# Inter-Office Memorandum

|  |  |  |  |
|---|---|---|---|
| To | Communication Protocols | Date | June 5, 1979 |
| From | Ed Taft | Location | PARC/CSL |
| Subject | Spruce protocols | File | [Maxc1]<Pup>SpruceProtocols.bravo |

XEROX

Attributes: technical, Distributed computing

Abstract: This memo documents the protocols used to communicate with printing servers running the Spruce program.

Press files are sent to Spruce servers by means of the Pup EFTP protocol, documented in [Maxc1]<Pup>EFTPSpec.press. For the printing application, the EFTP protocol has been extended slightly so as to permit Spruce to notify the sender of certain exceptional conditions. Additionally, Spruce has an additional server that will give out status information.

This document supercedes an earlier memo entitled Ears Protocols,'' October 17, 1976.

## Press file transfer

The EFTP receiver socket is the standard one, 20 (octal). The spruce server supports only one transfer at a time, and a request to begin a new transfer while one is already in progress will elicit a *receiver busy abort* without disturbing the ongoing transfer.

If a transfer is in progress but the server has not received any EFTP Data packets for some time, a request to begin another transfer will be honored and the old transfer forgotten, with no message to the original sender. (In this case, if the original sender later does attempt to resume the timed-out transfer, the first new EFTP Data packet will elicit an *out of sync abort*.)

The sender may abort a transfer in the middle by issuing an *external sender abort* and destroying the connection.

### *Abort codes*

The following abort codes may be generated by the server. They are an extension to the abort code set defined in the EFTP protocol, and they have precise interpretations that may permit a sender to initiate appropriate recovery activity. All aborts also contain a human-readable explanation that should be reported to the human user wherever possible.

*Out of sync abort* (4) defined in the EFTP protocol
: The server received a packet that was not the next packet in sequence for an ongoing transfer. This should occur only in response to other than the first packet of a transfer. It is most likely an indication that the server either crashed and restarted or timed out a transfer. The sender should immediately restart the file transfer.

*Receiver busy abort* (3) defined in the EFTP protocol
: The server is in the midst of receiving a file from some other sender. This abort should occur only in response to the first EFTP Data packet of a transfer. The sender may try

again after a short timeout (e.g., a few seconds), with every expectation of being able to transmit the file in the immediate future.

*Long wait abort* (6)
The server is not presently accepting new files for printing. This abort should occur only in response to the first EFTP Data packet of a transfer. The sender may try again later, but may have to wait an arbitrarily long time. Spruce generates this response if it has been told to stop spooling,’’ e.g., because the printer is down for maintenance.

*Medium wait abort* (7)
The server cannot accept a new file for printing at this time, but expects to be able to shortly. This abort may occur at any time during a transfer. The sender may try again after waiting a modest amount of time (e.g., a minute). Spruce generates this abort if its spooling queue fills up and it has to print for a while in order to make room for new files.

*File reject abort* (2) defined as *external receiver abort* in the EFTP protocol
The server rejects the file that is being transferred; i.e., the *contents* of the file are unacceptable to the server. This abort may occur at any time during a transfer. The sender should *not* attempt to retransmit the file. (Spruce will also generate this response if the file is too big for the server to print.)

*Suspend request* (8)
The server cannot continue accepting the current file due to a transient condition, but it is *not* aborting the transfer. The sender should attempt to resume transmission (starting with the EFTP Data packet that elicited the *suspend request*) after a short wait (a few seconds). Of course, if the sender gets tired of waiting, it may abort the transmission by means of an *external sender abort*. (Spruce may also generate this in response to the first EFTP Data packet of a transfer if it is not prepared to begin the transfer immediately but expects to be able to within a few seconds.)

Remember that *all* aborts, with the exception of the *suspend request*, cause the current transfer to be aborted if one is in progress. Except in the case of the *file reject abort*, the sender must restart transmission from the beginning after waiting an appropriate length of time.

## Status server

An additional server, running on socket 21 (octal), answers requests for status. A Pup of type 200 (octal) directed to this socket will yield a response Pup of type 201. The first content word of this Pup is a status code with the following interpretation:

1. Not spooling (EFTP receiver not running)
2. Spooler is idle (no EFTP transfer in progress)
3. Spooler is busy (EFTP transfer in progress)

The remainder of the Pup is a human-readable text string describing the printer’s status. If the server is not spooling, this text usually contains an explanation entered by whoever stopped the spooler. It should be reported to the human user wherever possible.

Spruce will answer status requests at any time (assuming it is up), regardless of whether or not an EFTP transfer or other activity is in progress. It should be noted that the Spruce status server runs even while Spruce is in its scan-conversion and printing phases, when it is not prepared to initiate EFTP transfers. Spruce will interrupt these phases after a short delay to service a new transfer request.

Consequently, a sender should be tolerant of a fairly long delay (up to 30 seconds) in the server’s response to the first EFTP Data packet. The sender should *not* conclude that the server is down’’ unless it receives no response from the status server.