[redacted]

LEVEL II

AD A109344

DTIC FILE COPY

093745

Massachusetts
Computer Associates, Inc.

81 12 28 122

12 63

LEVEL II

Final Technical Report

AO 2832

Title of Work:

Provide a National Software Works (NSW) Framework and Machine Connections

February 17, 1976
CADD-7602-1711

Name of Contractor:	Massachusetts Computer Associates, Inc. 26 Princess St., Wakefield, Mass. 01880
Principal Investigator:	Stephen Warshall - 617-245-9540
Project Scientist:	Stephen Warshall - 617-245-9540
Effective Date of Contract:	30 June 1974
Contract Expiration Date:	30 June 1975
Amount of Contract:	$698,548.00
Contract Number:	F08606-74-C-0068
Program Code Number:	4P10

Sponsored by
Advanced Research Projects Agency
ARPA Order Number 2832

TABLE OF CONTENTS

1. A General Introduction

1.1 Purpose

The National Software Works (NSW) is a facility, resident on the Arpanet, intended to support the construction, use, maintenance, modification, verification, and storage of programs and bodies of information on which these programs operate. It is principally aimed at the construction of programs and at providing software tools which can be used in the construction activity.

NSW is intended to facilitate both the administrative and technical aspects of these activities. Thus, it provides mechanisms for the exercise of fiscal and access control in the operation of a programming project, and also access and storage conveniences to programmers for the management of their files.

The salient factor in the conception of NSW is the expectation that the hardware, software, and human resources needed for the execution of a task may be geographically and administratively dispersed, although connected through the network. Tools whose use is to be coordinated may be resident at different computer installations, possibly under the control of different organizations, each with its own rules of operation.

1.2 Design

NSW as an entire system contains large collections of information about its users and the resources belonging to the system; it also contains the programmatic objects whose execution constitutes the operation of the system.

The software animating the NSW is called NSWExec; it is partitioned into independent processes on different processors in the network. These processes have individual names, such as Works Manager (WM), Front End (FE), Shop Foreman (SF), File Package (FP), Works Manager Operator (WMO), etc.

NSWExec appears, operationally, as a state-of-the-art time-sharing monitor. That is, it functions as a keeper and supplier of computational resources, and as a mediator between the user and these resources.

The essential functions of a time-sharing monitor which NSWExec is to provide in the larger network-wide environment are:

Logging in and out -- permitting the user to make, and break, contact with the monitor, and authenticating his right to use its services.

Maintaining a file system, with access protection and provisions for shared use.

Handling I/O with the user's terminal.

Interpreting and honoring the user's requests for resource usage ("Executive Commands").

Setting a specified program into operation ("running a tool") at the user's behest, and linking his terminal to the program in case the tool runs interactively.

Since the users, the resources, and the NSWExec software may all be dispersed throughout the network, creating analogues of these basic functions of a time-sharing monitor in this environment has raised some complex design problems. This overview will be organized as a consideration of these functions in the network environment, with a description of the design solutions adopted in the NSW. The functions we shall discuss are:

Maintaining a physically dispersed but conceptually integrated file system, with adequate access controls.

Managing communication between the separate components of the system.

Catering to the user at his on-line terminal -- connecting him to NSWExec, accepting and interpreting his raw input, protecting him from intervention by non-NSW programs (local host executives, etc.).

Responding to the user's requests for resource use and disposition -- i.e., the normal "Executive Commands", permitting operations on files, inspection of current information about resources and circumstances, invocations of tools, etc

Initiating execution of a tool, and providing a File-System interface so that the tool may obtain the input files it needs from the NSW File System, and deliver the output files it produces into the NSW File System, in a manner compatible with the file-system conventions of the host system where the tool is resident.

2. FILE SYSTEM

2.1 Design Considerations

As does any contemporary Operating System, NSW provides a file system to its users, with naming conventions, protection, access controls, and facilities for entering, deleting, copying, and renaming individual files. However, it had been determined at the beginning of the design that NSW would not "own" any on-line storage device, dedicated to the storage of NSW files.

A principal element of the NSW concept is to both facilitate and constrain file access and file sharing by the members of a programming project, in a manner which will allow the implementation of a wide variety of management policies. To this end, NSW has its own file-naming conventions and mechanisms for verifying access rights which are rather different from those of the hosts' operating systems.

In any case, the user must not be required to have any knowledge of the individual file systems on the hosts; rather, he must be able to use a uniform file-system vocabulary in any reference to his files, regardless of what component of NSW he is communicating with.

2.2 Design Solutions

The files in the NSW File System actually live in the various file systems of Arpanet hosts: on any host which can provide storage for NSW files, NSW "owns" one or more directories (accounts), with the maximum protection available, in which it keeps its files. Hosts providing file storage are called "Storage Hosts". There is also a "principal NSW host" in the Arpanet. This is the host on which the central elements of the NSWExec software are executed -- in the current implementation, a TENEX.

NSWExec contains an information retrieval system, resident on the principal NSW host in the Arpanet. The data base of this information retrieval system does not contain the NSW files themselves, but rather it constitutes the catalogue of the File System. Every file name known to the NSW File System has a record in this catalogue; part of the information in the entry for a particular file gives the location (identity of the host, plus file identification within that host's system) of any existing copies of the file itself. Note that the existence of multiple copies of a file is information which is not normally available to users.

Some of the operations which the user might wish to perform on files can be done merely by making changes in the catalogue, such as deleting a file, renaming a file, or removing a semaphore (access lock) which he had had set on a file; the user, of course, does not directly access the catalogue. But others require operations on the bodies of the files themselves, such as making a copy of a file within the NSW file system, importing a file from outside the NSW system, or exporting a file to a destination outside NSW.

For these operations of making physical copies of files, NSWExec calls upon a "black box" called the File Package, whose job is to understand file transmission across the Arpanet so thoroughly that it can accomodate any likely formats and perform any reasonable conversions necessary to cause a copy of a file at one place in the network to appear at another place.

The portion of NSWExec software which includes the file-catalogue information retrieval system is called the Works Manager (WM). It has a number of other functions, which we shall discuss at the appropriate places in this Overview. The host on which the WM runs -- called above "the principal NSW host" -- is termed the "WM host", in distinction to other hosts participating in NSW, such as a "Front End host", or a "Tool-bearing host".

The two NSWExec components mentioned in this section -- the Works Manager(including especially the file catalogue system) and the File Package -- are clearly new programs which have had to be written for NSW.

3. COMMUNICATION SYSTEM

3.1 Goals

For acceptable operation the network connection between the user's terminal and whatever system he is communicating with should be a fast, character-by-character, full-duplex link. But such links are expensive in Arpanet, in terms of traffic loads and response time, so they should be direct, and used only when a person is at one end of the connection.

Nonetheless, there will need to be frequent communication between the dispersed software components of NSW, and this communication should be as efficient as possible.

Normally, a program on a time-sharing system can be executed only by a logged-in user of that system. Within NSW, the user should only have to log in to NSWExec itself and not to any of the individual Tool-Bearing Hosts (TBHs).

3.2 Decisions

The individual components of NSW software will be configured as independent, coordinate, concurrent processes (even if they happen to reside on the same host).

The standard communication between two NSW processes will be by unitary messages, expressed in one of the PCP (Procedure Call Protocol) message formats (B8 or B36, as appropriate), and dispatched through a message-handler (named MSG), which is itself an independent process on each host participating in NSW.

> A process, then, does not call another as a subordinate, or subroutine. Rather, it sends a message requesting a service, and later receives a message in response.
>
> Two NSW processes may establish a direct networkconnection, if desirable for terminal response or for transmitting large volumes of data. But the initial communication between them, and the agreement to set up the direct connection, are accomplished via MSG.

In the Arpanet implementation, the several MSGs are privileged processes on their hosts, with exclusive rights to reserved network sockets. This permits bypassing the local host's login procedure when executing a process on that host.

MSG is a new program, written for NSW.

4. USER-TERMINAL CATERING

4.1 Situation

The user must be able to get in touch with NSWExec in a reasonably straightforward fashion, preferably without having to log in to any other systems along the way.

As discussed above, the direct connection between the user and NSWExec should be a fast, character-by-character full-duplex link (for most terminals), so that the user will receive rapid and convenient response to his typeins. It should include

Character echoing, perhaps with substitution for the input character, and suppression of echoing for passwords, and

Basic editing facilities, such as the ability to backspace (delete) a character or a word, retype what has been typed for inspection before confirming, or kill what has been typed so that the user can start over.

Any process the user may be connected to is actually running on some Arpanet host, under the host's own operating system. Operating systems generally have some reserved character which, when received from the terminal, causes them to interrupt communication between the terminal and the running process in order to allow the user to communicate directly with the operating system. The NSW user must be protected from the consequences of mistakenly typing such a character.

Conversely, if the NSW user is in communication with some process other than NSWExec, there must be some action he can take to temporarily suspend his communication with that process in order to communicate with NSWExec. Such actions (e.g., typing a special character) must be intercepted before they are transmitted to the connected process.

But the user will, in general, be only indirectly connected to the WM host, so that, if these terminal-catering functions were to be performed at the WM host, the response would be unacceptably slow.

Aside from these user-catering functions, the principal content of the communication between the user and NSWExec will be the the user's calling for the execution of executive commands (see below), and WMExec's displaying the response to these commands on the user's terminal. The amount of information necessary to specify such commands is not large, and the display of NSWExec's responses does not require two-way communication.

4.2 Decision

The user-catering function of NSWExec will be placed in a separate process, called the Front End (FE), to which the user will always be connected, and which will run on a machine as "close" to the user as possible.

In the initial system, the FE process resides on the principal NSW host; a user can achieve connection to this FE by either:

Logging in to this host in the normal fashion, CONNECTing to the appropriate directory, and running the program NSW; (this mode of operation will always be present as a fall-back option)

Executing, from his local host or TIP, an Initial Connection Protocol to a reserved socket on the NSW host.

In an early stage of development, the FE will run on a dedicated minicomputer, connected to the Arpanet either as a local host (through an IMP) or directly as a "smart TIP". The user will be have a broad-band connection to this minicomputer.

However the user's connection to the FE process is achieved, he will immediately be permitted (in fact, required) to LOGIN to NSWExec, identifying himself and giving a password. When the LOGIN is accepted by NSWExec, he will be able to issue any Executive Commands.

The FE process provides echoing on the user terminal, recognition and completion of abbreviated command words (if the user desires), editing functions on the user's typein, and more sophisticated display-control functions.

In the standard mode of operation, when the user is communicating with NSWExec or with "integrated" tools, the user's interactions with the terminal are driven by a "grammar" contained in the FE process, which elicits from the user the information needed to specify the operation he wishes performed. This information is tnen packed into an message and expedited through MSG to the appropriate recipient (Works Manager or tool).

An "integrated" tool is one which in fact handles its communications with the user in the above fashion -- via MSG.

A tool which has not been "integrated" is called an "old" tool. It will be put in contact with the user through a direct TELNET connection between the FE process and the tool process. Thus another feature of the FE is that it can establish and maintain this TELNET connection.

In the standard mode of operation, no danger exists that the user might send to the tool some special character which would place him unwittingly in contact with the tool-host's Operating System. In the TELNET-connection mode, either the FE or the Shop Foreman (see below) must filter out any such characters.

In either mode of operation, a reserved special character will be recognized by the FE process, having the effect of temporarily suspending communication with the tool, and returning communication to NSWExec.

For the duration of his session with NSW, all communication between the user and the system will thus be mediated by the FE, whether the user is conversing with the Works Manager or with one or more tools. This provides a consistent style of interaction with all elements of NSW, except perhaps from some "old" tools which will have their own conventions which the user must obey.

The FE Process is new software, designed and programmed by Charles Irby at SRI/ARC, who also designed and implemented the language for specifying grammars, the compiler for that language, and the interpreter for the compiled grammars.

5. EXECUTIVE COMMANDS

5.1 The component of NSWExec which implements the user's Executive Commands is called the Works Manager (WM). It resides on the principal NSW host, which is therefore called the WM Host.

The formats of the Executive Commands are specified in the Executive Grammar, which is always available to the FE Process. When the user has specified a command to his, and the FE's, satisfaction, it is packaged into a message which amounts to a call on some procedure within the WM; this message is then sent from the FE to the WM via MSG.

5.2 Executive Commands are essentially requests for the use of computing resources (including requests to inspect the status of resources). Hence the WM is fundamentally a purveyor and allocator of resources.

The WM maintains in its data base lists of the rights, privileges, and responsibilities of the users known to the system. When the user logs in, his right to use NSW is authenticated by checking this information. Whenever he requests the use of any resource, his right to use it is verified against these lists.

As mentioned above, the WM maintains in its data base the catalogue of the file system, and uses this to control the existence of copies of files on different hosts in the network. The catalogue, together with the user's rights information, allows the WM to control access to the files in the system.

The WM also maintains information on the tools available within NSW, which enables it to cause a tool process to be created and run on its appropriate host. This information, together with the user's rights information, allows the WM to control access to the tools.

5.3 As part of its resource-managing responsibilities, the WM provides to the (human) managers of programming projects within NSW facilities for admitting new users to the system, and specifying the rights the new user shall enjoy.

These management facilities will in fact be embodied in a separate "Management Tool", access to which will be restricted by the rights of the user seeking to execute it, as with any other tool.

5.4 The types of Works Manager commands are discussed in Chapter 1, "Works Manager Procedures."

6. TOOL INTERFACE

6.1 Design Considerations

In a contemporary interactive computer system, a tool runs under the control of the Operating System on its computer; the Operating System provides to the tool:

Means for communicating with the user's terminal;

Means for using the file system;

Other miscellaneous services more directly associated with the hardware, such as memory allocations, interrupt servicing, date-and-time and elapsed-time information, etc.

In the course of its operation, the tool will interact with the file system in several different ways:

It will need to open for reading (or modification) some already-existing files (input files).

It will need to create scratch files for temporary storage of information during its operation (and perhaps for re-start after a crash).

It may produce new files (or modifications of input files) which are to be delivered to the file system after the tool-run is completed (output files).

In NSW, where the tool, the user, and the "Operating System" (NSWExec) are, in principle, all on different network hosts, several considerations apply:

Communication between the tool and the user is handled through MSG (perhaps plus a TELNET connection), as discussed above; hence tool/user communication must be diverted from the host OS's terminal-handling mechanisms to some other process.

The tool's input files must come from the NSW File System, and not from the host's own file system, hence requests for input files must be diverted from the host OS to the WM. However, once the tool has obtained the file from the NSW File System, it must be able to work within the local file system for operations within the file -- reading or writing at particular locations within the file.

But to use the NSW file system for the storage of scratch files would be grossly inefficient, requiring frequent WM calls, and frequent updating of the WM's File System catalogue; hence, the tool should continue to use the host file system for these.

To keep the tool operation integrated within NSW, the tool's output files must be submitted to the NSW File System for storage. Hence, calls to the host OS to close, or deliver, output files must be intercepted and re-directed to the WM.

The last category of local OS services -- the miscellaneous ones -- must clearly be left intact, since it would be either impossible or expensive to provide them from the WM.

Since it is intended to be an easy task to adapt an existing program to run as an NSW tool, it is obviously desirable to minimize the amount of programming required to do this.

For instance, a tool should not have to include the software necessary to send and receive MSG messages.

6.2 Design

A new program, called the Shop Foreman (SF) must be written to run on each host which will provide tools.

The SF has at its disposal a number of empty file directories (accounts, workspaces) within the local file system, which it will provide to tools running on that host.

When the WM has decided to run a tool on a particular host, it sends a message to the SF on that host, asking it to load and start up an instance of the tool process. The SF will select one of its local directories and assign it for the tool to run "out of" (or "in").

When the tool wants to Open an input file, it has presumably gotten the NSW Filename of the file from the user. It passes this name to the SF, requesting the file. The SF then sends a message to the WM, requesting that a copy of the file be sent to the local directory assigned to the tool. When the copy has arrived in that directory, the SF returns to the tool the local directory name of the copy, which it will have "opened" for the tool in the local file system.

And similarly, when the tool wants to Close an output file, it passes the local directory name of the file, together with the NSW Filename the user has given, to the SF. The SF, in turn, sends a message to the WM, "Delivering" the file to the NSW File System -- that is, requesting the WM to make a copy of the file in one of the NSW's own directories on some host (perhaps the same host, if it is also a NSW Storage Host).

When the tool is ready to stop running, it notifies the SF so that control of the communication link to the user is not returned to the local OS.

Old tools which are to be made to run under NSW will need to be modified only to the following extent:

Points of communication between the tool and the user terminal must be detected and modifiedin one of two ways:

If it seems feasible to structure the communication in message blocks, these messages should be composed for transmission via MSG.

If, however, it is necessary to maintain TELNET-style single-character communication with the user, "system calls" for implementing this must be replaced by accesses to the TELNET connection to the FE.

Each file used by the tool must be identified as an input file, an output file, or a scratch file.

All places where the tool Opens an input file -- i.e., requests by name a (presumably) pre-existing file from the local file system -- must be identified and replaced by calls on the SF.

All places where the tool Closes an output file --i.e., delivers a file name and contents for storage in the local file system -- must be identified and replaced by calls on the SF.

The tool must notify the SF before terminating its execution.

CHAPTER 1

A Summary of Works Manager Procedures

1. Introduction

The Works Manager (WM) is the central software of NSWExec. Its job is to authenticate users' interactions with NSW, to carry out executive commands, and to control access to all NSW resources.

Operationally, the Works Manager is a "server process", which is brought to life when a "WM Call" is made by a Front-End (FE) process or a Shop-Foreman (SF) process: there exists no single Works Manager Process which remains continuously alive while dealing with multiple petitioners, as is the case, for example, with the MSG and FE processes.

From outside, the Works Manager appears as a collection of separately-callable procedures, each performing a specific function. Coordination of the separate procedures and synchronization of separate incarnations of the WM process are effected by jointly-accessed, interlock-protected, data structures. Each WM Call, either from a FE or a SF, is a call on a specific one of these procedures.

> The principal shared data structure is the Catalogue in the NSW Information Retrieval System, which contains all the long-lived data about all elements of NSW. This Catalogue is described (to some extent) in Chapter 2.
>
> Furthermore, whenever NSW is in operation, there are tables of current data, residing in the WM Host, which depict the momentary state of NSW -- e.g., a list of users currently logged in, a list of the tools currently running, etc. Almost every WM Call will result in some change being made to one or more of these tables. It is these "hot" tables which give the appearance of continuity of service by "the" WM on behalf of a user.

The purpose of this chapter is to list these WM Procedures and give a brief description of their effects.

> Since there are a number of items of information which are frequently used as arguments of call, or returned as values, by these procedures, these are discussed in a preliminary section. The full "meaning" of these data items may not be clear, however, until their pattern of use has been shown in the discussion of the procedures themselves.
>
> Some of these procedures can meaningfully be called only from a Front-End process (e.g., LOGIN, LOGOUT), and others only from a Shop-Foreman process (e.g., OPEN, DELIVER); the remainder may be called from either source. It will be indicated for each procedure, explicitly or implicitly, where it may be called from.

This chapter, then, is a programmer's overview of the calls which the WM will accept from other processes. It will be the task of a different writing to re-assort (some of) this same information (and add more) to describe the "Executive Commands" which the user is told he may employ in communicating with WMExec.

2. FREQUENTLY-USED ARGUMENTS

userid: INTEGER

The internal WM identifier of a logged-in user. It is assigned to the user at login by the WM, and is thereafter regularly used in all messages between the FE and the WM, so that each can be sure which user the message refers to.

id = (userid | 0)

WM procedures which can be called from either the FE or a tool require "id" as their first argument. If, in an actual call, the first argument is non-zero, then it is a userid, and the call is from a Front End. If it is zero then the call is from a tool. WM procedures which show "userid" as first argument can only be called from the FE. If any other first argument is shown (except for the procedures LOGIN and REATTACHTONSW which are only FE-callable), then the WM procedure can only be called by a tool.

cost: INTEGER

cost is returned by several WM procedures. It is to be interpreted as the cost in cents of the use of a tool or of an entire session, as appropriate. The user is given the opportunity to gripe about the cost by returning a non-null message when invited to protest.

tool-name: STRING

The name, e.g., NLS, TECO by which a tool is known to the WM. Retrievable under this tool-name in the WM Catalogue is a large block of data called the Interactive Tool Descriptor. This descriptor supplies whatever information the WM needs for successfully running the tool and servicing its file requests. More specifically, it lists the ARPAnet hosts (called "Tool-Bearing Hosts" (TBHs)) and process identifications of potential instances of the tool so the WM can cause an instance of the tool process to be readied for execution (see RUNTOOL); and it lists the file-attributes required for input files and those to be attached to output files (see OPEN and DELIVER).

tooluse-name: STRING

The name by which a particular instance of a given user's active tool is known. This argument is necessary to distinguish between, e.g., different concurrent uses of NLS.

NSW-filename: STRING,

NSW-filenames are discussed in Chapter 2. For the purposes of this chapter, a brief description of the form of the name will suffice. The "NSW-filename" is the full identification of the file in the NSW File System, which could amount to a rather long string of text. However, the user will never have to type in a full filename: instead, he will use either a "filespec" or an "entry-name", depending on the intended use of the file (see these terms below).

A (full) NSW-filename consists of two parts: the name-part, and the attribute-part, separated by a slash (/). The name part is a sequence of name-components, separated by periods (.); the order of the name-parts is significant. The attribute-part is a list of attributes, separated by semicolons (;); the order of the attributes is not significant. A file name might be enclosed in angle-brackets (< , >); if it is, the filename proper might be terminated by a comma (,), allowing additional information, meaningful to tools (though not to NSWExec), to be included in the brackets.

An example of a full NSW-filename might be:

```
    IVTRAN.PHASE1.PARSE.SYMBOL-HASH/
        BCPL-SRC;CR:ILLIAC+BOLDUC;DTC:1975:08:25:16:03:38
```

Name-parts do not necessarily designate unique files. NSW files have attributes and certain of these attributes (those supplied by tools) may be used for disambiguation. Thus it is entirely possible for a user to have a file with name-part A.B and attribute FORTRAN-SRC and another file A.B with attribute 360-FORTRAN-REL. The NSW-file-names of these two files are unambiguous and consist of name-part/ tool-supplied attributes. E.g., A.B/FORTRAN-SOURCE and A.B/360-FORTRAN-REL. The tool-supplied attributes consist of those file attributes which are supplied by tools through WARRANT, DELIVER, CLOSE. The exact form of the attributes is described in Chapter 2.

access-type: COPY | DELETE | ENTER

denotes a particular kind of access to the NSW file system.

filespec: STRING

A filespec is an abbreviated form of an NSW-filename, used in contexts where the name of an existing file is required -- i.e., COPY and DELETE accesses. A filespec need contain only enough parts of the NSW-filename to unambiguously denote the file. As explained below under "scope", an initial segment of the name-part can be automatically supplied, and need not be typed by the user. Any sequence of consecutive name-components which are not necessary for identifying the particular file may be replaced by three periods (...). Also, an attribute-part may be typed in a filespec, to distinguish between two files which differ only in attributes (e.g., the source-language and the relocatable binary forms of the same program).

Thus, the file named by the example under "NSW-filename" above, would be retrieved under the filespec

    IVTRAN...PARSE/BCPL-SRC   .

Any time a filespec is used, if it does not happen to designate a unique file, the WM will send to the Front End for diplay to the user an indexed list of the full filenames of all files which match the filespec; the user may indicate which one he intends by responding with the index number.

> More specifically: If the filespec matches a great many files, the WM Information Retrieval System will protest and refuse to retrieve them; the user will we asked to submit a more reasonable filespec. If the filespec matches few enough files to retrieve, but more than some user-settable limit ("maxlist"), the user will be informed of the number of files matched, and asked if he wants to see the list of names. Only if the number retrieved is less than maxlist will the list be displayed automatically. In any case, the user has the option, in response to any of these messages, of entering a different filespec.

entry-name: STRING

An entry-name is an appreviated form of an NSW-filename used in contexts where a new filename is to be created. As described below under "scope", the contents of the user's ENTER scope will be prefixed to the entry-name as typed. Aside from this scope abbreviation, however, the user must type the entire name-component of the filename -- that is, no ellipses (...) are permitted. No attribute-part is permitted, either, since the user may not assign attributes to files (his identity as creator of the file, and the date-and-time of creation attributes will be appended automatically).

Referring again to the NSW-filename example above, if we assume the user had an ENTER scope of "IVTRAN.PHASE1", the filename shown could have been created (minus the "BCPL-SRC" attribute, which could only have been appended by a tool), using the entry-name

    PARSE.SYMBOL-HASH   .

scope: STRING

Scopes are a method of abbreviating NSW-filenames for user convenience. There are three types of scopes -- COPY, DELETE, and ENTER -- corresponding to the three access-types. A user may have any number of COPY and DELETE scopes active, but only one ENTER scope. Whenever a filespec is typed by the user, that filespec, together with all of either his COPY or DELETE scopes (depending on the context of use) are submitted to the Information Retrieval System. When an entry-name is typed by the user, his ENTER scope is prefixed to the entry-name typed, in order to construct the full name-part of the file to be entered into the Catalogue.

Syntactically, a scope looks like a name-part: a sequence of name-components separated by periods (.).

Scopes are set and changed by the user to make it convenient to reference files in a relatively small region of filename-space, presumably because he expects to do most of his work with those files. If he wishes to access a file outside of his current scope, he may prefix a filespec or entry-name with a dollar-sign ($) (to be read as a barred "S", meaning: DON'T SCOPE), to override the automatic scoping mechanism.

qhelp: BOOLEAN

qhelp is used when a tool calls the WM and doesn't want the WM to directly contact the user at the FE for assistance. In this case qhelp is set to F.

<X>-attcode: INTEGER

An index into the Interactive Tool Descriptor, where a large list of required or known attributes can be referenced without a large amount of net transmission. "<X>" will be either "input" or "output" in the procedure descriptions.

external-name: STRING

Either an ARPAnet pathname with password or a device pathname with password. An external-name is needed for copying files from a source outside of NSW (see IMPORT, TRANSPORT) or copying to a destination outside of NSW (see EXPORT, TRANSPORT). An external-name argument is always accompanied by a password argument, for gaining access to the external directory, device, etc.

3. WORKS MANAGER PROCEDURES

3.1 Connection

LOGIN (project, node-name, password)
--> userid, node-profile, user-profile, system-message, qhave-mail

LOGIN connects a user to the NSW, establishing him as an active user with all the rights implied by the node at which he has logged in. Mistakes (i.e., non-recognition by the WM) in arguments will be handled by HELP returns. The user will then be permitted to retype the incorrect argument or abort and re-start the login.

project: STRING, node-name: STRING, password: STRING

This triple is collected from the user for the initial LOGIN call; it identifies and gives access to a node on the NSW project tree. The user is then considered to be logged-in "at" (or even "as") that node. All rights to access files, use tools, use WM procedures, and spend money are associated with the login node.

node-profile: BIT-STRING, user-profile: BIT-STRING

These are encoded instructions to the FE (and perhaps also to the WM), determining the style of communicating with the user; they include specifications for lengths of heralds and prompts to be displayed, degree of command-word recognition and completion desired, lengths of lists to be displayed, etc. The information in node-profile is peculiar to the node, while the information in user-profile applies to the person "owning" the node in NSW records, regardless of which of his nodes he may log in at (a person may own several nodes -- for example, a project manager will own the top node in his project, but might also set up some subsidiary nodes for his personal work).

system-message: STRING

This would be an important operational message from NSW (or perhaps Project) management, to be displayed to all NSW users at their next login ("system news").

qhave-mail: BOOLEAN

This, if true, will cause the FE to inform the user that there exists new mail addressed either to him personally, or to his present login node; to read his mail, the user should call a Readmail tool.

LOGOUT (userid)
    --> cost

LOGOUT disconnects a user from NSW. Normally, if any interactive tools are still running for this user, he will be asked to terminate those tooluses in the way appropriate for each tool, and re-call LOGOUT. Alternatively, the user may ask the NSW to terminate these tooluses for him (Command: Logout Fast); in this case, output files which the tool has already DELIVERed or CLOSEd will be entered into the NSW File system, and any other files will be lost. If the TBH has gone down while the user was running, the WM will try later to recover and DELIVER any files which the tool had CLOSEd before the crash. Batch tools are, of course, asynchronous with respect to user-NSW connection, and are not affected by logout.

A WM-procedure RELOG was originally planned, which would enable the user to move his login location from one node to another. This has been replaced by a Front-End command MOVELOG which executes successive calls on the WM-procedures LOGOUT and LOGIN.

REATTACHTONSW (project, nodename, password)
    --> userid, node-profile, user-profile, LIST [(tool-info)]

This procedure is intended to allow a user to resume his session in the event that his FE-machine goes down, by re-contacting NSWExec through another FE process. This is not a high priority procedure and will not be available in early versions of the WM. The LIST of "tool-info" in the returned items would contain, for each tooluse the user had initiated, the information necessary for the new FE to set up its tables as if it had been the one the user had been using, presumably: tool-process-ID, tool-name, tooluse-name, (perhaps) tool-grammar.

SCOPE (userid, access-type, scope, qadd)

SCOPE adds (if qadd is T) or deletes (if qadd is F) the scope of access-type. Assistance is obtained by HELP return in the event that there is difficulty (such as the user trying to set his scope to include files to which he does not have authorized access).

3.2 Tool Running

RUNTOOL (userid, tool-name, tooluse-name)
    --> tool-process-ID, tool-grammar

RUNTOOL verifies that the user has access to the tool called tool-name. It creates an instance of the tool process, and establishes a communication path. It returns to the FE a process identification for the FE to use in calling the tool, along with the tool grammar. The tooluse-name argument is provided so that several active instances of the same tool can be distinguished.

tool-grammar: ?

The tool-grammar is an encodement of the Command Meta Language (CML) specification of the commands provided to the user for interacting with the tool. When the FE process is on the WM Host TENEX, what is passed is the local name of the .REL file which embodies this grammar. When the FE process is on a separate machine, the grammar itself will be passed, in some format yet to be specified (perhaps BIT-STRING?).

ENDTOOL (tooluse-name)
--> cost, LIST [NSW-filenames of files with semaphore left set by tool], LIST [NSW-file-names of files DELIVERED]

ENDTOOL is called from the Shop Foreman when a tool indicates it has finished running; this procedure causes the WM to detach the tool from the FE and terminate the tool process. The "return items" shown above are actually sent to the FE (rather than returned to the SF), along with a message instructing the FE to remove this tooluse from its list of active tools and to break its communication link with the tool. All semaphores set during the tool's running are unset unless the Interactive Tool Descriptor indicates that this tool is one which understands use of the semaphore. If so, a list of files with semaphore set is sent to the FE so that the user can either confirm for each file that he wants to leave the semaphore set, or indicate that he wants it unset.

RERUNTOOL (userid, tooluse-name)
--> FE-tool-process-handle, tool-grammar

RERUNTOOL reestablishes the connection between a user and a tool which was running on a TBH which had crashed and has subsequently come back up. This procedure is not at all well defined yet and will not be available in early versions of NSWExec.

3.3 Files, No Movement

DELETE (id, filespec, qhelp)
--> NSW-filename

DELETE verifies that filespec designates a unique file to which the user (identified explicitly by userid, or implicitly if DELETE is called by a tool (first argument 0)) has DELETE access. This access is blocked by a set semaphore. If any assistance is required it is obtained via a HELP return (if qhelp is T or if DELETE were called by a batch tool) or by a direct FE HELP call (otherwise). Once a unique file has been found, its STATUS attribute is set to DEL. It will no longer be accessible to OPEN, COPY, RENAME, EXPORT, etc., but the actual file catalogue entry and file copies are not immediately deleted. The NSW-filename of the deleted file is returned. This return could be a HELP return, requiring confirmation before the actual delete occurs. Alternately, since the file does not immediately disappear, an UNDELETE operation could be supported.

RENAME (id, filespec, entry-name, qhelp)
--> old-NSW-filename, new-NSW-filename

RENAME verifies that filespec designates a unique file to which the user has DELETE access. This access is blocked by a set semaphore. If any assistance is required it is obtained via HELP return or direct FE call as above. RENAME forms a new NSW-filename using entry-name and the tool-supplied attributes of the old file. It verifies ENTER access and unambiguity. As usual assistance is sought should there be any difficulty. The NSW catalogue is then altered to reflect the new name-part and both old and new NSW-filenames are returned.

SETSEMAPHORE (filespec, qhelp)
--> NSW-filename

The WM verifies that the tool can use SETSEMAPHORE, that filespec designates a unique file to which the user has DELETE access, and that the semaphore is not already set. Assistance is obtained via HELP return or direct FE call as above. If all is well, the semaphore is set and the NSW-filename is returned.

UNSETSEMAPHORE (id, filespec, qhelp)
--> NSW-filename

The WM verifies that filespec designates a unique file to which the user has DELETE access. Assistance is obtained as usual. If all is well, the semaphore is unset and the NSW-filename returned.

WARRANT (attcode, NSW-filename)
--> new-NSW-filename

WARRANT adds the attributes referenced in the Interactive Tool Descriptor by attcode to the file whose current name is NSW-filename. Since tool-supplied attributes are part of NSW-filenames, the new NSW-filename is returned.

DISPLAY (userid, access-type, filespec, (mask|%))
--> (LIST [NSW-filenames] | number of files)

DISPLAY either lists selected portions of file catalogue entries for some set of files or else reports the cardinality of the set. The elements of the mask will refer to the parts of the file catalogue entry described in Chapter 2, but details of the encodement have not yet been specified.

CLOSE (output-attcode, local-filename, entry-name, qhelp)
--> NSW-file-name

CLOSE performs all the functions of DELIVER below except that: (1) no NSW file copy is made; and (2) an entry is made in the active tool use entry of the calling tool. The intent of CLOSE is to protect the user from TBH crashes. Should the TBH go down and then come back up again, then, if either the user is not logged in or chooses not to RERUNTOOL, the WM will make NSW file copies of the CLOSEd files. Thus the user's CLOSEd files will be in a well-defined consistent state when he again chooses to OPEN them. If a file has been CLOSEd with a given output-attcode and entry-name, and, later during the running of the tool, another file is CLOSEd or DELIVEREd with the identical output-attcode and entry-name, the WM assumes that it is the user's intent to replace the previously CLOSEd file. If a file has been CLOSEd, it is not necessary to DELIVER it. ENDTOOL will DELIVER all CLOSEd files that have not previously been explicitly DELIVEREd.

3.4 Files, Movement

COPY (id, filespec, entry-name, qhelp)
--> src-NSW-filename, dst-NSW-filename

COPY verifies appropriate accesses: COPY access for the source file, ENTER access for the destination file, plus DELETE access for the destination file if the copying would "overwrite" an existing file. It creates a new NSW catalogue entry and a new copy of the source file.

EXPORT (id, filespec, external-name, password, qhelp)
--> src-NSW-filename

EXPORT verifies appropriate access and sends a copy of the source file to the location designated by external-name.

IMPORT (id, external-name, password, entry-name, qhelp)
    --> dst-NSW-filename

IMPORT is the inverse of EXPORT.

TRANSPORT (id, src-external-name, password, dst-external-name, password)

TRANSPORT is an extended FTP for NSW users.

OPEN (input-attcode, filespec, qset, qhelp)
    --> NSW-filename, local-filename

OPEN is used by a tool to obtain a copy of an NSW file. The WM verifies that there is a unique file designated by filespec to which the user has COPY access and which has the attributes implied by input-attcode. Assistance is obtained as usual. Should the user also have DELETE access to the file, then WM will set the semaphore on the file if either the Interactive Tool Descriptor indicates that it should be set, or if qset is TRUE. If the semaphore is already set (and the user has DELETE access rights), then this tool's access to this file is blocked unless the user, in response to a message to the FE, indicates that he is willing to use a copy of the filed version, even though someone else may be planning to replace it soon. In any event, if the semaphore is set, the user is informed that it has been set, and by whom. The WM makes a copy of the file into the workspace used by the tool, performing whatever conversions are necessary and possible. The NSW-filename of the copied file and the local filename of the new copy are returned.

DELIVER (output-attcode, local-filename, entry-name, qhelp)
    --> NSW-filename

DELIVER is used by a tool to insert a file into the NSW file system. ENTER access and unambiguity are verified with assistance sought as usual. An entry is made in the NSW file catalogue and an NSW-owned copy is made of the file designated by local-filename. The attributes implied by output-attcode are appended to the file. The original file is left in the tool's workspace. The NSW-filename of the new entry is returned.

GETFILE (code)
    --> local-filename

GETFILE is used by certain tools, e.g., READMAIL, which require access to the NSW file system on behalf of a user, but to files to which the user himself does not have direct access. For example, there will be a canonical way, referenced by code in READMAIL's Interactive Tool Descriptor, of constructing an NSW filename of a mailbox in which someone has put mail for the user of READMAIL. Details are still unspecified.

code: INTEGER

An index into the Interactive Tool Descriptor, where an algorithm for constructing an NSW-filename and other pertinent information can be found.

PUTFILE (code, local-filename)

PUTFILE is the inverse of GETFILE. PUTFILE would be used by SENDMAIL. Both would be used by NLS (conceivably) for storing a user's NLS-profile.

READDEVICE (local-filename, device-code)

READDEVICE is used by tools which provide means for users to input via local tape, card reader, paper tape reader without making NSW files. The WM would figure out from the userid which FE the user was at and therefore what the external-name of the appropriate device is. After that, this procedure is just like TRANSFER.

device-code: STRING

crd = card reader
pun = card punch
ptr = paper tape reader
ptp = paper tape punch
mt7 = 7 track mag tape
mt9 = 9 track mag tape
dta = DEC tape

WRITEDEVICE (local-filename, device-code)

WRITEDEVICE is the inverse of READDEVICE.

CHAPTER 2

Catalogue Entries and File Names

1. Introduction

Crucial to the development of the NSW was the design of a nomenclature for NSW files. There were a number of conflicting goals which impacted on this effort. Most important, of course, was the necessity to have names which were unambiguous throughout the NSW system. This full file name should also contain information relating to the ownership, history and use of the file. On the other hand, the users of NSW within a project are not concerned with the possibility that one of their file names might conflict with one at a remote site. Furthermore, they will not be willing to specify full file names to avoid such conflicts, and this will translate into dissatisfaction with, and reduced usage of, NSW. A third criterion was imposed by the environment of NSW: since each NSW file in fact resides within some (and possibly more than one) host, its NSW file name must in some way be compatable with the naming conventions of the host's operating system. Finally, it was necessary to distinguish between various copies of the file: for any given file there may be copies at a number of places within the system, but the existence and location of these copies should be transparant to the user.

These issues are resolved by raising the level of the design from the file name to the catalogue entry. The catalogue entry contains full information about a file and all of its physical copies. It contains a fully unambiguous specification of the file: what we would think of as the file name, a list of attributes which are available to the user for examination and further specification of the file, and a list which is unavailable to the user which defines each of the physical copies of the file. In retrieving a file the user need give no more information than is necessary; in most cases this information will consist of components of the name, with missing components indicated by ellipses. When necessary, or if desired for other readers of a program, a command, or a teletype script, attributes can be appended to the specification.

The following sections describe the components of the NSW catalogue entry, in terms of both function and formal syntax.

2. NSW Catalogue Entry

An NSW catalogue entry has the following form:

name-part
use-type
creator
last-reader
semaphore
time-of-creation
LIST [local-copy-entry]

2.1 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)

2.2 Use-type

The use-type is a small integer which corresponds to a logical property of the file. The currently defined types (and the corresponding property) are:

0	null (no property)
1	COBOL-SRC
2	COBOL-LST
3	COBOL-LDM
4	TEXT
5	BINARY

2.3 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 2.1 above).

2.4 Last-reader

Same as 2.3.

2.5 Semaphore

The semaphore is set by a user (or tool) who is writing into the file, to warn other potential users that the file may be undergoing change. The semaphore is either 0 - meaning not set - or it is the pair (project, nodemame) indicating the setter of the semaphore.

2.6 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.

2.7 Local-copy-entry

There is an item in the list of local-copy-entries for each file system copy of the NSW file. That is, if one copy of a file is stored on BBNB and another on ISIC then there are two local-copy-entries. Copies made of a file for use by a tool are not file system copies.

Each local-copy-entry is a pair (structure, path-name) where structure is a (host dependent) list defining the structure of the copy of the file. As we gain experience with the possible range of file structures, we expect to be able to encode structure more succintly. For the present, the following structures are defined:

B4700:

use-type	LRS*	BLK	NRECS	FMT
COBOL-SRC	80	5	var	A
COBOL-LST	140	5	var	F
COBOL-LDM	100	1	var	B
TEXT	var	var	var	A
BINARY	var	var	var	B

TENEX:

use-type	BYTE-SIZE	NBYTES	FMT
TEXT	7	var	F
BINARY	8	var	B

where use-type (defined in 2.2 above) is included in the table for information only (it is not recorded as part of the structure).

---

* LRS is logical record size in characters
BLK is the number of records per block
NRECS is the number of records in the file
FMT is the format control character
var indicates that the value of a parameter is a variable of the structure (i.e., dependent on the particular file which the structure describes).
BYTE-SIZE is the size in bits
NBYTES is the number of bytes in the file

These structures are adequate for moving files between TENEX and B4700 so that COBOL source, text data, and listing files can be edited on TENEX, COBOL source, listing, data, and load module files can be stored on TENEX, and COBOL source files can be compiled on B4700. Installation of additional tools will require expansion of the list of file structures.

Path-name is the list (host, directory, local-file-name, password). The syntax of the components of this list is host-dependent.

2.8 Ambiguity

Two file catalogue entries are distinct if the name-parts and use-types are not identical.

2.9 Public/Private

A user of NSW is allowed to see any part of a file catalogue entry except the list of local-copy-entries.

3. NSW filename

3.1 Syntax

For the initial system, an NSW filename has the following syntax: ([n], [n,m], (,), {, } have their usual meta-syntactic functions.)

```
<NSW-filename>        ::=  <name-part> / <attributes>

<name-part>           ::=  <name-component>
                           {. <name-component> } [0,9]

<attributes>          ::=  <use-type>; <creator>; <last-reader>;
                           <semaphore>; <time-of-creation>

<use-type>            ::=  UT : ( null | COBOL-SRC | COBOL-LST |
                           COBOL-LDM | TEXT | BINARY )

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

<last-reader>         ::=  LR : ( null | <project> + <node-name> )

<semaphore>           ::=  SM : ( null | <project> + <node-name> )

<time-of-creation>    ::=  TC : <digit> [4]  ( : <digit> [2] ) [5]

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

<identifier>          ::=  <character> [1,12]

<character>           ::=  A|B|...|Z|a|b|...|z|0|...|9|-|_
```

3.2 Ambiguity

Two NSW filenames are distinct if their name-parts and use-types are not identical.

3.3 Retrieval

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

4. Full NSW filename syntax

```
<Filename>  ::=      %NSW  <Nmptail>
                   | <Identifier>  <Nmptail>
                   | <Nmptail>
<Nmptail>  ::=       ...  <Identifier>  <Nmpnd>
                   | ...  <Nmpnd>
                   | .  <Identifier>  <Nmpnd>
                   | <Nmpnd>
<Nmpnd>  ::=         /  <Attpt>
                   | <Ender>
                   | <Nmptail>
<Attpt>  ::=         <Ender>
                   | <Attribute>  <Attail>
<Attail>  ::=        <Ender>
                   | ;  <Identifier>  <Attail>
<Attribute>  ::=  [ COMMENT:  Any of the <Attributes> defined in 3.1
                above, plus others to be defined subsequently.]
<Ender>  ::=    [ COMMENT:  Any separator other than "...", ".",
                "/", ";"; or failure to recognize an  <Identifier> ]
```

CHAPTER 3

Black Boxes Called by the Works Manager

1. Introduction

1.1 What Black Boxes Are

The Works Manager requires that certain functions involving other hosts be performed. From the point of view of the Works Manager, the manner of performance of these functions is unimportant -- only their I/O map matters. That is, the Works Manager requires black boxes, each of which, given a certain input set, produces a desired output; the internal functioning of each black box does not matter. It is the purpose of this chapter to describe the desired I/O map for each of what currently seems an appropriate set of black boxes.

1.2 Meta-description

All the black boxes are described here as if there were an asynchronous process which could be called to perform these functions. Thus, every argument is explicitly given, even though it is possible to have an implementation in which some arguments would be implicit because of the place of call. Nevertheless, we have given all arguments in order not to influence those who will be charged with implementing these black boxes.

2. Common Arguments and Data Structures

2.1 Introduction

While different black boxes require different arguments, some of which are simple integers or strings and some of which are more complex data structures, it is the case that even for explicit specification of all possibly necessary arguments for each black box, only a small number of argument types are needed. The rest of this section contains the glossary of these argument types; taken together with the functional descriptions of the black boxes given in section 3 it provides as complete a specification of the black boxes as is now possible. While section 3 shows the argument list for each black box, the functional descriptions are essentially independent of the detailed specifications of the arguments given here.

2.2 Detailed Descriptions

For each argument we show (1) its name as used in section 3, (2) its data type, and (3) its function.

account designator - LIST[user-name, account, accounting-password]

Account designator provides the information necessary to authorize the use of a black box. User-name is a STRING and designates an authorized black box user. (In our case it will always be NSW, but we are leaving open the possibility of other users.) Account is an INTEGER; accounting-password is a STRING. They provide further and finer protection of the black boxes.

host - INTEGER

Host designates a node on the ARPAnet. It appears in section 3 variously as host, src-host, dst-host, TBH (tool-bearing host), or FE (front-end).

workspace - STRING

A workspace is what is commonly called a directory, but since the latter term is also used to refer to the catalogue of items in the workspace, we have chosen to use the terms workspace and catalogue for these two different objects.

filename - STRING

Filename names a file in a workspace. (Strictly, it refers to an entry in a catalogue which references a file in a workspace.) It appears variously as filename, src-filename, dst-filename.

file-password - STRING

We assume that, in general, files are protected by passwords. This protection may be implemented at any of the three levels of file specification - i.e., host, workspace, filename. Thus, we separate file-password from file specification when we designate a file.

cec - LIST[BOOLEAN, INTEGER, ?]

Cec is an abbreviation of condition/error code. It is always returned by a black box. It is divided into two parts. The first part is standard for all black boxes. It consists of a notification of success or failure, and the charges (in cents) incurred by execution of the black box (as distinct from charges which might be incurred because of tool execution begun by the black box).

The second part is black box dependent. It provides information as to the causes of any failure. Examples of the kind of possible failures are:

Account designator invalid

source/destination host invalid/dead

source/destination host net connection dead

source/destination workspace not found/invalid

source/destination password invalid

source filename invalid

net transmission error

source filename not found

source file too big

structure cannot be used on destination host

net/host failure

Note that these are examples of the kinds of failures which would cause error messages. We expect that the implementers will find it both necessary and possible to provide a larger vocabulary of error messages each of which will give at least as much information as to the cause of the error as do these examples. Precise definition of error messages will presumably be part of the effort of implementing the black boxes.

3. Black Boxes -- Functional Descriptions

3.1 Introduction

The functional descriptions of the black boxes required by the Works Manager are grouped by the type of activity which they mediate. These are:

File manipulation within a TBH

File manipulation between TBHs

Determining the status of a host or of the net

Manipulation of interactive tools.

3.2 File Manipulation Within a TBH

```
1) LOCALCOPY(account-designator, host, src-workspace,
             src-filename, src-file-password, dst-workspace,
             {dst-filename}, dst-file-password)

        -> cec
           {dst-filename}
```

LOCALCOPY copies a file within a host's file system. There is no transmission across the net. The source file is not disturbed. LOCALCOPY returns a generated dst-filename if one is not supplied as an argument of call.

```
2) DELETE(account-designator, host, workspace,
          ( {filename} [1, N] | ALL ), file-password)

        -> cec
           ( {filename} [0, M] | ALL )
```

DELETE deletes a file or set of files from a single workspace in a host's file system. It may be used to delete a single named file, a list of named files, or, if ALL is the argument, every file from a given workspace. It returns the names of all files deleted (or ALL if the entire contents of the workspace were deleted).

3) LOCALMOVE(account-designator, host, src-workspace, src-filename,
src-file-password, dst-workspace, {dst-filename},
dst-file-password)

-> cec
{dst-filename}

LOCALMOVE moves a file within a host's file system without copying. It is logically equivalent to a LOCALCOPY followed by a DELETE, but it is intended to be implemented by changing the host's file system catalogue (i.e., renaming). If dst-filename is not supplied, or if it is ambiguous in dst-workspace, then a generated filename is returned. Any such ambiguity is reported in cec.

4) MOVEWORKSPACE(account-designator, host, src-workspace,
src-file-password, dst-workspace, dst-file-password)

-> cec
{ <src-filename, dst-filename> } [0, N]

MOVEWORKSPACE is equivalent to a LOCALMOVE (with dst-filename the same as src-filename) for every file in src-workspace. If there is any ambiguity, then a new filename is generated for each src-filename which is ambiguous in dst-workspace. A list of pairs <src-filename, (generated) dst-filename> is returned, and the ambiguity is reported in cec.

5) CATALOGUE(account-designator, host, workspace, file-password)

-> cec
{filename} [0, N]

CATALOGUE obtains a listing of all the files in workspace.

3.3 File Manipulation Between TBHs

1) NETCOPY(account-designator, src-host, src-workspace, src-filename, src-file-password, src-structure, dst-host,
dst-workspace, {dst-filename}, dst-file-password,
dst-structure)

-> cec
{dst-filename}

NETCOPY causes the file identified by src-host, ... to be transmitted across the net and stored as dst-host, ... The source file is not disturbed. The success or failure of transmission is reported in cec. If a dst-filename is not supplied, or if it is ambiguous in dst-workspace, then a generated dst-filename is returned. Ambiguity is reported in cec.

Src-structure and dst-structure are presently undefined. They are intended to consist of a logical and a physical component The logical structure of a file will be a constant of the file, recorded in the NSW catalogue. Each copy of a file will have an associated physical structure which may vary for different copies - e.g., a file may be a B4700 COBOL load module (logical structure) and may exist in two copies, one on a TENEX in one format and another on a B4700 in a different format.

3.4 Status

1) HOSTPROBE(account-designator, host)

```
-> cec
   state-vector
```

HOSTPROBE queries the state of a host. The exact format of state-vector requires further determination. Some expected components are:

ALIVE/DEAD HOST

ALIVE/DEAD NETWORK CONNECTION

number of active users

file space used/available

length of batch queue

scheduled downtime

2) NETPROBE(account-designator)

```
-> cec
   state-vector
```

NETPROBE queries the state of the net. The exact format of state-vector requires further determination.

3.5 Interactive Tools

1) BEGINTOOL(account-designator, FE, userid, TBH, password, account, workspace, file-password, tool, startup-data)

-> cec
tooluse-id

BEGINTOOL causes the user identified by userid at the FE to be connected to the TBH, logged in under password and account, attached to the workspace, and finally causes tool to be started with startup-data provided. Note that this is a logical description of BEGINTOOL; the WM does not require (or even expect) that the sequence of actions which a user currently performs will be literally mimicked.

Userid is the INTEGER associated with a user by the WM at login. Account is probably a STRING but it could conceivably be more structured. Startup-data is a BITSTRING. Tooluse-id is an INTEGER and is used as an argument in other black boxes.

2) ENDTOOL(account-designator, tooluse-id)

-> cec
cost

ENDTOOL stops the use of a tool and severs the link which was established by a previous BEGINTOOL. The cost (in cents) of the tooluse is returned.

3) SUSPENDTOOL(account-designator, tooluse-id)

-> cec

SUSPENDTOOL ends the use of a tool and severs the link which was establshed by a previous BEGINTOOL. However, sufficient information is retained, indexed by tooluse-id, so that the link can be reestablished by RESUMETOOL.

4) RESUMETOOL(account-designator, tocluse-id)

-> cec

RESUMETOOL reestablishes the link broken by a previous SUSPENDTOOL.

5) TOOLPROBE(accounting-designator, tooluse-id)

-> cec
state-vector
cost

TOOLPROBE queries the state of a running tool. The details of state-vector require formulation. The currently accumulated cost in cents is also returned.

CHAPTER 4

Communication with the PDP11 / B4700

1. OVERVIEW OF NSW BATCH FACILITIES

1.1 NSW Batch Processes

Batch processes within NSW are those which run (for some period of time) without communication with the user. The batch processes easily divide into two classes: batch tools and batch jobs. The batch tool bears strong resemblance to the interactive tool: it is a reliable, production program with clearly defined inputs and results. On the other hand, the batch job offers the NSW user the ability to run a fairly arbitrary job on the batch machine.

1.2 Batch Tools

Batch tools are essentially the same as interactive tools with the important difference that the user is not attached to the tool while it is running, but is free to do other work.

When a user requests that a batch tool be run, all of the information required by the tool, which might normally be determined at various points during execution if the tool were running interactively, must be specified before execution can begin. NSW provides an Interactive Batch Specifier (IBS) to mediate a conversation between the user and the Works Manager for the purpose of fixing the information needed by the tool. There is a Batch Tool Descriptor for each batch tool which specifies the information needed by the tool, and the conversation mediated by IBS will determine

- Files to be processed
- Disposition of results

The user's right of access to the desired input and output tools is verified, and the output files are marked (by setting the semaphore -- see Chapter 2, section 2.5) as "being generated". The IBS then refers to a skeleton text in the Batch Tool Descriptor to generate the necessary JCL for running the tool, and this JCL is then submitted for execution (see below). When submission has been effected, IBS returns to the user, who can either submit another batch process or terminate IBS and return to WM command level.

4.3 Batch Jobs

Most tools are sufficiently general to run under control of JCL substantially different from the skeleton contained in the Batch Tool Descriptor. We use the term "batch job" to refer to the running of such a tool, or of a collection of tools, under control of a JCL file provided by the user. In this situation the IBS mediates a conversation between the user and the WM in which the rights of access to the files and tools specified in the user's JCL are verified. As before, output files are marked as "being generated". Note that the user-supplied JCL file may contain NSW filenames.

So far as possible, the JCL is checked for validity by verifying the following:

- Correct JCL -- verified for syntactic correctness
- Permitted accesses -- to both files and tools
- Sufficient funds -- so that the job (with time limit if necessary) will not overrun the user's account.

While the JCL file may be marked, once verified, to indicate the validity of the JCL syntax, both accesses and funds are dynamic and must be rechecked each time the JCL file is submitted.

The validated JCL is then submitted for execution. As before, the IBS returns to the user for either a new batch process or termination.

4.4 Job Execution

Once tool or job specification is complete, execution is initiated. Execution of batch processes is supervised by a batch operator component of the WM. When the user inquires about status, he is really conversing with the WM operator (WMO).

The WMO supervises transfer of files to the batch site, transfers a file of JCL to that site, and then requests that the JCL be executed. Ideally, the WMO could receive the log of the process execution (as produced by a B4700 SPO or TENEX typescript, etc.) so that the user could "connect" to the SPO to "watch" the process run, or type it periodically.

Once the batch machine completes the process, it signals the WMO that the results are available. The WMO is then free to dispose of the results according to the process specification. If an error was detected during the batch process and the result disposition is unclear, the results may be held in suspension pending a subsequent conversation with the user. The final step in the batch process is that the WMO signals the user that the process is complete - with an online or offline message depending on whether or not the user is logged in when the process completes.

What kind of visibility does the process have when it is under control of the WMO? That is if the user were to type < inquire > , how much could he be told? We envision several status messages:

Awaiting transmission

In transmission

Awaiting execution

In execution

Awaiting receipt

Complete

Awaiting disposition

"In execution" requires additional detail. Hopefully it would also report running time as of the probe. Optionally, a < watch > facility might be desirable: the process log would be typed on the user's terminal as received from the batch machine.

2. TENEX IP DESIGN

2.1 Introduction

IP was originally designed by ADR and SAI as the Interface Protocol between the B4700 and the PDP11 at Gunter. As such, it describes a variety of messages to be exchanged between so-called 11IP (running under ELF) and 4700IP running as a never-ending program under the B4700 operating system, MCPV.

The PDP11 at Gunter has been configured with 128K of core, with the intent of providing a comprehensive Arpanet Interface with the B4700, in the form of PCP, FP, etc. However, utilization of more than 28K on the 11 requires VM (Virtual Memory) ELF, which will not be available in the near term. So the first implementation of the B4700/Arpanet connection will have to be based on use of 28K in the 11. As (Real Memory) ELF requires 24K for ELF, NCP and its buffers, TELNET, and one terminal, the B4700 interface code is confined to 4K. Since the device driver for the 50KB line to the B4700 requires 2K, almost all processing of IP messages is being moved to TENEX until VM ELF becomes available.

In this limited configuration, the 11 is essentially a transparent interface between the Arpanet and the 50KB link from the 11 to the B4700. Communication is thus between the B4700 and TENEX. It should be emphasized that this design is for the special purpose of initial debugging of the B4700/Arpanet (and therefore NSW) interface. It does serve the needs of a batch TBH from NSW's standpoint, and therefore will serve as an interesting model.

2.2 User/Server relationship

B4700 is strictly a server. It listens for "request" messages, or under unusual circumstances, sends out an unsolicited message. When sent a message, it produces exactly one "response" message. There is one exception to this one-request/one-response rule: no explicit acknowledge message is sent over the net for each unit of file transfer.

TENEX is strictly a user, communicating only with B4700 IP. It sends request messages, and expects to receive for each exactly one response, with unsolicited messages which may be received any time a normal response message is expected.

2.3 Unsolicited Messages (interrupts)

In certain situations, the B4700 wishes to signal TENEX of change in status. Currently, two such events are defined:

- Job done
- B4700 operator requests service termination

Of course, status could be polled; rather, at any time the B4700 can send an unsolicited message to TENEX. Such messages arrive with a zero "handle" (a field in the header portion of a message) and are immediately recognizable by TENEX. These messages become interrupts to WMO processes supervising the B4700 operations.

2.4 Message Format

All IP messages (both request- and response-type) have the same format: an 8-byte header followed by a variable length portion. The length of the variable portion is always in the last two bytes of the header in units of 8-bit bytes, and is between 0 and 200, inclusive (the limit of 200 may be relaxed later). Thus, an IP message works as follows:

```
+----------------+        +---------------->>>>-----+
|                |        |                         |
|     header     |        |    variable portion     |
|                |        |                         |
+----------------+        +---------------->>>>-----+
```

The format of the header is as follows:

```
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |               |               |               |
|subsys |  fn   |    handle     |   modifier    |    length     |
|       |       |               |               |               |
+-------+-------+-------+-------+-------+-------+-------+-------+
```

where:

subsys is a character (all characters (in IP) are 8-bit ASCII) as follows:

A - File Transfer System (FTS)
B - Catalog System (CAT)
C - Work Order Executive (WOE)
Z - System messages;

fn and modifier are characters (3 in all) used to request particular functions of the subsystem;

handle associates messages with processes and is a two-byte integer;

length was described above and is a two-byte integer.

(In IP a "byte" is an arbitrary bit pattern, while "character" will describe a limited ASCII character set which is quite digestible by "unit record" machines like the B4700: it specifically excludes the format effectors CR, LF, TAB, FF, EOL, etc. Rather, (optional) format of ASCII records is specified as an ASA format code in column 1 of each record (see the discussions on file transfers for more detail)).

2.5 Handles

The 50KB line between the B4700 and the PDP11 is viewed by them as a single pair of physical channels. 4700IP consists of several asynchronous processes, however. In the original design, the PDP11 assigned a unique (mod 2**16) handle to each request it sent to the 4700 so that the 4700 could process several requests concurrently. The B4700 was expected to return the request handle in the handle field of the corresponding response, so that the PDP11 could establish the correspondence.

With the decision to temporarily move the user end of IP to TENEX, TENEX will now assign handles to IP messages as they are sent out over the Net. As before, the 4700 will echo the handle in its response (but remember that unsolicited messages are sent with zero handles). Each message is assigned a handle equal to one greater than the previous message. The handle of the IP message following an NSW cold start is 1, as is the handle of the message following one with a handle of 2**16-1.

3. FILE TRANSFER IN IP

Part 1 - Introduction and Catalog Subsystem

3.1 Introduction

Every file transfer in IP is done in two stages:

1) Negotiation with the 4700 catalog
2) Transfer of file contents

In the 4700 file system, every file on disk has a unique name in a single catalog. Along with the name, logical record size (lrs), records per block (rpb), number of records in file (nrecs), and allocation information (areas on the 4700) are recorded. In order to satisfy NSW file transfer requirements, the WM must be cognizant of lrs, rpb, and nrecs for every file which it imports to or exports from the 4700. Negotiation with the catalog, therefore, consists of reading or setting those parameters in preparation for transfer of file contents. Once the catalog information is established, the file contents may be transferred by a transfer of each logical record.

These tasks are split between the CAT and FTS subsystems as follows:

CAT never reads file contents but is solely concerned with reading and writing the catalog.

FTS sends and receives file contents, never changing the catalog, except for changing the record count as required by file transfer to the 4700.

3.1.1 CAT Messages

Currently, four basic functions are supported by CAT:

0. Read Name - tests the catalog to see if a particular file is cataloged, and if so, returns its catalog entry.

1. Enter Name - (where the name need not be completely specified) enters a new name in the catalog with lrs and rpb supplied by NSW, allocated on the basis of a supplied record count (nrecs) but is initially empty.

2. Purge File - deletes (and expunges) a name and its associated file contents.

3. Rename - changes only the name of a catalog entry to a new name. The new name must not be in use.

3.2 Message Description Technique

The format of IP messages was described earlier. For the purposes of detailing the format of described messages, we will use the notation

CAT(fn,modifier,variable part)
FTS(fn,modifier,variable part)
WOE(fn,modifier,variable part)
SYS(fn,modifier,variable part)

where

fn is a character,
modifier is two characters,
variable part is a BNF metavariable, described later.

Unsolicited messages are indicated by an asterisk appended to the IP subsystem name; e.g. SYS*(3,00,) is an unsolicited message requesting service termination. All other messages have handles assigned as described earlier.

3.2.1 CAT Messages

Read Name = CAT(0,00,<file name>)
requests catalog information on the specified file name. Normal response is the complete <file spec> for the requested file. Returns:

File Present = CAT(0,00,<file spec>) if found
Nonexistent = CAT(0,82,<file name>) if not
4700 I/O err = CAT(0,88,<file name>) if error

Enter Name = CAT(1,00,<partial file spec>)
establishes a new name in the 4700 catalog. The file spec name portion may optionally contain "wild card" characters ('?') in which case the 4700 is free to substitue any character in order to create a unique name. The first character in a name may not be a wild card. Normal response is actual name used. Returns one of the following:

Succcessful Catalog = CAT(1,00,<file name>)
Catalog Full = CAT(1,84,<partial file spec>)
Name not unique = CAT(1,86,<partial file spec>) if unique name could not be created.
4700 I/O Error = CAT(1,88,<partial file spec>)

Purge File = CAT(2,00,<file name>)
deletes and expunges catalog entry and file contents of an existing 4700 file. Normal response is positive acknowledgment. Returns one of the following:

```
Successful Purge = CAT(2,00,<file name>)
File not found   = CAT(2,82,<file name>)
4700 I/O Error   = CAT(2,88,<file name>)
```

Rename File = CAT(3,00,<file name pair>)
renames the (existing) file specified in the first name to be the (non-existent) second name. Normal response is positive acknowledgment. Returns one of the following:

```
Successful Rename   = CAT(3,00,<file name pair>)
Old Name Not Found  = CAT(3,82,<file name pair>)
New Name Not Unique= CAT(3,86,<file name pair>)
4700 I/O Error      = CAT(3,88,<file name pair>)
```

3.3 BNF

The following BNF defines the metavariables used by the catalog system definition.

```
<digit>           ::= 0|1|...|8|9
<number>          ::= <digit>|<digit><number>
<letter>          ::= A|B|...|Y|Z|a|b|...|y|z
<alphanumeric>    ::= <letter>|<digit>
<xalphanum>       ::= <alphanumeric>|?
<file name>       ::= <user name>|<print label>
<user name>(1)    ::= <letter>|<user name><alphanumeric>
<partial
  file name>(1) ::= <letter>|<RJE#>|<partial file name><xalphanum>
<print label>(1)::= <RJE#>|<print label><alphanumeric>
<file name pair>::= <file name>;<file name>
<file spec>       ::= <file name>;<file map>
<partial
  file spec>      ::= <partial file name>;<file map>
<file map>        ::= <lrsc>,<rpb>,<nrecs>
<lrsc>(2)         ::= <number>
<rpb>(3)          ::= <number>
<nrecs>(4)        ::= <number>
<RJE#>(5)         ::= 6
```

Notes

(1) Length limited to 6 characters.

(2) Logical record size in characters must be doubled by 4700IP to produce the lrs in digits(!) as required by MCPV. Even, and less than 200 (for now).

(3) Records per block must be such that lrsc*rpb is less than 1000, because of core limitations within the 4700.

(4) Number of records has no real limit. Space is allocated so that nrecs will fill 10 areas on the 4700 disk. Thus, the file can double in size before access (with more than 20 areas) becomes inefficient. Noa (maximum number of areas) for new files will be set to 20 by 4700IP to keep NSW users reasonable.

(5) This needs confirmation by GAFB.

Part 2 - Transfer of File Contents

3.4 Introduction

IP deals only with sequential byte files. Every file on the 4700 is of this general class, so that

1) Any file on the 4700 can be copied by IP with enough information so that it could be restored - as in an archival system.

2) Sequential byte files (of reasonable byte size) can be shared among 4700 and TENEX tools.

The sequential byte files are broken into record groups of explicit length of three different types:

A) ASCII - in which the records are sequences of IP characters (no format effectors);

F) Formatted ASCII - in which the records are sequences of IP characters, the first of which is always an ASA format specifier;

B) Binary - in which the records are sequences of 8-bit bytes, with all 256 bit patterns permitted.

3.5 File Transfer Sequence

Transfer of logical records within IP is an exception to the one request/one-response rule, as receipt of data does not cause the receiver to return an IP message, but to simply ask his NCP for the next message(s). Actual sequence is as follows, assuming required CAT operations have been performed:

TENEX -> 4700

TENEX requests "start send"	
	4700 responds "ok to send" or "error"
TENEX sends records	4700 stores records
.	.
.	.
.	.
TENEX sends "eof"	
	4700 closes file and acknowledges transfer complete
TENEX receives confirmation, completing the transfer.	

Note that once the transfer has begun, TENEX will not expect to hear from the 4700 until after eof. I/O errors or service interrupts will change this, however. I/O errors at TENEX are handled by sending an "I/O error" message to the 4700 instead of the next record or the eof. On the other hand, TENEX has to listen for possible I/O while it is sending records, which can be done by testing the channel for message waiting, for example.

4700 -> TENEX

TENEX	4700
TENEX requests "start get"	
	4700 responds "starting get"
TENEX receives acknowledgement	
TENEX receives records	4700 sends records
.	.
.	.
.	.
	4700 sends "eof" and terminates the transfer
TENEX receives the "eof" and terminates the transfer.	

Errors on 4700 will cause TENEX to receive an error message instead of a record, but the 4700 must listen for possible error messages from TENEX while it is sending records.

3.6 FTS Messages

All file transfers are done with FTS IP messages, as follows:

Start Send = FTS(0,01,<xfr spec>)
initiates an IP send operation. <xfr spec> contains only 4700 file name and format mode. Normal response is the assignment of a "transfer number" within 4700IP (up to five file-content transfers may be in progress simulataneously. The referenced file name must exist, and have an assigned lrsc, rpb, and an area allocation appropriate to the number of records in the new file. Any old file contents associated with the name are deleted. Response is one of the following:

Ok to Send	= FTS(<xfr no>,00,<xfr spec>)
No Room	= FTS(0,81,<xfr spec>) if the maximum number of transfers are in progress
File not Found	= FTS(0,82,<xfr spec>)

If the transmission mode is F(Formatted ASCII) lrsc must be between 1 and 133 and rpb must be 5.

Send Record = FTS(<xfr no>,00,<record>)
sends the next record of the specified file transfer. Normally expects no response. Instead, checks the incoming net connection for a possible error message, and finding none, proceeds to send the next record of the file or a eof message. May get responses of

Transmission Error= FTS(<xfr no>,83,)
4700 Write Error = FTS(<xfr no>,84,)

The 4700 is expected to store the incoming <record>s as follows, under control of the format established during "start send" as follows:

A(ASCII): <record> contains not more than lrsc IP characters. The buffer set up within the 4700 for the next record of the file is initialized to lrsc EBCDIC blanks. The characters of <record> are converted to EBCDIC and written into the leading positions of the buffer.

F(Formatted ASCII): <record> contains not more than lrsc IP characters. The buffer set up within the 4700 for the next record of the file is initialized to be a MCPV PBD record with 132 EBCDIC blanks and formatting based on the incoming <record>'s column 1. The remaining characters of the record are converted to EBCDIC and filled into the print-line positions of the buffer. The algorithm for doing this needs specification by SAI. (See Burrough's Medium System TN on "Device Alternates" for the format of PBD (Printer Back-up Disk files).

B(Binary): Each <record> contains exactly lrsc bytes. The bytes are copied into the buffer for the lnext record without translation.

Normal EOF = FTS(<xfr no>,03,)
indicates that TENEX has reached EOF on the specified <xfr no>. Normal response is "transfer complete". The 4700 closes the newly transferred file and frees the <xfr no>. Response is one of the following:

Transfer Complete = FTS(<xfr no>,05,)
Transmission Error= FTS(<xfr no>,83,)
4700 Write Error = FTS(<xfr no>,84,)

Abnormal EOF = FTS(<xfr no>,04,)
indicates that TENEX wishes to abort the file transfer. Normal response is "transfer deleted". The 4700 scratches the (partially) transferred file and frees the <xfr no>. Always responds with

Transfer Deleted = FTS(<xfr no>,06,)

Start Get = FTS(0,02,<xfr spec>)
intiates an IP get operation - to "get" a file back from 4700 to TENEX. <xfr spec> contains only 4700 file name and format mode. Normal response is in fact a series of responses: an "ok to get" acknowledgement, followed directly by a number of data transfers, followed by normal eof followed finally by "transfer complete". During this sequence, TENEX stores away the records (contained in the data transfer messages) and sends no messages to the 4700 unless it wishes to abort the transfer.

As with the send operation, a transfer number is assigned and returned in the "ok to get" response. The referenced file must exist. If format mode is F, then the file must also have lrsc=140 and rpb=5.

The first response is always one of the following:

Ok to Get = FTS(<xfr no>,00,<xfr spec>)
No Room = FTS(0,81,<xfr spec>) as in send
File not Found = FTS(0,82,<xfr spec>)

Following an "ok to get", several (one for each record in the file) data transfer responses are made by the 4700:

Gotten Record = FTS(<xfr no>,00,<record>)

The 4700 is expected to format the outgoing <record>s under control of the format established during "start get" as follows:

A(ASCII): Trailing blanks may be truncated. The (remaining) characters of the next 4700 record are converted to ASCII to create the variable portion of the IP data transfer message.

F(Formatted ASCII): Trailing blanks may be truncated. The (remaining) characters and formatting information in the next 4700 record are converted into one or more ASA format effector records and converted to ASCII. As in send, the conversion algorithm needs specification by SAI.

B(Binary): <record> contains exactly lrsc bytes as taken from the next 4700 record.

When 4700 reaches normal eof, it will send

```
Normal EOF = FTS(<xfr no>,03,0)
```

to complete the get sequence. Abnormal events can be indicated at any time in this sequence by sending an error message instead of the next message in the normal sequence. They are:

```
4700 Read Error = FTS(<xfr no>,85,)
Abnormal EOF    = FTS(<xfr no>,04,)
```

TENEX I/O errors will cause "Abnormal Eof" to be sent to 4700.

3.6.1 BNF

(Refer also to BNF in description of CAT)

```
<xfr no>          ::= 1|2|3|4|5
<xfr spec>        ::= <file name>/<fmt spec>
<fmt spec>        ::= A|F|B
<record>(6)       ::= <char record>|<byte record>
<char record>     ::= <IP char>|<char record><IP char>
<byte record>     ::= <byte>|<byte record><byte>
<byte>            ::= any 8-bit code
<IP char>(6a)     ::= <alphanumeric>|<blank>|<punctuator>
<blank>           ::= a blank
<punctuator>      ::= !|"|#|$|%|&|´|(|)|*|+|,|-|.|/
                          |:|;|<|=|>|?|@|[|\|]|^|_|`|{|||}|~
```

Notes

(6) Length is recorded in header.
(6a) All characters with ASCII (octal) codes #040 through #176.

3.7 BATCH JOB EXECUTION UNDER IP

Running batch jobs under IP consists of several stages:

1) Transmitting input and command files
2) Submitting the command file to MCPV for execution
3) Retrieving result files

Stages 1) and 3) are accomplished through CAT and FTS; stage 2) is performed via WOE (Work Order Executive). Two functions are currently supported:

1. Submit Deck
2. Query Status

Submit Deck = WOE(1,00,<file name>)
causes the referenced file to be converted to an MCPV pseudo-reader and the resulting reader scheduled for execution. Normal response is "submit successful" and an assigned <job id>. The referenced file must have lrsc=80 and rpb=5. Responds with one of the following:

Submit Successful	=	WOE(1,00,<job id>)
File Not Found	=	WOE(1,82,<file name>)
Job Refused (by 4700IP)	=	WOE(1,88,<file name>)
4700 I/O Error	=	WOE(1,88,<file name>)

If submission was successful, WOE polls the MCPV "mix table" so that when the job is finished it can send an unsolicted message back to TENEX:

Job Done = WOE*(4,00,<job summary>)

The summary details time and charges for the job. On completion of the job, 4700IP must locate the @<number> form of the listing file name and change it to the <print label> located in the label record of the listing file. The "job done" message is not sent until this has been successfully accomplished.

Delete Job = WOE(5,00,<job id>)
causes the indicated job to be deleted from 4700IP's tables. This request is sent in "response" to the unsolicited Job Done message. The job is not deleted by the Job Done sequence as TENEX might be down. Rather, TENEX sends this request after all output has been retrieved.
Response is one of the following:

```
Job Deleted     = WOE(5,06,<job id>)
Job Not Found   = WOE(5,00,<job id>)
```

Query Status = WOE(3,00,<job id>)
is used to query the status of the referenced job.

Current status definitions are the following:

```
00 - not found
01 - running
02 - stopped
03 - scheduled
04 - waiting loading
05 - finished
```

The actual response is always

```
WOE(3,<job status>,<job id>)
```

where <job status> is one of the above.

3.7.1 BNF

```
<job id>(9)       ::= <number>
<job status>      ::= 00|01|02|03|04|05
<job summary>     ::= <job id>;<cpu time>,<charges>
<cpu time>(7)     ::= <number>
<charges>(8)      ::= <number>
```

Notes

(7) Units unknown
(8) Units unknown
(9) Maximum unknown

3.8 IP SYSTEM MESSAGES

System messages are defined for the purpose of exchanging system status information. They contain no information relating to any particular file or any particular job. The following are currently defined:

Echo = SYS(0,00,<record>)
requests the B4700 to respond by simply returning the request to TENEX unaltered. No side effects are produced by this message. Expected response is

Echo = SYS(0,00,<record>)

Invalid Message = SYS(1,00,)
as a "request" (sent by TENEX) means that a response or unsolicited message arrived at TENEX garbled. Further processing in this event is currently undefined, but during debugging it will be used to trap internal errors.

If the PDP11 or the B4700 should receive an illegal request it will respond instead with the unsolicited message

Invalid Message = SYS*(1,00,)

with similar effect.

Should an error persist on the 50KB line between the PDP11 and the B4700, the unsolicited message

Communications Error = SYS*(2,00,)

will be sent to TENEX. TENEX will periodically probe the B4700, and when service resumes, a restart sequence will be initiated (currently undefined).

When the B4700 operator requests service termination, the unsolicited message

Service Termination = SYS*(3,00,)

is sent to TENEX. TENEX will not inititate any file transfer operations or start any jobs until service is resumed.