For Xerox Internal Use Only -- November 17, 1977

Pup File Transfer Protocol Package

This package is a collection of modules implementing the Pup File Transfer Protocol. The package is used by the FTP subsystem and the Interim File System and runs on Altos and Novas.

1. Overview

This document is organized as a general overview followed by descriptions of each of the modules in the package. A history of revisions to the package is included at the end.

Before beginning the main documentation, some general comments are in order.

a. The File Transfer Protocol is (alas) complex; this package requires the Pup package and all of its supporting packages plus some other packages not specific to Pup. This documentation is less tutorial than normal Alto package descriptions so you should be prepared to consult its author.

b. This document describes the external program interfaces for a particular implementation of the File Transfer Protocol, and does not deal with the internal implementation nor the reasons for design choices in the protocol or this implementation. Before considering the details of this package, you should read <Pup>FtpSpec.ears to get the flavor of how the File Transfer Protocol works. The <Pup> directory also contains descriptions of the lower level protocols on which FTP is based. Detailed knowledge of these protocols is not necessary to use this package, but you must be familiar with the operation of the Pup package.

c. This package and the protocol are under active development so users should expect modifications and extensions.

d. This package is designed to run on both Altos and Novas, under several operating systems and with several file systems. Functions are carefuly split into protocol-specific and environment-specific modules. This package provides the protocol modules; you must write the matching environment-specific modules.

1.1. Organization

The FTP package comes in four modules: Server, User, Utilities, and Property lists. The utility and property list modules are shared by the User and Server.

The User and Server modules implement their respective halves of the protocol exchanges.

The Property List module generates and parses property lists, filesystem-independent descriptions of files. When passed between User
↑L

and Server FTPs through the network byte stream, their form is defined by protocol as a parenthesized list. When passed between these protocol modules and the user-supplied modules in a program, they take the form of a data structure defined by this package.

The Utility module contains protocol routines shared by the User and Server modules and some efficient routines for transferring data between a network stream and a disk stream.

1.2. File Conventions

The FTP package is distributed as file FTPPackage.dm, and contains the following files:

User	
FtpUserProt.br	User protocol common to file and mail
FtpUserProtFile.br	User file commands
FtpUserProtMail.br	User mail commands
Server	
FtpServProtFile.br	Server file commands
FtpServProtMail.br	Server mail commands
Property lists	
FtpPListProt.br	Property list protocol
FtpPList1.br	Implements a 'standard' property list
FtpPListInit.br	Initialization
Utility	
FtpUtilB.br	Common protocol
FtpUtilXfer.br	Unformatted data transfer
FtpUtilDmpLd.br	Dump/Load data transfer
FtpUtilA.br	Assembly-language utility code
FtpUtilInit.br	Initialization
Definitions	
FtpProt.decl	Protocol parameters and structures
Command files	
CompileFtpPackage.cm	Compiles all files
DumpFtpPackage.cm	A list of all binary files
FtpPackage.cm	A list of all source files

All of these modules are swappable, and are broken up into pieces no larger than 1024 words. Modules whose names end in "init" are initialization code which should be executed once and thrown away.

The source files are kept with the subsystem sources in FTP.dm and are formatted for printing in a small fixed-pitch font such as Gacha8 (use the command 'Gears/s @FtpPackage.cm@').

1.3. Other Packages

FTP is a level 3 Pup protocol, and this package uses a number of other Alto software packages. As always, files whose names end in "init" may be discarded after initialization (except ContextInit.br).

For Xerox Internal Use Only -- November 17, 1977

```
Pup Package
    PupBSPBlock.br      PupBSPStreams.br    PupBSPProt.br  PupBSPa.br
    PupRTP.br           PupDummyGate.br     PupRoute.br
    Pup1b.br            PupAl1a.br          Pup1Init.br
    PupAlEthb.br        PupAlEtha.br        PupAlEthInit.br
Context Package
    Context.br          ContextInit.br
Interrupt Package
    Interrupt.br        InterruptInit.br
Queue Package
    AltoQueue.br
Timer Package
    AltoTimer.br
Time Package
    CTime.br
ByteBLT Package
    AltoByteBLT.br
CmdScan Package
    Keyword.br          KeywordInit.br
Strings Package
    StringUtil.br
Template Package
    Template.br
```

1.4. Principal Data Structures

The following data structures are of interest to users, and together with the procedures described later, constitute the package interface.

FPL	File Property List, is this implementation's internal representation of the protocol-specified property list. An FPL structure will be referred to as a 'pList' from here on.
FTPI	File Transfer Package Interface, contains pointers to the network byte stream, user disk stream, log stream, the file buffer, and various flags.
FTPSFI	FTP Server File Interface, is a vector of user-supplied procedures constituting the interface between the protocol and environment-specific modules in a file Server.
FTPSMI	FTP Server Mail Interface, same as an FTPSFI except for a mail server.
FtpCtx	FTP Context, is the process-global storage for a User or Server FTP process. It consists of an FTPI, and if the process is a Server, an FTPSFI or FTPSMI. This is a convenient place for the user-supplied modules to keep process-private data. You can do this by adding items to the FtpCtx definition and then recompiling everything.

The entire FtpCtx need not be filled in all of the time. For each group of procedures, the items they require will be specified. A general description of the contents of the FTPI part of an FtpCtx is in order here.

```
bspSoc              a pointer to a BSP socket open to a remote FTP
                    process.
↑L
```

bspStream	a pointer to the stream in the above BSP socket. Pup package experts will recognize that this is redundant, but it is often convenient and makes the code clearer.
dspStream	a pointer to a stream to which this package will output generally useful information, including copious amounts of debugging information if debugFlag is true. The only operation that need be defined is 'Puts'.
diskStream	a pointer to a disk stream. It should always be opened in byte mode.
buffer	a pointer to a block of memory which can be used for block transfer I/O operations. The bigger this is the faster things will go.
bufferLength	the length in words of the above buffer
debugFlag	a boolean. If true, the protocol exchanges for this context are output to dspStream as text, along with some other useful information. Use this! It will save you much head-scratching.
connFlag	a boolean. This should be true if bspSoc is open. The package will cooperate in maintaining this flag, which is valuble when finishing.
serverFlag	a boolean. This flag is tested by procedures in the shared modules to determine whether the caller is a User or Server.
savedBSPErrors	the default BSP error procedure is saved here. This package handles certain errors itself.

1.5. Programming Conventions

This package can be used with the Bcpl Overlay package. File FtpOEPInit.br contains a procedure which will help do this, but you should consult with the author.

This package does a lot of string manipulation, and uses the following conventions:

a. All strings are allocated from 'sysZone'.

b. Strings are represented in data structures (such as property lists) as addresses. Zero means no string is present.

All of the procedures in this package expect to execute in contexts (in the sense of the Context package), and expect CtxRunning (defined by the Context package) to point to an appropriately filled in FtpCtx.
↑L

For Xerox Internal Use Only -- November 17, 1977

1.6. Timeouts

If a Get or Put operation times out, the bspStream Get and Put routines are changed so that all subsequent operations fail immediately. This will cause the current command to fail quickly, so that its caller can take appropriate action. This package makes timeouts look the same as if the stream closed, and treats them as unretryable. Two timeouts are used by the package and kept in statics.

getCmdTimeout
: This timeout is used in situations involving human user interaction and should be fairly long. Its default value is defGetCmdTimeout, defined in FtpProt.decl.

getPutTimeout
: This timeout is used when transferring data and should be fairly short. Its default value is defGetPutTimeout, defined in FtpProt.decl.

2. Server

The FTP Server module consists of two files: FtpServProtFile.br, a file server, and FtpServProtMail.br, a mail server. The internal organization of both files is the same; they just implement different sets of commands. Each file has one external procedure:

FtpServProtFile() or FtpServProtMail()
: which carry out protocol commands received over bspStream by calling the user-supplied procedures in FTPSFI or FTPSMI. When the BSP connection is closed by the remote FTP User process, these procedures return.

This module uses the following fields in FtpCtx: dspStream, bspStream, bspSoc, and FTPSFI or FTPSMI. All of the primary command slots (Version, Store, Retrieve, StoreMail, etc.) must contain procedures. If you do not wish to implement a command, it suffices to point the command's slot at the following procedure:

```
and NYI(nil) = valof
   [
   FTPM(markNo,1,"Unimplemented Command")
   resultis false
   ]
```

in which case any subsidiary procedures for that command (such as StoreFile and StoreCleanup for the Store command) need not be filled in. FTPM is described in more detail below. For the remainder of this section, 'FtpServProt' refers to 'FtpServProtFile' or 'FtpServProtMail'.

2.1. Version Command

By convention, Version is the first command exchanged over a newly opened FTP connection. The User sends its protocol version number and
↑L

For Xerox Internal Use Only -- November 17, 1977

a string such as "Maxc Pup Ftp User 1.04 19-Mar-77". When FtpServProt receives this command, it replys with its protocol version number and then calls

```
     (CtxRunning>>FtpCtx.Version)()
```

which should generate some herald text:

```
     Wss(CtxRunning>>FtpCtx.bspStream, "Alto Pup FTP Server ")
```

to which FtpServProt will append a string of the form "1.13 14-May-77".

2.2. Retrieve Command

When the remote FTP User process sends the command 'Retrieve' and a property list describing the files it wants to retrieve, FtpServProt parses the property list and calls

```
     (CtxRunning>>FtpCtx.Retrieve)(remotePList,localPList)
```

which should decide whether to accept the command. Retrieve's decision may involve checking passwords, looking up files, and other actions using the information in remotePList plus other environment-specific information, such as whether the requester has the correct capabilities, etc. To refuse the request, Retrieve should call

```
     FTPM(markNo, code, string)
```

and return false. To accept the command, it should return a new pList describing a file matching remotePList which Retrieve is willing to send. FtpServProt will return this pList as 'localPList' in the next call to Retrieve, so that it can be deallocted. On the first call, localPList will be zero. Some FTP implementations require a minimum set of properties here, but the whole subject of who should specify what properties is rather involved and beyond the scope of this description. For more information, consult the FTP specification. This package provides a fast procedure (in the Utility module) for deciding the 'type' of a file (text or binary) which you may find useful.

Property lists in retrieve requests may specify multiple files, so FtpServProt will continue to call Retrieve until it returns false. On each call, remotePList will be the same original pList sent from the remote User, and localPList will be the last pList returned by Retrieve. If Retrieve supports multiple file requests then it must save some information so that the next time FtpServProt calls it, it can generate the next file. If Retrieve does not support multiple file requests then it should do its thing during the first call and remember that it is finished. The next time it is called it should return false having only deallocated localPList (it should not call FTPM).

If Retrieve returns true, FtpServProt sends the returned property list back to the User to more fully describe the file. At this point the User may back out of the transfer, in which case the next procedure will be skipped, and RetrieveCleanup will be called immediately. If the User indicates a willingness to proceed, FtpServProt then calls

```
        (CtxRunning>>FtpCtx.RetrieveFile)(pList)
↑L
```

For Xerox Internal Use Only -- November 17, 1977

to transfer the file data. This package provides a procedure (in the Utility module) for transferring data from a disk Stream to a BSP Stream, but you are free write your own. When RetrieveFile has finished the transfer, it should return true if everything went OK.

Next, FtpServProt calls

```
(CtxRunning>>FtpCtx.RetrieveCleanup)(pList,ok)
```

where 'ok' is false if RetrieveFile returned false or the User backed out of the command. Note that if Retrieve returned true, RetrieveCleanup will always be called, but RetrieveFile may not. If Retrieve allocates any resources (such as opening a file) they should be deallocated here.

Finally, FtpServProt calls Retrieve again, and the process repeats until Retrieve returns false.

2.3. Store Command

When the remote FTP User process sends the command 'Store' followed by a property list describing the file, FtpServProt parses the property list and calls

```
(CtxRunning>>FtpCtx.Store)(pList)
```

which should decide whether to accept the command. To accept, Store need only return true; no property list is sent back in this command. To refuse the command Store should call FTPM(markNo, code, string) and return false, in which case the next procedure (StoreFile) is not called.

If Store returns true, FtpServProt tells the User process to go ahead and send the file, and then calls

```
(CtxRunning>>FtpCtx.StoreFile)(pList)
```

to transfer the file data. This package provides a procedure (in the Utility module) for transferring data from a BSP Stream to a disk Stream, but you may write your own. When StoreFile has finished the transfer, it should return true if everything went OK.

Finally, FtpServProt calls

```
(CtxRunning>>FtpCtx.StoreCleanup)(pList,ok)
```

where 'ok' is true if StoreFile returned true and the User indicated that everything went ok. If 'ok' is false, StoreCleanup should delete the file, since it is almost certainly damaged. Note that if Store returned true, StoreCleanup will always be called, but StoreFile may not. If Store allocates any resources (such as opening a file) they should be deallocated here.

2.4. Delete Command

When the remote FTP User process sends the command 'Delete' followed by a property list describing the files which it wants to delete, FtpServProt parses the property list and calls
↑L

```
     (CtxRunning>>FtpCtx.Delete)(remotePList,localPList)
```

which should decide whether to accept the command. Don't delete anything yet! The User may still back out. To refuse the delete request, Delete should call FTPM(markNo, code, string) and return false. To accept the command, it should return a new pList with every property it can find, so that the User can be sure of the identity of file to be deleted, and return true. FtpServProt will return this pList as 'localPList' in the next call to Delete, so that it can be deallocted.

Property lists in delete requests may specify multiple files, so FtpServProt will continue to call Delete until it returns false. On each call, remotePList will be the same original pList sent from the remote User, and localPList will be the last pList returned by Delete. If Delete supports multiple file requests then it must save some information so that the next time FtpServProt calls it, it can generate the pList for the next file. If Delete does not support multiple file requests then it should do its thing during the first call and remember that it is finished. The next time it is called it should return false having only deallocated localPList (it should not call FTPM).

If Delete returns a Plist, FtpServProt will send it back to the User and wait for confirmation. If the User still wants to delete the file, FtpServProt calls

```
     (CtxRunning>>FtpCtx.DeleteFile)(pList)
```

which should delete the file. Finally, FtpServProtFile calls Delete again, and the process repeats until Delete returns false.

2.5. Directory Command

When the remote FTP User process sends the command 'Directory' followed by a property list naming the files about which it wants information, FtpServProt parses the property lists and calls

```
     (CtxRunning>>FtpCtx.Directory)(pList)
```

which should decide whether to accept the command. To refuse the request (because for example the requestor does not have the correct access capabilities) Directory should call FTPM(markNo, code, string) and return false. To accept the command it should return a pList describing a file.

Property lists in directory requests may specify multiple files, so FtpServProt will continue to call Directory until it returns false. If Directory supports multiple file requests then it must save some information so that the next time FtpServProt calls it, it can generate the pList for the next file. If Directory does not support multiple file requests then it should do its thing during the first call and remember that it is finished. The next time it is called it should return false having only deallocated localPList (it should not call FTPM).
↑L

2.6. Rename Command

When the remote FTP User process sends the command 'Rename' followed by two property lists describing the old and new files, FtpServProt parses the property lists and calls

```
     (CtxRunning>>FtpCtx.Rename)(oldPList,newPList)
```

which should decide whether to accept the command. The FTP protocol does not require that user access information be present in newPList, so access checking should be done on oldPlist only. To refuse the rename request, Rename should call FTPM(markNo, code, string) and return false. Otherwise it should rename the file returning true if successful. If the rename operation fails, Rename should call FTPM(markNo, code, string) and return false.

File FtpServProtMail.br implements the server part of the Mail Transfer Protocol. This description ignores various critical sections and other vital considerations which must be handled by the user-supplied routines in order to provide a reliable mail service. For the semantics of the protocol see <Pup>MailTransfer.ears.

2.7. StoreMail Command

When the remote FTP User process sends the command 'StoreMail' followed by a property list, FtpServProt parses the property list and calls

```
     (CtxRunning>>FtpCtx.StoreMail)(pList)
```

which should decide whether to accept the command. To accept, StoreMail need only return true; no property list is sent back in this command. To refuse the command StoreMail should call FTPM(markNo, code, string) and return false, in which case the next procedure (StoreMailFile) is not called.

If StoreMail returns true, FtpServProt tells the User process to go ahead and send the mail, and then calls

```
     (CtxRunning>>FtpCtx.StoreMailFile)(pList)
```

to transfer the file data. When StoreMailFile has finished the transfer, it should return true if everything went OK.

Finally, FtpServProt calls

```
     (CtxRunning>>FtpCtx.StoreMailCleanup)(pList,ok)
```

where 'ok' is true if StoreMailFile returned true and the User indicated that everything went ok. If 'ok' is false, StoreMailCleanup should delete the file, since it is almost certainly damaged. Note that if StoreMail returned true, StoreMailCleanup will always be called, but StoreMailFile may not. If StoreMail allocates any resources (such as opening a file) they should be deallocated here.
↑L

For Xerox Internal Use Only -- November 17, 1977

2.8. RetrieveMail Command

When the remote FTP User process sends the command RetrieveMail followed by a property list, FtpServProt parses the property list and calls

```
     (CtxRunning>>FtpCtx.RetrieveMail)(pList)
```

which should decide whether to accept the request. To refuse, RetrieveMail should call FTPM(markNo, code, string) and return false. To accept, it should return true; no property list is sent back in this command.

If RetrieveMail returns true, FtpServProt then calls

```
     (CtxRunning>>FtpCtx.RetrieveMailFile)(pList)
```

which should transfer the file. When RetrieveMailFile has finished, it should return true if everything went OK.

Next, FtpServProt calls

```
     (CtxRunning>>FtpCtx.FlushMailBox)(pList)
```

which should flush the contents of the mailbox. If this operation fails, FlushMailBox should call FTPM(markNo, code, string) and return false, otherwise it should return true.

2.9. MoveMailToFile Commmand

When the remote FTP User process sends the command MoveMailToFile followed by a property list, FtpServProt parses the property list and calls

```
     (CtxRunning>>FtpCtx.MoveMailToFile)(pList)
```

which should decide whether to accept the request. To refuse, MoveMailToFile should call FTPM(markNo, code, string) and return false. To accept the request, it should perform the operation and return true. If the operation fails, MoveMailToFile should call FTPM(markNo, code, string) and return false.

3. User

The FTP User module (files FtpUserProt.br, FtpUserProtFile.br, and FtpUserProtMail.br) implements the User protocol exchanges.

Many of the procedures in this module report results by returning a word containing an FTP mark code in the right byte and a subcode in the left byte (referred to below as 'subcode,,mark'). Marks and subcodes are the first two arguments to the FTPM procedure which is described in more detail in the Utility section. If the mark type is 'markNo', the subcode describes the reason why the Server refused; your modules may be able to fix the problem and retry the command. The package will output to dspStream text accompanying No, Version, and Comment marks.
↑L

For Xerox Internal Use Only -- November 17, 1977

3.1. Common User Protocol

File FtpUserProt.bcpl contains routines shared by FtpUserProtFile.br and FtpUserProtMail.br. It uses the bspStream, bspSoc, and dspStream fields in its FtpCtx and contains the following external procedures:

UserOpen(Version) = true|false
UserOpen should be called after the BSP Connection is open. It sends a version command and aborts the connection returning false if the Server's protocol is incompatible. Otherwise it calls

```
Version(stream,nil)
```

which should generate some herald text:

```
Wss(stream, "Alto Pup FTP User ")
```

to which UserOpen will append a string of the form "1.13 15-May-77", and then return true. The herald string received from the Server is output to dspStream.

UserClose(abortIt [false])
UserClose closes the FTP connection, aborting it if 'abortIt' is true.

UserFlushEOC() = true|false
flushes bspStream up to the next command, and returns true if it is EndOfCommand. If the stream closes or times out, it returns false. It calls UserProtocolError if it encounters anything except an EOC.

UserGetYesNo(flushEOC) = subcode,,mark
flushes bspStream up to the next command, which must be 'Yes' or 'No'. If flushEOC is true, it then calls UserFlushEOC and returns the Yes or No mark and accompanying subCode. If the stream closes or times out, it returns false. UserGetYesNo calls UserProtocolError if it encounters anything except Yes or No followed by EOC.

UserProtocolError()
Writes an error message to dspStream and then calls UserClose to abort the connection.

3.2. User File Operations

File FtpUserProtFile.br implements the User protocol for standard file operations. It uses the bspStream, bspSoc, and dspStream fields in its FtpCtx and contains the following external procedures:

UserStore(pList, StoreFile) = subcode,,mark
Attempts to send the file described by 'pList' to the remote Server, calling the user-supplied procedure 'StoreFile' to transfer the data. It returns zero if something catastrophic happens (such as the Server aborts the connection), in which case retrying is probably futile.

UserStore sends pList to the Server for approval. The Server can refuse the command at this point, in which case UserStore returns subcode,,markNo. If the Server accepts the command, UserStore calls

↑L

StoreFile(pList)

which should transfer the file data. This package provides procedures for transferring data from a disk stream to a network stream, but you are free to write your own. StoreFile should return true if the transfer went successfully. If some environment-specific thing goes wrong (such as an unrecoverable disk error), StoreFile should call FTPM(markNo, code, string, true) before returning false. UserStore then asks the Server if the transfer went successfully and returns subcode,,mark. If mark is 'markYes', the file arrived at the Server safely.

UserRetrieve(pList, Retrieve) = subcode,,mark
Attempts to retrieve the file described by 'pList' from the remote Server, calling the user-supplied procedure 'RetrieveFile' to transfer the data. UserRetrieve returns zero if some catastrophic error occurs, markNo if the Server refuses the command, and markEndOfCommand if the everything goes OK.

UserRetrieve sends pList to the Server and waits for approval. The Server can refuse the command at this point, in which case UserRetieve returns subcode,,markNo. If the Server can handle property lists that specify multiple files, then the following steps are taken for each file:

If the Server has no more files matching the original pList, UserRetrieve returns subcode,,markEndOfCommand (subcode is undefined in this case). Otherwise the Server sends a fully-specified property list describing a file which it is willing to send. UserRetrieve parses this into pList and calls

Retrieve(pList)

which should decide whether to accept the file. To skip the file, Retrieve should return false. UserRetrieve so informs the Server and then loops. To accept the file, Retrieve should return a procedure which UserRetrieve can call to transfer the data. Don't open the file yet, because the Server can still back out, in which case UserRetrieve skips the next step and just loops. If Retrieve returns true, UserRetrieve tells the Server to send the file and then calls

RetrieveFile(pList)

which should open the file, transfer the data, and close the file. This package contains procedures for transferring data from a network stream to a disk stream, but you are free to write your own. When RetrieveFile is done, it should return true if everything went OK. UserRetrieve then loops.

UserDelete(pList,Delete) = subcode,,mark
Requests the remote Server to delete the files described by 'pList', calling the user-supplied procedure DeleteFile before allowing the server to actually delete anything. UserDelete returns zero if some catastrophic error occurs, markNo if the Server refuses the command, and markEndOfCommand if the everything goes OK.

```
    UserDelete sends pLi̇st to  the Server and waits for  approval.  The
↑L
```

Server can refuse the command at this point, in which case UserDelete returns subcode,,markNo. If the Server can handle property lists that specify multiple files, then the following steps are taken for each file:

If the Server has no more files matching the original pList, UserDelete returns subcode,,markEndOfCommand. Otherwise the Server sends a fully-specified property list describing a file which it is willing to delete. UserDelete parses this into pList and calls

```
Delete(pList)
```

which should return true to confirm deleting the file described by 'pList'. UserDelete passes this answer on to the Server and then loops.

UserDirectory(pList, Directory) = subcode,,mark
Requests the remote Server to describe in as much detail as it can files matching 'pList', and then calls the user-supplied procedure Directory when the answers come back.

UserDirectory sends pList to the Server and waits for an answer. The Server can refuse the command at this point, in which case UserDirectory returns subcode,,markNo. If the Server can handle property lists that specify multiple files, then the following steps are taken for each file:

If the Server has no more files matching the original pList, UserDirectory returns subcode,,markEndOfCommand. Otherwise the Server sends a property list which UserDirectory parses into pList and calls

```
Directory(pList)
```

and then loops.

3.3. User Mail Operations

File FtpUserProtMail.br implements the user part of the Mail Transfer Protocol. This description ignores various critical sections and other vital considerations which must be handled by the user-supplied routines in order to provide a reliable mail service. For the semantics of the protocol see <Pup>MailTransfer.ears.

UserStoreMail(pList,StoreMail)
Attempts to send mail to the mailbox described by 'pList' at the remote Server, calling the user-supplied procedure 'StoreMail' to transfer the data. It returns zero if something catastrophic happens (such as the Server aborts the connection), in which case retrying is probably futile.

UserStoreMail sends pList to the Server for approval. The Server can refuse the command at this point, in which case UserStoreMail returns subcode,,markNo. If the Server accepts the command, UserStoreMail calls

```
         StoreMail(pList)
↑L
```

which should transfer the mail. StoreMail should return true if the transfer went successfully. If some environment-specific thing goes wrong (such as an unrecoverable disk error), StoreMail should call FTPM(markNo, code, string, true) before returning false. UserStoreMail then asks the Server if the transfer went successfully and returns subcode,,mark. If mark is 'markYes', the mail arrived at the Server safely.

UserRetrieveMail(pList,RetrieveMail) = subCode,,mark
Attempts to retrieve the contents of the mailbox described by 'pList' from the remote Server, calling the user-supplied procedure 'RetrieveMail' to transfer the data. UserRetrieveMail returns zero if some catastrophic error occurs, markNo if the Server refuses the command, and markEndOfCommand if the everything goes OK.

UserRetrieveMail sends pList to the Server and waits for approval. The Server can refuse the command at this point, in which case UserRetieveMail returns subcode,,markNo. Otherwise UserRetrieveMail calls

```
RetrieveMail(pList)
```

which should transfer the file data. When RetrieveMail is done, it should return true if everything went OK.

UserMoveMailToFile(pList) = subCode,,mark
requests the server to move the contents of the mailbox described by 'pList' to the file also described by pList. UserMoveMailToFile returns zero if some catastrophic error occurs, markNo if the Server refuses the command and markYes if everything goes OK.

4. Utility Routines

The utility module (files FtpUtilB.br, FtpUtilA.br, FtpUtilXfer, FtpUtilDmpLd, and FtpUtilInit.br) contains protocol routines shared by the User and Server modules, and some routines for efficiently manipulating disk streams.

InitFtpUtil()
builds some internal tables and streams, getting space from sysZone. You must call this procedure before starting a Server or issuing any User commands.

FTPM(mark, subCode [0], string [], eoc [false], par0, par1, par2, par3, par4)

sends the FTP command 'mark' to the remote FTP process, including 'subCode' if the command requires one, and 'string' if one is present. Then, if 'eoc' is true, an EOC command is sent. 'String' is written to bspStream using the Template package, and may contain imbedded format information. 'Par0' through 'par4' are passed as arguments to the PutTemplate call. The subcode and string arguments further explain certain commands. For markNo, subCode is a machine-readable explanation of why a request was refused, and 'String' is human-readable text such as "UserName and Password required". Codes are tabulated in an appendix to <Pup>FtpSpec.ears. New codes may be registered on request.

↑L

For Xerox Internal Use Only -- November 17, 1977

GetCommand(timeout [30000]) = subCode,,mark
flushes bspStream up to the next command and returns the mark and subcode (if any). Returns false if the stream closes or it hangs for 'timeout' miliseconds while waiting for a byte. Comment commands are ignored. GetCommand writes the strings accompanying Version, No, and Comment commands to dspStream.

The utility module makes three 'process-relative streams' for use by the rest of the package. The only operation defined is 'Puts'.

lst	writes to dspStream
dls	writes to dspStream if debugFlag is true
dbls	writes to bspStream and if debugFlag to dspStream

For example, Wss(dls,string) writes 'string' to the running process' dspStream if the process' debugFlag is set.

4.1. Unformatted Data Transfer

File FtpUtilXfer.br contains procedures for performing efficient operations on disk Streams. They use the following fields in FtpCtx: bspSoc, bspStream, dspStream, diskStream, buffer, and bufferLength. The following Alto operating system disk stream procedures are used: SetFilePos, FilePos, FileLength, ReadBlock, WriteBlock, plus the generic stream operations: Gets, Puts, Resets, and Endofs.

DiskToNet() = true|false
Transfers bytes from diskStream to bspStream up to end-of-file, and returns true if everything went OK. Before starting the transfer, DiskToNet outputs "...transferring..." to dspStream, and before returning it outputs "xxx bytes...".

NetToDisk() = true|false
Transfers bytes from bspStream to diskStream until it encounters another FTP command returning true if everything went smoothly. Before starting the transfer, NetToDisk outputs "...transferring..." to dspStream, and before returning it outputs "xxx bytes...".

FileType() = Text|Binary
Resets diskStream, scans it looking for high order bits on, and then Resets it again. As soon as it encounters a byte with the high order bit on, it returns 'Binary', otherwise (having read the entire file) it returns 'Text'. This routine does not use the bspSoc or bspStream fields in FtpCtx.

4.2. Dump Format Data Transfer

File FtpUserDmpLd.br contains procedures for transferring data between a disk and an FTP connection in dump format. They may be used as the inner loops of the user-supplied data transfer procedures passed to UserStore and UserRetrieve and will create and unbundle dump-format files on the fly. If you don't want to handle dump format, you don't need this file. Dump-file format is described in an appendix to the Alto Executive documentation.

For Xerox Internal Use Only -- November 17, 1977

These procedures use the same fields in FtpCtx and the same Alto OS routines as the unformatted transfer routines. Buffer must be at least 130 words long. Making it longer does not speed up the transfer.

DumpToNet(filename []) = true|false
    Dumps 'filename' from diskStream to bspStream converting it to dump format, returning true if things go OK. DumpToNet outputs "...xxx bytes" to dspStream before returning. To terminate a dump file, call DumpToNet without a filename.

LoadFromNet() = string or zero
    Loads files from bspStream to diskStream (if it is non-zero), converting them from dump format, returning a string when it encounters a name block and zero when it encounters an 'end block'. The caller should not modify the returned string. LoadFromNet outputs "...skipped" or "...xxx bytes" to dspStream for each component file in the dump file.

5. Property Lists

The property list module (files FtpPListProt.br, FtpPList1.br, and FtpPListInit.br) translates between this package's internal representation of a property list and the protocol-specified network representation.

The FTP protocol specifies the syntax of a property list and the syntax of a set of properties sufficient for standard file operations, but states that property lists are extensible. Therefore the property list module comes in two parts: a part that knows the syntax of property lists, and a part which knows the syntax of individual properties. To add new properties you need only modify the latter.

The principal data structure in this module is the File Property List Keyword Table, or fplKT. This table, built by InitFtpPlist, contains (propertyName,propertyObjects) pairs. PropertyNames are strings such as "Byte-size". PropertyObjects know how to Scan (parse) properties into pLists, Generate properties from pLists, initialize properties from a pList full of default values, and Free properties stored in pLists.

5.1. Property List Protocol

File FtpPlistProt.br implements four operations on property lists. This is the module that knows the syntax of a property list, but not the syntax of individual properties. Procedures in this file use the bspStream, bspSoc, and dspStream fields of the FtpCtx and contain the following external procedures:

InitPList(defaultPList []) = pList
    Creates an empty pList, and initializes it to be a copy of 'defaultPList' if one was supplied.

```
FreePList(pList)
    Destroys 'pList' and returns 0 to facilite writing pList =
    FreePList(pList).  If pList is zero, FreePList returns zero without
    doing anything.
↑L
```

ScanPList() = pList|false
Expects to find a property list in bspStream. ScanPList parses this property list and returns a pList if it had proper syntax. If the property list is malformed, ScanPList calls FTPM(markNo, code, string) and returns false. If the connection closes or ScanPList waits for more than 30 seconds while trying to read from bspStream, it returns false.

GenPList(pList)
Generates a property list in network format from 'pList' and sends it to bspStream.

5.2. The 'Standard' Properties

Files FtpPlist1.br and FtpPlistInit.br implement the standard properties. These files know the syntax of individual properties; they contain the operation procedures for the standard property objects. These files are used by the FTP subsystem and IFS and are sufficient for performing 'standard' file operations. If you wish to add properties, these are the modules which you must change. In addition to the property operations which are rather specialized to their task, there are a few generally useful procedures which are made external:

InitFtpPList()
which makes the standard property objects and builds fplKT, getting space from sysZone. This procedure must be called before calling any of the procedures in FtpPlist.br (which typically means before starting a server or calling any procedures in the User module).

Nin(string,lvDest) = true|false
Interprets 'string' as a decimal number and leaves the result in 'lvDest', ignoring leading blanks and terminating on end of string. A null string results in lvDest getting 0. Returns false if the string contains any characters other than 0-9 and <space>.

ParseDate(string,lvRes) = true|false
Parses the string format date into an Alto format date which it puts into the two word vector at 'lvRes'. Returns true if it could parse the date. ParseDate expects the format of the string to bear some similarity to "day-month-year hour:minute:second".

WriteDT(stream,dt)
converts 'dt' from 32 bit Alto date format to a string of the form "dd-mmm-yy hh:mm:ss" and writes it to 'stream'.

6. Example

The following example program makes use of most of the facilities in the User part of the Ftp Package. I have run it and it works. It is a rock-bottom minimal User Ftp with no redeeming features whatsoever. More extensive and realistic examples can be found by looking at the sources for the Ftp subsystem.

The main procedure FtpUserExample performs initialization, which consists of augmenting SysZone, initializing the Ftp and Pup packages, and creating and starting a context running the procedure User.
↑L

For Xerox Internal Use Only -- November 17, 1977

User opens a BSP connection to Maxc, sets up its FtpCtx, gets and fills a blank pList, and calls UserRetrieve. When UserRetrieve returns, User closes the connection, releases its resources and commits suicide.

```
//FtpUserExample.bcpl - Example Ftp User

//last modified October 24, 1977  6:05 PM

// The load command file is:
// Bldr/l/v 600/W FtpUserExample ↑
// ↑
// FtpUserProt FtpUserProtFile ↑
// FtpPListProt FtpPList1 ↑
// FtpUtilb FtpUtila FtpUtilXfer ↑
// ↑
// PupBspStreams PupBspProt PupBspBlock PupBspA ↑
// PupRtp PupNameLookup ↑
// Pup1B PupAl1A PupRoute PupDummyGate ↑
// PupAlEthB PupAlEthA ↑
// ↑
// Context ContextInit Interrupt ↑
// AltoQueue AltoTimer AltoByteBlt ↑
// Template CTime StringUtil Keyword ↑
// ↑
// FtpPlistInit FtpUtilInit KeywordInit ↑
// Pup1Init PupAlEthInit InterruptInit

get "FtpProt.decl"
get "Pup.decl"

external
[
//incoming procedures
InitFtpUtil; InitFtpPList; InitPupLevel1
GetFixed; CallSwat; AddToZone; Allocate; Free
InitializeContext; CallContextList; Enqueue
GetPartner; OpenLevel1Socket; OpenRTPSocket; CreateBSPStream
InitPList; FreePList; NetToDisk
UserRetrieve; UserOpen; UserClose; NetToDisk
ExtractSubstring; OpenFile; Closes; Wss
```

```
//incoming statics
sysZone; dsp; CtxRunning; UserName; UserPassword
]

let FtpUserExample() be
[
let v = GetFixed(10000)
if v eq 0 then CallSwat("GetFixed failed")
AddToZone(sysZone,v,10000)
let ctxQ = vec 1; ctxQ!0 = 0
InitFtpUtil()
InitFtpPList()
InitPupLevel1(sysZone,ctxQ,10)
Enqueue(ctxQ,InitializeContext(Allocate(sysZone,1000),
 1000,User,lenExtraCtx))
CallContextList(ctxQ!0) repeat
]

and User(ctx) be  //a context
[
↑L
```

For Xerox Internal Use Only -- November 17, 1977

```
let soc = Allocate(sysZone,lenBSPSoc)
let maxcPort = vec lenPort
unless GetPartner("Maxc",dsp,maxcPort,0,socketFTP) do
   CallSwat("GetPartner failed")
OpenLevel1Socket(soc,0,maxcPort)
unless OpenRTPSocket(soc) do
   CallSwat("OpenRTPSocket failed")

CtxRunning>>FtpCtx.bspStream = CreateBSPStream(soc)
CtxRunning>>FtpCtx.bspSoc = soc
CtxRunning>>FtpCtx.dspStream = dsp
CtxRunning>>FtpCtx.buffer = Allocate(sysZone,256)
CtxRunning>>FtpCtx.bufferLength = 256
CtxRunning>>FtpCtx.debugFlag = true
unless UserOpen(Version) do
   CallSwat("UserOpen failed")

let pList = InitPList()
pList>>FPL.UNAM = ExtractSubstring(UserName)
pList>>FPL.UPSW = ExtractSubstring(UserPassword)
pList>>FPL.SFIL = ExtractSubstring("<system>Pup-Network.txt")

let mark = UserRetrieve(pList, Retrieve)
if mark ne markEndOfCommand then
   CallSwat("UserRetrieve failed")
FreePList(pList)
UserClose()
Free(sysZone,soc)
Free(sysZone,CtxRunning>>FtpCtx.buffer)
finish
]

and Version(stream,nil) be Wss(stream,"Example FTP User")

and Retrieve(pList) = RetrieveFile

and RetrieveFile(pList) = valof
[
let s = OpenFile(pList>>FPL.NAMB,ksTypeWriteOnly,charItem)
CtxRunning>>FtpCtx.diskStream = s
unless NetToDisk() do CallSwat("NetToDisk failed")
Closes(s)
resultis true
]
```

7. Revision History

March 30, 1977

First release.

May 15, 1977

Added Directory and Rename commands. Server now handles property lists which specify multiple files. Added User and Server mail operations.

June 8, 1977
↑L

Overlay machinery was changed and some bugs were fixed. Some structure definitions changed, so recompilation of user programs is necessary.

July 17, 1977

DiskToNet and NetToDisk moved out of FtpUtilb into a new file FtpUtilXfer. Property lists reorganized, causing changes to the calling interface in FTPSFI. Plist module now uses the Keyword routines in the CmdScan package. Recompilation of user programs is necessary. FtpUserDmpLd renamed FtpUtilDmpLd. Timeouts cleaned up.

October 24, 1977

Example program added.
↑L