TO: MSPM Distribution

FROM: P. G. Neumann

SUBJECT: BF.2.22, The Registry File Maintainer

DATE: 03/01/68

The attached BF.2.22 is a major revision of the former issue, 08/14/67. Many of the calls have been redesigned, although the module is functionally the same.

Published: 3/01/68
(Supersedes: BF.2.22 08/14/67)

Identification

The Registry File Maintainer
S. I. Feldman

Purpose

Every I/O device known to a system has an associated file called its Registry File (RF). The Registry File Maintainer (RFM) is called by outer modules and other interested users to get information from or to store information into a Registry File. This section describes the functions of Registry Files, describes the calls to the RFM and the implementation thereof.

Registry Files

Protection of all I/O devices is implemented by file system protection of their Registry Files. Non-volatile information about the device is stored in this file for use by the I/O System. The following is a brief discussion of the form, content, and use of these files.

The RFs are organized by type of device into "type" directories. These type directories are immediately inferior to the Registry File Directory (path name ">rfd"). A Registry File is identified by two 32-character strings, called the "type" and the "name". The type is the entry name of the type directory in the Registry File Directory; the name is the entry name of the RF in its type directory. In addition to the regular RFs, these type directories also contain certain other files: an I/O Assignment Table (IOAT), normally accessible from the hardcore ring only by the I/O Assignment Module (IOAM); see BF.2.26). This table lists names of the users who possess and control the file and its associated devices at any time. There may be several read-only files in the directory for use in storing certain critical RF-related information that cannot be left in the normal RFs for security reasons. Finally, there are usually one or more normal Registry Files called prototype RFs. These files are used to created new RFs of the given type.

The normal RFs are made up of several distinct parts. The first is of standard form and content. This part of the RF is used by the Attachment Module (see BF.2.23) and the Registry File Maintainer (RFM; see below). This fixed part has information on the RF, and has "pointers" to certain RFs. (These pointers consist of the type and name of the target RF). Each RF has a "level" number associated with it. The smaller the level number, the more directly related the device is to the GIOC. Thus, a typewriter channel has level equal to 1, while the typewriter has level equal to 2. "Down" is defined to be the direction of decreasing level numbers. The up and down chaining represents the connections between devices. A file may point to more than one up and one down RF, although this is not the usual case.

Some of these links between RFs are permanent, representing physical wiring. Others are transient, and change with the configuration or for other reasons. See below for a discussion of logical channels. As an example of a temporary link, a typewriter is associated with a channel only while it is dialed in. The association is broken when it hangs up. The following is a plausible RF arrangment for tapes: The level 4 RF represents a tape reel. There is a temporary association (down link) with the level 3 RF for the tape drive upon which the reel is presently mounted. There is also a temporary up link from the drive to the reel file. There is a permanent link down from the level 3 drive RF to the level 2 controller RF. There are many permanent up links from the controller, one for each of the drives connected to it. There are also some temporary down links from the level 2 controller file to the level 1 files for the High Performance Channel RFs. There is a temporary up link from each of these channel files to the appropriate controller file. The association between controller and channel can be changed by using the peripheral switch; any modification will be reflected in the Device Configuration Table. (See the discussion of logical channels.)

The fixed part of the RF contains, in addition to names of RFs in the chain, information to guide the Attachment Module in following the list. There is also information on the type of DCM to be used, and part of the mode string to be passed to that DCM. A RF has at least one device associated with it. If several devices are closely related, they may be considered as a single resource and may therefore share a single RF. For example, a full-duplex typewriter channel can be implemented by connecting two half-duplex typewriter GIOC channels to a single data set. There would be a single RF for the pair of channels, but there is information on both devices separately in the RF.

Each device has a "device profile" in the RF. Once the size is set (either when the RF is created or when the first rfm$set_profile call is made), it does not change. This profile is meant to contain relatively constant physical device information. For example, the tab settings of typewriters will be stored in the profile. However, data as changeable as the line and column number are not stored in the profile. A tape profile might include information on density and the names of other reels that make up a single multi-reel file.

In addition to the profile, there is a behavior log for each device. This log usually contains information on faulty performance by the device. Whenever such a thing happens, the appropriate I/O System module calls rfm$add_log to store this information at the end of the chain of log entries. Each log entry contains the identification of the process and user in control, the time, and an arbitrary bit string supplied by the caller. The log can be read and entries can be deleted by appropriate calls to the RFM. The log itself is a chained list of structures, [illegible] ing a relative pointer to the next in the list. The fixed part of the RF contains a relative pointer to the first and last blocks on the chain.

At some installations, the assocation between certain devcies and certain GIOC channels may depend upon the settings of the peripheral switch. Registry Files have the ability to handle such a connection. When the Attachment Module finds that the "logchans" bit is on in the RF, it calls a hardcore supervisor procedure, the Device Configuration Table Manager, which will translate a "logical channel" name stored in the RF to the real RF name. The Attachment Module then stores that information in the RF and proceeds as if that were the down name stored in the RF normally.

Above, we mentioned the existence of certain special read-only files in the type directories. These files contain information relating to Universal Device Manager processes. Such a process is a system process that can handle several devices of a given type. The Attachment Module needs to know the process group id of the relevant UDMP and also the name of a certain data base used by the process, called its PDT. This information is placed in the special read-only file. Only certain types of RFs require these files. These are the last RFs examined by the Attachment Module while it traces through a chain of files. Typically, that file represents the channel or controller. For the RF with name "X", there is a file with name "X_ro" in the same type directory. Since several devices can use the same read-only file, a single file will most likely have several names.

Registry File Declaration

The following is the EPL declaration for a Registry File. The first two declarations, rf and rfx, together form the fixed part of the RF discussed above; there are two parts for implementation reasons. The third declaration, rf_ro, is the special read-only file which exists for reasons of system security.

```
dcl 1 rf based(rfp),
  2 level fixed bin(35),       /*level=1 for a GIOC channel, 2 for
         "                       a device connected directly to
         "                       a GIOC channel, etc.*/
  2 force_udmp bit(1),         /*if 1, force the use of a universal
         "                       device manager process*/
  2 in_use_switch bit(36),     /*set ON at attach time and OFF
         "                       at detach time*/
  2 hangupable bit(1),         /*if ON, device can hang up*/
  2 logchans bit(1),           /*if ON, the down_names for this
         "                       device are to be filled in by a call
         "                       to the hardcore ring to get the present
         "                       RF name corresponding to the logical
         "                       channel names.  If this bit is ON,
         "                       no more RFs are to be searched.*/
  2 allocate bit(1),           /*if ON, Reserver should be called
```

```
        "                          with each resource_name as argument.*/
  2 temp_link bit(1),            /*connection with next file is
        "                          temporary.  Blank out down_name
        "                          entries upon detachment*/
  2 nup fixed bin(35),           /*number of entries in up array*/
  2 ndown fixed bin(35),         /*number of entries in down array*/
  2 ndev fixed bin(35),          /*number of entries in devices array*/
  2 ntypes fixed bin(35),        /*number of entries in att_types array*/
  2 present_type_index fixed bin(35), /*index in att_types array of
        "                          type with which device was last
        "                          attached*/
  2 down_slot fixed bin(35),     /*position of upname for this file
        "                          in up array of next registry file*/
  2 alloc_type char(32),         /*use this type in calls to the
        "                          Reserver alloc$resource
        "                          entry*/
  2 lock bit(144),               /*for locking RF when threading
        "                          or deleting behavior log entries or
        "                          modifying the profile.*/
  2 up(rfp->rf.nup),             /*registry files pointing to this one*/
    3 uptype char(32),
    3 upname char(32),
  2 devices(rfp->rf.ndev),       /*entries for devices associated
        "                          with this registry file*/
    3 resource_name char(32),    /*name used in calls to the Reserver
        "                          and the Device Assignment Module*/
    3 profile_relp bit(18),      /*relp to device profile for this
        "                          device*/
    3 profile_length fixed bin,  /*number of bits in this profile*/
    3 oldest_log_relp bit(18),   /*relp to oldest entry in behavior log*/
    3 newest_log_relp bit(18),   /*relp to most recent entry in
        "                          behavior log*/
    3 nlog fixed bin,            /*number of entries in behavior log*/
    3 device_type fixed bin(35),
  2 rfxrelp bit(18),             /*relative pointer to RF extension*/
  2 free_storage area((15000));
/*


*/
dcl 1 rfx based(rfxp),
  2 att_types(rfp->rf.ntypes),/*special information for each type
        "                          by which this device may be known*/
    3 type_name char(32),
    3 ccm_type char(32),         /*name of code conversion driving
        "                          table to be used*/
    3 trace_down bit(1),         /*if ON, trace down to next registry
        "                          file.  Otherwise, stop here*/
    3 alloc_down bit(1),         /*if ON, must call Reserver to
        "                          allocate a device of type
        "                          down_type, and use returned
        "                          resource_name as down_name(1).
```

```
         "                         In either case, find next RF by
         "                         using down_type and down_name(1)*/
    3 look_only bit(1),          /*keep tracing down to other RFs
         "                         under trace_down control, but
         "                         only to compute code conversion
         "                         driving table name*/
    3 down_type char(32),        /*used as described above*/
    3 down_name(rfp->rf.ndown) char(32), /*used as described above*/
    3 logical_channel (rfp->rf.ndown) char(32),  /*array of
         "                         names to be used in call to get present
         "                         equivalent RF name from
         "                         info in DCT.  Used only if the
         "                         logchans bit is on*/
    3 extra_mode char(32),       /*character string to be
         "                         concatenated with mode to be
         "                         passed to DCM*/
    3 dcm_type char(32),         /*used as type in attach call to
         "                         DCM if trace_down is OFF or
         "                         look_only is ON*/
    3 dcm_name char(32);         /*used as ioname2 of attach call to
         "                         DCM if trace_down is
         "                         OFF or look_only is ON*/
/*


*/
dcl 1 rf_ro based(p),            /*special Registry File.  There is a
         "                         file of this format associated with
         "                         each regular RF, with name equal to
         "                         the name of the normal RF concatenated
         "                         with "_ro".  This file contains
         "                         certain data that must be protected
         "                         against tampering and is therefore
         "                         read-only to most users.*/
  2 pdt_name char(32),           /*name of PDT in DMP*/
  2 udmp_user_id char(50);       /*user_id of universal device manager
         "                         for this device, if any*/
```

<u>Calls and Arguments</u>

```
call rfm$get_devices(type,name,device_types,resource_names,cstatus);
call rfm$set_profile(type,name,devnumber,dataptr,nbits,cstatus);
call rfm$get_profile(type,name,devnumber,dataptr,nbits,cstatus);
call rfm$get_nlog(type,name,nlogs,cstatus);
call rfm$add_log(type,name,devnumber,dataptr,nbits,cstatus);
call rfm$delete_log(type,name,devnumber,first,number,cstatus);
call rfm$read_log(type,name,devnumber,first,number,infoptr,cstatus);
call rfm$get_ups(type,name,uptypes,upnames,nreturned,cstatus);
call rfm$get_down(type,name,down_type,down_names,cstatus);
call rfm$link(toptype,topname,att_type_index,down_index,
     bottomtype,bottomname,upindex,cstatus);

declare
```

```
type char(*),                  /*type of device*/
name char(*),                  /*name of device*/
toptype char(*),               /*type of device with larger level number*/
topname char(*),               /*name of that device*/
bottomtype char(*),            /*type of device with smaller level
    "                            number*/
bottomname char(*),            /*name of that device*/
devnumber fixed bin,           /*index in devices array for
    "                            this device*/
dataptr ptr,                   /*pointer to bit string of
    "                            length nbits into which profile
    "                            is to be stored or from which profile
    "                            or log entry is to be copied*/
nbits fixed bin,               /*number of bits in bit string.  See
    "                            above*/
first fixed bin,               /*index of first behavior log entry
    "                            (starting from the oldest) which
    "                            is to be read or deleted*/
number fixed bin,              /*how many log entries are to be
    "                            read or deleted*/
device_types(*) fixed bin,     /*array into which device types are
    "                            to be stored*/
resource_names(*) char(32),    /*array into which the resource names
    "                            are to be stored
infoptr ptr,                   /*pointer to the following structure*/
1 info(number) based(infoptr),/*array of structures which will
    "                            contain information on the behavior
    "                            log entries.  The length of this array
    "                            tells the RFM how many to read out*/
  2 time fixed bin(71),        /*time when RFM stored log entry*/
  2 proc_id bit(36),           /*process id of caller when RFM stored
    "                            log entry*/
  2 user_id char(50),          /*user id of above user*/
  2 nbits fixed bin,           /*number of bits in log entry*/
  2 dataptr ptr,               /*pointer to the log entry*/
uptypes(*) char(32),           /*list of uptypes for the RF*/
upnames(*) char(32),           /*list of upnames for RF*/
nlogs(*) fixed bin,            /*list of number of behavior log
    "                            entries for each device*/
down_type char(*),             /*down type for RF*/
down_name(*) char(32),         /*list of down_names*/
down_index fixed bin,          /*index in down_name array where
    "                            down_name is to be stored*/
up_index fixed bin,            /*index in upnames array where up_type
    "                            and up_name are to be stored*/
nreturned fixed bin;           /*number of upnames returned*/
```

Implementation of Calls

get devices

This call is made to get a list of device types and resource names from a given RF. In response to the call, the following steps are taken:

1. If the RF with type _type_ and name _name_ is non-existent, set bit 1 of _cstatus_ and return. If the file is not readable by this user, set bit 8 of _cstatus_ and return.

2. Store as many device_types as possible in the _device types_ array. If there are more in the RF, set bit 18 of _cstatus_. If the array is larger than the array in the RF, fill in the remaining elements with zeroes.

3. Store as many resource names from the RF into the _resource names_ array as possible. If there are more in the RF, set bit 18 of _cstatus_. If the array is larger than the array in the RF, fill in the remaining elements with blanks.

4. Return.

set profile

This call is used to store a device profile. In response to the call, the following steps are taken:

1. Find the RF with the given type and name. If the file does not exist, set bit 1 of _cstatus_ and return. If the file is not readable by this user, set bit 8 of _cstatus_ and return. If the file is accessible but not writable by this user from the caller's validation level, set bit 4 of _cstatus_ and return.

2. Lock the RF using rf.lock as a lock structure.

3. If _devnumber_ if less than one or greater than rf.ndev, set bit 2 of _cstatus_, unlock the RF, and return.

4. If the above arguments are all right, then if rf.devices(devnumber).profile_relp is not zero, go to step 5. Otherwise, store _nbits_ in the rf.devices(devnumber).profile_length, allocate a bit string of the appropriate size in the area, and store the relp to that bit string in rf.devices(devnumber).profile_relp. If the area is not large enough, set bit 7 of _cstatus_, unlock the RF and return.

5. Store the bit string (length _nbits_) pointed to by _dataptr_ in the appropriate device profile and return. The rules of bit assignment relating to padding and truncating apply.

get profile

This call is made to read out a profile. The following steps are taken in response to the call:

1. If the file is non-existent ,set bit 1 of _cstatus_ and return. If this user does not have read permission for the file, set bit 8 of _cstatus_ and return.

2. If devnumber is less than one or greater than rf.ndev, set bit 2 of cstatus and return.

3. Lock the RF using rf.lock as a lock structure.

4. If the arguments are valid, then if rf.devices(devnumber).profile_relp is zero, set the bit string pointed to by dataptr (length nbits) equal to the null string, unlock the RF, and return.

4. Set the bit string described above equal to the devnumberth device profile using bit string assignment rules. Unlock the RF and return.

get nlog

This call is made to find out how many behavior log entries for the devices associated with the RF. In response to the call, the following steps are taken:

1. If the RF with the given type and name is non-existent, set bit 1 of cstatus and return. If this user does not have read permission for the file, set bit 8 of cstatus and returrn.

2. Store as many of the nlog entries in rf.devices into the nlogs array as possible. If the argument array has more elements than there are devices, fill in the rest of the array with zeroes. If there are elements in the RF that have not been returned, set bit 18 of cstatus.

3. Return.

add log

The following call is made to add a behavior log entry for a device associated with the file. In response to the call, the following steps are taken:

1. If the RF with the given type and name is non-existent, set bit 1 of cstatus and return. If this user does not have read permission for he file, set bit 8 of cstatus and return. If the file is not writable from the caller's ring, set bit 4 of cstatus and return.

2. Lock the RF.

3. If devnumber is less than one or greater than rf.ndev, set bit 2 of cstatus, unlock the RF and return.

4. Otherwise, allocate a structure like the following in the area in the RF:

```
dcl 1 log_entry based(p),
```

```
2 next_relp bit(18),
2 time fixed bin(71),
2 proc_id bit(36),
2 user_id char(50),
2 nbits fixed bin,
2 data bit(nbits);
```

See the declaration of all arguments of RFM calls for the meaning of the elements of the structure. If the area is not large enough, set bit 7 of cstatus, unlock the RF, and return.

5. If the allocation succeeds, copy the bit string of length nbits pointed to by dataptr into the appropriate part of the structure. Store the present process id, user id, and timer value in the structure, as well as nbits. Set the next_relp equal to zero. If rf.devices(devnumber).oldest_log_relp is zero, set it and the corresponding newest_log_relp equal to the offset of the newly allocated structure. If the oldest relp is nonzero, then set the relp in the structure pointed to by rf.devices(devnumber).newest_log_relp equal to the offset of the new structure, and then store that same offset in newest_log_relp.

6. Increment rf.devices(devnumber).nlog by one, unlock the RF, and return.

delete_log

This call is used to delete a set of behavior log entries for a particular device associated with a Registry File. In response to such a call, the following steps are taken:

1. If the RF with given type and name is non-existent, set bit 1 of cstatus and return. If the file is not readable by this user, set bit 8 of cstatus and return. If the file is not writable from the caller's ring, set bit 4 of cstatus and return.

2. Call the Locker and lock the RF using rf.lock structure.

3. If devnumber, first, or number is less than one or if devnumber is greater than rf.ndev or if the sum of first and number is greater than rf.devices(devnumber).nlog, then unlock the RF, set bit 2 of cstatus and return.

4. Otherwise, follow the chain of relps for the behavior log of the device. Starting at the first element (counting the oldest link on the chain as number 1), free number of them. Make the next_relp of the last entry not freed point to the one after the gap, or set it to zero if it is now the last entry on the chain. Modify oldest_log_relp and newest_log_relp in the devices array of the RF as necessary.

5. Unlock the RF and return.

read log

This call is used to read out a set of log entries (non-destructively). In response to the call, the following steps are taken:

1. If the file is non-existent, set bit 1 of cstatus and return. If the file is not readable by this user, set bit 8 of cstatus and return.

2. If devnumber, first, or number is less than one, or if devnumber is greater than rf.ndev or if the sum of first and number is greater than rf.devices(devnumber).nlog then set bit 2 of cstatus and return.

3. Chase the chain of relps until the entry with index equal to first is found. Copy out the contents of the log entry into the corresponding elements of the info array. (The next_log relp is not copied and a pointer to the data string is stored in dataptr). If there are more entry logs on the chain, set bit 18 of cstatus. Return.

get ups

This call is used to get the uptypes and upnames (identifications of the RFs that point down to this one). In response to the call, the following steps are taken:

1. If the file is non-existent, set bit 1 of cstatus and return. If the file is not readable by this user, set bit 8 of cstatus and return.

2. Copy as many elements of the uptype and upname arrays (elements of rf.up) into the uptypes and upnames character array arguments. Set nreturned equal to the number of complete pairs of RF names. If there are more than this, set bit 18 of cstatus.

3. Return.

get down

This call is used to find the next RF in a chain, assuming they have been previously linked. In response to the call, the following steps are taken:

1. If the RF with type type and name name is non-existent or inaccessible, set bit 1 of cstatus and return.

2. If rf.present_type_index is not greater than zero and less than or equal to rf.ntypes, set bit 5 of cstatus and return.

3. Otherwise, set down type equal to rfx.att_types(r[illegible]pe_index).down_type and store as many

of the corresponding down_names into the down names array. If there are more element of the argument array than down names, fill in the extra elements with blanks. If there are more down names, set bit 18 of cstatus. Return.

link

This call is used to link up two Registry Files, an upper one and a lower one. This call is meant for use by certain Device Control Modules. In response to the call, the following steps are taken:

1. Find the Registry File with type toptype and name topname. If the file does not exist, set bit 1 of cstatus and return. If the file is not readable by this user, set bit 8 of cstatus and return. If the file is not writable by this user from the caller's ring, set bit 4 of cstatus and return.

2. Do the same checking for the file with type bottomtype and name bottomname.

3. If any of the following conditions holds, set bit 2 of cstatus and return:

upindex is less than one or greater than rf.nup in the lower file.
down index is less than one or greater than rf.ndown in the upper file.
att type index is negative or greater than rf.ntypes in the upper file. If that argument is zero, then if rf.present_type_index is less than one or greater than rf.ntypes.

4. If att type index is non-zero, store it in rf.present_type_index for the upper file. Call the value of the present_type_index N.

5. Store bottom name in rfx.att_types(N).down_name(down_index) in the upper file.

6. Store bottomtype and bottomname in the corresponding elements of rf.up(upindex) in the lower file.

7. Return.

Summary of Cstatus Bits

1 File non-existent
2 Number out of range
3 Typename not found
4 File not writable (set profile, add log, delete log, and link calls only)
5 Unlinked Registry File (get down call only)
6 Non-RFM unexpected error

7 Area too small (add log and set profile calls only)
8 File not readable by this user
18 More data available (array not large enough)