MULTICS TECHNICAL BULLETIN MTB-605

To: Distribution

From: Robert S. Coren

Date: 12/02/82

Subject: Planned Improvements for IMFT in MR 10.2

SUMMARY

The following improvements are planned for the Inter-Multics File Transfer (IMFT) facility in MR 10.2:

o -- Communications independence: divorce IMFT from HASP so that IMFT can be used with a variety of communications protocols;

o -- Request for remote transfer ("pull"): allow a user to request transfer of a file or subtree from a remote site;

o -- Automatic deferral: enable an IMFT driver to defer any request that would take longer than a specified amount of time to transmit.

This MTB describes the general mechanisms proposed for each of the above features, followed by estimates of the amount of time required to implement each feature. The changes required to the IMFT Reference Guide (CY73) are summarized at the end of the MTB.

Comments on this MTB should be communicated:

in forum (method of choice):
meeting >udd>m>Palter>forums>IMFT

via Multics mail:
Coren.Multics on System M or MIT

by Telephone (method of last resort):
Robert Coren
492-9329
HVN 261-9329

COMMUNICATIONS INDEPENDENCE

IMFT, as currently implemented, can only transfer objects over a HASP connection; imft_io_, the I/O module that IMFT uses, sends and receives HASP records. In order to make it possible to use IMFT over other kinds of connections (e.g., X.25), knowledge of HASP must be moved out of imft_io_.

Because the IMFT protocol uses fixed-format control records for communication between the two drivers, imft_io_ itself must read and write records rather than stream data. For each communications method to be supported, an intermediate I/O module will be spliced in between imft_io_ and the standard I/O module that implements the communications method; the intermediate module translates between IMFT records and whatever form the standard I/O module uses for its data. These intermediate modules will have names of the form imft_to_<MODULE>, where <MODULE> is the name of the standard I/O module. Thus, imft_to_hasp_host_ turns IMFT records into HASP records on output, and conversely on input; imft_to_tty_ sends IMFT records as stream data in rawo mode, and receives stream data (in rawi mode) that it packages into IMFT records.

Each IMFT record begins with a fixed-length header that contains the length of the body of the record in bytes, an indicator of whether it is a data record or a control record, and other indicators as needed. This makes it easy for a module such as imft_to_tty_ to recognize IMFT records in the input stream and pass them intact to imft_io_.

The present imft_io_ converts between binary data as kept by the storage system and 7-bit characters for transmission across a HASP link. This conversion task will be moved to the various intermediate I/O modules, since, for example, tty_ is perfectly capable of handling 8-bit bytes, and because this allows for the possibility of an I/O module that acts on binary data, doing any necessary packing/unpacking itself.

Initially, the only intermediate I/O modules provided will be imft_to_hasp_host_, imft_to_hasp_workstation_ (which will probably be essentially the same program), and imft_to_tty_. The latter, of course, can be used to interface to X.25. A module may be added later that implements a block protocol over an asynchronous communications channel, in order to provide for error detection; for the initial release, however, sites will be advised that, although IMFT may be used over an asynchronous dialout channel, no protection is provided against line noise or lost data.

REQUEST FOR REMOTE TRANSFER

Currently, a user who wants to transfer a file or subtree from one Multics site to another must issue the request at the sending site; i.e., IMFT can only be used to "push" objects from one system to another. This is very inconvenient for a user who, while logged in to system A, discovers, remembers, or decides that he wants a file from system B, and must now log into system B for the sole purpose of queueing the transfer request. The ability to request that an object be "pulled" from system B to system A would eliminate this inconvenience.

This problem can be solved fairly simply by providing an additional request type for "pull" requests, and configuring the output driver on system A to have two minor devices, one that services the "push" queues and one that services the "pull" queues. The output driver will alternate between "push" requests and "pull" requests. When a request is found for a "pull", the driver simply sends the request itself, appropriately flagged; when the input driver on system B receives it, it adds it to the local "push" queue for eventual sending as if the request had originated on system B.

The user interface for remote requests will be the addition of a control argument to enter_imft_request: "-source <site_name>", which specifies the name of the foreign site from which the files and/or subtrees are to be transferred. The -source and -destination control arguments will, of course, be mutually exclusive, and the default, for compatibility, will be "-destination imft".

This scheme requires two request types and associated sets of queues for each foreign site. The request types will have names of the form From_<Site_name> and To_<Site_name>, and will be defined separately in the I/O Daemon Tables. As a result, the output of "print_request_types -generic_type imft" will not really be adequate to tell the user of enter_imft_request what site names are valid; a new command, print_imft_sites, will be provided to allow a user to find out the names of possible sources and destinations.

The double-queuing scheme has the disadvantage that, on a heavily-loaded connection, a user might have to wait quite a while for his file, since he first has to wait for his request to be transferred, and then has to wait again until the remote system is ready to send it. Furthermore, since he has no way to interrogate the queues on the remote system, it is harder for him to make an informed estimate of which priority queue he should be using. (It is presumed that a remote request would be requeued at the same priority as the original request if possible.) These are minor disadvantages, however. The only alternative method that comes to mind is to have an output driver send control records at suitable intervals that ask the remote input driver if there are any outstanding "pull" requests. An affirmative answer would take the form of a reply record containing the request, which the output driver would immediately process. The principal difficulty with this approach is that there is currently no mechanism whereby an input driver serves a queue, and adding one would not only be a lot of work, but might require either violation or extension of the protocols used within the I/O daemon itself.

Access Implications

The access control segment currently used to control the ability of a foreign user to transfer files into a local user's hierarchy (>udd>LProject>LPerson>FSite.imft.acs) will also be used to control the ability of said foreign user to transfer files from the local user's hierarchy by means of a remotely-issued request. The foreign user must have "r" access to the ACS in order to transfer files by remote request, and "w" access will be required to transfer files into the local hierarchy. This is an incompatible change in the meaning of "r" access on the IMFT ACS, but it makes more intuitive sense ("r" means you can get at objects, "w" means you can modify them). An SRB notice to this effect will be provided. Of course, the actual transfer will be protected in the same way as if it the request had been originally issued at the source system.

Some concern has been expressed about the possibility that the "pull" request might give privileged users on a foreign (not necessarily trusted) site the power to obtain local files to which they did not otherwise have access. Therefore, an option will be provided to allow a site to restrict "pull" requests from a specified foreign site to objects to which the IMFT daemon has explicit access. That is, if the option is enabled, a file or subtree cannot be pulled unless its ACL includes a term that explicitly contains the person ID of the IMFT daemon.

The "push" request submitted by an IMFT driver process in response to a "pull" message will be added to the appropriate queue by means of the queue_admin_ gate, so that it can be treated as a request from the local manifestation of the original requestor. The queue_admin_ mechanism does not currently check to see if the user on whose behalf the request is being added has "a" extended access to the queue; this should probably be changed, so that a user cannot use "pull" to submit a request that he could not have submitted using "push".

AUTOMATIC DEFERRAL

A mechanism already exists and is used by printer drivers to optionally defer requests that are estimated to take more than a specified amount of time to print. IMFT could use a similar mechanism; the only hard part is estimating the time required to fulfill a request. The output driver can maintain an average of the time per bit required to transmit each object, thereby providing a reasonable estimate of the amount of time required to transmit an object of a given size. This average will presumably be weighted according to the number of files per object; i.e., it presumably takes longer to transmit a subtree containing 10 files of 10 records each than a single file of 100 records. An initial estimate would be based on the baud rate of the connection, with a percentage added to allow for overhead generated both by the hierarchy dumper and by IMFT itself.

TIME ESTIMATES

Task	Time (weeks)
Modify imft_io_ to build standard IMFT records	2
Turn the remainder of imft_io_ into imft_to_hasp_(host workstation)_	2
Write imft_to_tty_	2
Add argument to io_daemon_tables for IMFT I/O module	1
Documentation changes	2
Testing	2
Subtotal for communications independence	11
Add minor device driver for "pull" queue	1
Modify input driver to queue remote requests	2
Extra access checks and -source control argument	1
Documentation changes	1
Testing	1
Subtotal for remote requests	6
Automatic deferral	2
Total	19

DOCUMENTATION

The IMFT Reference Guide is riddled with explicit references to HASP. The necessary changes are not presented in this MTB; it is simply noted that all such references have to be either deleted or changed to describe a generic communications configuration, with references to the appropriate documentation for individual protocols.

The remainder of this MTB includes the following: a revised version of the description of the I/O daemon tables entries from Section 2; a revised command description of enter_imft_request; and a command description of print_imft_sites. The command descriptions of list_imft_requests, cancel_imft_request. and move_imft_request will be revised to add the -source control argument and change the description of the -destination control argument in accordance with the description of enter_imft_request.

I/O Daemon Table

I/O daemon tables define the devices and Request_types to be used with the I/O daemon. A source file consists of a sequence of statements and substatements that define and describe each device and Request_type. It is not the intent of this section to present a full description of I/O daemon tables, but only the device and Request_type definitions required for IMFT I/O daemon definition. For a full description of I/O daemon tables, refer to the Bulk I/O manual.

The pathname of the source of the I/O daemon tables is usually:

```
>ddd>idd>iod_tables.iodt
```

Once you have edited the appropriate information into the I/O daemon tables source, they must be compiled via the iod_tables_compiler command (see the Bulk I/O Manual). It is recommended, for convenience, that the pre-compiled version of the I/O daemon tables be stored in the same directory as the compiled version with the name iod_tables.iodt.

I/O DAEMON DEVICE AND REQUEST TYPE DEFINITION FOR IMFT

The IMFT driver requires that you define two major devices in the I/O daemon tables: one for the input driver and one for the output driver. These devices must specify use of the "imft_driver_" driver module. The major device for the output driver may have either one or two minor devices defined: one for transferring files to the remote site, and one for requesting transfers from the remote site (in order to process requests entered with the -source control argument).

The IMFT driver does not support the "line: variable;" construct. Additionally, the HASP subchannels used by a driver are specified in the args statement. Therefore, the line statement used for the IMFT driver must be "line: *;".

The args statement specifies the direction of transfer for the driver, the "attach" descriptions for the subchannels used by the driver, the Person_ids used to validate the local and remote systems, and whether to initiate transfers automatically when the physical connection is established. See "Access Considerations", below, for an explanation of the purpose of the Person_ids.

Two Request_ty[illegible] r each foreign site, one for requests for tran[illegible] reign site, and one for transfers from the foreign [illegible] same Request_type must be specified in the "default_ty[illegible] atement of both the minor device of the output driver used r transfers to the foreign site and the input driver, for any given IMFT connection. The name of this Request_type must be the same as the foreign system ID specified in the args statement of the drivers, prefaced by the string "To_". The definition of the Request_type must include two "device" statements; one of these statements identifies the input driver for the connection and the other identifies the minor device of the output driver used for transfers to the remote site.

A second Request_type may be defined to allow users to request transfers from the remote site. The name of this Request_type must be the same as the foreign system ID specified in the args statement of the drivers, prefaced by the string "From_". This Request_type must be specified in the "default_type" statement of the minor device of the output driver used to request transfers from the remote site. The definition of the Request_type must include a "device" statement identifying the same minor device.

Currently, the IMFT facility does not charge users for use of the facility. Therefore, the Request_type defined for an IMFT driver must include the statement "accounting: nothing;".

If a hardwired connection is used and the channel is configured as described above, it is recommended that all four I/O daemon driver processes for the connection be logged in automatically by either the system_start_up.ec or the start_up.ec executed by Utility.SysDaemon. Additionally, it is recommended that all four drivers specify "mode= automatic" in their respective args statements. By including these specifications, the IMFT connection will run automatically without operator intervention whenever both systems are running.

If the Access Isolation Mechanism (AIM) is enabled, to ensure proper operation of the daemon the definitions of the Request_types used for the input and output drivers must include the statement:

```
     max_access_class:  system_high;
```

If this statement is omitted, the coordinator will leave requests in the queue indefinitely. See "Access Isolation Mechanism Considerations" above for more detail.

DRAFT: MAY BE CHANGED 2-11

By default, the IMFT user commands use the Request_type "imft". If you wish to define a default remote system for transfer requests, define the "imft" Request_type in the I/O daemon tables with the same values specified for the driver_userid, default_queue, max_queues, max_access_class, and device statements as are specified for the actual Request_type for transfers to that remote system. In addition, the following commands should be issued after using the create_daemon_queues command to make the "imft" Request_type a synonym of the actual request type:

```
     delete imft_*.ms
     add_name To_Site-Name_(1 2 ... N).ms imft_(1 2 ... N).ms
```

where To_Site-Name is the Request_type name as given in the I/O daemon tables and N -4 is the number of queues defined for that Request_type.

Device Definition for IMFT

IMFT requires the following device statement and substatements to define the input and output driver:

Device: <name>;

Defines the name of a major device and denotes the beginning of a device description. Any subsequent substatements (see below) apply to this device until the next Line, Device, or Request_type statement is encountered. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces.

driver_module: <name>;

For IMFT, <name> must be "imft_driver_".

line: <name>;

For IMFT, <name> must be "*".

args: <string>;

Defines the characteristics of this device. <string> is a quoted string consisting of a series of keyword/value pairs separated by commas. The syntax of a keyword/value pair is:

```
keyword= value
```

No space is permitted after the keyword and before the equal sign. If the value contains spaces, commas, quotes, or equal signs, it must be quoted.

Define the following keyword/value pairs for IMFT:

direction= <inout>

Specifies whether this is an input or output driver. <inout> must be either "input" or "output".

local_system= <Person_id>

Specifies the name of the local system. This name is used to validate the connection. See "Access Considerations" for more information. This keyword is required.

foreign_system= <Person_id>

Specifies the Person_id of the remote system. This Person_id is also used to validate the connection. See "Access Considerations" for more information. This keyword is required and must also be used in constructing the Request_type names, as indicated above.

input_description= <quoted_string>

Specifies the attach description for the input subchannel for this driver. Note that a quoted string is required thereby creating an entry with double quotes (see example). This keyword is required if the communications protocol being used is restricted to unidirectional channels (e.g., HASP).

output_description= <quoted_string>

Specifies the attach description for the output subchannel for this driver. Note that a quoted string is required thereby creating an entry with double quotes (see example). This keyword is required if the communications protocol being used is restricted to unidirectional channels (e.g., HASP).

io_description= <quoted_string>

specifies the attach description for the single channel used by this driver for both input and output. Note that a quoted string is required thereby creating an entry with double quotes (see example). This keyword should be used if the communications protocol being used permits bidirectional channels; otherwise, the input_description and output_description keywords (above) must be used.

mode= <auto/manual>

Specifies whether this driver is to operate with or without operator intervention. <auto/manual> must be either "automatic" or "manual". This keyword is optional and defaults to "manual". Use of "automatic" implies either "auto_receive= yes" or "auto_go= yes" as appropriate, causes the driver to wait indefinitely for completion of the connection sequence, and causes the driver to wait for the remote system's driver to reconnect again whenever the remote driver disconnects. Use of "manual" implies either "auto_receive= no" or "auto_go= no" as approoriate, causes the driver to wait no more than five minutes for completion of the connection sequence, and causes the driver to logout whenever the remote system's driver disconnects.

auto_go= <yes/no>

Specifies whether an output driver should immediately begin to transmit files and subtrees to the remote system or should instead wait for an operator command after the connection is established. <yes/no> must be either "yes" or "no". This keyword cannot be specified for an input driver. This keyword is optional and defaults to "no" if "mode= manual" is specified and defaults to "yes" if "mode= automatic" is specified.

auto_receive= <yes/no>

Specifies whether an input driver should immediately wait for files and subtrees from the remote system, or should wait for an operator command after the connection is established. <yes/no> must be either "yes" or "no". This keyword may not be specified for an output driver. This keyword is optional and defaults to "no" if "mode= manual" is specified and defaults to "yes" if "mode= automatic" is specified.

allow_remote_request= <yes/no>

Specifies whether an input driver should accept requests from the remote system for the transfer of files from the local system. <yes/no> must be either "yes" or "no". This keyword may not be specified for an output driver. This keyword is optional and defaults to "no".

explicit_access= <yes/no>

Specifies whether an explicit ACL term is required for requests for remote transfer. <yes/no> must be either "yes" or "no". If it is "yes", then if a request for transfer of a file to the remote system originated at the remote system (through use of the -source control argument to the enter_imft_request command), the transfer cannot take place unless the Person_id of the local system appears explicitly in the ACL of the file. This keyword may not be specified for an input driver. This keyword is optional and defaults to "yes".

max_access_class= <quoted_string>

Specifies the maximum access class for data that may be transferred across this connection. The access class specified must be less than or equal to the common access class ceiling between the two systems (as defined in Section 4 of this manual). If given, this keyword must be specified for both the output driver on one system and the corresponding input driver on the other system. If not given, the common access class ceiling is used as the limit for data transfer.

minor_device: <name>;

Defines the name of a minor device and denotes the beginning of a minor device description. Any subsequent substatements (see below) apply to this minor device until the next Line, Device, or Request_type statement is encountered. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces. The minor_device statement is used to identify each of the minor devices of the output driver. The following substatement is required for each minor device:

default_type: <name>;

Identifies the Request_type serviced by this minor device. <name> must be the same as the name of the Request_type that identifies this minor device; it must be the same as the Person_id of the foreign system (see above), prefaced by either "From_" or "To_".

Example 1

```
Device:                 system_m_ft_out
 driver_module:           imft_driver_;
 line:                    *;
 args:                    "direction=output, local_system=MIT,
                             foreign_system=System-M
                             ods=""hasp_host_ -comm hasp
                                   -tty b.h203.rdr1 -device reader"",
                             ids=""hasp_host_ -comm hasp
                                   -tty b.h203.pun1 -device punch"",
                             explicit_access=no,
                             auto_go=yes";
minor_device:           to;
 default_type:            To_System-M;
minor_device:           from;
 default_type:            From_System-M;
```

In the above example, a device is defined for an IMFT output driver.

Example 2

```
Device:                 mit_file_transfer_in;
  driver_module:          imft_driver_;
  line:                   *;
  args:                   "direction=input, local_system=System-M,
                             foreign_system=MIT,
                             ids=""hasp_workstation_ -comm hasp
                                  -tty b.h203.rdr2 -device reader"",
                             ods=""hasp_workstation_ -comm hasp
                                  -tty b.h203.pun2 -device punch"",
                             allow_remote_request=yes,
                             auto_receive=yes";
  default_type:           To_MIT;
```

In the above example, a device is defined for an IMFT input driver.

IMFT Request Type Definition

IMFT requires the following Request_type statement and substatements. If you wish to use a queue other than the default queue, use the max_queue and default_queue: statements (see the Bulk I/O manual).

Request_type: <name>;
Defines the name of the Request_type and denotes the beginning of a Request_type description. Any subsequent statements (see below) apply to this Request_type until the next Line, Request_type, or Device statement is encountered. <name> must be the <name> specified as the foreign system name in the input and output driver definitions, prefaced by the string "From_" or "To_".

generic_type: <name>;
IMFT requires the generic type <name> to be: imft.

driver_userid: <Person_id.Project_id>;
Must identify the user selected above to run this connection.

accounting: <name>;
IMFT requires <name> to be "nothing".

max_access_class: system_high;
Must be specified exactly as shown or the I/O

coordinator will leave requests in the queues indefinitely.

device: <name>;

Specifies the devices that can be used to process requests of the associated type. If the Request_type is for requesting transfers from the remote site, one device statement is required, specifying the corresponding minor device of the output driver; if the Request_type is for transferring files and subtrees to the remote site, two device statements are required: one for the input driver and one for the corresponding minor device of the output driver (see example).

Example

```
Request_type:          To_System-M; /* used by input and
                                 output drivers */
  generic_type:         imft;
  driver_userid:        IMFT.Daemon;
  default_queue:        3;
  accounting:           nothing;
  max_access_class:     system_high;
  device:               system_m_ft_out.to;
  device:               system_m_ft_in;

Request_type:          From_System-M; /* used by output
                                         driver only */
  generic_type:         imft;
  driver_userid:        IMFT.Daemon;
  default_queue:        3;
  accounting:           nothing;
  max_access_class:     system_high;
  device:               system_m_ft_out.from;
```

In the above example, a pair of Request_types is defined for an IMFT connection with the system named "System-M".

SECTION 4

USER COMMANDS

The Inter-Multics File Transfer Facility (IMFT) allows files and subtrees to be transferred between Multics systems. IMFT is queue driven, i.e., your requests are placed in a queue for later action similar to an output request. IMFT lets you enter, list, cancel, or move requests via the following commands:

- enter_imft_request (eir)
  submits a request to transfer files or subtrees

- list_imft_request (lir)
  lists the requests in the IMFT queues

- cancel_imft_request (cir)
  cancels requests in the IMFT queues

- move_imft_request (mir)
  moves IMFT requests from one queue to another

The user may request IMFT to transfer files from the system at which he/she is logged in (the "local" system) to some other system (the "remote" or "foreign" system), or to transfer files from the foreign system to the local system. In the discussion below, the system from which the files are to be transferred is referred to as the source system, and the one to which they are to be transferred is referred to as the target system.

ACCESS REQUIREMENTS

To transfer a file or a subtree from the source system to the target system, the conditions detailed below must be met.

For files, the user on the source system must have at least "r" access to the file; for subtrees, the user must have at least "s" access to the root of the subtree and each directory contained therein and at least "r" access to each file in the subtree.

The daemon process on the source system that transfers the file or subtree must also have the same type of access as described above for the source system's user. Additionally, the daemon must also have at least "s" access to the directory containing the file or subtree in order to verify that the user has the proper access. The identity of the daemon can be determined using the print_request_types command.

The user on the target system must have "sma" access to the directory into which the file or subtree is to be placed. The source system user and the target system user are the same unless the -foreign_user control argument is specified.

The daemon process on the target system that receives the file or subtree must also have "sma" access to the directory into which the file or subtree will be placed. In addition, this daemon must have at least "s" access to the directory containing that directory in order to validate that the target user has the proper access.

The ability of a user on the local system (LPerson.LProj) to transfer files to or from the foreign system is controlled by the access granted to the local user by the user on the foreign system (FPerson.FProj) to the segment:

```
>udd>FProj>FPerson>LSite.imft.acs
```

on the foreign system where LSite is the name of the local system. In order to request that files be transferred from the foreign system, LPerson.LProj must have read access to the above-named segment; in order to transfer files to the foreign system, LPerson.LProj must have write access to the segment. (Note: when setting write access on an ACS, it is advisable to set its maximum length to 0, to prevent it from acquiring contents.)

In the case of remote requests (i.e., use of the -source control argument), the foreign and local users must have all the same access as if the request had been issued at the source system, in addition to read access to the ACS as indicated above. Further, the site administration of the source system may restrict transfer of files by remote request to those files whose ACLs have explicit terms for the IMFT daemon; i.e., an ACL term of "r *.*.*" would not be sufficient to permit the file to be transferred.

The identity of the daemon on the foreign system and the name of the local system used to form the name of the ACS segment above can be determined by using the print_request_types command on the foreign system.

Assume that the user Kelley.SysMaint on MIT wishes to send the file:

```
   >udd>sm>pbk>test>new_version.pl1
```

to the directory:

```
   >udd>ssa>pbk>imft>mit
```

on System-M where his user ID is PKelley.SiteSA. Further assume, that the daemon on both systems is IMFT.Daemon and that the names of the source and target systems as given by print_request_types are MIT and System-M respectively.

On MIT (the source system), Kelley.SysMaint must issue the following set_acl commands to ensure that he and the daemon have proper access:

```
set_acl >udd>sm>pbk>test>new_version.pl1 r Kelley.* r IMFT.Daemon
   set_acl >udd>sm>pbk>test s IMFT.*
```

Note that any ACL term which grants appropriate access is sufficient. In other words, an ACL term on >udd>sm>pbk>test for IMFT.Daemon.*, IMFT.*.*, *.Multics.*, or even *.*.* is sufficient to give the daemon proper access; it is not necessary to use an ACL term for IMFT.Daemon.* explicitly although, of course, that is also acceptable.

On System-M, PKelley.SiteSA must issue the following set_acl commands to ensure proper access to receive the file:

```
   set_acl >udd>ssa>pbk>imft>mit sma PKelley.* sma IMFT.Daemon
   set_acl >udd>ssa>pbk>imft s IMFT.Daemon
   set_acl >udd>SiteSA>PKelley>MIT.imft.acs w Kelley.SysMaint
```

Once proper access is established, Kelley.SysMaint can then issue the command line:

```
   eir >udd>sm>pbk>test>new_version.pl1 -tpn
   >udd>ssa>pbk>imft>mit>=== -fu PKelley.SiteSA -ds System-M
```

For a related example, suppose that the same user wished to transfer the same file, but wished to issue the request while logged in at System-M as PKelley.SiteSA. In that case, all the access described above must be established, but in addition, Kelley.SysMaint must issue the following command at MIT:

```
   set_acl >udd>SysMaint>Kelley>System-M.imft.acs r PKelley.SiteSA
```

PKelley.SiteSA may now request the transfer by issuing the commanu line:

```
   eir >udd>sm>pbk>test>new_version.pl1 -tpn
>udd>ssa>pbk>imft>mit>=== -fu Kelley.SysMaint -source MIT
```

NOTE TO DOCUMENTATION: In the following section "Notes on AIM", on page 4-4.1, all references to "local" and "foreign" system must be changed to refer to "source" and "target" system respectively.

Name: enter_imft_request, eir

The enter_imft_request submits requests to transfer files or subtrees to or from remote Multics systems using the Inter-Multics File Transfer (IMFT) facility.

Usage

```
     eir transfer_specs -control_args
```

where:

1. transfer_specs
   specify the files or subtree to be transferred and has the following format:

   path {-target_pathname equal_path},
   path {-tpn equal_path}
   path specifies the relative pathname of files and/or subtrees to be transferred. The star convention is accepted. If supplied, the equal_path is the relative pathname of where the files and subtrees will be placed on the target system. The equal convention is accepted. The target pathname is converted to an absolute pathname relative to the working directory on the local system. If not given, the files and subtrees are given the same pathname on the target system.

2. control_args
   may be chosen from the following:

   -file, -f
   specifies that transfer requests should be issued only for files which match the transfer_specs. If a transfer_spec does not use the star convention and there is no matching file, an error message is issued. (Default -- issue requests for matching files and subtrees).

   -subtree, -subt
   specifies that transfer requests should be issued only for subtrees which match the transfer_specs. If a transfer_spec does not use the star convention and there is no matching subtree, an error message is issued.

-chase
specifies that transfer requests should be issued for the targets of any links which match the transfer_specs. (Default -- chase links for any transfer_specs that do not use the star convention; do not chase links for any transfer_specs that use the star convention)

-no_chase
specifies that transfer requests are not issued for the targets of any links which match the transfer_specs.

-destination STR, -ds STR
identifies the remote system to which the files and subtrees are to be transferred. (Default -- imft). STR must be one of the names listed by the print_imft_sites command.

-source STR, -sc STR
identifies the remote system from which the files and subtrees are to be transferred. STR must be one of the names listed by the print_imft_sites command. If neither -destination nor -source is specified, the default is -destination imft.

-queue N, -q N
specifies that the requests be entered in priority queue N where N is an integer between 1 and 4 inclusive. (Default -- depends on the destination or source specified)

-brief, -bf
suppresses the messages providing the request IDs of the requests entered by this command.

-long, -lg
prints the messages providing the request IDs of the requests entered by this command. (Default)

-long_id, -lgid
prints the long form of the request ID in any messages.

-short_id, -shid
prints the short form of the request ID. (Default)

-absolute_pathname, -absp
: prints the absolute pathname of the file or subtree along with the request ID for each request entered by this command.

-entryname, -etnm
: prints only the entry name of the file or subtree along with the request ID for each request entered by this command. (Default)

-notify, -nt
: sends notification of successful initiation and completion of each transfer request. The notifications are sent on the the local and remote systems. (Default)

-no_notify, -nnt
: suppresses notifications of successful transfer on both systems. Any errors detected during transmission will still cause mail to be sent regardless of the use of -no_notify.

-merge_directories, -mdr
: specifies that if there is a directory on the target system with the same name as one of the names on the root directory of the subtree being transferred, the contents of the source subtree will be merged with the target subtree. If the target entry is not a directory, processing will continue as though -replace_directories had been specified. Any directories within the subtree are treated in a similar fashion with respect to name duplications. See the Notes for a description of the treatment of files within the subtree. (Default)

-replace_directories, -rpdr
: specifies that if there is an entry on the target system with the same name as one of the names on the root directory of the subtree being transferred, that name will be removed from the target entry; if the target entry has only one name, it will be deleted.

-foreign_user Person.Project, -fu Person.Project
: specifies the identity of the user at the remote system on whose behalf the transfer requests are being entered. Notifications on the remote system are sent to this user. See "Access required" below for further information. (Default -- the same as the user on the local system)

Notes

If conflicting control arguments (e.g., -notify and -no_notify, or -destination and -source) are given on the command line, the rightmost control argument takes effect.

If there is an entry on the target system with the same name as one of the names on the file being transferred, that name will be removed from the target entry; if the target entry has only one name, it will be deleted. No distinction is made between files specified in a transfer_spec and files contained in a subtree with respect to the handling of duplicate names on the target system.

Examples

```
eir **.pl1 -tpn <x>===.new -ds MIT
```

transfers all files and subtrees in the working directory whose name ends with the pl1 suffix. If the local working directory is >udd>m>gmp>w, a file named "foo.pl1" will appear on the remote system as >udd>m>gmp>x>foo.pl1.new

```
eir my_subtree -ds System-M -mdr
```

transfers the subtree named "my_subtree" in the working directory to the same point in the hierarchy on the remote system. Assume (1) that there already is a foreign directory named my_subtree, (2) that the local my_subtree contains two files named file1 and file2 and a directory named subdir1, and (3) that the foreign my_subtree also contains two files named file1 and file3. After the transfer is completed, the foreign my_subtree will contain three files -- file1 and file2 from the local system and file3 from the foreign system -- and one directory -- subdir1 from the local system along with the contents of the local subdir1.

```
eir >udd>sm>Kelley.profile -tpn >udd>m>PKelley.= -source MIT
     -fu Kelley.SysMaint
```

transfers the segment >udd>sm>Kelley.profile from MIT on behalf of the MIT user Kelley.SysMaint, and places it in >udd>m>PKelley.profile on the local system.

Name: print_imft_sites

The print_imft_sites command displays the names of foreign sites that can be used with the -source or -destination control arguments of enter_imft_request.

Usage

```
print_imft_sites
```