.IEN: 36
Section: 2.5.2.4.3

MASSACHUSETTS COMPUTER ASSOCIATES, INC.
26 Princess Street
Wakefield, Massachusetts 01880

File Package:

The File Handling Facility for the National Software Works

by
Paul M. Cashman
Ross A. Faneuf
Charles A. Muntz

Revised
December 27, 1976
CADD-7612-2711

This work was supported by the Advanced Research Projects Agency of the Department of Defense and by Rome Air Development Center. It was monitored by Rome Air Development Center under contract number F30602-76-C-0094.

TABLE OF CONTENTS

PREFACE

Charles Muntz and Paul Cashman were responsible for the first version of the File Package Design Specification (MCA Document Number CADD-7602-2011). This document was dated February 20, 1976. Ross Faneuf is the chief author of this revison of the File Package Design Specification (MCA Document Number CADD-7612-2711).

The reader of this document is assumed to have read "MSG: The Interprocess Communication Facility for the National Software Works" (Massachusetts Computer Associates, Inc., Document Number CADD-7612-2411, Bolt Beranek and Newman, Inc., Report Number 3237).

Chapter 1:

Overview

The primary function of the NSW File Package (FP) is the transportation of physical copies of user files into and out of NSW file storage space. This usually (and most importantly) occurs in order to place acceptable physical copies of NSW files in tool workspaces as input files, and to move output files from these workspaces back into the NSW file system. Part of this function is to make output from one tool available and acceptable to another tool, even if these reside on different hosts. Secondary functions include "importing" user files into NSW space under direct user request; "exporting", the complementary operation, and maintenance of NSW file space (i.e. phyusical copy deletion).

A File Package resides on every NSW Tool Bearing Host (TBH). Every such host also includes some NSW-controlled file space which provides storage space for physical copies of NSW files. Only NSW (Works Manager and/or FP) has access to this space. (Additionally, NSW will control the workspaces used by tools; the FP considers these to be external to NSW file space). The Works Manager (WM) contains a catalog of NSW files; the NSW file system consists of all files entered in this catalog. At one level, an NSW file consists of an NSW file name, a list of physical copy descriptors for each copy of the file, and a global type descriptor which records the physical and structural attributes of the file. (See appendix A) The NSW name is (usually) assigned by the user and is syntactically uniform for the entire NSW file system. Each physical copy descriptor includes the network address of its storage host, and all information required to access the file; this information is meaningful only to the operating system of the storage host, and is not uniform accross the file system. The global type descriptor contains the information needed to encode/decode the file for transmission of the file between hosts in different families.

The multiple physical copies are logically indistinguishable, and the choice of which will be used in a given operation is of no concern to the user -- indeed, cannot be affected by him, as he has no access to the physical copy list. The logical equivalence is clear when physical copies reside on several machines running the same operating system (a "host family"), but less so for copies residing on different host types. We assert equivalence based on the following notion: at the logical level, the physical copies represent identical sequences of lines (or records); whenever the NSW file system has been able to capture and encode the physical/structural attributes of the file on its native operating system, translation of the file between different hosts, preserving logical equivalence, becomes possible.

The WM grants access to given files by given users: once granted, it is the FP's task to make a suitable physical copy available for the desired access (direction of the file motion is

immaterial). Two FP's are involved in a copy operation if no local source copy exists: "receiver" FP on the host desiring the copy; and "donor" FP on a host containing an existing copy. The receiver FP drives the copy procedure: it has the task both of choosing the donor copy, and creating a copy with equivalent logical structure.

The receiver FP is given the right to select among multiple physical copies if more than one exists.

Three cases exist:

1). Local copy: There is an original on receiver's host.

2). Family copy: There is an original on a foreign host in the same operating system family, supporting the local file formats.

3). Non-Family copy: There is an original on a host which does not support the local file formats. Translation is forced.

Local Copy

The most efficient way to make the copy is by using an original on the local host. The local copy procedure can be implemented entirely within the local operating system, but serves to identify procedures common to other modes of copying. Only the one FP is involved in a local copy operating.

Family Copy

The analogy with archival of files on magnetic tape is useful. Suppose the host operating system supports archival. Such a process encodes an arbitrary file in serial fashion so that the original file may later be reproduced in programmatically indistinguishable fashion. We characterize the save portion of the operation by the following steps:

A1. Locate the contents of a named file.

A2. Read its physical structure characteristics.

A3. Record its physiccal structure.

A4. Until end-of-file

A4R. Read a "block" (machine dependent unit) of the file and serially encode it.

A4W. Write the block on tape.

A5. Close the file.

A similar procedure is used to restore the file.

The "save" procedure is exactly what the donor's FP needs to send a family copy, while the "restore" procedure is used by receiver; a one-way MSG direct connection assumes the role of the magnetic tape. Thus, two hosts in the same family can exchange files with as much fidelity as one finds with save/restore. Host families will use a private (i.e., determined by a consensus of implementors in that family) dump/restore encoding for file transfers among members.

Non-Family Copy

When no family copy exists, the receiving FP must attempt to reproduce the logical structure of a file in the local file format. The receiver exercises its right to choose the donor copy. Once selected, the donor's FP will be asked to send an encodement of the file, which the receiver FP must then decode and store.

The file encodement has two components: one component is a global file type descriptor, which describes the physical and structural attributes of the file. This descriptor is available from the WM, keyed by a global file type string. There is a unique descriptor for each native file format recognized by NSW.

The second component is an intermediate language (IL) which encodes the file contents. The syntactic productions which may legally appear in the encodement of a file depend on its global file type -- for instance, IL sequence numbers may not appear in the encodement of a file with no keys.

Together, these components form a system for encoding files for transmission and translation between unlike hosts. The system has several properties:

1) It must preserve a maximum of the information contained in the logical structure of the file as created on its native operating system

2) It should be host independent

3) It should include compaction for efficient network transmission

4) It should be reasonably easy to encode and decode

Several chapters of this document are devoted to the specification of this system. We feel that no existing physical file encodement adequately satisfies the above criteria, and therefore use a (new) technique that is nobody's encodement. Its compression techniques apply equally well to binary and text files, and is still a reasonable encodement for the private intra-family dump/restore copy.

Notes:

This specification only details operations on sharable devices - disks. The status of non-sharable devices is not currently clear enough to allow a detailed presentation. We still omit a discussion of the following issues:

- ☆ Device assignment
- ☆ Device control
- ☆ Operator communications
- ☆ Standard device names

Two hooks in the FP allow for expansion toward non-sharable devices:

1) The physical copy descriptor (see) allows host-dependent strings for the physical copy name and location) physical device descriptor. These elements can name and locate files on non-sharable devices.

2) The IL includes productions which allow files on a tape volume to be moved as a related group. (See Chapter 4.)

Chapter 2:

File Package Functions

I. Introduction

This chapter deals with the functions performed by the File Package. We first describe in some detail the several main kinds of information that are passed to FP's as arguments -- particularly the physical copy descriptor and global file descriptor. We then describe the externally callable FP procedures, and finally the internal procedures which implement cross-network copy operations.

Many of the above are described twice -- first, in rather general terms in order to convey the conceptual elements of the specification, and second to give an exact specification for the calling sequences, form of arguments, etc. The reader is assumed to be familiar with the spec for MSG, ... with the NSWB8 message encodement, and the NSWTP for inter-component procedure calls.

II. The Physical Copy Descriptor, Global File Descriptor, and and the NSW File Catalog.

1. Introduction

Four pieces of information about files are passed to FP's. Depending on the function (procedure) invoked, all may be present, or only a subset. The particular cases will be clear in the specification of the FP interface. The four items are:

1) Physical Copy Descriptor
2) Filespace (directory) password
3) Global File Type string
4) Global File Descriptor

We may quickly dispose of (2) and (3). The directory password grants access to the physical file copy filespace specified in (1). It is a separate item so that it may be omitted where possible (i.e. for NSW filespace). As a security precaution.

Items (3) and (4) are in direct relation: (3) is the key which locates (4) in the WM's Global File Type Table. (See appendix ...). At this point, it may be fruitful to see appendix B.
Both are included for convenience (efficiency). A FP for which the global file type is native will not need to refer to the global file descriptor.

We now proceed to a discussion of (1).

2. The Physical Copy Descriptor

It must be possible to locate each physical instance of a file which participates in a FP operation; the Physical Copy Descriptor (PCD) contains this information. A PCD is a list of following file components:

HOST	(integer)	Network address of host where file is stored
DIRECTORY	(string)	Name of directory in which file is stored
NAME	(string)	Host-dependent file name
PHYS	(string)	Host-dependent location information
ILFLAG	(boolean)	TRUE if file copy is an IL encodement

The first four items continually refine the specification of the location of the physical file copy. The encodement flag is included in the PCD because it must be set on a per-copy basis.
We now discuss the four location-specifying items.

A. HOST

This integer must be correctly set for any existing physical file copy, and correctly returned by a FP generating a new physical copy. It is the only component of a PCD meaningful to any FP, no matter whether the file copy described is in the local file space or not: Indeed, a FP bases its analysis of the kind of copy operation specified on this information (local, family, non-family). The network addresses are interpreted as precisely those used by MSG as a component in process addresses. (Note: the local host address is available to a FP through the WhoAmI primitive in MSG).

B. NAME

This string contains the file's name, necessarily in the format of the operating system where the physical copy resides. We expect it will not be meaningful to allow a PCD for an existing physical file copy to have a null NAME. The format of this string, its length and syntax, depend on the operating system of the storage host. Only the FP on that host need interpret the string. (The same comment applies to DIRECTORY and PHYS).

C. DIRECTORY

If meaningful, this string names the file space directory in which the file resides.

D. PHYS

This item contains any further information required to locate the file copy, including any structural or device-dependent information.

We have noted that only the storage host's FP need interpret the NAME, DIRECTORY and PHYS. A FP not in the same family as the storage host will simply pass these items on without looking a them -- indeed, this may be true even for family copy operations. The FP at the storage host will use these strings to locate the file copy (more precisely, it knows how to map the "canonical" name formed from these strings into a name which makes sense, and is complete, for its operating system).

Therefore, for each family H in the NSW system, the content of NAME, DIRECTORY and PHYS is left to the H-family File Package implementors. This includes the decision whether any meaning will be assigned to DIRECTORY and PHYS, and the mapping from the PCD's "canonical" name into the H-family operating system structure.

Examples:

We give examples of the four location items for files in the TENEX and OS/360 families. (These are not to be construed as final or required mappings, but as examples only). Our expectation and hope is that the PCO is sufficiently flexible and complete to allow similar mappings for any operating system family which may be added to NSW.

TENEX

File is MTA0:<BABBAGE>DIFF.ANALYZER;12 with password 'CHAS' on storage host BBNB.

```
HOST        =   49
DIRECTORY   =   'BABBAGE'
NAME        =   'DIFF.ANALYZER;12'
PHYS        =   'MTA0:'
```

(If needed the password will be sent as a separate argument)

OS/360

File is NSW.PDS.PROGRAMS (FILEPKG). It is uncatalogued, but resides on a disk with serial number NSWDSK, on UCLA-CCN.

```
HOST        =   5
DIRECTORY   =   null
NAME        =   'NSW.PDS.PROGRAMS (FILEPKG)'
PHYS        =   'UNIT=3330, VOL=SER=NSWDSK'
```

If the above file were catalogued, a possible variant would be:

```
DIRECTORY   =   'NSW.PDS.PROGRAMS'
NAME        =   'FILEPKG'
PHYS        =   null
```

3. The Global File Descriptor

It is possible to locate and access any physical file copy, given its PCO and, perhaps, directory password. We now address the problem of encoding and decoding the file for non-family transmissions.

Our starting point was to conceptually divide the world into two groups -- record-oriented machines and paper-tape oriented machines. Record machines see files as sequences of logically grouped bytes, these "groups" being called records. Paper tape machines see files as streams of bytes with no logical grouping structure overlaid on the bytes. (The term "paper tape" does not refer literally to the device only). Most of our attention has been devoted to text rather than binary files, as the automatic

translation of text files between hosts as required for various NSW tool accesses has been our most pressing problem. For text files, a further characterization appears to be that record machines enjoy file formats that are structurally rich, and paper tape machines have files that are rich in format effectors (see chapter..., Translation Semantics)

We feel that it is possible to translate a given file from machine to machine by preserving the notion that in each incarnation of the file, it is formed of a logically equivalent sequence of "lines" or "records", and therefore all physical copies of a single NSW file are logically equivalent. It follows that there must exist a single characterization of these multiple copies which is global to the NSW system, i.e., which applies to each distinct storage or tool bearing host family where a physical copy may appear. This characterization is associated with the host creating the file in the first place.

This leads us to the notion that every file in the NSW file system has what we call a Global File Type (GFT), which is specified at the point of origin of the file. This GFT characterizes identically each physical copy of the file, whether these copies reside on a host with the same operating system as the original creator or not.

Let us give an example: consider a simple EBCDIC text file generated by an editor on OS/360. We assign this a GFT of '360-TXT' (n.b. this is an example, and not necessarily a GFT in the NSW system). Then we imagine this file is transmitted to a TENEX, via encodement into IL by the originating 360 and decoding from IL by the receiving TENEX. The GFT of the new TENEX copy is still '360-TXT', and is associated with the NSW catalog entry for the file, rather than the PCD for the TENEX copy. (In fact, the copy probably has exactly the physical characteristics of some file format native to TENEX, e.g., '10X-TXT' or '10X-SOSTXT.', but its GFT remains '360-TXT' nonetheless).

The immutability of GFT across non-family host file transfers is crucial, as it allows the NSW to base all such file transfers/translation on data stored centrally in the WM's data base and passed to FP's with initial procedure calls, rather than relying on FP's eventually getting enough information from each other to allow file translation. It also allows receiver FP's to immediately reject a file transfer if no local equivalent of the GFT is known (or currently implemented).

To expand on this: for each operating system family H in NSW, it is the responsibility of the H-family File Package implementors to specify the GFT's for each file format in their operating system which will be visible to NSW, and to generate the corresponding Global File Type Table (GFTT) entries. Then the implementors of FP's on other hosts must figure out their equivalent of these GFT's on their operating systems. Each GFT with such an equivalent has a (potential) implementation of the cross-host file translation encodement, and the rejection of any GFT without such an equivalent

is always determined and well known. Thus, at any point in the development of the NSW system, the set of possible cross-host translations is well known, and depends on information in the NSW data base rather than on information embedded in any physical file encodement, and unavailable until transmission has actually begun.

The infomation required by a FP willing to create a local equivalent of a foreign GFT is in the Global File Descriptor (GFD), which itself is a strightforward encodement into NSW88 of the appropriate GFTT entry. It is a list of six items:

CLASS	(integer)	Binary or Text
KEYS	(integer)	No keys or keys
DIMENSION	(integer)	Overprint characteristic for text files
KEYDESCRIPTOR	(integer)	Number of digits in sequence number (key) in IL
TABDESCRIPTOR	(structure)	Describes all horizontal and vertical TAB's for text files
ILBYTESIZE	(integer)	Number of bits in byte in IL encodement and network transmission

These items correspond exactly to the items in the description of the GFTT (See Appendix B). We reiterate that description here, with slightly different emphasis.

A. CLASS

Currently, this item simply specifies whether the file is invariably text, or binary - in essence, non-text. We may choose at later date to further subdivide the class of binary files -- e.g., to identify an NLS file.

At the current time, we are chiefly interested in the distinction between text and non-text files, as we expect most cross-family file transfers to involve text files. The only exception known at present is the output of cross-compiler or cross-assembler tools, and these are in the class of strictly binary files, requiring only mapping from one operating system file structure to another. By contrast, text files also require content mapping, e.g., of format effectors.

The fill character for files compressed by the IL encodement is implied by CLASS. For text files, the fill character is the blank; for binary files, the fill character is ZERO.

The values for CLASS are:

1. Text
2. Binary

B. KEYS

This item currently specifies whether a file type has keys in the OS/360 sense. A file type with keys allows (or requires) the appearance of IL sequence numbers in its IL encodement. Treatment of keys is particularly important for mapping files between paper tape and unit record families.

Values for KEYS are:

1. No keys
2. Keys.

C. DIMENSION

This item extends the information supplied by (A) and (B), partly overlapping. It applies particularly to text files formatted for line printing, helping to identify the complexity of the paper tape/unit record transformation. The values assigned to DIMENSION are to be interpreted as follows:

1. Stream data, unkeyed, e.g., an ASCII text file on TENEX without line sequence numbers. The basic paper tape file format, but not necessarily text.

2. Record/line oriented file, e.g., TENEX SOS file or ordinary blocked record file on OS/360. Not necessarily a text file.

3. A text file containing non-overprinting format effectors, particularly one formatted for a line print, e.g., containing form feeds.

4. A text file like (3), but including overprinting format effectors,-e.g., backspace. We anticipate using values (3) and (4) specifically to identify line printer files.

D. KEYDESCRIPTOR

This item simply specifies the number of digits (characters) in the IL sequence number for keyed files.

E. TABDESCRIPTOR

This structure specifies all the tab-like format effectors which may appear in the file. These are separated in horizontal and vertical tab characters, with the native operating system

interpretation of each. We allow both regularly and irregularly spaced tabs.

F. ILBYTESIZE

Is simply the byte size used for cross-family transmission of the IL encodement of this file type. The minimum value is 8.

Example: The GFD for a TENEX SOS (line-oriented, sequence-numbered) file might be:

```
CLASS          =   1
KEYS           =   2
DIMENSION      =   2
KEYDESCRIPTOR  =   5
TABDESCRIPTOR  =   ((horizontal tab every 8th column)
                    (form feed is 60 lines))
ILBYTESIZE     =   8
```

4. The NSW File Catalogue

The File Package does not have direct access to the NSW File Catalogue, which is owned by the Works Manager. All the information required by the FP arrives in its initial procedure call argument list. From the FP's point of view, it needs PCD's and GFD's and is not concerned with the WM's means of obtaining this information.

The File Catalogue Entry is specified in (Appendix A). The FP receives PCD's and GFD's, never sees attributes, and is ignorant of the NSW File Name. Its main interaction with the File Catalogue is to confirm that a requested operation is complete so that the WM may update its data base.

However, for the import and expert operations, FP implementors are given some minimal additional information if they choose to implement a local cross-reference directory for backup purposes. This is the NSW file name string. With this, a FP can set up a cross-reference directory which identifies the physical copy in its NSW file space with its WM-assigned NSW file name. Creation of such a directory is an implementor option. On TENEX, for example, the local directory contains NSW file, TENEX file name, FP login directory, and calling WM process address. It allows the NSW operator to restore user files to the data base in the event of some data base failure.

III. File Package Functions

1. Introduction

There are six externally callable FP procedures. Four are WM-callable procedures, and two are FP-callable. Four involve file motion (copy) operations; one is a delete (file space maintenance) operation; and the last requests information (file analysis). There is also a single internal proccedure which implements the receiver side of cross-network file copying. The seven procedures are:

A. Copy Operations

1. WM-callable

   a. Import a file into NSW filespace (FP-IMP)
   b. Export a file out of NSW filespace (FP-EXP)
   c. Transport a file between non-NSW filespaces (FP-TRANSP)

2. FP internal procedure:

   a. Receive a file from the network (NETRECEIVE)

3. FP-callable

   a. Send me a file (FP-SENDME)

B. Delete Operation

1. WM-callable

   a. Delete your physical copies of a file (FP-DEL)

C. Analyze Operation

1. FP-callable

   a. Analyze a physical file copy (FP-ANAL)

FP-IMP is used by the WM to implement the user NET IMPORT and COPY commands, and the internal DELIVER procedure. FP-EXP is used to implement the user NET EXPORT command, and the internal OPEN procedure. FP-TRANSP implements the user NET TRANSPORT command. FP-SENDME is the procedure used to set up network file copy between two FP's. FP-ANAL is currently unused, but provided if some kind of file analysis becomes necessary.

In this chapter, we will specify each function in terms of the choices which must be made and the actions which must be taken. The next chapter exactly specifies the calling sequences for each function, including the NSWB8 encodement of all arguments. That chapter also describes the outer-level communications of the FP, i.e., the MSG-level dialogues. Another chapter sketches the WM-FP interactions, including the algorithms involved in making FP procedure calls within the WM.

2. Copy operations.

A. Wm-callable procedures

At a low level, FP-IMP, FP-EXP and FP-TRANS involve use of an identical file copy facility, and it might have been possible to replace the three with a single FP-COPY procedure. We have chosen to specify these three separate functions because each has quite different arguments, choice patterns, and dissimilar communications patterns (for which see chapter ---)

FP-IMP is used to incorporate a file into the NSW file system. External files may be taken from a tool's workspace (DELIVER), or from an external file space or I/O station as specified by the user (NSW EXEC: NET IMPORT command). In addition, the NSW EXEC: COPY command is treated as an importation, although the source file is already in NSW file space. FP-IMP is characterized by:

a. Unique source file physical copy
b. Destination filespace (NSW) maintained by FP
c. Possible maintenance of cross-reference directory
d. Usually a local operation without file translation,
d. NSW file always having same GFT as source file.

FP-EXP is used to create a physical copy of an NSW file in an external filespace. This is frequently due to a tool's need for some specific access requirement, requiring an appropriately translated physical copy (OPEN). FP-EXP also implements the NSW EXEC: NET EXPORT command. FP-EXP is characterized by:

a. Choice between multiple physical copies of NSW source file

b. Possible forced translation of file to satisfy tool's needs, as source and destination may have different GFT'S.

FP-TRANSP implements the NSW EXEC: NET TRANSPORT command. It is the simplest of the copy operations, characterized by:

a. No choices in source, destination, or possible translations

b. Does not involve NSW file space

The three copy operations above have one important shared characteristic: the procedure calls always go to a FP on the host where the new physical copy is to reside -- i.e., to the receiver host. Thus, a FP receiving one of these procedure calls will always be seeking to create a new file in the file space of the local host, whether this is NSW filespace or not. Indeed, FP-IMP must assume that local NSW filespace is involved. It is an error for FP-EXP or FP-TRANSP to receive a destination specified foreign to the local host.

File motion in copy operations is divided into three classes:

1. Local copy. Source and destination filespaces are on local host. This operation involves use of the local operating system procedures only. Since this is host dependent, the local copy operation has no specification in this document.

2. Family copy. Source and destination filespaces are on hosts using the same operating system. File transfer requires use of NETRECEIVE/FP-SENDME, but does not require IIL encodement of the source file. The transmission protocol is private, determined by the FP implementor for each host family rather than being specified in this document.

3. Non-Family. Source and destination filespaces are on hosts not sharing the same operating systems. File transfer requires use of NETRECEIVE/FP-SENDME as above, and also requires the IL encodement and inter-family transmission protocol specified here.

B. FP Internal Procedure

NETRECEIVE implements the receiver side of network file copy for both family and non-family file transfers. It is the issuer of FP-SENDME calls to remote FP's. It is the procedure responsible for decoding IL files into the local file formats.

C. FP-callable procedure

FP-SENDME is the function required by FP-IMP, FP-EXP and FP-TRANSP to request transfer of a source file residing on a foreign donor host. The calling sequence is identical whether or not the donor is in the receiver's family or not. The procedure involves

a dialogue between two FP's, one on each host, and is one of only two instances of an NSW component which talks to an instance of itself (the other being WM's in a distributed data base system).

The operation is driven by the receiver FP, i.e., the NETRECEIVE process. The FP-SENDME call specifies the source file and network transmission parameters. The donor FP's only responsibilities are to locate the file, correctly encode it into IL if necessary, and transmit it. The receiver FP has the responsibility for correctly translating the file for the desired access. FP-SENDME is characterized by:

a. FP-FP dialogue
b. Only operation directly using network connections
c. File translation into/out of IL

3. Delete opeation (WM-callable)

The FP-DEL procedure deletes a list of physical file copies on a single host. Maintenance of the file catalogue is done by the WM, and the FP's only procedure is to delete each specified file copy. These need not, in fact, be copies of the same NSW file. FP-DEL characteristics are:

a. All PCD's are on local host

4. Analyze Operation (FP-callable)

The FP-ANAL procedure is currently unspecified, but included as a hook in case a future need for such a function arises. The information gathered by the ANALYZE function specified in the preliminary specification is now available in the GFD and FP-SENDME calling sequence (q.v.).

WE now proceed to a more detailed specification of each function.

5. FP-IMP function

The arguments to the FP-IMP procedure are:

a. PCD, password, GFT and GFD of source file
b. NSW file name string
c. Flag to indicate deletion of source file

The results of FP-IMP are:

a. PCD of new file copy in NSW filespace

The FP-IMP procedure call always goes to a FP on the host where the WM wants to create the file copy. Note that the NSW-controlled file space into which the new file will go is not a FP-IMP argument. The receiver FP will select a file space, relieving

the WM of the burden of maintaining filespace(s) on each storage host. This implies the existence in each FP of a "File Space Data Base" which describes the local NSW file space. The design of this data base is left to the implementors of each host family FP.

A File Package receiving a FP-IMP procedure call performs the following operations: (Message handling, argument translation and checking, etc., is always assumed).

a. Locate and access some NSW file space for the new file copy.

b. Generate a unique local file name for the new copy. This and the file space description are entered into a new (result) PCD.

c. Locate and access the source file. This will usually be in some file space on the local host.

d. Copy the source file into NSW file space. If the file is on a foreign host, this requires a FP-SENDME call to a FP on that host.

e. (Option). Create a cross-reference directory entry for the new file using argument (b) and the information in the new file PCD.

f. Delete the source file, if requested by argument (c). Confirm to the WM the source file deletion (if requested) and creation of local cross-reference directory entry.

g. This will often be set for files being delivered by a tool.

h. Confirm to the WM the source file deletion (if requested) and creation of local cross-reference directory entry.

6. FP-EXP function

The arguments to the FP-EXP function are:

a. A list of PCD's of the NSW file to be exported
b. GFT and GFD of the NSW file
c. PCD of result file in non-NSW file space, and filespace password
d. List of desired GFT's for result file
e. Flag indicating whether to preserve a copy of the file in NSW file space, and the NSW file identifier string.

The results of FP-EXP are:

a. Completed PCD for result file
b. PCD for NSW copy, if created
c. GFT of result file.

FP-EXP is a more complex function than FP-IMP, and will provide a good motivation for a more complete discussion of the file copy operation. We now discuss the arguments, and the choices that must be made between them. Recall first that the FP-EXP procedure call goes to a FP on the host where the new external file is to be created: thus, the result file PCD must specify a file space on the local host, and the GFT's in the desired result list must all be native to the local host and known by the local FP.

The receiver FP is given the complete list of PCD's of the NSW file, and must choose one PCD as the source for the copy operation. This choice is based on considerations of efficiency, and thus may vary between implementations on different host families. A possible algorithm (and one implemented for the TENEX FP) is as follows.

i. Use a copy on the local host if it exists
ii. Use a copy on another host in the same family if one exists
iii. Use a copy on a foreign host which is already encoded in I.L.
iv. Use any other copy

Note that the translation problem for the file is independent of the source of the physical file copy, and depends only on the GFT of the NSW file.

The result file PCD may be incomplete: if the FP-EXP call is the result of a Foreman request to OPEN a file, the file name will usually not be specified. This must be created by the FP. The filespace description must be complete, and specify the local host. The PCD with name specified becomes the external file result PCD.

FP-EXP is more complex than FP-IMP not only because of the choice of source copy, but also because of the possibility of translating between source and result GFT's. Argument (c) is a list of desired result GFT's; there will be one entry in this list if the FP-EXP call results from an NSW EXEC: NET EXPORT command, and possibly more than one if the FP-EXP results from an OPEN call. (And the list might be empty, in which case FP-EXP defaults to the source (NSW) GFT). The listed is sorted with the most desired GFT first. The algorithm for choosing the result is:

a. Result GFT is GFT in list for which a translation from the source GFT is known, and most easily carried out.

If no known translation from source GFT to any desired result GFT is known, FP-EXP fails. Most translations will be between GFT's native to different hosts: e.g., from 10X-TXT to 360-TXT or 360-MAC20 REL to 10X-MAC20KEL; in these cases, the translation problem is identical to the translation problem between hosts discussed elsewhere. That is, the translation reduces to

knowing whether or not some foreign GFT is known to the non-family copy routine embedded in FP-SENDME. Some cases may involve translations between types native to single host families, e.g., between 10-SOSTXT (sequenced SOS file) to 10X-TXT (unsequenced ASCII file). Each such case is identified and defined by the implementors of each family FP.

The reason that the result GFT is the one for which translation is easiest, rather than the most desirable, is to allow tools able to do their own translation to do so, without burdening the FP. Thus, the TENEX SOS editor might have a desired GFT list of (10X-SOSTXT, 10X-TXT), and the FP would choose 10X-TXT if its source is also 10X-TXT, although the translation between the two is known.

Note that the GFT translation problem does not affect the choice of source physical copy. Each such copy of the NSW file has the same GFT and the same possibility of translation to the result GFT as any other. The choice of source copy has a significant effect on the efficiency and ease of the copy/translation operation, but no effect on its achievability.

Finally, FP-EXP may be requested to leave a copy of the file in local NSW filespace. This request will be ignored if a source copy in local file space already exists. The new physical copy will be in one of two forms:

a. If source GFT is native to the local operating system, file copy will be in that form
b. Otherwise, new physical copy will be in IL

The new copy is entered in the local cross-reference directory, exactly as done by FP-IMP. However, since source file deletion is not an option with FP-EXP, no confirmation is required (see section IV-3, WM/FP interactions).

After argument decoding, the FP-EXP function performs the following operations:

a. Choose the source physical copy of the NSW file.
b. Choose the result GFT of the result file
c. Copy the source file into the specified external file space. Simultaneously create a copy in NSW file space if requested.
d. If an NSW copy was created, enter it in the local cross-reference directory.
e. Return results to caller WM.

7. FP-TRANSP function

The arguments to the FP-TRANSP function are:

a. Source file PCD, password, GFT and GFD
b. Destination file PCD, password, GFT and GFD

FP-TRANSP has no results (except the standard NSWTP error descriptor).

FP-TRANSP is exactly FP-EXP with no choices as to source physical copy or result GFT, and no option to create an NSW filespace copy. If the translation from source to destination GFT is known, the copy operation is performed; otherwise, FP-TRANSP fails. So the operations carried out by FP-TRANSP are simply:

a. If possible, copy the source file into the destination file space

8. The NETRECEIVE function

The arguments to the NETRECEIVE function are:

a. Source file PCD, password, GFT and GFD
b. Primary result (destination) file PCD, password and GFT
c. Secondary result file PCD and password

The NETRECEIVE function implements the receiver side of all network transfer operations. It is responsible for translating from IL to the native result GFT, and implements the creation of the two local copies which may be required by FP-EXP. Since this is an internal FP procedure, we only require that the functionality of this procedure be supplied, and not that it be an actual procedure in any given FP implementation.

The source file must be on a foreign host, and both primary and secondary result filespaces must be on the local host. The primary result file is the usual result of the network copy operation; it is NETRECEIVE'S responsibility to translate the file received from the foreign host into the specified result GFT which must be a native type, and thus NETRECEIVE implements both the decoding of IL files and any translations between native GFT's. The secondary result file is created only when FP-EXP is asked to create a file copy in NSW file space; in this case, the copy's GFT is the source GFT, and the file will be in the local format corresponding to the GFT if it is a native type, or in IL if the file was received in IL.

NETRECEIVE does a FP-SENDME to a FP on the donor host in order to initiate the transfer. It transmits the source file information as received, without attempting to check it, using only the host address from the PCD. NETRECEIVE also chooses the maximum transmission byte and record sizes which it will accept. The byte size will generally depend on the operating system for intra-family transfers, and be the IL bytesize from the source GFD for

inter-family transfers. The transmission record size depends on local operating system constraints.

The response to the FP-SENDME specifies the actual transmission byte and record sizes, a connection identifier, and estimated file size. If required, NETRECEIVE uses the estimated file size to allocate local storage space for the result file(s). It then opens a direct binary receive connection to the donor host with the MSG OPENCONN primitive, using the connection identifier and byte size received from the FP-SENDME call. It then receives the source file over the connection, and closes the connection after receiving an end-of-transmission signal over it.

Either simultaneously with receiving the file, or after transmission is complete, NETRECEIVE creates the result file(s) requested. If it is creating a secondary result file in IL, it concatenates the transmission records and removes the transmission record syntactic tokens and byte counts, producing a pure IL encodement.

The operations performed by NETRECEIVE are:

a. Choose acceptable byte and record sizes
b. Execute FP-SENDME to donor FP
c. Receive FP-SENDME response, and open direct receive connection using supplied parameters
d. Receive the file, and create the result file(s)
e. Close the direct connection

9. FP-SENDME function

The arguments to the FP-SENDME function are:

a. Source file PCD, password, GFT and GFD
b. Receiver host address
c. Maximum transfer byte size and record size
d. Private intra-family information

The results of FP-SENDME are:

a. Connection identifier
b. transfer byte size and record size
c. File size (in bits)

The FP-SENDME function implements the donor side of all network transfer operations. The procedure call is identical for both family and non-family transfers. It is responsible for establishing the parameters of the network transfer, and for encoding sources files into IL for non-family transmissions.

The source file PCD must describe a file in the local file space. If the file must be encoded into IL, it is FP-SENDME's responsibility to correctly encode it according to the GFT/GFD.

Since the GFT will be a native type if translation is required, the GFD will not be required. The GFD will be required, however, if the source copy is already in IL; for in that case the FP must parse the IL in order to correctly break the file into transmission records.

The receiver host i.d. (MSG network addreess) is provided so that the FP can find out whether the transfer is family or non-family. It is also available from the parameter block provided for the MSG receive generic call that initiated the FP-SENDME, but is included as an argument for convenience.

The receiver FP specifies the maximum network transmission byte size it is willing to accept, and the maximum number of bytes it can accept in one transmission (record size). The donor FP handling the FP-SENDME then chooses a byte and (maximum) record size, and returns this to the receiver. We expect that the byte size will usually be identical to that in the GFD, but we allow FP's to choose a larger byte size if this is a convenience.
One network transmission byte always contains one right-adjusted IL byte in non-family transmission mode. The contents of a byte in family transmission mode are defined by the implementors of the FP for each family.

Argument (d) is also defined by the implementors of the FP for each given family, and may contain an arbitrary amount of information. It must be empty for non-family transfer.

Besides the byte and record sizes, the FP-SENDME function returns a connection identifier and estimated file size. The connection identifier is the argument required by the MSG OPENCONN primitive, allowing the donor and receiver to open the direct network connection for the file transfer. The file size is in bits, and is provided for FP's which must allocate disk space; it is considered an estimate of size only.

After sending its results to the calling receiver FP, the FP-SENDME procedure opens a direct network connection to, the receiver FP via the MSG OPENCONN primitive, using the connection identifier and size it specified in its response. Once the connection is open, the file is transmitted over it to the receiver. The transmission protocol will be the inter-family protocol described in chapter (---), or the private intra-family protocol created by each implementor.

In either case, the end-of-transmission signal is transmitted over the direct connection with the file data. (The end-of-transmission mark is part of the inter-family protocol. Each intra-family protocol must use an analagous technique). Once the end-of-transmission signal is sent, the FP SENDME procedure closes its end of the direct connection using the CLOSECONN primitive.

The FP-SENDME procedure is responsible for all encodement of files into IL, and for blocking IL encodements into transmission records for the direct connection. Note that the IL syntax forbids requires that IL records not be broken when the IL encodement is blocked for transmission.

The operations performed by FP-SENDME are:

a. Confirm existence and accessibility of source file, and lock it
b. Choose actual transmission byte and record sizes, and MSG connection identifier
c. Send response to receiver FP. (End of action if error response)
d. Open the direct binary send connection
e. Transmit the file using the appropriate protocol
f. Close the direct connection

10. The FP-DEL function

The arguments to the FP-DEL procedure are:

a. A list of PCD's of files to be deleted

The results of FP-DEL are:

a. A list of error descriptors

FP-DEL only supports the deletion of physical copies of NSW files. Thus the list of PCD's does not include passwords, as it is presumed the FP has acess to NSW filespace.

Each PCD in the argument list must specify a file in local NSW filespace. These need not be copies of only one NSW file, and the Works Manager is free to request the deletion of physical copies of many NSW files in one FP-DEL call.

FP-DEL invokes the usual local operating system procedures for file deletion, removing names from catalogs or directories. Updating the local cross-reference directory is each implementor's choice.

The result list will be empty (length 0) for successful deletion of all files. If any error occurs in attempting to delete any file copy, a result list is created with the same length as the argument list, and an error descriptor analogous to the NSWTP error descriptor is placed in the entry ccorresponding to the argument PCD which gave the error. FP-DEL then attempts to delete the remaining PCD's -- one error does not terminate FP-DEL.

The operations of FP-DEL are simply:

a. Delete the file specified by each argument PCD

11. The FP-ANAL function

This function is currently unspecified. Possible need for it arises if:

a. A GFT exists for which information about a particular PCD cannot be saved in the WM data base -- e.g., number of records

b. We implement a user function which returns file size (e.g., number of characters in listing file)

IV. File Package Interfaces via MSG

1. Introduction

This section of the description of FP functions focuses on the patterns of communication between a File Package component and the Works Manager or another File Package. In particular, we are interested in the kinds of messages received and sent by a FP using MSG. There are two subsections of this section: the first describes the kind of primitives invoked in MSG for each external FP procedure; the second describes the exact calling sequences and results of each procedure.

2. MSG Message Primitive interactions

Every FP procedure is initiated by the receipt of a message from MSG, and each at least sends a result message to the caller via MSG. A given FP instance handles at most one procedure call at a time, and the operations to implement this procedure are largely bounded by the receipt of the procedure call and the sending of the result.

Every message received or sent by the FP is encoded in NSWB8, and has the NSWTP format. In this section, we specify each message by showing the contents of the NSWTP header. (See latest NSWB8/NSWTP specification). This descriptor is:

(type, transaction-identifier, parameter, argument-list)

type will be 1 for a procedure call and 2 for a reply

transaction-identifier will always have a unique non-zero value created by the caller

The exaction specification of argument-list is the subject of subsection 3 below

We now summarize the communications for each externally callable FP procedure.

A. FP-IMP

FP-IMP is invoked by a procedure call sent to any FP instance on the destination storage host:

i. WM to FP: SendGenericMessage(1,tid,"FP-IMP",args)

The FP-IMP procedure responds to the caller with a specific result message:

ii. FP to call WM: SendSpecificMessage(2,tid,error-descriptor,results)

FP-IMP then receives a confirming message from the WM; only then does FP-IMP complete its action (deletion of source and entry in local cross-reference directory)

iii. Caller WM to FP: SendSpecificMessage(1,tid,confirmation,LIST(0))

confirmation: CHARSTR: "FP-IMPOK" success
"FP-IMPBAD" failure.

FP-IMP then re-confirms its final actions:

iv. FP to caller WM: SendSpecificMessage(2,tid,error-descriptor, LIST(0))

B. FP-EXP

FP-EXP is invoked by a procedure call sent to any FP on the destination storage host:

i. WM to FP: SendGenericMessage(1,tid,"FP-EXP",args)

FP-EXP then responds to the specific WM instance that called it:

ii. FP to caller WM: SendSpecificMessage(2,tid,error-descriptor,results)

C. FP-TRANSP

FP-TRANSP is invoked by a procedure call sent to any FP on the destination storage host:

i. WM to FP: SendGenericMessage(1,tid,"FP-TRANSP",args)

As above, FP-TRANSP responds to the caller WM:

ii. FP to caller WM: SendSpecificMessage(2,tid,error-descriptor,results)

D. FP-SENDME

FP-SENDME is invoked by a procedure call from the FP instance desiring a file copy to any FP on the host where the source file is stored:

i. FP to FP: SendGenericMessage(1,tid,"FP-SENDME",args)

FP-SENDME replys to the specific caller FP instance:

ii. FP to caller FP: SendSpecificMessage(2,tid,error-descriptor,results)

FP-SENDME then opens a direct connection, transfers the file, and closes the connection.

E. FP-DEL

FP-DEL is invoked by a procedure call to any FP on the host from which the NSW physical file copies are to be deleted:

i. WM to FP: SendGenericMessage(1,tid,"FP-DEL",args)

And FP-DEL's response to the calling WM instance is:

ii. FP to caller WM: SendSpecificMessage(2,tid,error-descriptor,results)

F. FP-ANAL

Not specified at this time.

Error recovery at the MSG interface level is specified in (---).

3. External procedure calling sequences -- arguments & results.

The following subsection specifies the argument lists and result lists for each procedure call mentioned in (2) above.

Frequently Used Arguments

NSWB8 is assumed as the format of all messages containing File Package arguments.

Note on empty arguments:

Whenever an argument is empty, the following will be used rather than the NSWB8 EMPTY data type:

INDEX	INDEX with value 0
CHARSTR	CHARSTR with length 0
LIST	LIST with length 0

LIST(n) and CHARSTR(n) mean a LIST or CHARSTR of length n.

1. Physical-copy-descriptor (pcd):

```
LIST ( host:                INDEX,
       directory:           CHARSTR,
       name:                CHARSTR,
       phys:                CHARSTR,
       ilflag:              BOOLEAN      )
```

2. Global-file-descriptor (gfd):

```
LIST ( class:               INDEX,
       keys:                INDEX,
       dimension:           INDEX,
       key-descriptor:      INDEX,
       tab-descriptor:      LIST,
       il-bytesize          INDEX        )
```

class: INDEX

value = 1 for text
(Implies fill character is blank)
value = 2 for binary
(implies fill character is zero)

keys: INDEX

value = 1 for no keys
value = 2 for keys

2.a. tab-descriptor

```
LIST ( horizontal-tab-descriptor:  LIST ( tab-char:  CHARSTR(1)
                                          stop-specifier:  INDEXI
                                          LIST (INDEX .... ))

       vertical-tab-descriptor:  LIST ( tab-char:  CHARSTR(1)
                                        stop-specifier:  INDEXI
                                        LIST (INDEX ... ))
```

A given descriptor is interpreted as a stop-increment if it contains a single stop-specifier, as a stop-position list if it contains a LIST (even if the list contains but one element)

3. Result-descriptor

File Package results are returned in the standard NSWTP header as specified in the NSWBB and NSWTP speccification. (Except see the description of the results of the delete operation)

4. File-identifier: CHARSTR (fid)

A string formed by the WM, the <file-identifier> as specified in the description of the NSW file name syntax. This is an interim argument, and is used by the FP to create a local file directory containing this identifier and the local file system name of each file imported into the NSW file system.

5. Global-file-type: CHARSTR (gt)

The string which identifies global file type, and is used by the WM as access key to the global file type table.

6. Password: CHARSTR (pswd)

Local file space (directory) password.

File Package calling sequences:

1. WM call on FP to import a file into the NSW file system:

FP-IMP (ext-pcd,ext-pswd,ext-gt,ext-gfd,fid,qdel)

← new-int-pcd

qdel: BOOL

is TRUE iff the file described by ext-pcd, etc, is to be deleted in conjunction with the importation

2. WM call on FP to export a file out of NSW file space:

FP-EXP (LIST(int-pcd),int-gt,int-gfd,
ext-pcd,ext-pswd,LIST(desired-ext-gt),qkeep,fid)

← new-ext-pcd,new-int-pcd, new-ext-gt

LIST(desired-ext-gt)

The first global file type in the list is preferred, with any others being acceptable. If the list has length zero, the new-ext-gt (result) is defaulted to int-gt.

3. WM call on FP to transport a file between external file spaces:

FP-TRANSP(src-pcd,src-pswd,src-gt,src-gfd,
dst-pcd,dst-pswd,dst-gt,dst-gfd)

← (no results, i.e. LIST(0))

4. WM call on FP to delete the physical copies of an NSW file.

FP-DEL(LIST(int-pcd)) ← LIST(rd)

The result list has the following interpretations:

If all the specified physical copies are successfully deleted, then the result will be LIST(0).

Otherwise, it will be a list with the same number of entries as the input list. For each input pcd, the corresponding item in the result list will be LIST(0) if the pcd's file was successfully deleted, and a three element list for each failure:

LIST(errclass, errnumber, errstring)

This is similar to the error descriptor in the NSWTP header.

5. Receiver FP call on donor FP to send a file over the network:

FP-SENDME(src-pcd,src-pswd,src-gt,src-gfd,receiver-host-id,
max-byte-size,max-record-size,family-info)

← connection-identifier,transfer-byte
size,transfer-record-size,file-bit-size

receiver-host-id: INDEX

network (MSG) address of receiver host

max-byte-size: INDEX

maximum byte size acceptable to receiver

max-record-size: INDEX

maximum number of bytes (as above) which receiver can buffer in one transmission

connection-identifier: INDEX

connection identifier to be used in MSG OPENCONN call

transfer-byte-size: INDEX

byte size donor will actually use in transmission

transfer-record-size: INDEX

maximum number of bytes donor will send in one transmission

file-bit-size: INTEGER

donor's estimate of file size, in bits.

family-info: EMPTY for normal cross-family protocol, i.e., transmission of I.L.

non-EMPTY for private family protocol, chosen by each File Package implementor. May be non-EMPTY only for family copy operation.

Chapter 3:

File Package Structure

1. Introduction

This chapter suggests an organization for the File Package, i.e., the modules which implement the functions described in this specification. This chapter is meant as a guide to implementors, and does not ordain a mandatory organization of the File Package.

We begin by giving a graphic representation of the File Package, then describe athe modules shown. These may then be further refined.

2. Top-Level FP structure

A. FLPKG (Top Level Control) ;FLPKG is the MSG generic name
- B. Initializer
- C. Procedure Call Handler
- D. Dispatcher
- E. FP-IMP Processor
- F. FP-EXP Processor
- G. FP-TRANSP Processor
- H. FP-SENDME Processor
- I. FP-DEL Processor
- J. FP-ANAL Processor
- K. NETRECEIVE Processor
- L. MSG Utilities
- M. Local File System Utilities
- N. NSWTP/NSWB8 Utilities
- O. FP data base Utilities

2.A. FLPKG

The File Package main routine simply calls the Initializer, Procedure Call Handler, and Dispatcher serially. Depending on its host's MSG, it then either cycles or executes a StopMe primitive.

2.B. Initializer

Any initialization is done by this routine. It must reset the FP data base if the FP cycles rather than stopping. It is responsible for issuing a WhoAmI to identify the local host.

2.C. Procedure Call Handler

This module uses the MSG utilities to issue the ReceiveGenericMessage which contains the File Package procedure call to be processed, and decodes the message into the FP data base.

2.D. Dispatcher

The Dispatcher passes the procedure argument list to the appropriate function processor.

2.E. FP-IMP Processor

The structure of this processor is:

i. FP-IMP control
ii. Read Arguments
iii. Locate source host
iv. Dispatch on copy type
v. Local copy
vi. Family copy
vi. Non-Family copy
vii. Send results of FP-IMP
viii. Receive WM confirmation
ix. Enter in cross-reference directory & delete source
x. Send WM final confirmation

2.F. FP-EXP Processor

The structure of this processor is:

i. FP-EXP control
ii. Read arguments
iii. Choose source copy
iv. Set up local (result) PCD(s)
v. Dispatch on copy type
vi. Local copy
vii. Family copy
viii. Non-Family copy
ix. Send results to WM

2.G. FP-TRANSP Processor

The structure of this processor is:

i. FP-TRANSP control
ii. Read arguments
iii. Locate source host
iv. Dispatch on copy type
v. Local copy
vi. Family copy

vii. Non-Family copy
vii. Send results to WM

2.H. FP-SENDME Processor

The structure of this processor is

i. FP-SENDME control
ii. Read arguments
iii. Locate & lock source (local) file
iv. Send results message
v. Open data connection
vi. Dispatch on transmission protocol
vii. Family transmission protocol
viii. Non-Family transmission protocol
ix. IL encoder
x. Close connection
xi. Release local file

2.I. FP-DE Processor

The structure of this processor is:

i. FP-DEL control
ii. Read arguments
iii. Loop & delete files
iv. Send results to WM

2.J. FP-ANAL Processor (unspecified)

2.K. NETRECEIVE processor

The structure of this processor is:

i. NETRECEIVE control
ii. Open local (result) file(s)
iii. Issue 'FP-SENDME' to source host
iv. Receive results
v. Open data connection
vi. Dispatch on transmission protocol
vii. Family transmission protocol
viii. Non-family tranmission protocol
ix. IL decoder
x. Close connection
xi. Complete local file(s)

2.L. MSG Utilities

These include:

a. Execute primitive - at lease SendGenericMessage, ReceiveGenericMessage, SendSpecificMessage, Receive SpecificMessage, OpenConn, CloseConn, WhoAmI, Stopme

b. Build Parameter Blocks

c. Create/Read Process name

2.M. Local File System Utilities

These at least include:

a. Verify legality of PCD

b. Connect to directory

c. Open/close File

d. Catalog/enter a file in directory

e. Delete a file

f. Read/write a file

g. Read/write file catalog parameters

h. Create unique file name

i. Receive/Send data over direct connection

2.N. NSWTP/NSWB8 Utilities

These utilities are responsible for at least:

a. Writing from NSWB8 to internal data base form

b. Writing from internal data base form to NSWB8

c. Reading/writing NSWTP headers

2.O. FP data base utilities

NSW component implementors typically use an internal data base for each component to insulate themselves from changes and inconveniences in the NSWB8 encodement. These utilities create/maintain the internal data structures which are analogs

to the NSWB8 forms.

Chapter 4:

An Intermediate Language for File Transfer

1. Introduction

This document describes the grammar used to encode NSW file contents for transfer of binary and text files between NSW TBH's of differing families. The donor FP encodes the file according to the grammar and the file's global file type, and the receiver FP parses the resulting intermediate data and recreates the file as close to its original form as possible.

The complete grammar for file transfer consists of the global file type, corresponding global file type table (GFTT) entry, and Intermediate Language (IL) grammar described here. There are three main requirements for this complete grammar: first, it should be possible to encode files with a minimal loss of structural information; second, it should capture the structural information of the file system of any TBH (and be extendable to new types of TBH as they are brought into NSW); third, it must support a physical transmission protocol compatible with the network I/O system of each TBH family. The fulfillment of the first requirement depends on fulfilling the second. The third requirement could be fulfilled by a transmission protocol independent of the encodement, but there are some practical advantages to including this in the IL, and we have chosen to do so.

We have not yet attempted to capture all the complexity of, say, the OS/360 file system, although the complete grammar is quite rich and easily expanded. The encodement of structural information and data is completely separated, both the ease future expansion and provide more conceptual clarity. All structural information, including byte size, sequence number and format effector descriptors, text/binary, etc., is contained in the GFTT - global file type. The I.L. grammar describes data encodement for the file, and includes productions which specify the network transmission protocol.

In moving text files from paper type-oriented machines to unit record-oriented machines we must inevitably deal with the problem of ASCII format effectors. Part of the function of the global file type table is provide a "dictionary" defining the vertical format effectors and horizontal tab spacing for each global file type (text) in the NSW file catalog. The receiver FP consults this entry on encountering a format effector during the parsing of an encoded file, and tasks the action most appropriate to its operating system and/or the tool which will use the file. For example, horizontal tabs may be expanded into blanks, vertical tabs into blank lines or blank lines with ASA format effectors as first characters, etc. Similarly on transmission from a unit record TBH to a paper tape TBH, leading blanks may be collapsed by the receiver FP into horizontal tabs, etc. The grammar makes no assumptions and imposes no restrictions on how the receiver

FP will store the parsed file; its purpose is to specify as completely as possible how the sending FP stores the file.

The problem of moving text files with ASA format effectors to paper tape-oriented machines is not quite as severe. There are syntactic tokens (see section III) which enable the sending FP to say "begin the next record, skipping two (or three, or more)". Thus a 133-byte print line with a zero as the first byte would be encoded as "begin next record, skipping two" followed by an encoding of 132 bytes of data. The receiving FP could reproduce the effect by inserting the right number of carriage returns.

We assume that the donor FP can always encode a file native to its host operating system if it knows the global file type, and that the encodement will contain only the format effectors, etc., defined by the corresponding GFTT entry.

The transmission of an encoded file between TBH's always uses the byte size specified in the GFTT; the minimum byte size is 8 bits. The receiver FP, of course, must use a storage technique that loses no bits, but is free to store the file in any form it wishes. Transmission is by transmission records; the size of each succeeding record is specified by a transmission record descriptor. which is a production of the I.L. grammar.

In the following section, the productions of the grammar are presented in groups, and the meaning of each group is discussed. Extended BNF is used. When a number is quoted ("255") it represents an 8-bit (i.e., the low-order 8 bits of a byte which is the same size as the width of the connection) control item (see section III). An unquoted number is an 8-bit integer (unless otherwise noted). Where a non-terminal of the grammar is followed by "(p:q)" it signifies the presence of from p to q copies of the non-terminal.

It is not possible to present the transmission protocal and the file encodement grammar together in BNF; so we first give the transmission protocol, indicate how the file data is formed from that, and specify the relationship between the productions of the transmission protocol and the encodement grammar.

2. The File Transmission Grammar.

A. <FILE-TRANSMISSION> ::= <TRANSMISSION-RECORD> (1:n)"251"

An NSW transmission is defined as a sequence of transmission records terminated by an end-of-transmission byte.

B. (a) <TRANSMISSION-RECORD> ::= "240" <NB> <FILE-BYTE> (NB)
(b) <NB> ::= unsigned quantity formed by concatenating two transmission bytes: the number of file data bytes

which follow in the transmission.

(c) <FILE-BYTE> := one byte of file.

The byte size is taken from the GFTT for all the above.

The encoded text file is the concatenation of all file bytes received.

C. <TEXT-FILES> := concatenation <FILE-BYTE> (1:m)

The donor FP may choose to place as many file bytes in a transmission record as desired, except that the receiver's maximum transmission record size may not be exceeded, and I.L. records may not be broken (see below).

D. (a) <TEXT-FILES> ::= <TEXT-FILE>
(a') |<SUBFILES>
(b) <SUBFILES> ::= <SUBFILE>(1:p)
(c) <SUBFILE> ::= <TEXT-FILE>"255"

The file to be transferred may be a single file (Da) or a concatenated sequence of files (Da', Db), each of which is a single file followed by an end-of-file byte (Dc).

E. (a) <TEXT-FILE> ::= <RECORD>(1:q)
(b) <RECORD> ::= <DATA-REC>
(c) <DATA-REC> ::= <REC-CTL><SEQ-NUM>(0:1)<ITEM>(1:s)

A file is a sequence of one or more records (Ea), each of which is a data record (Eb). A data record (Ec) consists of a record control byte (or bytes; see below), an optional sequence GFTT number (which must conform to the descriptor in the), and a sequence of items. (If the GFTT has indicated that sequence numbers of length n are present, the n bytes following the <REC-CTL> item are to be interpreted as the <SEQ-NUM>.) A transmission record may not begin or end within an encodement record.

F. (a) <ITEM> ::= <STRING>
(a') |<REPEAT>
(a'') |<FILL>
(b) <STRING> ::= <STR-LEN><CHAR>(0:r)
(c) <REPEAT> ::= <REP-LEN><CHAR>
(d) <STR-LEN> ::= "0"|"1"|...|"127"
(e) <FILL> ::= "128"|...|"191"
(f) <REP-LEN> ::= "192"|...|"223"

An item is a string (Fa), a repeated character (Fa'), or a fill character indicator (Fa''). The string is a length factor followed by that number of characers (Fb). The repeated character is a factor followed by the character (Fc). The string length explicitly ranges from 0-127. The fill number k is expressed as 128+k, and the repeat factor k as 192+k. (The compression implied by these productions

is optional, e.g., a string of blanks may be passed uncompressed if the implementer so desires.)

G. (a) <REC-CTL> ::= "224"|...|"239"|"249"|"250"
|"252"<N1>|"253"<N2>

(b) <CHAR> ::= any 8-bit pattern

(c) <N1> ::= unsigned 8-bit quantity

(d) <N2> ::= unsigned 16-bit quantity (formed by concatenating *2* 8-bit bytes)

See next section of this chapter for a further description of (Ga).

3. Syntactic Tokens

The following is a table of syntactic tokens which represent record and text control information.

Bit Pattern	Value Range	Meaning
0xxxxxxx	0-127	String of length 0 <= n <= 127
10xxxxxx	128-191	Fill character repeated 0 <= n <= 63 times
110xxxxx	192-223	Repeat next char. 0 <- n<= 31 times
11110001	240	Begin transmission record
11110xxx	241-247	(Reserved)
11111000	248	Begin header
11111010	250	Form feed
11111011	251	End file transfer
11111100	252	Begin new record, skipping 0-255 records
11111101	253	Begin new record, skipping 0-(2**16-1) records
11111110	254	Begin control record
11111111	255	Delimiter of subfiles

The above set of syntactic control items allow for an easy implementation of a generator and a parser for the grammer. Every control items is one byte long and indicates, explicitly or implicitly, the length of the following data item.

Chapter 5:

Translation Semantics

File translation is a major problem which the File Package must address. When the FP is forced to translate a file, it must map a foreign representation of the file onto its "logical equivalent" in the local file system.

Definition of this equivalence seems beyond the current state of the art. One can, however, state some desired properties. Suppose identical line printers are connected to donor and receiver. Then receiver's copy is logically equivalent to the original only if their listings are visually identical. The same applies to decks of punched cards (both binary and text) which may be read or punched. The hope is that equivalence at this level will extend to tools. Indeed, most tools tend to deal with their text input files in units of lines, irrespective of file organization. Solving the listing problem is very close to solving the "line of text" problem which should provide the desired mapping of the output of one tool into the input of another.

What kind of difference in file systems causes this problem? It is that some file systems represent text files as "unit-records" and others as "paper-tapes". Sequential access to lines of text is available at the file system level in a unit-record machine. That is, a low level access method can easily produce the next line of text; that line will consist of only characters and blanks.

The situation is entirely different in a paper-tape machine. A low-level access method is used to read bytes from the file which must be interpreted in order to produce a line of text in the above sense. Besides characters and spaces, "format effectors" are stored in paper-tape files. Indeed the paper-tape file organization is a descendant of the paper-tape controlled typewriter. Format effectors may be grouped into horizontal ones (e.g., carriage return (CR), backspace (BS), and horizontal tab (HT)) and vertical ones (e.g., line feed (LF), form feed (FF), and vertical tab (VT)). Vertical effectors have no horizontal component, and vice-versa. Vertical motion is always down the page.

Those format effectors which cause horizontal motion to the left have no analog in unit record machines and a mapping onto a reasonable equivalent must be found. Of course, if such motion is paired with vertical motion no problem arises; in fact, sender is requested to treat CR-LF and LF-CR pairs as record delimiters in the intermediate language sense. The basic question is this: if the paper tape sequence "b BS /" is to become the two records:

```
(record)          b
(skip 0)          /
```

should the sequence "b BS BS a"

become

```
(record)          b
(skip 0)          a
```

or

```
(record)          ab          ?
```

For purposes of display the single record version is both correct and optimal, but for tool use, it should be left to FP and tool implementors to make the choice.

A brief discussion of each of the format effectors is in order:

HT - never poses a translation problem as it is equivalent to one or more blanks. However, hosts are free to choose their own tab-stop conventions; therefore, we have included a description of them in the file's HEADER record.

BS - has already been discussed.

CR - unless paired with a vertical format effector is equivalent to back-spacing to column 1.

LF - causes the beginning of a new record beginning at current column. The new record is blank to the left of current column.

FF - easily translates to record environments, but with the same column proviso attached to LF.

VT - is similar to HT. Stops are specified in HEADER record.

The following algorithm sketch may be useful. It describes a procedure by which consecutive characters from a paper tape are encoded into a compact record file - compact because "a BS BS b" produces only one record. It is called by the following control program:

```
CHARACTER PT(LTH)   /*PAPER TAPE*/
INTEGER LTH         /*LENGTH OF PT*/
```

'FILE PACKAGE DESIGN SPECIFICATION

```
/*CHARACTER ARRAY FOR BUILDING RECORD-EQUIVALENT OF PT*/

        CHARACTER C(MAXLRECL,LPP,MAXPAGES,MAXOVPRT)

/*PARAMETERS OF C*/

        INTEGER MAXLRECL,       /*MAX LENGTH OF RESULTANT RECORDS*/
                LPP,            /*LINES PER RESULTANT PAGE*/
                MAXPAGES,       /*PAGE CAPACITY OF C*/
                MAXOVPRT,       /*MAX OVERPRINT CHARACTERS
                                  PER CHARACTER POSITION*/
                BOTTOM,         /*BOTTOM MARGIN OF PAGE
                                  IN LINES (BOTTOM < LPP)*/
                HTSTOP,         /*HORIZONTAL TAB STOP,
                                  IN COLUMNS*/
                VTSTOP          /*VERTICAL TAB STOP,
                                  IN LINES*/

/*INDEX VARIABLES FOR ACCESSING THE ELEMENTS OF C*/

        INTEGER COL,            /*NUMBER OF COLUMN WITHIN LINE*/
                LINE,           /*NUMBER OF LINE WITHIN PAGE*/
                PAGE            /*NUMBER OF PAGE WITHIN C*/


/*PRE-BLANK THE ENTIRE CHARACTER ARRAY, C, AND INITIALIZE
  COLUMN, LINE, AND PAGE TO THE UPPER LEFT-HAND
  CORNER OF PAGE 1.*/

        C(*,*,*,*) = ' '

        COL  = 1
        LINE = 1
        PAGE = 1

        FOR I = 1 TO LTH DO
                CALL UREC(PT(I))
```

The procedure UREC will build the character array C whose slices C(*,LINE,PAGE,OVPRT) represent unit records which result from a compacting paper-tape conversion. For illustration, PT can contain TWO-LINES and THREE-LINES (skip to next multiple of 2 or 3 lines) as well as HALF-PAGE and THIRD-PAGE vertical format effectors.

```
        PROCEDURE UREC(CHAR):

        CHARACTER CHAR
```

```
        PROCEDURE FMFEED:
        BEGIN
                LINE = 1
                PAGE = PAGE + 1
        END FMFEED

/*PROCEDURE OF UREC TO ADVANCE TO A VERTICAL TAB STOP
  WHEN STOPS ARE EVERY "FRAME" LINES ON A PAGE OF "LPP" LINES.*/

        PROCEDURE ADVANCE(FRAME):
        BEGIN
                INTEGER NEWLINE, FRAME

                NEWLINE = FRAME*CEIL(LINE/FRAME)+1
                IF NEWLINE > LPP THEN CALL FMFEED
                        ELSE LINE = NEWLINE
        END ADVANCE

/*IS-GRAPHIC MEANS CHARACTER VISIBLY PRINTS*/
    IF IS-GRAPHIC(CHAR) THEN
        BEGIN
                IF COL > MAXLRECL THEN SIGNAL("LRECL EXCEEDED")
                IF C(COL, LINE, PAGE, MXOVPRT) NE ' ' THEN
                     SIGNAL("TOO MANY OVERPRINTS")
                OVPRT = 1
                UNTIL C(POS, LINE, PAGE, OVPRT) EQ ' '
                     DO OVPRT = OVPRT + 1
                C(COL, LINE, PAGE, OVPRT) = CHAR
                COL = COL + 1
        END

        ELSE

        CASE OF CHAR

        SPACE:          COL = COL + 1
        BKSP:           IF COL > 1 THEN COL = COL - 1
        CR:             COL = 1
        HTAB:           COL = HTSTOP*CEIL(COL/HTSTOP) + 1
        LF:             IF LINE > LPP - BOTTOM THEN CALL FMFEED
                            ELSE LINE = LINE + 1
        VTAB:           CALL ADVANCE(VTSTOP)
        HALF-PAGE:      CALL ADVANCE(LPP/2)
        THIRD-PAGE:     CALL ADVANCE(LPP/3)
        TWO-LINES:      CALL ADVANCE(2)
        THREE-LINES:    CALL ADVANCE(3)

        RETURN

        END UREC
```

APPENDIX A.

This appendix to the File Package Specification describes the internal structure of an NSW file catalog entry in the Works Manager data base, and gives the syntax of an NSW file name.

1. NSW FILE CATALOG ENTRY

1.0 Introduction

An NSW file catalog entry contains a file's complete NSW file name, and a list locating all physical copies of the file. The NSW file name contains not only an ordinary name part, but also the file's semantic type, global file type, and system attributes.

1.1 Top level structure

The top level structure of an NSW file catalog entry is simply:

NSW-filename
list (physical-copy-descriptor)

1.2 The NSW file name

The top level structure of the NSW-filename component is:

file-identifier
system-attributes

The file-identifier has three components:

name-part
semantic-type
global-file-type

Two file catalog entries are distinct if their file-identifiers are distinct. Thus the components of the file-identifier are exactly those required for name disambiguation.

1.2.A Name-part

A name-part consists of 1 to 10 ordered name-components. Each name-component is 1 to 12 characters from the alphabet:

AB ... Z
ab ... z
01 ... 9
- (hyphen)
_ (underline)

(S Schaffner)

1.2.B Semantic-type

Examples of semantic-type are FORTRAN-SRC, FORTRAN-REL, COBOL-REL, etc. The semantic-type is specified by tool WARRANT. Semantic-types are generally asserted on output from a creator tool and are of interest to a file user tool. The complete list of semantic-types is maintained separately.

1.2.C Global-file-type

The global-file-type is a key into the global file type table, which specifies the file's physical storage properties needed to do network transfer of the file between storage hosts. The global-file-type is independent of the semantic-type, although many pairs are meaningless. Examples of global-file-type are 10X-TXT, 10X-BIN, 360-BIN, etc.

1.2.D System-attributes

System-attributes record information which may be used in file specification, but which is not needed for name disambiguation. The semaphore has the further property of providing an access lock to the file.

Currently defined system-attributes are:

semaphore:

> The semaphore is either 0 - meaning not set - or it is the pair (project, node-name) indicating the setter of the semaphore.

creator:

> The creator of a file is identified by the pair (project, node-name). Each member of the pair has the same syntax as a name-component (see 1.2.A above).

time-of-creation:

> Time-of-creation is a 14-digit decimal integer consisting of (left to right):
>
> | | |
> |---|---|
> | year | (4) |
> | month | (2) |
> | day | (2) |
> | hour | (2) |
> | minute | (2) |
> | second | (2) |

The time reference is GMT.

last-reader:

Same as creator.

time-of-last-read:

Same as time-of-creation.

last-modifier:

The last modifier of a file is the last person who replaced the file - i.e., did RENAMEGLOBAL, COPYGLOBAL, PUT etc. with qreplace set to T. The distinction between creation and modification is thus that a file is created if an explicit DELETE is done before entering a new filename, and a file is modified if the DELETE is done implicitly by setting qreplace to T. The structure of last-modifier is the same as creator.

time-of-last-modification:

Same as time-of-creation.

1.3 Physical-copy-descriptor

There is an entry in the list of physical-copy-descriptors for each file system copy of an NSW file. Thus, if there is a file copy on each of BBNB, ISIC, and ISID, there will be three physical-copy descriptors in its catalog entry list. File copies exported from the file system into user filespace or tool workspace are not in the file system and have no physical-copy-descriptors.

The top level structure of a physical-copy-descriptor is:

intermediate-language-flag
location-attribute

The location-attribute has four components:

host
directory
name
phys

1.3.A Host

This item contains the network address of the storage host whose file system contains the file copy.

1.3.B Directory

This item specifies the directory containing the file copy if the host's file system supports directories, or is empty.

1.3.C Name

This item contains the complete file copy name in the local file system syntax.

1.4.C Phys

This item may contain volume, unit, or other specifications appropriate to the local file system.

1.4 Public/Private

An NSW user is allowed to see all parts of a catalog entry comprising the NSW-filename. He is not allowed to see the physical-copy-descripters.

2. NSW FILENAME SYNTAX

An NSW file name has the following, where the following are meta syntactic:
[n], [n,m], (,), {,}

```
<NSW-filename>              := <file-identifier> ; <system-attributes>

<file-identifier>           := <name-part>/<semantic-type>;
                               <global-file-type>

<name-part>                 := <name-component> {.<name-component>} [0,n]

<semantic-type              := ST = {FORTRAN-SRC|FORTRAN-REL|...}

<global-file-type>          := GT = <host> - <type>

<host>                      :=  {10X|360|MULTICS|...}

<type>                      := {TXT|SEQ-TXT|BIN|...}

<system-attributes>         := <semaphore>; <creator> ; <last-reader> ;
                               <last-modifier> ; <time-of-creation> ;
                               <time-of-last-read> ;
                               <time-of-last-modification>

<semaphore>                 := SM = {<project-node> |0}

<creator>                   := CR = <project-node>

<last-reader>               := LR = {<project-node>|0}

<last-modifier>             := LM = <project-node>

<time-of-creation>          := TC = <time>

<time-of-last-read>         := TR = {<time>|0}

<time-of-last-modification>     := TM = <time>

<project-node>                  := <project> + <node-name>

<time>                          := <digit> [4] (:<digit> [2]) [5]

<name-component>,
<project>,
<node-name>                     := <identifier>

<identifier>                    := <character> [1,n]

<character>                     := A|B| ...|Z|a|b|...|Z|0|..|9|-|←
                                   (7-bit ASCII)
```

<digit> := 0|1| ...|9| (7-bit ASCII)

2.1 Example

An example of a complete NSW file name is:

```
WALDO.HENRY.REL/ST=FORTRAN-REL;GT=360-SIN;
  SM=IPTO+CARLSON;CR=IPTO+CARLSON;LR=COMPASS+FANEUF;
  LM=IPTO+CARLSON;TC=1976:11:13:06:31:59;TR=1976:11:13:11:19:02;
  TM=1976:11:13:06:31:59
```

2.2 Blanks

An NSW filename may contain no embedded blanks.

3. FILESPECS and ENTRYNAMES:

Any portion of an NSW filename may be specified by user for retrieving a file. Missing name components may be indicated by ellipses - e.g. WALDO...REL. Attributes may be given in any order - see example below.

3.1 Names returned by Works Manager procedures.

Works manager procedures return the file-identifier component of an NSW filename.

APPENDIX B.

This appendix to the File Package Specification describes the Global File Type Table in the Works Manager data base.

1. GLOBAL FILE TYPE TABLE

1.0 Introduction

The global file type table is maintained in the NSW data base. It is a static table with one entry for each global-file-type known to the file system. Each entry in the table contains information needed by a File Package instance to correctly interpret the intermediate language representation of a file native to a different host.

In general, a File Package instance will only need the global-file-type in order to deal with a file native to its own host - in particular, to translate that file into File Package intermediate language. It will require the complete table entry to translate a foreign file from FP intermediate language to a corresponding native form. The Works Manager is responsible for obtaining the entry contents from the global file table and passing them to the File Package (See File Package interface specification).

1.1. Global File Table Entry

The top level structure of a global file table entry is:

structural-attributes
physical-attributes

The structural-attributes component describes the gross structural properties of the file, and the physical-attributes component gives the description of the intermediate language (IL) entities so implied. Taken together, they imply the legal syntactic elements which will appear in the IL representation of the file.

1.2 Structural-attributes

This item has three components:

class
keys
dimension

1.2.A Class

This item currently specifies whether the file is a text or binary file.

1.2.B Keys

This item specifies whether the file has keys or not.

1.2.C Dimension

This item contains the file dimension as an integer, based on a concept due to Charles Muntz of COMPASS and elaborated by Robert Braden of UCLA. The value may currently be between one and four, with the following interpretation:

value 1: Stream data, e.g. a file of ASCII text not blocked into lines. Such a file may not have keys.

value 2: Record/line oriented file, e.g. a TENEX SOS file.

value 3: A text file in line printer format, with a class of format effectors excluding overprints.

value 4: A text file in line printer format including overprint format effectors.

1.3 Physical-attributes

This item has three components:

key-descriptor
TAB-descriptor
IL-bytesize

1.3.A Key-descriptor

This item is an integer specifying the number of bytes in an IL sequence number if the file has keys.

1.3.B TAB-descriptor

A TAB-descriptor has two components:

horizontal-tab-descriptor
list (vertical-tab-descriptor)

Tab-descriptors in each case are identical in form having two components:

control-character
stop-descriptor

where a stop-descriptor is either an integer stop increment, or a list of integer stop positions

1.3.C IL-bytesize

The IL-bytesize is an integer, the number of bits in the IL byte for this file.