To: Distribution

From: Robert S. Coren

Subject: Maintenance of the I/O Daemon and Associated Data Bases

Date: 01/25/74

This document describes the procedures necessary to enable the I/O Daemon to run normally; in particular, it will describe the tools available for creating and preparing the data bases required by the I/O Daemon. Some familiarity with MOSN 6.4.3, "Operation of the I/O Daemon," is assumed.

All relevant I/O Daemon data bases must reside in a single directory. For regular service operation, this directory is >daemon_dir_dir>io_daemon_dir. For testing purposes, the daemon may run using data bases in some other directory (see Section IV, "Operating the I/O Daemon in 'Test' Mode").

I. BRIEF DESCRIPTION OF DATA BASES

A. Permanent Data Bases

These data bases stay around between I/O Daemon sessions, and must exist in their correct form in order for the I/O Daemon to run properly.

io_daemon_parms

This is an ascii segment suitable for on-line editing. It contains all the basic parameter information for the I/O Daemon subsystem; it is the control file on which all other I/O Daemon data bases depend. Its contents and syntax are described in Section II of this document.

User request queues

These are message segments in which user output requests are placed by the dprint and dpunch commands for processing by the I/O Daemon. They are described in Section III of this document, where the create_daemon_queues command is discussed.

daemon_search_rules_

This is an ascii segment suitable for interpretation by the parse_search_rules_ subroutine (which is described in the Multics Systems Programmer's Supplement to the MPM). It contains the search rules to be used by the I/O Daemon; these rules must include the library >tools.

B. Temporary data bases

The I/O Daemon uses a variety of temporary data bases which are reinitialized or re-created whenever the I/O Daemon Coordinator process is logged in. The information in them reflects the current state of the I/O Daemon; the initial values are mostly derived directly or indirectly from io_daemon_parms.

II. THE "io_daemon_parms" FILE

The "io_daemon_parms" segment is an ascii file with a syntax suitable for use with parse_file_. It should be input and modified by means of a text editor. The segment consists of statements of the form:

```
keyword: value;
```

where keyword is one of the keywords described below, and value is a character string consisting of letters, digits, or any of the characters, ">", "_", and "$". (The special syntax for subkeywords to the "remote" keyword is described later.) Comments may be included in the file for easier reading; any characters appearing between the strings "/*" and "*/" are considered comments and ignored by the procedures that interpret the file. Blanks and tabs not embedded in keywords or values are likewise ignored.

A. Definitions of Terms

As used in this document, the word device refers to a printer or card punch mentioned on a PRPH card in the BOS configuration deck. The device name is the unique identification of the device as given on the PRPH card (e.g., "prta", "puna", etc.); it is also the name by which the device is attached by the I/O Daemon.

A device class is a group of devices which are considered interchangeable; in particular, all devices in any one class take their output requests from the same set of queues. (Thus a user selects the device _class_ which he wishes to perform his request, rather than the particular device.) Device class names, which are 8 characters or less in length, should be chosen so as to be mnemonic, but are not otherwise restricted.

As a matter of convenience, there is generally a correspondence between each device class and a registered login id for a driver process, such that the login id is composed of the device class name prefixed by the characters "Iod_". This correspondence is not enforced within the I/O Daemon, however; in theory, any registered user with appropriate access and privileges could run as a device driver.

Keywords fall into four general categories: _global_ keywords, which give information pertaining to the entire I/O Daemon subsystem; _definition_ keywords, which give the names of devices and device classes; _default_ keywords, which supply default attributes for all devices in a given class; and _specific_ keywords, which are specific to individual devices.

B. Global Keywords

There are only three global keywords; they must all be present in the file. All global keywords begin with capital letters.

Time

The "Time" keyword sets the time period during which requests are to be saved after they have been performed. Its value is a decimal integer giving the number of minutes each request is to be saved.

Max_queues

The "Max_queues" keyword gives the maximum number of priority queues that may exist for any device class. Its value is a decimal integer. Owing to a present implementation restriction, Max_queues should not be greater than 4.

The "Time" and "Max_queues" keywords must be the first two keywords in the file, but it does not matter which one comes first.

End

The "End" keyword indicates the end of the io_daemon_parms file. It has no value, but is followed immediately by a semi-colon.

C. Definition Keywords

Device_class

The "Device_class" keyword defines a device class. Its value is a string of at most 8 characters, which is the name of the device class being defined (e.g., "printer", "punch", etc.). No two "Device_class" keywords may have the same value.

device

The "device" keyword defines a single device. Its value is a string of at most 8 characters, which is the name of the device being defined. As explained above, this name is the same as the device identifier on the PRPH card for the device (except in the case of "remote" devices, as explained below). If the same device is to be defined in more than one device class, it should be defined by a separate "device" keyword within the device-class definition of each device class to which it belongs.

The remaining keywords define attributes of devices. A default keyword, which begins with a capital letter, applies as a default to all devices within a class; a specific keyword, which begins with a lower-case letter, applies only to the specific device within whose definition it appears.

A <u>device-class definition</u> consists of the "Device_class" keyword, followed optionally by one or more default keywords, followed by one or more device definitions (see below). A device-class definition is terminated by a new "Device_class" keyword or an "End" keyword.

A <u>device definition</u> consists of a "device" keyword, followed optionally by one or more specific keywords. A device definition is terminated by a new "device" keyword, a new "Device_class" keyword, or an "End" keyword.

The following general rules apply to the specification of device attributes:

1. A specific keyword present for any definition of a device overrides any corresponding default keyword that might apply to that device.

2. The same specific keyword must not be supplied more than once with different values for the same device.

D. Default keywords

Accounting

The "Accounting" keyword defines the type of accounting to be done for use of devices in the device class by the I/O Daemon. Its value is either the character string "system" or the absolute pathname of a subroutine to be called to account for each request. If the value is "system", then the submitter of the request is charged for the number of lines printed; a subroutine name presumably refers to a user- or installation-supplied subroutine which will be used to record per-request accounting information. Such a subroutine would be supplied for devices whose driver processes run on a project (specified by the "Project" keyword) which is being charged like a regular user for CPU time, etc.

If the "Accounting" keyword is not supplied for a device class, "system" is assumed for that device class.

Dim

The "Dim" keyword names the default DIM (Device Interface Module) through which devices in the current class are to be attached. It applies to any device defined in that class for which no "dim" specific keyword is supplied. Its value is a string of up to 32 characters.

Element

The "Element" keyword gives the default value by which the bit count of a file should be divided to obtain the number of elements to be output; in other words, it is the "element size" used by the relevant DIM, expressed in bits. Its value is a decimal integer. If the "Element" keyword is not supplied, a default value of 9 is used (i.e., the element for output is assumed to be one character).

Project

The "Project" keyword indicates what project drivers for the device class are logged in under. Its value is a string of up to 9 characters. It is used by the dprint and dpunch commands to ascertain if the driver has sufficient access to process a user's file. If it is omitted, the project name "SysDaemon" is assumed.

Type

The "Type" keyword is used to indicate whether the devices in a given class are printers are card punches. Its value must be either "print" or "punch"; if it is not supplied, "print" is assumed. The value "punch" is valid only if the device class name begins with the characters "pu".

E. Specific keywords

dim

The "dim" keyword names the DIM to be used with the device for which it is specified. It overrides the "Dim" keyword, if any, for the device class. Either a "Dim" keyword or a "dim" keyword must apply to every device defined in the file.

element

The "element" keyword defines the element size for a given device. If neither the "element" keyword nor the "Element" keyword is supplied for a device, a default value of 9 is used.

default_class

The "default_class" keyword specifies the name of the device class in which a device is to run if no device class name is specified when its driver process is brought up (see Section II of MOSN 6.4.3, "Bringing Up the I/O Daemon"). If no "default_class" keyword is supplied, the device class must always be specified explicitly when the driver comes up.

remote

The "remote" keyword is used to specify a variety of special parameters for a remote device (or device combination). The value for the "remote" keyword is a series of subkeywords connected to values by "=" signs, separated from each other by commas, and ending with a semicolon, thus:

```
          remote:   subk1=val1,
                    subk2=val2,
                    .
                    .
                    subkn=valn;
```

where the subk*i* are the various subkeywords, and the val*i* are their respective values.

A remote device may really consist of both a printer and a card punch, in which case it is conceptually two devices in two different device classes (from the point of view of the I/O Daemon); these two devices are defined separately in the io_daemon_parms file under separate "device" names, but connected by a generic name for the remote device as a unit. This name is defined by the "name" subkeyword described below; it is the name supplied to the driver of a remote device when it first comes up.

If a device is defined with the "remote" keyword, then the name given as the value for the "device" keyword is *not* the name on a PRPH card; it is used internally by the I/O Daemon and may be any 8-character name (provided, of course, that it is not the same as any other device name).

A remote device always runs in a default device class. This means that if a device is defined with the "remote" keyword, the "default_class" keyword must also be supplied for that device. This also means that a remote device cannot be defined more than once with the same "device" keyword in different classes.

F. Subkeywords for remote devices

With the exception of the "name" keyword, each subkeyword described below need only be specified once per remote device.

name

The "name" subkeyword has as a value a string of up to 8 characters, which is the generic name of the remote device. It must always be supplied with the "remote" keyword, and must be the *first* subkeyword following the "remote" keyword.

tty_name

The "tty_name" subkeyword is used if the remote device has a dedicated hard-wired communications line reserved for it, in which case the value for this keyword is the 6-character channel identifier for the communications line as it appears in the "lines" file.

password

the "password" subkeyword defines an 8-character password which must be input when the remote device dials up. If the "tty_name" subkeyword appears, the "password" subkeyword is optional; otherwise it is required.

user_input_dim

The "user_input_dim" subkeyword defines the DIM to be used to read the "user_input" stream from the remote device. It must be supplied once for each remote device.

user_output_dim

The "user_output_dim" subkeyword defines the DIM to be used to write the "user_output" and "error_output" streams on the remote device. It must be supplied once for each remote device.

Examples 1 and 2 following show sample io_daemon_parms files. Example 1 is a simple case to allow the system to run 2 printers and one punch. Example 2 illustrates a few more of the features described above; it would be used to run 2 printers, one card punch, and one remote printer-punch combination. It describes a configuration in which the system (i.e., "SysDaemon" processes) would normally run one printer and the punch, while the other printer and the remote device are run by processes on the Multics project; however, the system is also capable of running the second printer.

EXAMPLE 1: Simple io_daemon_parms file

```
/* Everything possible is defaulted: all drivers run on SysDaemon project,
   use "system" accounting; printers use 9-bit element size */

Time:               60;

Max_queues:         3;


Device_class:       printer;
Dim:                prtdim;

 device:            prta;

 device:            prtb;


Device_class:       punch;
Type:               punch;
Dim:                pun21;
Element:            1;        /* pun21 DIM uses 1-bit elements */

 device:            puna;


End;
```

EXAMPLE 2 : More complete io_daemon_parms file

```
Time:                   60;         /* keep requests for one hour */

Max_queues:             3;


Device_class:           printer;    /* regular on-site printer */
Dim:                    prtdim_;
Element:                9;          /* This could be omitted */
Accounting:             system;     /* So could this */
                                    /* runs on SysDaemon project */

device:                 prta;       /* first printer */
 default_class:         printer;    /* defaults to this class */

device:                 prtb;
 default_class:         cprinter; /* this printer defaults to */
                                    /* different class but can */
                                    /* run in this one */

Device_class:           cprinter; /* special printer class */
Type:                   print;      /* optional */
Accounting:             >udd>m>lib>e>io_acct;
                                    /* uses different accounting */
Project:                Multics;    /* Not SysDaemon */

device:                 prtb;       /* also runs in "printer" class */
 dim:                   prtdim_;


Device_class:           punch;      /* Card punch */
Element:                1;          /* 1 bit = 1 element */
Dim:                    pun21;
Type:                   punch;      /* required */
Project:                SysDaemon; /* optional */

device:                 puna;       /* note absence of default class */
                                    /* class must always be supplied */

Device_class:           remote;     /* remote printer class */
Accounting:             >udd>m>rsc>remote_acct;
Project:                Multics;

device:                 prtrem;     /* internal name of printer */
 dim:                   g115_print;
 default_class:         remote;

 remote:                name=mohawk,
                        tty_name=tty613,
                        user_input_dim=g115_reader,
```

EXAMPLE 2: More complete io_daemon_parms file (cont.)

```
                     user_output_dim=g115_tprint;

Device_class:        pun_rem;  /* remote punch class */
Project:             Multics;
Accounting:          >udd>m>rsc>remote_acct;
                               /* same accounting as for */
                               /* remote printer */

device:              punrem;
 dim:                g115_punch;
 default_class:      pun_rem;
 remote:             name=mohawk;
                               /* rest of info is already */
                               /* supplied by definition of */
                               /* remote printer */

End;
```

III. THE "create_daemon_queues" COMMAND

The create_daemon_queues command is used to create user request queues, which are message segments in which user requests are placed by the dprint and dpunch commands.

Names: create_daemon_queues, cdq

Usage: create_daemon_queues -control_args-

The following are acceptable control arguments:

-directory path -dir path	The queues will be created in the directory whose pathname is path, which may be either an absolute or a relative pathname. The io_daemon_parms file to be used must be in the directory path. If this control argument is omitted (as it would be for normal use), the directory >daemon_dir_dir>io_daemon_dir will be used.
-default class_name -df class_name	The device class specified by class_name will be used as the defualt when a user enters a dprint command without the "device_class" control argument.
-default_punch class_name -dfp class_name	The device class specified by class_name will be used as the default when a user enters a dpunch command without the "device_class" control argument; the first two characters of class_name must be "pu".

The effect of the command is to create a group of message segments for each valid device class, where the valid device classes are determined from the contents of io_daemon_parms. Each queue has a name of the form:

CLASS_n.ms

where CLASS is the device class name, and n is the priority level of the queue; one such queue is created in each class for all n from 1 to "Max_queues" as specified in io_daemon_parms. In addition, the names:

```
default_n.ms
```

for _n_ = 1 to Max_queues are added to the queues for the class specified by the "-default" control argument, if it is present; similarly, the names:

```
pun_dflt_n.ms
```

are added to the queues for the class specified by the "default_punch" control argument.

If any of the queues already exist, they are left alone (except for possibly having default names added or removed); when a queue is created, it is given the following extended access control list:

```
adros     IO.SysDaemon.*
aros      *.*.*
```

If desired, the extended ACL can be modified by means of the message_segment_setacl (mssa) command. For these queues, "s" access allows a user to find out how many requests are in the queue; "a" and "o" together allow him to add requests and to read and delete his own requests; "r" allows him to read any request; "d" allows him to delete any request.

When a default class is not specified, any existing default is left alone; if a default class is specified, then if the default names were present on a different class of queues, they are removed from the queues for that class.

The print_io_devices (pid) command described in the MPM can be used to find out what classes exist and what the default classes are at present.

IV. OPERATING THE I/O DAEMON IN "TEST" MODE

The I/O Daemon can be tested within the regular Multics command environment by using special "test" entries. A separate process must be logged in for the Coordinator and for each driver to be tested. To set up a Coordinator in "test" mode, enter the command:

```
iodc_overseer_$test DIR
```

where DIR is the absolute pathname of the directory to be used for all data bases. The io_daemon_parms, daemon_search_rules_, and user request queue segments must already exist in DIR.

For each ordinary driver process to be tested, enter the command:

```
iodd_overseer_$test DIR
```

where DIR is the same as for the Coordinator. For a driver for a remote device, enter the command:

```
iodd_overseer_$r_test DIR
```

The operation of the daemon in "test" mode is exactly as described in MOSN 6.4.3, "Operation of the I/O Daemon," except that the following additional commands are valid for both Coordinator and driver processes in "test" mode.

debug

The Multics debug command will be invoked. After exit from the debug command (by means of the .q request), the process will await a further command.

pi

If entered after a QUIT, this command will signal the condition "program_interrupt". This command is useful if the Multics debug command (or a program invoked from the debug command) produces large amounts of unwanted console output. The "pi" command is invalid if not preceded by a QUIT.

return

This command causes the Coordinator or driver to return to its caller; in general, this will mean a return to Multics command level. From the internal point of view, it has a similar effect to the I/O Daemon "logout" command, but the "testing" process is not destroyed.

In addition, if an I/O Daemon process gets an unclaimed signal in "test" mode, it will, after printing the usual error message, invoke the "debug" command. Upon exit from debug, a driver process will continue with its normal error recovery. A Coordinator in "test" mode, however, will await a further command after return from debug: a "return" command will return the process to Multics command level; a "start" command will resume processing from the point of error (like the regular Multics "start" command); any other response (including a blank line) will cause the Coordinator to continue with its normal error recovery procedures.