Communications Server z/OS V1R5 and V1R6 Technical Update

FTP Client API on z/OS®

eserver

Topics

e-business

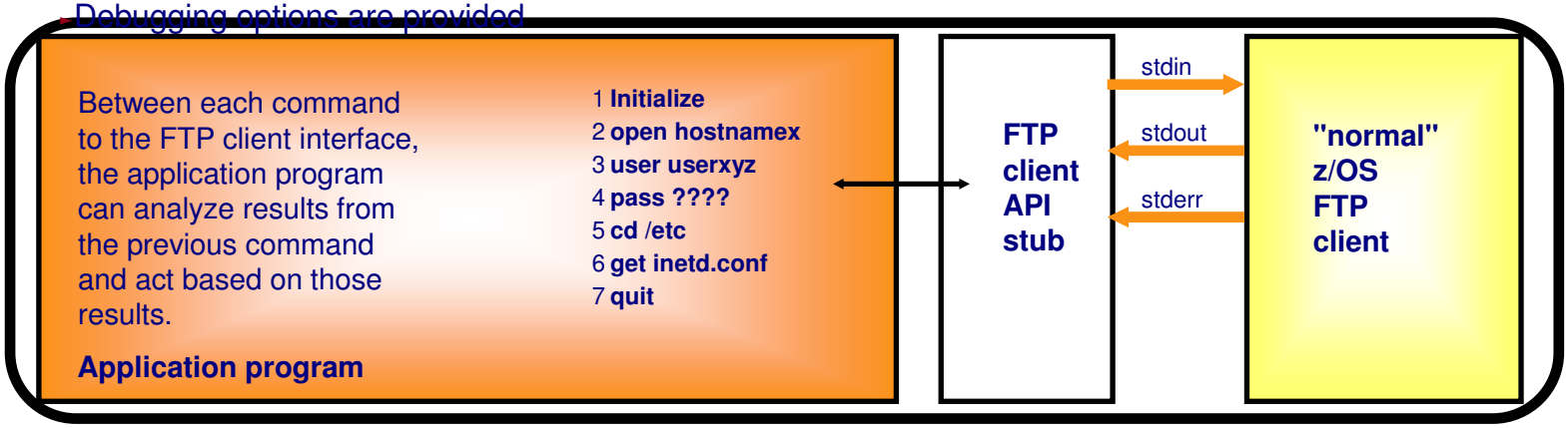
- z/OS V1R6
  - FTP client programming interface
  - Modernized DBCS/MBCS handling by FTP
- Recent APAR activity

FTP Client Application Programming Interface

z/OS FTP Client programming interface for improved automation and integration of z/OS file transfers in z/OS V1R6

- Provides an interface that allows an application to programatically invoke the FTP client on z/OS from common environments (unix shell, TSO, or MVS batch job)

- Characteristics of the interface:
  - z/OS V1R6 provides a callable interface to be used from Assembler, Cobol, PL/I (or any z/OS supported programming language that supports a call interface) - plans to add C and REXX APIs in z/OS V1R7
  - Interface is reentrant and does support multiple parallel FTP client sessions by tasks within an address space
  - For communication between the program and the interface, a simple set of commands and data areas are used. (Mappings for common programming languages are provided)
  - Both blocking (wait for a response), and non-blocking (polling-mode) calls are supported
  - In non-blocking mode, progress replies can be returned to the calling application as the transfer progresses
  - The simple commands tell the interface what to do, for example: initialize, terminate, execute an FTP client command, process output from the FTP client command that was executed, poll for command completion.
  - Results are returned as structured fields in communication area control blocks (return codes from interface and server replies or possibly local command) along with free-format replies from the FTP client code
  - Debugging options are provided



Why do we need an FTP client programming interface?

- Customers utilize the z/OS FTP Client and Server in highly critical applications to transform and transfer data between hosts and platforms. The z/OS FTP Client handles requests and drives the Server in performing these tasks.

- The z/OS FTP Client can be invoked in several environments and accepts subcommands either entered by an interactive user or contained in a pre-built script

- An interactive user can evaluate request results immediately and decide whether and how to proceed - however, as an "application", an interactive user is slow and costly

Large-scale applications generally use a pre-built script, which limits options for conditional execution

- Currently, an application using a script must choose whether to exit at the first error in an eligible subcommand or to ignore all errors and continue processing subcommands

- This limitation requires minute differentiation of tasks within separate steps to enable granular conditional execution

- z/OS V1R6 introduces a new interface to the z/OS FTP Client that allows customers to automate and manage not only routine tasks, but also mutable and exceptional events in the Client and Server, in an informed and directed fashion

FTP client programming interface - introduction

- The FTP Callable Application Programming Interface addresses these requirements and includes the following features
  - Uses a standard call interface
  - Is reentrant and reusable
  - Does not establish a new enclave within the application run unit
  - Permits the use of multiple instances of the interface by one program
  - Polls the Client until a subcommand completes (wait mode) or returns to the application to enable multitasking (no-wait mode)
  - Allows the application to poll the client for status of a subcommand
  - Returns results of the request and collects the output that was generated by the z/OS FTP Client

- Information returned by the FTP Callable API
  - Results from the interface
    - Overall result code (also returned in the return code register)
    - Status code
    - Interface error code
    - Interface service error return and reason codes
  - Results from the z/OS FTP Client and Server
    - Client error code
    - Last Server reply code received
    - Subcommand code
  - Statistics about any output generated by the request
  - Not all results are applicable for any given request

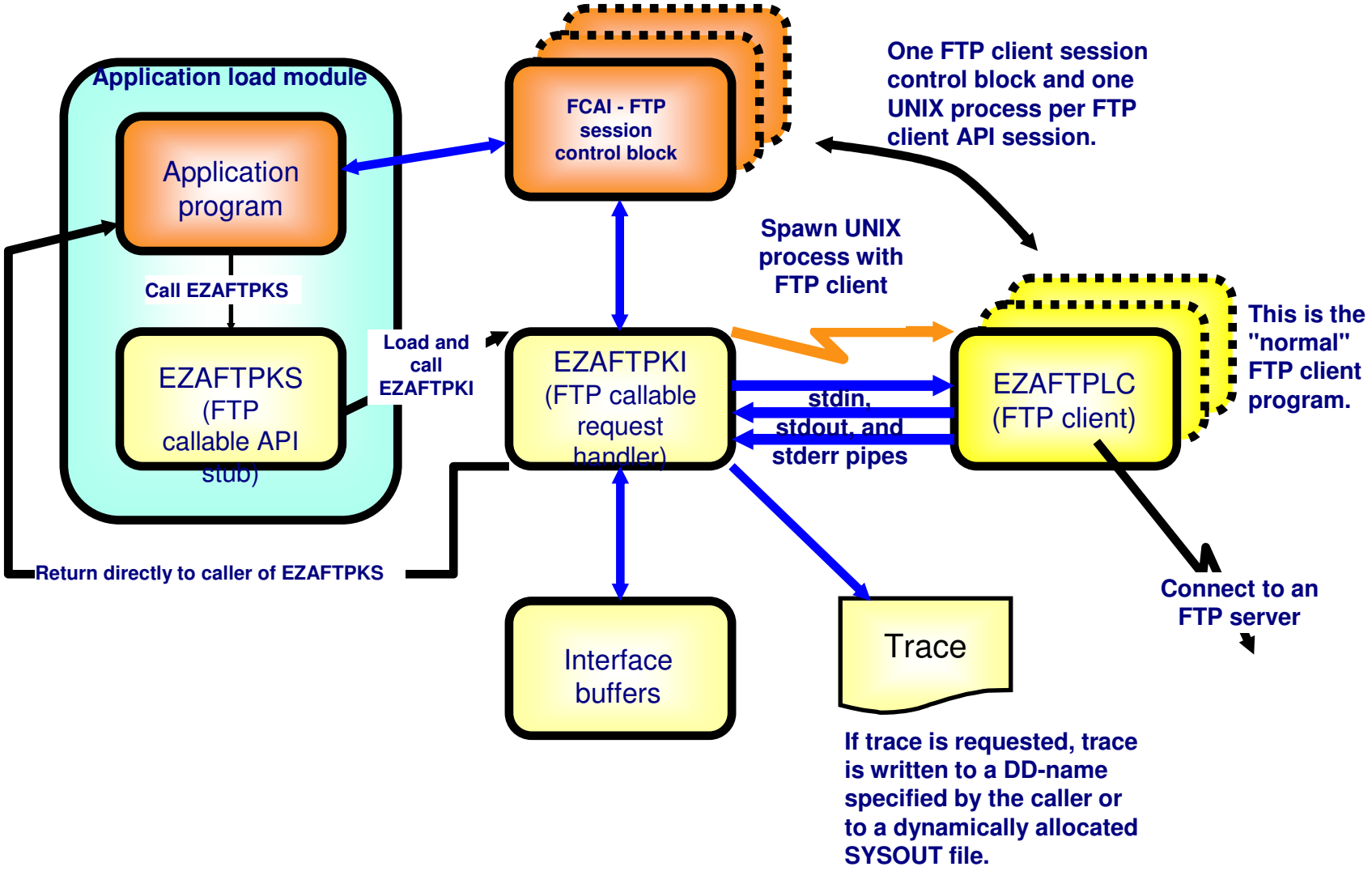
FTP client programming interface - introduction

- Elements of the FTP Callable API
  - The application program (or driver) that sends requests to the interface and interrogates the results
  - An FTP Callable API control block (FCAI) to define each instance of the interface
  - The FTP Callable API stub program (EZAFTPKS) - link edited with the application or loaded dynamically and reused
  - The FTP Callable API request handler (EZAFTPKI) - loaded by the stub program and reused
  - An FTP Client child process for each instance of the interface, which executes in the same or new address space
  - An interface buffer for each instance that contains the results from the z/OS FTP Client for the preceding initialization or subcommand request

FTP client programming interface - introduction

- Requirements for using the FTP Callable API
  - Interface programs must be able to access storage acquired by the application, which includes FCAI control block(s) and optional buffer(s)
  - All requests with the same FCAI must execute under the same TCB
  - Standard CALL interface with samples for COBOL, PL/I and Assembler
  - The only addressing mode currently supported is 31-bit (AMODE 31). The application program can reside below the line (RMODE 24).
  - OMVS segment defined or defaulted for the application
  - The API has no signal handlers and raises no explicit signals
  - The application can catch an implicit SIGCHILD when the Client ends
  - So that the interface can trap certain ABENDs, specify TRAP(ON,NOSPIE) to disable invocation of the ESPIE macro when the application program executes within an LE enclave. For example, specify the following execution parameter for a COBOL application program: PARM='/TRAP(ON,NOSPIE)'
    - For instructions on specifying run-time options and parameters for LE languages, refer to "Using Run-Time Options" in z/OS Language Environment Programming Guide.
  - EZAFTPKS (the interface stub program) must be linked edited with the user program or loaded dynamically at execution
    - Shipped in CSSLIB
    - Minimal function to ensure upward compatibility
  - EZAFTPKI (the interface request handler module) must be available from the linklist or in the STEPLIB/JOBLIB for the application program

Structure of the FTP client API implementation



Programming references

- In library hlq.SEZACMAC
  - EZAFTPKA Assembler version of FCAI_Map
- In library hlq.SEZANMAC
  - EZAFTPKC Enterprise COBOL copy member for FCAI-Map
  - EZAFTPKP PL/I include member for FCAI_Map
- In library hlq.SEZAINST
  - EZAFTPAW Sample assembler application
  - EZAFTPAX Sample Enterprise COBOL application
  - EZAFTPAY Sample PL/I application
- The application program communicates with the interface by passing parameters on each call and by using the FCAI control block, which it must acquire in primary space prior to initialization
- The definitions for the FCAI include equates, constants, or level-88 names for setting and interpreting FCAI fields
- The FCAI must persist for the life of the instance of use of the interface
- The IP Programmer's Reference contains detailed information on the FCAI fields, how to set and interpret them, and tips on diagnosing error conditions

FCAI - Assembler layout - part 1 of 4


```
*********************************************************************
*                                                                   *
*        FTP Callable Application Interface (FCAI) control block    *
*                                                                   *
*        Each field in the mapping is marked with one of the        *
*        following:                                                 *
*           :I     Input field that is set by the user program     *
*           :I*    Input field that is set by the user program and *
*                  is further defined by equated values (see       *
*                  the values at the end of the mapping)           *
*           :O     Output field that is set by the interface       *
*           :O*    Output field that is set by the interface and   *
*                  is further defined by equated values (see       *
*                  the values at the end of the mapping)           *
*           :R     Reserved for use by the interface               *
*           :U     Application work area                           *
*                                                                   *
*        Fields marked  must be set by the user program.    *
*                                                                   *
*********************************************************************

FCAI_Map                 DSECT ,  map the FCAI
FCAI_Eyecatcher          DS   CL4 eyecatcher = 'FCAI'     :I
FCAI_Size                DS   H   size of FCAI area       :I
*                                 set to FCAI_NumInterfaceBytes
FCAI_Version             DS   XL1 version of FCAI         :I*
```

FCAI - Assembler layout - part 2 of 4


```
*-------------------------------------------------------------------*
*        See "Programming notes for the FTP Callable API" in        *
*        IP Programmer's Reference for a description of             *
*        FCAI_PollWait and FCAI_ReqTimer.                           *
*-------------------------------------------------------------------*
FCAI_PollWait             DS   XL1 seconds to wait before POLL read :I
*                                   0 = always wait 1 second (default)
*                                 >0 = max progressive wait value
FCAI_ReqTimer             DS   XL1 Request completion timer         :I
*-------------------------------------------------------------------*
*        Fields used in the API tracing function                    *
*-------------------------------------------------------------------*
FCAI_TraceIt              DS   XL1 trace indicator                  :I*
FCAI_TraceID              DS   CL3 ID used in a trace record        :I
FCAI_TraceCAPI            DS   XL1 TRACECAPI FTP.DATA statement     :O*
FCAI_TraceStatus          DS   XL1 status of the trace              :O*
FCAI_TraceSClass          DS   CL1 SYSOUT class for trace           :I
FCAI_TraceName            DS   CL8 ddname of the trace file         :O
*-------------------------------------------------------------------*
*        Interface token and request ID                             *
*-------------------------------------------------------------------*
FCAI_Token                DS   F   interface token                  :O
FCAI_RequestID            DS   CL4 last request (for example, SCMD) :O
```

FCAI - Assembler layout - part 3 of 4


```
*-------------------------------------------------------------------*
*        Request Completion Values                                  *
*        See "Interpreting results from an interface request" in the  *
*        IP Programmer's Reference for more information.            *
*-------------------------------------------------------------------*
                         DS   0F  start of request completion values
FCAI_Result              DS   X   result of last request           :O*
FCAI_Status              DS   X   status code for a request         :O*
FCAI_IE                  DS   X   interface error (IE)             :O*
FCAI_CEC                 DS   X   Client Error Code (CEC)          :O*
FCAI_ReplyCode           DS   H   server reply code (or 0 if none) :O
FCAI_SCMD                DS   X   client subcommand code           :O*
                         DS   XL1 reserved                         :R
FCAI_ReturnCode          DS   F   return code (see FCAI_IE values  :O
*                                 for conditions that set this)
FCAI_ReasonCode          DS   F   reason code (see FCAI_IE values  :O
*                                 for conditions that set this)
*-------------------------------------------------------------------*
*        Statistics about output in the interface buffer            *
*-------------------------------------------------------------------*
FCAI_NumberLines         DS   F   number of lines of output        :O
FCAI_LongestLine         DS   F   size of the longest output line  :O
FCAI_SizeAll             DS   F   size of all lines of output      :O
FCAI_SizeMessages        DS   F   size of all message output lines :O
FCAI_SizeReplies         DS   F   size of all reply output lines   :O
FCAI_SizeList            DS   F   size of all LIST or NLST output  :O
FCAI_SizeTrace           DS   F   size of all trace output         :O
```

FCAI - Assembler layout - part 4 of 4


```
*-------------------------------------------------------------------*
*        End of fields currently defined to the interface            *
*-------------------------------------------------------------------*
FCAI_ReservedForInterface DS   45F reserved                        :R
FCAI_NumReservedBytes     EQU  *-FCAI_ReservedForInterface (180)
FCAI_NumInterfaceBytes    EQU  *-FCAI_Map Total interf bytes (256)
FCAI_UserArea             DS   0C  User work area                  :U
```

Call syntax

- The FTP Callable API is invoked by calling EZAFTPKS -- the interface stub program -- from the application program
- The calling program must obtain storage for an FCAI and initialize selected fields in the FCAI before the first call to EZAFTPKS
  - Storage for the FCAI can be static storage in the calling CSECT or it can be acquired dynamically via STORAGE Obtain or by other means.
- EZAFTPKS call format for COBOL programs
  - CALL 'EZAFTPKS' USING FCAI-Map, request_type, parm1, parm2, ...
- EZAFTPKS call format for assembler programs
  - CALL EZAFTPKS,(FCAI_Map, request_type, parm1, parm2, ...),VL
- EZAFTPKS call format for PL/I programs
  - CALL EZAFTPKS (FCAI_Map, request_type, parm1, parm2, ...);
- Call types:
  - INIT - Initialize an FTP client session
  - TERM - Terminate an FTP client session
  - SCMD - Submit an FTP client command (such as: open, user, get, put, locsite, etc.)
  - GETL - Retrieve output lines from previous SCMD command interaction
  - POLL - Query status of asynchronous SCMD command

Format of data returned over the API to calling application

Columns	Description	Contents
1	Line type	'M' - message 'R' - reply 'L' - LIST/NLST 'T' - Client trace
2-3	Length of following text	0 to 2400
4-n (unless length is 0)	Text of line	Any

```
M 0025 >>> PORT 127,0,0,1,4,109
R 0020 200 Port request OK.
M 0009 >>> LIST
R 0019 125 List started OK
L 0010 total 3960
L 0067 -rw-r-----   1 IBMUSER  SYS1        855 Jan 28  2002 CACertRaw.b64
L 0066 -rw-r-----   1 IBMUSER  SYS1       8192 Jun 30 17:42 Document.txt
L 0066 dr-xr--r--   2 IBMUSER  SYS1       8192 Nov 12  2001 Nov2001_cert
L 0064 -rw-r-----   1 IBMUSER  SYS1       1230 Jun 30 16:31 Readme.txt
L 0069 -rw-r-----   1 IBMUSER  SYS1         21 Sep  4  2001 zos_ebcdic_file
L 0069 -rw-r-----   1 IBMUSER  SYS1          0 Jan 22  2003 |touch testfile
R 0032 250 List completed successfully.
M 0024 Command(00-14-LIST-250):
```

Sample assembler program
Part 1 of 5


```
          PRINT NOGEN
*
         EZAFTPKA
*
FTPAPIS1 INIT 'Sample FTP API Client program 1',RMODE=24
*
         OPEN  (SYSPRINT,(OUTPUT)),MODE=31
         LA    R10,FCAI
         USING FCAI_Map,R10
*
* Initialize FCAI
*
         MVC   FCAI_Eyecatcher,=CL4'FCAI'
         MVC   FCAI_Size,=AL2(FCAI_NumInterfaceBytes)
         MVC   FCAI_Version,=AL1(FCAI_Version_Number)
         MVC   FCAI_TraceIt,=AL1(FCAI_TraceIt_Yes)
         MVC   FCAI_TraceSClass,=CL1'X'
         LA    R2,2                *Max wait 2 minutes
         STC   R2,FCAI_ReqTimer
*
* Initialize the API
*
         CALL  EZAFTPKS,(FCAI,APIINIT),VL
         LTR   R15,R15             *INIT OK ???
         BNZ   INITFAIL

.....

         DS    0D
FCAI     DC    XL(FCAI_NumInterfaceBytes)'00'
```

Only reason for RMODE=24 is in-line QSAM DCBs (nothing to do with the FTP API).

I have the FCAI defined as part of this CSECT.

Eyecatcher, size, and version must be initialized before calling INIT.

I enable tracing to a SYSOUT file with output class=X

I will max wait 2 minutes when doing synchronous calls to the API

Sample assembler program
Part 2 of 5

```
*
* Send an OPEN command
*
         CALL  EZAFTPKS,(FCAI,APISCMD,CMDOPEN),VL
         LTR   R15,R15
         BNZ   OPENFAIL
*
* Send a USER command and expect a password prompt
*
         CALL  EZAFTPKS,(FCAI,APISCMD,CMDUSER),VL
         CH    R15,=AL2(FCAI_Result_Status)
         BNE   USERFAIL
         CLC   FCAI_Status,=AL1(FCAI_Status_PromptPass)
         BNE   USERFAIL

......

CMDOPEN  DC    AL2(L'OPENTXT)
OPENTXT  DC    C'OPEN 127.0.0.1'
*
CMDUSER  DC    AL2(L'USERTXT)
USERTXT  DC    C'USER USER1'
```

First thing after initializing the API in this sample is to connect to an FTP server at the loopback address:

OPEN 127.0.0.1

If open is successful, we proceed to log on to the FTP server:

USER USER1

The expected result of the USER command is to receive a prompt for a password, which is indicated in one of the FCAI status fields.

Sample assembler program
Part 3 of 5

```
*
* Send a password as response to the password prompt
*
         CALL  EZAFTPKS,(FCAI,APISCMD,CMDPASS),VL
         LTR   R15,R15
         BNZ   PASSFAIL
*
* Send a DIR command
*
         CALL  EZAFTPKS,(FCAI,APISCMD,CMDDIR),VL
         LTR   R15,R15
         BNZ   DIRFAIL

.....

*
CMDPASS  DC    AL2(L'PASSTXT)
PASSTXT  DC    C'PASS xxxxxx'
*
CMDDIR   DC    AL2(L'DIRTXT)
DIRTXT   DC    C'DIR'
*
```

If prompted for a password, we send a PASS command:

PASS xxxxxx

We're now logged on to the remote FTP server.

In this very simple sample, we now just send a DIR command to the server

DIR

Sample assembler program
Part 4 of 5


```
* Retrieve all output lines and print them to SYSPRINT
*
GETLNEXT EQU   *
*
         CALL  EZAFTPKS,(FCAI,APIGETL,
               GETLFIND,GETLTYPE,GETLSEQ,GETLBUF),VL
         LTR   R15,R15
         BZ    GETLGOOD
         CH    R15,=AL2(FCAI_Result_NoMatch)
         BE    GETLDONE
         B     GETLFAIL
GETLGOOD EQU   *
         MVI   PRTDATA,C' '
         MVC   PRTDATA+1(L'PRTDATA-1),PRTDATA
         MVC   PRTTYPE,LINEID
         SR    R2,R2
         ICM   R2,B'0011',LINELEN
         LTR   R2,R2
         BZ    LINEFAIL
         CVD   R2,DORD
         OI    DORD+7,X'0F'
         UNPK  PRTLEN,DORD
         BCTR  R2,0
         EX    R2,MVCLINE
         PUT   SYSPRINT,PRTLINE
         B     GETLNEXT
MVCLINE  MVC   PRTDATA(*-*),LINE
GETLDONE EQU   *
```

In a real application, we would have analyzed the output lines from the DIR command - here we simply format them and print them to a SYSPRINT file.

We retrieve the lines one by one using the GETL request type - FIND sequential selected line types (in this example: A for all).

- M Message from the client.
- R Reply from the server.
- L List data from a DIR or LS subcommand.
- T Trace output from debug or dump.
- A Any type of output line.

```
GETLFIND DC    CL4'FIND'
GETLTYPE DC    CL1'A'
GETLSEQ  DC    CL1'N'
GETLBUF  DC    A(BUFFER,0,1024)
```

Sample assembler program
Part 5 of 5


```
*
* Send a QUIT command
*
         WTO   '*** Calling SCMD with Quit'
         CALL  EZAFTPKS,(FCAI,APISCMD,CMDQUIT),VL
         LTR   R15,R15
         BNZ   QUITFAIL
*
* Send TERM
*
DONE     EQU   *
         CALL  EZAFTPKS,(FCAI,APITERM),VL
         CLOSE (SYSPRINT),MODE=31
         TERM
```

When we're done, we send a QUIT command for the client to disconncet from the server and terminate.

Finally we terminate the FTP session with callable interface.

For more details and instructions on how to program to the callable FTP client programming interface, refer to:

IP Programmer's Reference - Chapter 12 "FTP Callable Application Programming Interface", SC31-8787

Improved transfer progress feedback

- The z/OS FTP Client has always issued one of two messages at 10-second intervals to indicate the progress of a long-running inbound or outbound transfer
- In support of the FTP Callable API, the content of the transfer progress messages has been enhanced to include the bytes transferred during the interval and the rates of transfer
- The interval is now configurable or the messages can be suppressed entirely
  - An interval value of 0 suppresses the messages
  - Otherwise, the minimum (and default) interval value is 10 seconds
- All users of the z/OS FTP Client can take advantage of these changes, whether or not they invoke the Client from the API
  - EZA1485I number bytes transferred - interval second interval rate KB KB/sec - Overall transfer rate KB KB/sec
  - EZA2509I number megabytes transferred - interval second interval rate KB KB/sec - Overall transfer rate KB KB/sec
- New LOCSITE parameter:

```
PROGRESS = {10 | number}
```

- New FTP.DATA statement:

```
PROGRESS = {10 | number}
```

  - number - specifies the interval in seconds between progress report messages generated in the FTP client during an inbound or outbound file transfer. A value of zero turns progress reporting off in the FTP client. The default value is 10 seconds

Operation of the FTP Client under the API Prompting

There are several places in the z/OS FTP client where the client prompts the user for a response after a subcommand is processed. For example, after the USER subcommand is processed, the client prompts for a password. The FTP Callable API minimizes the number of situations that require a prompt. This simplifies the reactions required by the user program.

- Prompt for IP address if not supplied as a start parameter on the INIT request
  - The FTP client prompts immediately if the IP address was not supplied. **The FTP Callable API does not pass this prompt to the user program.** The user program should use SCMD to send an OPEN subcommand as soon as it wants a session with the FTP server.
- Prompt for userid after an OPEN subcommand
  - The FTP client prompts for a userid for login after the session is set up with the server. **The FTP Callable API does not pass this prompt to the user program.** The user program should use SCMD to send a USER subcommand as soon as it wants to login with the FTP server. The user program can provide the password as well as the userid as parameters with the USER subcommand.
- Prompt for password after a USER subcommand
  - The FTP client prompts for a password to complete a login if one was not passed with the USER subcommand. **The FTP Callable API passes this prompt to the user program using FCAI_Status_PromptPass.** The user program should use SCMD to send a PASS subcommand as the next subcommand. If any subcommand other than PASS is sent, the request fails with FCAI_IE_PassPromptErr.
- Prompt for subcommand after a PROXY subcommand
  - The FTP client prompts for a subcommand if PROXY is entered without a subcommand parameter. **The FTP Callable API does not support PROXY without a subcommand.** If the client receives PROXY without a subcommand the request fails with FCAI_CEC_PROXY_ERR.

Operation of the FTP Client under the API Prompting

- Prompt for accounting information after a USER, PASS, or CD (CWD) subcommand
  - Some FTP servers prompt the FTP client for accounting information after a USER, PASS, or CWD command. **The FTP Callable API passes this prompt to the user program using FCAI_Status_PromptAcct.** The user program should use SCMD to send an ACCT (or ACCOUNT) subcommand as the next subcommand. If any subcommand other than ACCT (or ACCOUNT) is sent, the request returns FCAI-IE-AcctPromptErr.
  - Tip: When a PASS or ACCT (or ACCOUNT) subcommand is expected, the interface refuses any other SCMD request until the prompt is satisfied. The user program can issue GETL or TERM without satisfying the prompt. TERM generates a QUIT subcommand that is accepted and stops the client process.
- Prompt for confirmation for MGET, MPUT, and MDELETE subcommands
  - The FTP client prompts for confirmation for these subcommands if the prompting subcommand has toggled prompting on. (Note that this is the state in which the FTP client starts unless the "-i" start parameter is specified.) **The FTP Callable API does not pass this prompt to the user program.** The subcommand is executed as if prompting were turned off.
- General command prompt: EZA2121I Command (***ee-ss-cccc-rrr***):
  - ***ee*** is the 2-digit decimal client error code for the subcommand (00 if none))
  - ***ss*** is the 2-digit decimal subcommand code (this field is blank when INIT does not cause an implicit OPEN to be performed)
  - ***cccc*** is the final 4-character FTP command sent to the Server (blank if none)
  - ***rrr*** is the numeric code from the last Server reply (blank if none)

Operation of the FTP Client under the API Miscellaneous

- The z/OS FTP client that is used by the FTP Callable API is described in *IP User's Guide and Commands* and *IP Configuration Reference*. The z/OS FTP client, when started with the FTP Callable API, operates essentially as it does when invoked in an interactive environment under the z/OS UNIX shell.
- When the z/OS FTP client is invoked from a batch job or from TSO, data sets and files can be allocated to DD names for use by the client. When the z/OS FTP client is spawned from the FTP Callable API, DD names associated with the application are not available to the client process. Specifically, the use of the following DD names is not supported by the FTP Callable API:
  - SYSFTPD and SYSTCPD
  - NETRC
  - INPUT (SYSIN) and OUTPUT
- Transfer of data sets by DD name is not possible in the spawned client process. If the application sends a transfer subcommand (PUT, GET, etc.) that includes //**DD:ddname**, the client returns FCAI_CEC_FILE_ACCESS.
- When the z/OS FTP Client starts, options (parameters) are processed that affect the operation of the client. The user program uses the START-PARM parameter on the INIT request to pass its options to the FTP Callable API which passes them on to the client. All of the options defined for the z/OS FTP client are accepted when the client is started with the FTP Callable API but note the following:
  - The "-e" and "EXIT" options are ignored by the FTP Callable API. These are intended to affect the operation of the FTP client by causing it to stop when an eligible subcommand encounters an error. In the FTP Callable API those errors are passed back to the user program as a Client Error Code so the application can process the error and decide whether and how to continue.
  - The "-i" option to disable prompting for m* commands has no effect on the API. m* command prompting is always off.

Operation of the FTP Client under the API Miscellaneous

- When the z/OS FTP client is executed within the z/OS UNIX shell, a backslash '\' is required before the '(' that signals the start of the MVS-type parameters. Do not use the backslash when invoking the client with the FTP Callable API.
- The z/OS FTP client describes how you change local site defaults using FTP.DATA. The search order for locating the FTP.DATA configuration file for the client under the FTP Callable API is:
  - $HOME/ftp.data
  - userid.FTP.DATA
  - /etc/ftp.data
  - SYS1.TCPPARMS(FTPDATA)
  - tcpip_hlq.FTP.DATA
- The IP Configuration Reference defines the FTP.DATA statements that can be used to change local site defaults for the z/OS FTP Client. One of the statements is CLIENTERRCODES which controls return code settings in the client. When the Client is started by the FTP Callable API, the value on the CLIENTERRCODES statement does not affect the reporting of results by the interface.
- When the z/OS FTP Server prompts for a password or accounting data, whatever is entered next from the Client is used to satisfy the prompt. Under the FTP Callable API, the application cannot issue an SCMD request other than the one expected, but it does have the option to issue GETL or TERM. If the request is TERM, the interface generates a QUIT subcommand which is accepted, terminates the connection with the Server, and stops the Client process.

Modernized DBCS/MBCS support in FTP

Enhanced DBCS and MBCS codepage support in FTP

- Enhance Multi-Byte Character Set (MBCS) - primarily support for Asian languages
- Current FTP support for Double Byte Character Set (DBCS) is based on an imbedded support in TCP/IP for selected conversions and is not ready for the latest z/OS character conversion technology
- z/OS V1R4 provides MBCS encoding support only for Chinese code standard GB18030
- z/OS V1R6 enhances MBCS to include the DBCS code pages currently supported by the existing old imbedded support
  - Some conversion parameters are not supported with the new method (they aren't standard)
- The new support is based on use of the standard FTP protocol (type ASCII) and use of SITE commands that are compatible with single byte (SBCS) conversion:
  - ENCODING SBCS/MBCS and
  - SB/MBDATACONN=(file_system_code_page , network_code_page)
- Original codepage support for DBCS using LOADDBCSTABLES is still supported, but we recommend moving to the new support if at all possible
- Objective is to make FTP independent of any specific code page - as long as the underlying z/OS conversion supports a code page conversion - so will FTP
- Currently FTP uses iconv() conversion services, but will eventually move to the Unicode Conversion Services

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.