This presentation covers FTP enhancements that are not related to security

FTP client API background

- The FTP client API was first introduced in V1R6
  - Provides better control over z/OS “programmed” FTP client operations
  - Allows better control over individual FTP client functions
  - Lets users and ISVs provide detailed monitoring of FTP client functions
- Support for new programming environments has been added over multiple releases
  - Assembler, REXX™, C/C++

z/OS Communications Server provides an FTP client API that can be used to programmatically control the z/OS FTP client. z/OS V1R6 provides an FTP client API for assembler that can be used by applications written in programming languages that support a callable interface. This FTP client API is suitable for applications written in languages such as assembler, PL/1 and Cobol. z/OS V1R7 extends the FTP client API to support the C and C++ programming languages. z/OS V1R8 further extends the FTP client API to support the REXX programming language.

The FTP client API provides improved control over existing options, such as using batch jobs or command lists. Using the FTP client API, an application can programmatically handle replies and errors received from both the FTP client and the FTP server.

FTP client API background

- Elements of the FTP client API
  - Stub program (EZAFTPKS) – link edited with the application or loaded dynamically and reused
  - Request handler (EZAFTPKI) - loaded by the stub program and reused
  - Created FTP client child process for each instance of the interface

The FTP client API has three main elements. It includes a stub program that is link-edited with the application or loaded dynamically and reused, a request handler, and a created FTP client. Each instance of the interface will create a new FTP client address space.

Why do a JAVA™ API?

- Without a client API, it is difficult to use FTP on z/OS from a Java application
  - There is limited existing Java FTP support
  - No support for all the features of z/OS
- Difficult for a Java application to programmatically control the z/OS FTP client
  - Must use the Java Native Interface (JNI) to communicate with the z/OS FTP client
  - JNI is a complicated interface

Controlling the z/OS FTP client from within a Java application is difficult. z/OS V1R7 added an FTP client API for C and C++, but a Java application must use the Java Native Interface (JNI) to use this API. JNI is a complicated, low-level API and is not intended for use by the typical Java programmer.

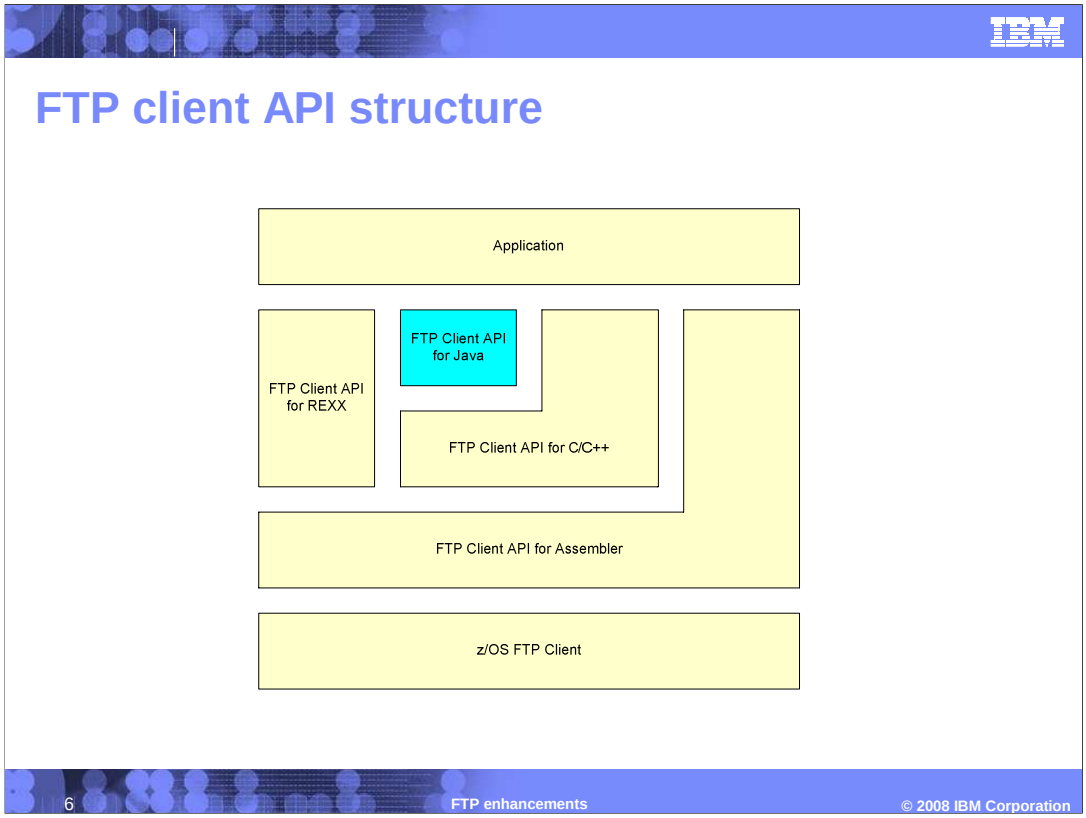
There are other options for using FTP from within a Java application without interfacing with the z/OS FTP client. The core Java programming language has limited support for FTP using URL syntax. Using this, an application can retrieve a file from a remote FTP server and receive the results into the application program's memory. There are also vendor Java FTP libraries available which allow applications to use FTP to retrieve files from or store files on a remote FTP server. None of these options support z/OS features, such as MVS datasets.

FTP client API for JAVA elements

- In V1R10, the FTP client API provides a Java interface
  - Includes a package of Java classes to control the z/OS FTP client
    - com.ibm.zos.net.ftp
  - Includes native JNI code to allow the Java classes to interface with the z/OS FTP client
- Also provided: a sample program and extensive documentation using Javadoc

To provide access to the z/OS FTP client from within a Java application, support for the Java programming language is added to the set of FTP client APIs that ship with z/OS Communications Server. The Java FTP client API consists of two parts. First, there is a package of Java classes that is used by a Java programmer to access the FTP client API. Second, there are native routines written in C++ that are invoked by the Java package using the Java Native Interface (JNI). Third, a sample program, ftpcapij.java, that uses the FTP client API for Java is provided.

The FTP client API for Java is documented using Javadoc. The Javadoc files consist of a set of HTML pages that describe the Java classes and the class methods. The Javadoc can be downloaded to a local workstation, the Javadoc files extracted, and viewed using any available Web browser. This is a new method of documentation for z/OS, but is standard for Java.



An application running on z/OS can use one of four flavors of the FTP client API.

The FTP client API for assembler provides a callable interface to send subcommands to the FTP client. The FTP client API for assembler sends the subcommands to a created z/OS FTP client.

The FTP client API for C/C++ allows applications written in either C or C++ to send subcommands to the FTP client. The FTP client API for C/C++ uses the FTP client API for assembler.

The FTP client API for REXX allows applications written in REXX to send subcommands to the FTP client. The FTP client API for REXX uses the FTP client API for assembler.

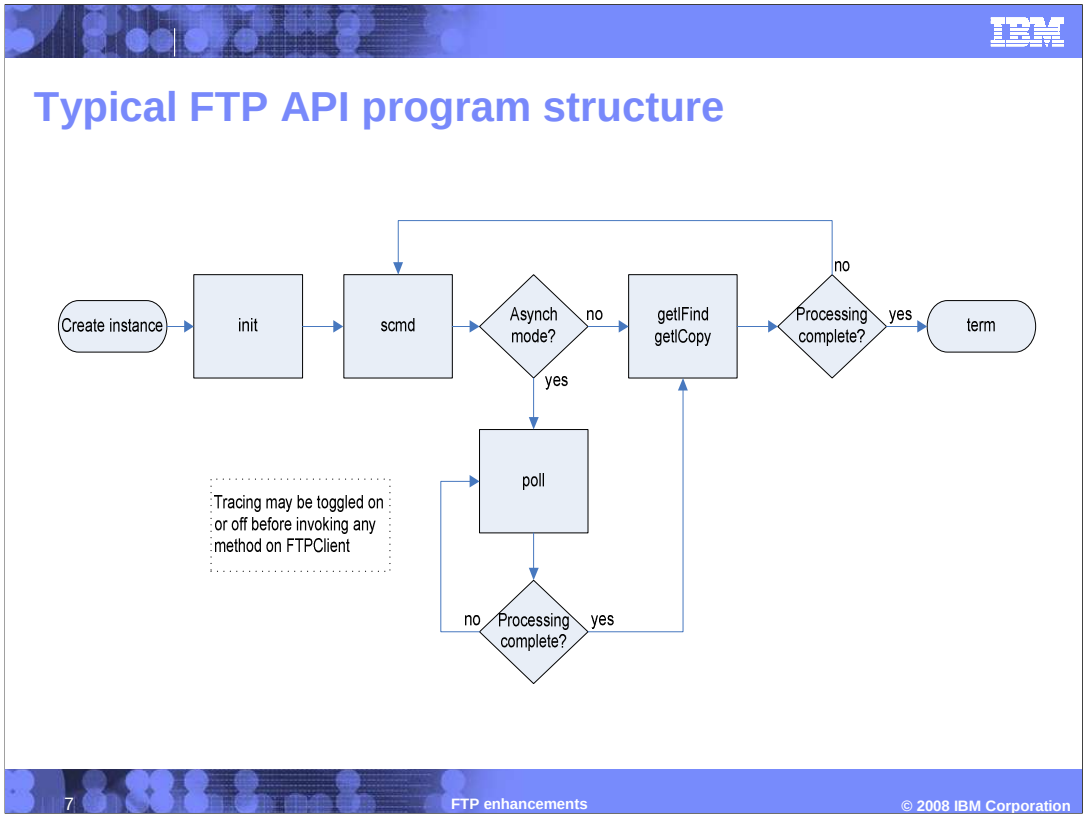
The FTP client API for Java allows applications written in Java to send subcommands to the FTP client. The FTP client API for Java uses the FTP client API for C/C++, which in turn uses the FTP client API for assembler.

The FTP client API for Java provides an interface to the z/OS FTP client. This interface enables a user program written in Java to send subcommands for the client to process. The interface also enables you program to retrieve output that includes the messages from the client, replies from the FTP server, and other data generated as the result of the request.

Each `FTPClient` object represents a single instance of the interface. A user program can create multiple instances of FTPClient. This allows a single user program to establish multiple simultaneous connections to the same FTP server or different FTP servers. Your program can multitask to different instances of the interface by requesting that the API not wait for the completion of an FTP subcommand before returning.

The z/OS FTP client that is used by the FTP client API is the standard z/OS FTP client. It is described in *File Transfer Protocol (FTP)* in *z/OS Communications Server: IP User's Guide and Commands File Transfer Protocol* in *z/OS Communications Server: IP Configuration Reference*. The z/OS FTP client, when started with the FTP client API for Java, operates essentially as it does when invoked in an interactive environment under the z/OS UNIX® shell. See *z/OS FTP client behavior when invoked from the FTP client API* in the *z/OS Communications Server: IP Programmer's Guide and Reference* for a description of the differences.

This package uses JNI to interface with the z/OS FTP client using the FTP client API for C. See the *z/OS Communications Server: IP Programmer's Guide and Reference* for more details on the FTP client API for C.



When using the FTP client API for Java, a Java program can create one or more instances of FTPClient. Each instance of FTPClient provides an interface to a single instance of a z/OS FTP client.

Once the FTPClient object has been created, the Java program must initialize the FTP client API interface. This is accomplished by invoking the init method on FTPClient. This call must be made before invoking scmd, poll, getlCopy, getlFind or term. The Java program can optionally pass an initialization string and 0-9 environment variables to the z/OS FTP client.

Once the interface is initialized, the Java program can begin to send subcommands to the FTP client. The Java program might choose to wait for the z/OS FTP client to complete processing the subcommand or for the z/OS FTP client to return immediately regardless of whether the subcommand has completed. If the Java program chooses not to wait for the z/OS FTP client to process the subcommand, the Java program must use poll to retrieve the results from the subcommand. You cannot start a new subcommand with this FTPClient until the subcommand is complete.

Once the subcommand is complete, the Java program can retrieve output that is returned by the FTP client. The FTP client API buffers all lines of output produced by the FTP client. The Java program can use getlFind to retrieve one line of output at a time, or can use getlCopy to retrieve all lines of output. The Java program can then process the line or lines.

Once the Java program finishes processing the output from the FTP client, it can then send another subcommand to the FTP client and repeat the processing above. Once the program is finished with the FTP session, the Java API should send a “QUIT” subcommand to the FTP client and then invoke the term method to terminate this instance of the interface. If the term invocation detects that a QUIT subcommand has not yet been issued to the client, it generates a QUIT subcommand. The generated QUIT subcommand request will be in the interface trace (if active) with (Generated by TERM) appended to the request record along with any results that are retrieved.

At any time, the Java program can enable or disable tracing using the traceIt method. This method results in the FTP client API for assembler trace being enabled or disabled for subsequent invocations of methods for this FTPClient. Before enabling the trace for the first time, the Java program can use setTraceID. This sets the identifier used in FTP client API for assembler trace records, and setTraceSClass to set the SYSOUT class for the FTP client API for assembler trace file.

FTP client API for Java documentation

- This function is documented in a Javadoc. Here's how to access it:
  - Download the Javadoc to your workstation
  - Extract the documentation using the Java jar utility
    - `jar –xvf EZAFTPdoc.jar`
    - Using the jar utility requires that you have Java installed on your workstation
  - Use a Web browser to view
    - Open the extracted file `index.html` with a Web browser

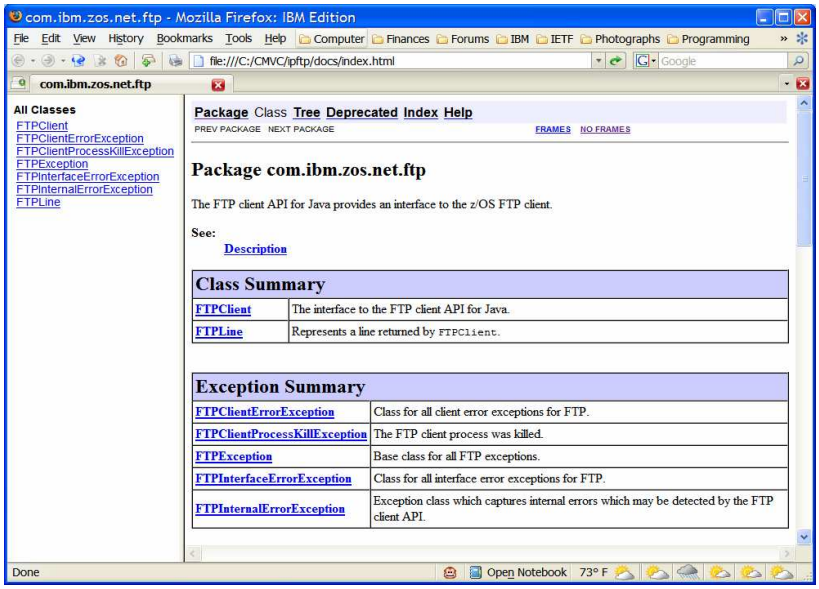
The FTP client API for Java is documented using Javadoc. The Javadoc for the FTP client API for Java provides documentation for the FTP client API for Java package, and the classes and exceptions found within the package. All public, external constants, constructors, and methods are documented. Constructors and methods contains a description of the constructor or method, all parameters, return values, and exceptions that might be thrown. All of the documentation contains appropriate hyperlinks to other sections of the documentation.

The Javadoc for the FTP client API package is stored in /usr/include/java_classes/EZAFTPdoc.jar. To view the documentation, copy it from z/OS to your workstation. The file should be transferred as a binary file, and can be retrieved using FTP or another program.

Once the Javadoc has been downloaded, it must be extracted from the JAR file. Use the jar utility that ships with Java to extract the contents; this might require you to download and install Java on the workstation if it is not already installed. Although not required, it is recommended you place the output files into a separate directory or folder.

Once the Javadoc has been extracted from the JAR file, use a Web browser to open the extracted file index.html. This file will be placed in the root directory or folder into which you extracted the Javadoc.

Example Javadoc screen capture



This slide shows an example screen capture of the FTP client API Javadoc. As you can see, it is viewed in a browser just like any other hyperlinked Web pages and provides links to detailed information on provided classes, exceptions, and other provided Java objects.

FTP dataset contention background

- FTP PUTs to an MVS dataset
  - Server requires exclusive access
  - Client requires shared access
- FTP GETs to an MVS dataset
  - Server requires shared access
  - Client requires exclusive access

Whenever a FTP PUT or GET request is processed, FTP must obtain exclusive or shared access to the required data sets.

For each request, two resources are needed: one for the FTP client and the other for the FTP server.

FTP dataset contention background

- If the FTP Get or Put subcommand cannot obtain appropriate access to the MVS data set, the request fails
  - In V1R9 and earlier, you have no idea what job or process is holding the MVS data set
  - JOB must be resubmitted for processing

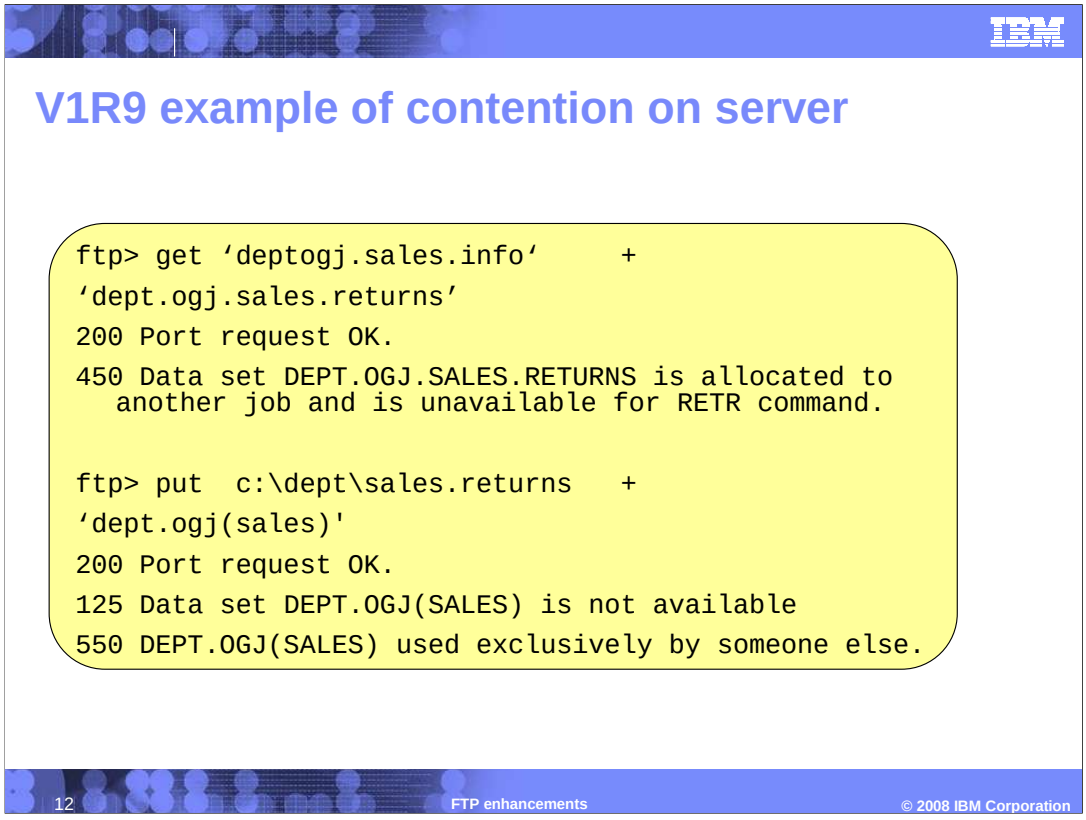
If either resource cannot be obtained, the FTP request fails.

When this occurs, in a V1R9 or early z/OS FTP, you have no idea who is using the data set when the failure occurred.

If the failure occurred in a batch job, you might not know for some time that it failed. This can become a problem if time sensitive information needs to be transferred between sites.

Upon discovering the job failed, the job must be resubmitted again. Delaying the transfer of data can have a business impact.

If you know what job prevented the transfer, you can improve scheduling to avoid conflicts.



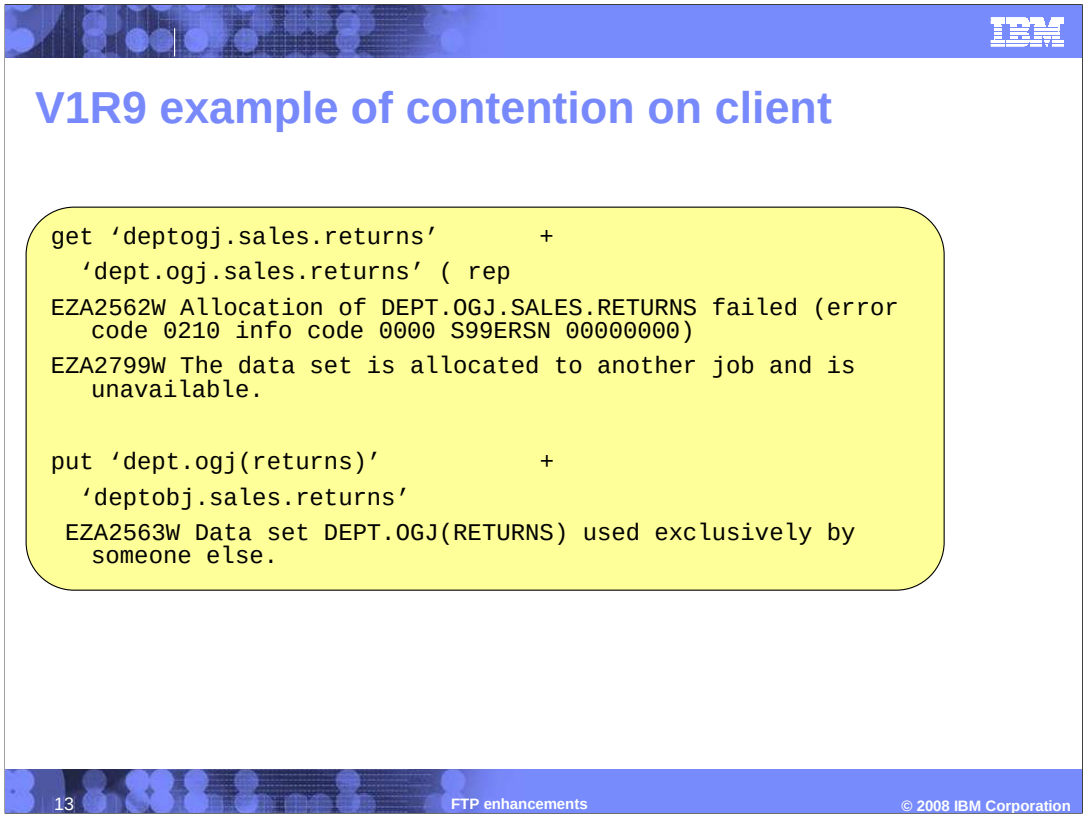
This slide shows an example of V1R9 FTP get and put subcommands failing because the MVS data set is held by another job on the server.

In the first example, the get subcommand fails because the FTP server is not able to obtain shared access to the MVS sequential file ‘deptogj.sales.info’. Another job on the FTP server site has exclusive access to this MVS data set.

In the second example, the put subcommand fails because the FTP server requires exclusive access to the member ‘sales’ in the MVS partitioned data set ‘dept.ogj’. Another job on the FTP server site has exclusive access to this member.

In both cases, FTP is only able to display a notification as to why the request failed, but not what job prevented FTP from transferring the data.

The display lines that begin with a 3-digit numeral are replies sent from the server to the client; the 3-digit prefix is called a reply code. The 450, 125 and 500 reply codes indicate a problem that the server has in satisfying the request.



This slide shows an example of V1R9 FTP get and put subcommands failing because another job is holding the MVS data set on FTP client system.

With the get subcommand, a job is currently accessing the MVS sequential file 'deptogj.sales.returns'. With the put subcommand, the 'returns' member in the 'dept.ogj' partitioned data set is being held by another job.

Because there are no reply codes associated with these messages, the failure is occurring on the z/OS system that the client is running on.

Like the previous example, FTP informs you why the FTP subcommand fails but does not provide any meaningful information about which job prevented the transfer.

Report holder of an MVS data set

- z/OS FTP will now search z/OS QUEUES for jobs accessing an MVS data set
  - Provide the job names that are accessing the MVS data set and preventing FTP access
  - Provide the type of access
    - Shared or Exclusive
    - Access that FTP needs
- Support provided for MVS datasets only
  - Not for z/OS UNIX file system files

In V1R10, z/OS FTP reports the holder of an MVS data set when it is held by another job. FTP can obtain the job names holding this MVS data set by searching a few queues.

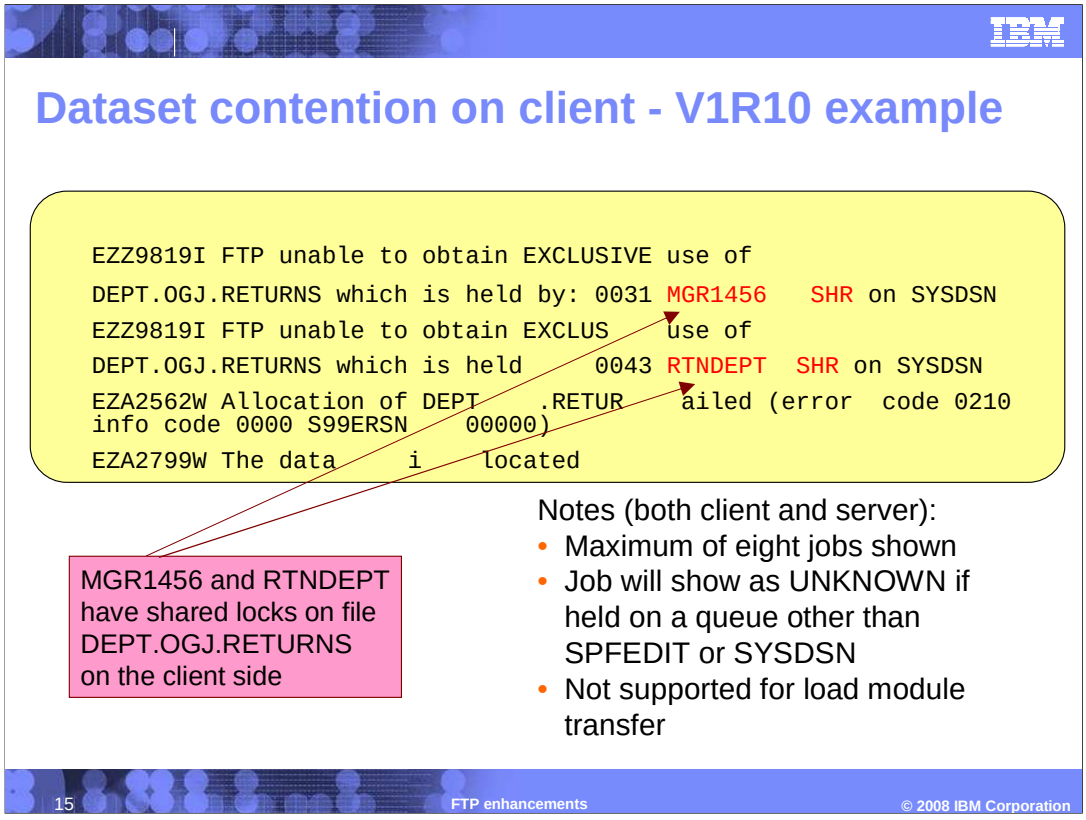
There are over 70 queues used by z/OS to serialize access to MVS data sets which can result in FTP not acquiring a data set. FTP focuses on two main queues: SPFEDIT and SYSDSN.

The SPFEDIT queue is used by z/OS to control who is holding a member of a PDS or PDSE. This queue is used by FTP and TSO edit to place a hold on the member of the PDS when they will be updating it. When FTP cannot obtain a hold on a member, the SPFEDIT queue is searched to locate the job holding the member.

The z/OS SYSDSN queue is used to serialize access to an MVS sequential data set. This is done by z/OS whenever an application accesses an MVS sequential data set. FTP searches this queue whenever it cannot obtain an MVS data set.

If FTP detects jobs on these queues which are holding the resource, it provides information on up to eight jobs that are found. It also provides the type of access FTP requires and the type of access the jobs holding the MVS data set have.

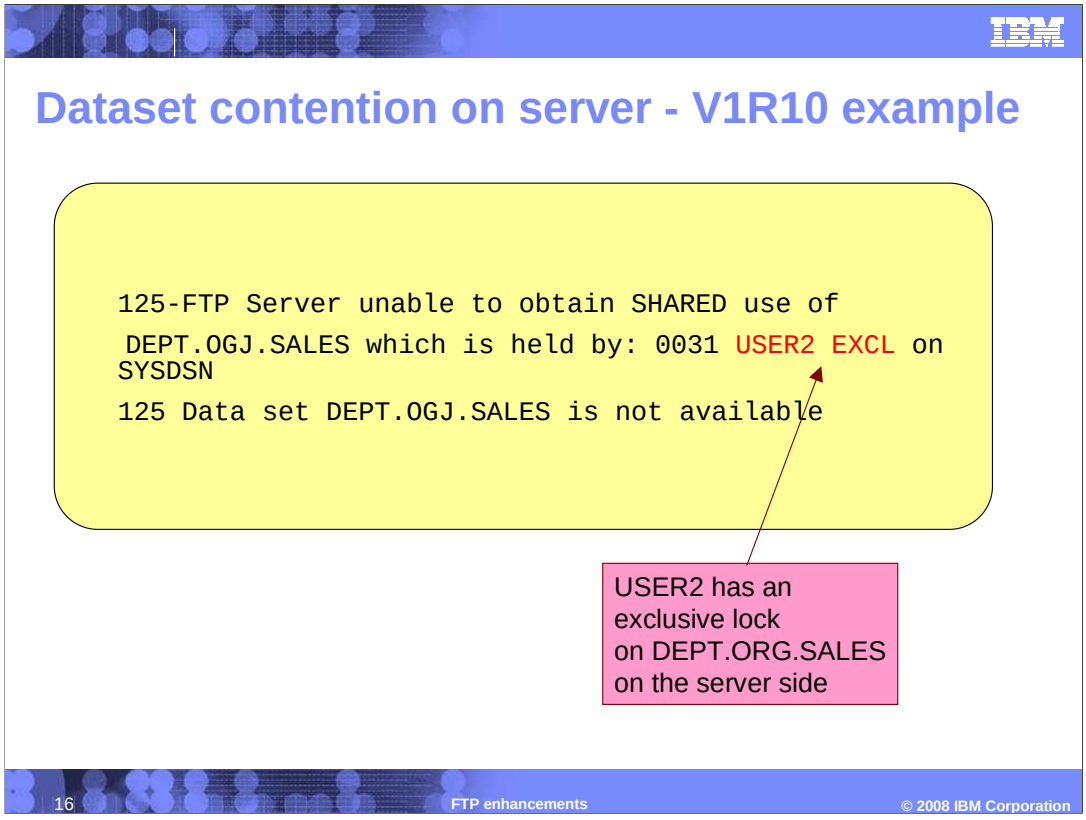
Now, whenever FTP cannot obtain an MVS data set, a message or reply is issued containing information that identifies who is holding the MVS data set. The message also identifies the access that the existing holder has, and the access that FTP needs.



This slide shows an example of messages from a client if an MVS data set cannot be obtained.

In this example the FTP client could not acquire the MVS data set because several jobs were holding shared locks on the data set on the FTP client's z/OS system.

For both the client and the server, a maximum of eight jobs are shown and a job will show as UNKNOWN If held on a queue other than SPFEDIT or SYSDSN. When a load module transfer fails because a job is already holding a PDS or PDSE member, FTP is not able to determine the holder of the member. Load module transfers are accomplished by calling the IEBCOPY utility.



This slide shows an example of replies from the server if an MVS data set cannot be obtained.

In this example, only one job was holding the MVS data set on the server with an exclusive lock. The job and the fact that the lock is exclusive are provide don the 125 reply from the FTP server.

Even better: wait for availability of MVS data set

- In V1R10, z/OS FTP also provides capability to re-access the failing MVS data set before terminating the request.
  - for a configurable period of time
- By automatically retrying, FTP can prevent batch jobs from failing
  - because there is no user around to resubmit the job

When V1R9 FTP cannot access an MVS data set because another job already is holding it, FTP fails the request. Now FTP can report the job that was holding the data set, but the job will still need to be rerun when there is no conflict. This can be a problem, especially for batch jobs when there is no user around to resubmit the job.

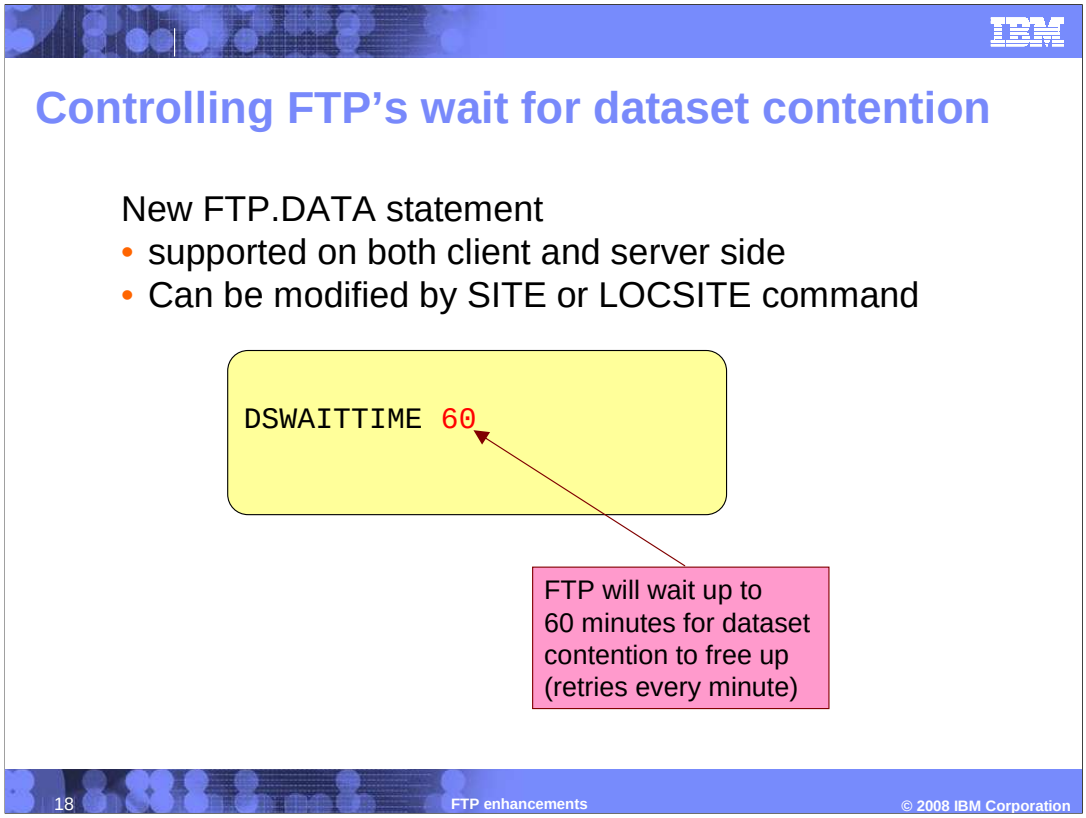
The solution to avoid rescheduling is to retry accessing the resource until it becomes available.

You can use the FTP client API or REXX clist to do this, but this requires additional effort and programming skills which you might not have.

To resolve this situation, FTP now provides the capability to re-access the MVS data set when a failure occurs because some other job is holding the MVS data set.

If either the FTP server or client cannot obtain the MVS data set, attempts to re-access it will be made at one minute intervals, up to a specified limit.

Ability to set a default in the FTP.DATA file and by the SITE and LOCSITE subcommands enables you to customize the amount of time that FTP takes to re-access the MVS data set.



The DSWAITTIME statement can be coded in the FTP.DATA configuration file. This statement has been added to both the EZAFTPAC sample for the FTP client and the EZAFTPAS sample for the FTP server.

It specifies how long FTP should try to obtain access to an MVS data set.

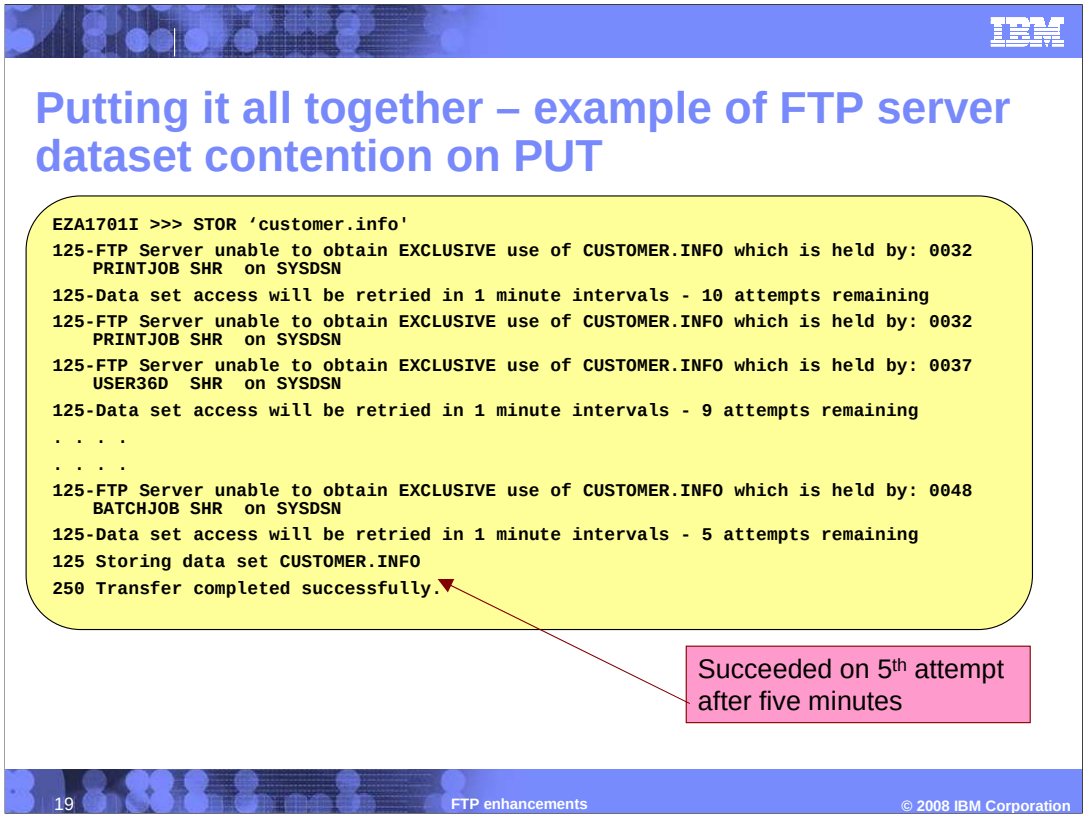
Values range between 0 and 14400 minutes.

FTP will attempt to acquire an MVS data set every minute. Therefore the DSWAITTIME value also specifies the number of times that FTP attempts to acquire the MVS data set.

For consistency with earlier releases, the default interval is 0. This means that if an MVS data set connection fails because the data set is already in use, the FTP request is terminated.

The FTP client can issue the SITE subcommand to modify the DSWAITTIME value at the FTP server.

The LOCSITE subcommand can be issued from a z/OS FTP client to modify the DSWAITTIME for the amount of time that the client should use in retrying to obtain a local MVS data set.



This slide shows the information that is displayed when FTP tries to obtain an MVS data set.

After the server has received the STOR command, FTP attempts to obtain the MVS data set exclusively, but the job PRINTJOB is already holding this MVS data set in SHR mode. Information is sent back to the server with the 125- reply code along with an additional 125- reply code indicating that FTP will retry in one minute intervals, and that 10 attempts remain. The “initial“ *attempts remaining* number is the setting of the DSWAITTIME value. The dash following the 125 reply code indicates that there is another 125 reply code to follow.

When FTP tries again, it locates two jobs (PRINTJOB and USER36) preventing it from obtaining the MVS data set. It replies with the 125- reply code for the jobs that have the MVS data set and the number of attempts remaining. The attempts remaining is a countdown timer. FTP must be able to obtain the MVS data set before it reaches 0, or the request fails.

In this example, FTP is able to successfully obtain the MVS data set after five attempts or with five minutes remaining to obtain the MVS data set. It ends with a 125 reply code with no dash to end the 125 reply, and then follows with the “250 Transfer completed successfully” reply.

DSWAITTIME restrictions

- DSWAITTIME is supported for
  - Client: GET and MGET / Server: RETR
  - Client: PUT and MPUT / Server: STOR
- DSWAITTIME not supported for:
  - PDS/PDSE with RECFM=U (load modules)
  - Client: APPEND, DELETE, MDELETE,RENAME,SUNIQUE
  - Server: APPE, DELE, STOU

Only a subset of FTP subcommands for GET and PUT are supported for DSWAITTIME, as listed on this slide.

DSWAITTIME is not supported when a partitioned data set has record format undefined. The partitioned data set is treated as a load library and IEBCOPY is used to process the request.

In addition there are subcommands for the client and commands for the server that do not support DSWAITTIME. For client contention, it is not supported for APPEND, DELETE, MDELETE, RENAME, or SUNIQUE. For server contention, this function is not supported for APPE, DELE or STOU (store unique).

Application data for FTP sockets

- Starting in V1R10, z/OS FTP will set application data for each FTP socket
  - Associate a text string with each FTP socket
  - Provide user ID and other fields of interest in the application data string
- Application data available to:
  - NMIs EZBNMIFR and SYSTCPCN
  - NETSTAT reports
  - SMF Type 119 connection termination records

In v1r9, z/OS Communications Server introduced the ability to associate application data for TCP sockets. Application data is a 40-byte string that your application can create and associate with a socket. You can display the socket application data with the Netstat command; you can inspect APPLDATA in the SMF type 119 connection termination record, and you can filter sockets by application data from an NMI application.

By setting application data for FTP sockets, the FTP information that network monitoring applications use is available not only to NMIs, but to Netstat output and to SMF type 119 connection termination records.

FTP server application data

Offset	Description
1 – 8	**The string “EZAFTP0S”**
9	**Blank**
10	**C for a control connection socket** **D for a data connection socket**
11	**Blank**
12-20	**User ID used to log into FTP**
21	**Blank**
22	**Security protection** **C for Clear; S for Safe; P for private;** **L for Clear but previously safe or private**
…	**More fields (see *IP Configuration Reference* for details)**

This figure shows the first fields of the FTP server APPLDATA. This is an example of FTP server application data. The FTP client and daemon provide similar data.

The string “EZAFTP0S” is the first field, and identifies this socket as an FTP server socket.

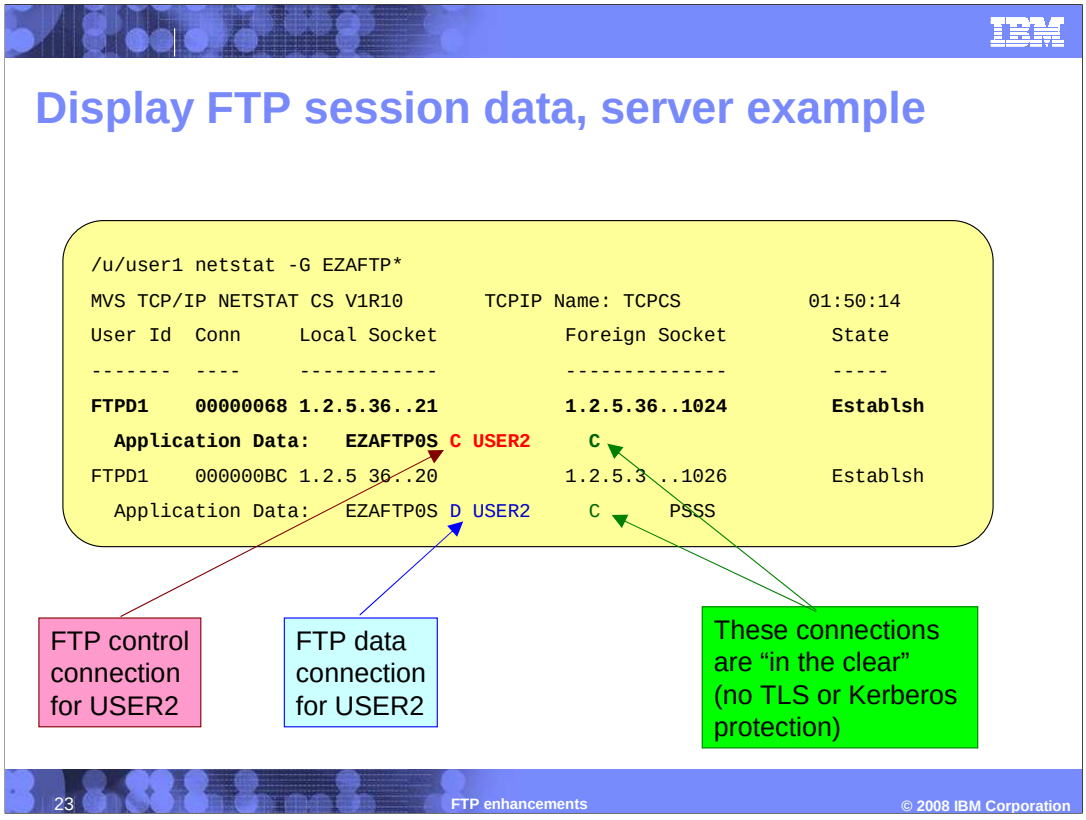
The next nonblank field is either C or D, to distinguish control connection sockets from data connection sockets. The control connection is the connection used to log into FTP, and is active for the entire session. For each file transfer that occurs, FTP establishes a temporary connection called the data connection that lasts as long as the file transfer lasts.

The next non blank field is the user ID used to log into FTP.

The next field identifies the level of security used on the connection. ‘Clear’ means data flows on the connection with no encryption or integrity checking; S and P imply that either TLS or Kerberos is used to protect the privacy or integrity of the data on this socket.

This data is meant to be exploited by network management applications and can also be displayed using netstat.

A full list of the fields provided can be seen in the IP Configuration Reference.



This example demonstrates using the Netstat report to display application data associated with FTP server sockets. This was done using the USS netstat command with the –G parameter to indicate you want to display only those sockets with application data that begins with the string “EZAFTP”. Some lines were removed from the report to keep the example on one page.

In this example, a user logged into ftp as USER2, and transferred a data set. You can see application data for the server, for the control connections and data connection. The data connection lasts only as long as the file transfer, so you might not often capture application data for FTP data connection socket as was done here.

You can deduce that the two lines in this display belong to the FTP server from the local port numbers and job name, but now you can note the application ID is set to EZAFTP0S in each case.

Note the C or D between preceding the string ‘USER2’; this identifies each of these sockets as control or data connection sockets. The C which follows the string ‘USER2’ corresponds to offset 22 in the FTP server application data format and indicates ‘Clear’: a connection that is not secured with TLS or Kerberos.

Keep-alive on data connection background

- Network devices cancel idle sessions
- Keep-alive packets sent periodically keep connection active
- FTP Control Connection already provides keep-alive support
- FTP Data Connection is exposed to being canceled if connection stays idle too long
  - Keep-alive interval on TCPCONFIG statement might not be active, or the interval might exceed a network device's cancellation timer

Any TCP/IP connection is subject to monitoring by the network. The session might be canceled if no activity on the session is detected within a defined period of time as determined by the network device. Cancelling the session prevents any further communication between the session partners. In cancelling the session, the session partners might not be notified of this cancellation which can result in a hung session if the session partners do not provide for this situation.

To prevent cancellation, TCP/IP sends keep-alive packets. A keep-alive packet contains one byte of data and uses a sequence number of a packet that was already sent. The remote session partner discards the data packet because it has already received the packet. The benefit is that any device monitoring the session will detect that the session is active.

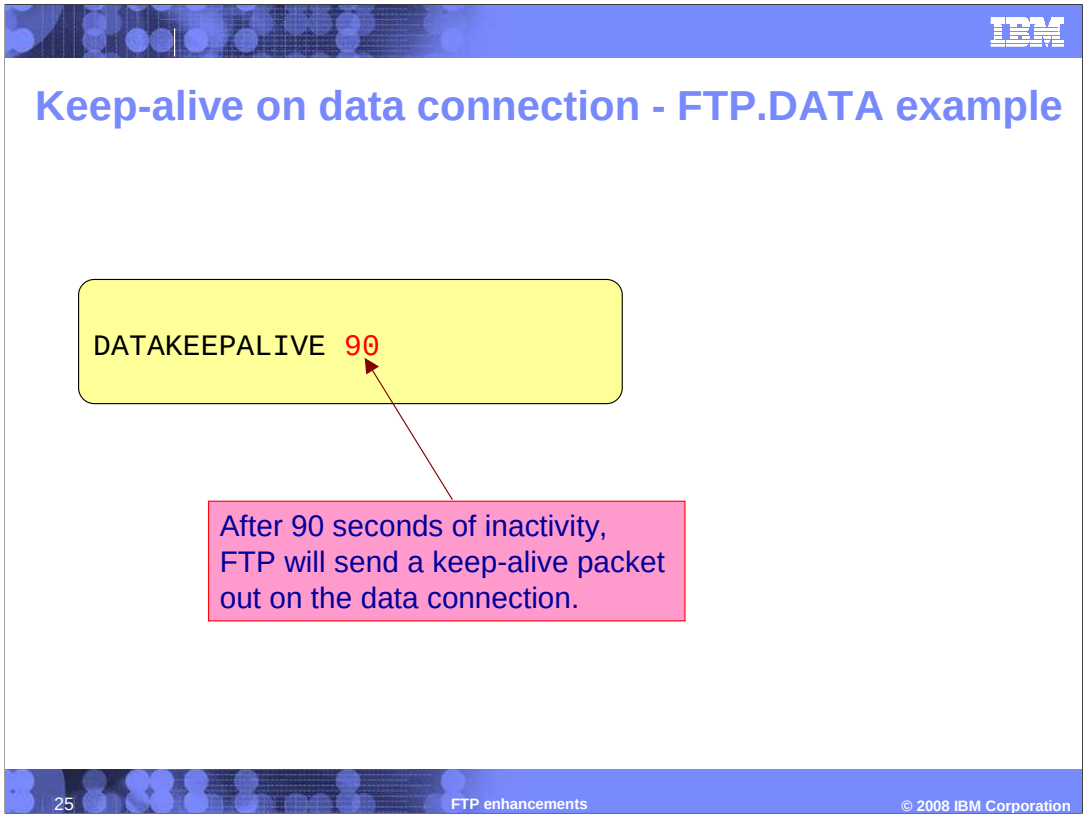
The FTP control connection is the connection over which FTP commands are sent from the client to the server. The keep-alive interval can be customized by the KEEPALIVE statement in the FTP.DATA configuration file instead of utilizing the TCP/IP stack's keep-alive interval.

The FTP data connection, over which file data flows during a file transfer between the client and server, does not support any customization of the keep-alive interval in V1R9. This session is susceptible to being cancelled if the connection stays idle too long. A long running DB/2 query or a job submitted to JES that has not completed can cause this to occur.

While the TCP/IP stack provides the ability to configure when keep-alive packets are generated, this interval might exceed that needed by FTP.

Without the ability to customize a keep-alive interval on the FTP data connection, FTP must rely on the keep-alive interval defined to the stack. The FTP connection might be monitored by a device whose cancellation timer is less than the stack keep-alive timer.

Each network has its specific needs and a single stack keep-alive interval might not be able to cover all of these networks.



The DATAKEEPALIVE statement can be coded in the FTP.DATA configuration file to cause FTP to initiate a keep-alive on the data connection after the configured period of inactivity. This statement is added to both the EZAFTPAC sample for the FTP client and the EZAFTPAS sample for the FTP server.

Initially, this statement is commented out in both samples and is defined with a value of 0. This is the default value and means FTP relies on the stack interval for keep-alives.

If the DATAKEEPALIVE value is specified, the FTP keep-alive interval is used for the data connection, overriding the TCP/IP stack interval.

The DATAKEEPALIVE value is specified in seconds and ranges from 60 to 86400 seconds which is equivalent to 24 hours.

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 REXX z/OS

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

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

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 or changes in the products or programs 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 (for example, 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.