions on the form of the entry names to be matched with the star name. Such names may contain null components which will only match star name components of * or **.

The following examples illustrate some common forms for star names. The entry name

```
*.pl1
```

identifies all two-component entries in the user's working directory which have pl1 as their second component; the path name

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

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

---

(1) Asterisk characters are loosely referred to as stars in the remainder of this memo.

*

and

.

identify, respectively, all one-component and two-component entries in the working directory. The entry name

my_prog.**

identifies all entries with my_prog as the first (and possibly only) component;

*.**.my_seg

identifies all entries with two or more components of which the last is my_seg;

**.pl1

identifies all entries with pl1 as the last (and possibly only) component; and

**

identifies all entries in the user's working directory. The entry name

prog*.pl1

identifies all two-component entries whose first component begins with prog and has four or more characters, and whose second component is pl1;

*_data

identifies all one-component entries whose first component ends with _data and has five or more characters; and

interest_*_data.*.*

identifies all three-component entries whose first component begins with interest_, ends with _data, and has fourteen or more characters. Finally, the entry name

ad?

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

```
!?????????????
```

identifies all fifteen character one-component entries beginning with !; and

```
sub_dir>prog?.**.pl1
```

identifies 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 five characters and begins with prog, and the last of which is pl1. (2)

Extended Equal Names

Some commands which accept pairs of path names as their arguments (e.g., the rename command) allow the final entry name of the first path to be a star name, and the final name of the second path to be an equal name. An equal name is an entry name containing special characters which represent one or more characters from the entry names identified by the star name. Commands which accept equal names provide a powerful mechanism for mapping certain character strings from the first path name into the second path name of a pair. Such a mechanism helps to reduce the typing required for the second path name, and it can be essential for mapping character strings from the entry names identified by the star name into the equal name, because these character strings are not known when the command is issued.

Under the extended Equals Convention, the mapping of character strings from the star name into the equal name will be performed according to the rules for constructing and interpreting equal names given below:

6) An equal name is an entry name. Therefore, it is composed of a string of 32 or fewer ASCII printing graphics or spaces, none of which may be the less-than (<) or the greater-than (>) character.

7) An equal name is composed of one or more non-null components. This means that an equal name may not begin or end with a period (.), and may not contain two or more consecutive periods.

---

(2) A name which matches the star name !????????????? is called a unique name because the unique active function and the unique_chars_ subroutine create names of this form which are guaranteed to be unique in the Multics Storage System.

8) Each percent (%) character appearing in an equal name component is treated as a special character. The percent represents the character in the corresponding component and letter position of the entry name identified by the star name. An error occurs if the corresponding character does not exist.

9) Each equal sign (=) appearing in an equal name component is treated as a special character. The equal sign represents the corresponding component of the entry name identified by the star name. An error occurs if the corresponding component does not exist. An error also occurs if an equal sign appears in a component which also contains a percent character. Only one equal sign may appear in each equal name component, except for a double equal sign component, as noted in the next rule.

10) An equal name component consisting only of a double equal sign (==) is treated as a special component. The double equal sign component represents all components of the entry names identified by the star name which have no other corresponding components in the equal name. From this definition, it follows that if the double equal sign component represents (i.e., corresponds to) any components of the entry name identified by the star name, then the equal name will have the same number of components as the entry name. Only one double equal sign component may appear in an equal name.

Note that the rules above do not require that equal names contain equal signs or percent characters. Therefore, an entry name which does not contain either of these special characters can be used as an equal name, as long as it does not contain any null components. Note too that the rules above impose no restrictions on the form of the entry names identified by the star name. These names may contain null components. However, the rename and addname commands cannot be called with an entry name which contains null components, because these commands treat their arguments as either star names or equal names. The fs_chname command may be used to rename entries if names containing null components are accidentally created.

The following examples illustrate how equal names might be used in rename and addname commands. The command

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

is equivalent to

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

```
addname  world.data  =.statistics  =.census
```

is equivalent to

```
addname  world.data  world.statistics  world.census
```

The command

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

is equivalent to

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

The star convention is used in the command

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

to rename all two-component entry names with data_base as their second component to have, instead, a second component of data. The command

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

is in error because the first name of the pair does not contain a component corresponding to the equal sign in the second name. The command

```
rename  program.pl1  old_=.=
```

is equivalent to

```
rename  program.pl1  old_program.pl1
```

and

```
addname  data  first_=_set
```

is equivalent to

```
addname  data  first_data_set
```

The next rename command, which contains a double equal sign component,

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

is equivalent to

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

and

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

is equivalent to

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

Note that, in the two examples above, the first name has components which are represented by the double equal sign in the second name of each pair. As a result, the number of components represented by the equal name is the same as the number of components in the first name. On the other hand, in the command

```
addname  able  ==.baker.charlie
```

which is equivalent to

```
addname  able  baker.charlie
```

the double equal sign does not represent any component of the first name. Component able of the first name is represented in the equal name by baker. As a result, the equal name represents a greater number of components than there are in the first name. The command

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

uses the star convention to add a 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. Finally, the command

```
rename  ???*.data  %%%.=
```

renames all two-component entry name which have a last component of data and a first component containing three or more characters to have a first component which has been truncated to the first three characters. Note that the command

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

may result in an error if the first component of any name matching *.data has less than three characters.

Changes in the Conventions

The rules stated above for the construction and interpretation of star names embody the following changes from the current Star Convention:

A) Under the extension, each character S of a star name will have to meet the following requirements:

```
" " <= S <= "~"
     S ~= "<"
     S ~= ">"
```

Currently, the programs which implement the Star Convention do not enforce these requirements. (3)

B) Spaces will be allowed in star names. Currently, the star name is assumed to end when the first space is encountered.

C) Question marks will be interpreted as special characters. Currently, they are interpreted as normal characters.

D) At most, only one asterisk will be allowed in each star name component, except for a double star component. This asterisk will be interpreted in the same way, whether or not other characters appear in the component. An undocumented function of the program which implements the current Star Convention interprets the asterisks in star names of the form, !********** or ***, as matching any character which appears in the corresponding component and letter positions of the entry name.

E) A double star component will be permitted as <u>any</u> component of the star name. Currently, it is only permitted as the last component.

The rules stated above for the construction and interpretation of equal names embody the following changes from the current Equals Convention:

F) Under the extension, each character S of an equal name will have to meet the requirements listed in change A above. Currently, the program which implements the Equals Convention does not enforce these requirements. (4)

---

(3) Remember that a star name is the <u>final</u> entry name of a path. Entry names may not contain less-than or greater-than characters.

(4) Remember that an equal name is the <u>final</u> entry name of a path. Entry names may not contain less-than or greater-than characters.

G) Spaces will be allowed in equal names. Currently, the equal name is assumed to end when the first space is encountered.

H) Percent characters will be interpreted as special characters. Currently, they are interpreted as normal characters.

I) An equal sign will be permitted in an equal name component which contains non-equal sign characters. Currently, this is not allowed.

J) A double equal sign will be permitted as any component of the equal name. Currently, it is only permitted as the last component.

In addition, two unusual uses of the Equals Convention which are currently in error, but which are documented as working in the Multics Programmers' Manual, will work differently under the extension. Currently, the MPM states:

> If an equal sign appears in a component for which there is no corresponding component in the first entry name, then that component (the equal sign) in the second name is discarded. That is,
>
> rename alpha beta.=.gamma
>
> is equivalent to
>
> rename alpha beta.gamma

This statement is incorrect. The current implementation of the Equals Convention declares an equal name to be in error if the first entry name does not contain a component which corresponds to the equal sign in the second entry name. The rules for constructing extended equal names given above correctly document such use of the Equals Convention as being in error.

The second MPM documentation problem is similar in nature. 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. For example,
>
> rename aa.bb.cc dd.==.ff
>
> would result in the entry dd.bb.cc since the ff is dropped.

This statement is incorrect. Currently, equal names of the form dd.==.ff are in error because the double equal sign may only appear in an equal name as the final component. Under the extension, the command

rename aa.bb.cc dd.==.ff

will be a legal command which is equivalent to

rename aa.bb.cc dd.bb.ff

Minimizing the Effects of the Changes

Changes B, E, G, I, and J are new features which are upwards compatible with the existing Star and Equals Conventions. These changes should not cause problems for users.

Changes A and F enforce published restrictions on the set of characters which may be included in entry names. Therefore, these changes can only cause problems for users attempting to use star and equal names to manipulate directory entries which are improperly named. Hopefully, any problems which arise will help to convince these users to follow Multics segment naming conventions.

Change D will have a serious affect on those users who make use of undocumented star names of the form: !********** or ***. Rule 4 above will prohibit star names which have more than one star in each component (with the exception of a double star component). To minimize the affect of this change, we intend to install the extension to the Star Convention in two phases. During the first phase, a partial extension will be installed. It will not implement Rule 4 above, but instead will treat the stars in star names of the form shown above as if they were question marks. During phase one, users will have the

opportunity to change their habits and their exec_coms to use question marks instead of the stars. After users have had about two months for this conversion, phase two will be installed. This phase will implement the full extension described above, including Rule 4.

Changing the two features of the current Equals Convention which are documented incorrectly in the MPM will not affect any users. These features cannot currently be used as they are documented because an error results.

The program equal_, which implements the Equals Convention, will be replaced as part of the extension by get_equal_name_, a program having a calling sequence which more closely follows Multics system programming standards. equal_ will become an obsolete interface, but a version of equal_ which accepts extend equal names not containing spaces will still exist. After get_equal_name_ has been installed, users can gradually change their programs to use get_equal_name_ rather than equal_.

Name: match_star_name_

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

```
declare match_star_name_ entry (char(*), char(*),
        fixed bin(35));

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

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

2) star_name is the star name it is to be compared with. (Input)

3) code is a status code which may be:

0 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)

Notes

Refer to the MPM writeup for the ncs_$star_ routine to see how to list the directory entries which match a given star name.

Refer to the MPM writeup for the check_star_name_ routine to see how to validate a star name.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (End)

Name: check_star_name_

This procedure validates an entry name to insure that it has been formed according to the rules for constructing star names. These rules are given in the MPM Reference Guide Section 1.5, "Constructing and Interpreting Names". It also returns a status code which indicates whether the entry name contains asterisks or question marks, and whether it is a star name which matches every entry name.

Entry: check_star_name_$path

This entry point accepts an absolute path name as its input. It validates the final entry name in that path, as described above.

Usage

```
declare check_star_name_$path entry (char(*),
        fixed bin(35));

call check_star_name_$path (path_name, code);
```

1) path_name is the path name whose final entry name is to be validated. Trailing spaces in the path name character string are ignored. (Input)

2) code is one of the following status codes. (Output)

0 the entry name is valid, and does not contain stars or question marks.

1 the entry name is valid, and does contain stars or question marks.

2 the entry name is valid, and is a star name which matches every entry name. This means that the entry name is either "**", or "*.**", or "**.*".

error_table_$badstar
the entry name is invalid. It violates one or more of the rules for constructing star names.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Entry: check_star_name_$entry

This entry point accepts, as input, the entry name to be validated.

Usage

```
declare check_star_name_$entry entry (char(*),
        fixed bin(35));

call check_star_name_$entry (entry_name, code);
```

1) entry_name is the entry name to be validated. Trailing blanks in the entry name character string are ignored. (Input)

2) code is as above. (Output)

Notes

Refer to the MPM writeup for the hcs_$star_ routine to see how to get a list of directory entries which match a given star name.

Refer to the MPM writeup for the match_star_name_ routine to see how to compare an entry name with a given star name.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (End)

Name: get_equal_name_

This procedure accepts an entry name and an equal name as its input, and constructs a target name by substituting components or characters from the entry name into the equal name, according to the Multics Equals Convention. Refer to the MPM Reference Guide Section 1.5, "Constructing and Interpreting Names", for a description of the Equals Convention and for the rules used to construct and interpret equal names.

Usage

```
declare get_equal_name_ entry (char(*), char(*), char(32),
        fixed bin(35));

call get_equal_name_ (entry_name, equal_name, target_name,
     code);
```

1) entry_name is the entry name from which the target is to be constructed. Trailing blanks in the entry name character string are ignored. (Input)

2) equal_name is the equal name from which the target is to be constructed. Trailing blanks in the equal name character string are ignored. (Input)

3) target_name is the target name which was constructed. (Output)

4) code is one of the following status codes. (Output)

   0 the target name was constructed properly.

   error_table_$badequal
   the equal name was not formed properly.

   error_table_$longeql
   the target name to be constructed would be longer than 32 characters.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Note

If the error_table_$badequal status code is returned because there were insufficient components in entry_name to correspond to an equal sign in the equal_name, or because there were insufficient characters in a component of the entry_name to correspond to a percent character in the equal_name, then a target_name which has been constructed by ignoring the excess equal sign or percent character is returned with the status code.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (End)