



## **FileTek FTP Data Unloader Manual**

StorHouse/RM Release 3.4

Publication Number  
900137 Rev. F

September 18, 2008





All rights reserved. No part of this publication may be reproduced, translated, stored in any electronic retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of FileTek, Inc.

Copyright © 1999-2008 FileTek, Inc. As an Unpublished Licensed Work.  
Publication Number: 900137 Rev. F

#### NOTICE: U.S. GOVERNMENT USERS

This notice applies to all acquisitions of this work by or for the U.S. Government ("Government"), or by any prime contractor or subcontractor (at any tier) under any contract, cooperative agreement or other activity with the Government. By accepting delivery of this work, the Government agrees that this work and the Licensed Program(s) described herein qualify as "commercial" computer software within the meaning of the acquisition regulation(s) applicable to this procurement. The terms of conditions of the license for the Licensed Program(s) shall pertain to the Government's use and disclosure of this work and the Licensed Program(s), and shall supersede any conflicting contractual terms or conditions. If the license for this work and the Licensed Program(s) fails to meet the Government's need or is inconsistent in any respect with Federal law, the Government agrees to return this work and the Licensed Program(s), unused, to FileTek, Inc. The following additional statement applies only to acquisitions governed by DFARS Subpart 227.4 (October 1988) "Restricted Rights - Use, duplication and disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 (OCT. 1988)." Unpublished licensed work property of FileTek, Inc. Unauthorized use, duplication or distribution prohibited. All rights reserved. A copyright notice on this work and/or on the Licensed Program(s) by itself does not constitute publication or public disclosure of this work or the Licensed Program(s). The contractor/manufacturer is:

FileTek, Inc.  
9400 Key West Avenue  
Rockville, Maryland 20850

Information in this document is subject to change without notice and does not represent a commitment on the part of FileTek, Inc. Further, FileTek, Inc. reserves the right to supplement the document with information not available at the time of creation of the document. FILETEK, INC. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, AND CANNOT WARRANT THE RESULTS YOU MAY OBTAIN USING THE DOCUMENT. IN NO EVENT SHALL FILETEK, INC. BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OR DATA, INTERRUPTION OF BUSINESS, OR FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, EVEN IF FILETEK, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES ARISING FROM ANY DEFECT OR ERROR IN THIS PUBLICATION. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

FileTek and StorHouse are registered U.S. trademarks of FileTek, Inc. VRAM is a U.S. trademark of FileTek, Inc. All other brand or product names are trademarks or registered trademarks of their respective owners.

Documentation for FileTek's StorHouse product. Protected by the following U.S. Patents: 4,864,572; 5,247,660; 5,727,197; 6,049,804.

# Contents

**Welcome ..... ix**

- The StorHouse family of products ..... ix
  - StorHouse/SM ..... ix
  - StorHouse/RM ..... x
  - StorHouse/Control Center ..... x
- Purpose of this document ..... xi
- Audience ..... xi
- What’s in this guide ..... xi
- For more information ..... xii
- For quick reference ..... xii
- Conventions ..... xiii

**Chapter 1: The basics ..... 1-1**

- What’s an unload? ..... 1-1
- How the Unloader works with FTP ..... 1-2
  - An FTP session ..... 1-3
  - Transfer types ..... 1-4
- What you need ..... 1-4
  - Your client FTP tool ..... 1-4
  - StorHouse account information ..... 1-5
- The unload process ..... 1-6

If an unload fails .....	1-6
Replies .....	1-7
Symbolic names in replies .....	1-7
Numeric SQL codes in replies .....	1-7
Limits .....	1-8
Locks .....	1-8
Result data .....	1-8
Data records and data fields .....	1-9
Data record formats .....	1-9
Text data .....	1-10
Fixed-length data .....	1-10
Variable-length data .....	1-11
Considerations for binary result data .....	1-11
Considerations for VAR-type result data .....	1-12
Considerations for LOB data .....	1-13
Unloading LOB data with the other result data .....	1-13
Creating LOB files on your client computer .....	1-14
Creating LOB files on a remote system .....	1-15

## **Chapter 2: Preparing an UNLOAD statement .....2-1**

About the control file .....	2-1
Control file conventions .....	2-2
Format conventions .....	2-2
Format of UNLOAD .....	2-3
Specifying the character set of result data .....	2-6
Format of the CHARACTERSET clause .....	2-7
Example CHARACTERSET clause .....	2-7
Unloading data to a StorHouse VRAM file .....	2-8
Format of the OUTFILE clause .....	2-8
Example OUTFILE clause .....	2-8
Formatting result data .....	2-9

Guidelines for delimiting data fields .....	2-10
Format of the FIELDS clause .....	2-13
Example FIELDS clauses .....	2-15
Identifying an escape character .....	2-16
Format of ESCAPED BY clause .....	2-17
Example ESCAPED BY clause .....	2-17
Appending a character to the end of result records .....	2-17
Format of the RECORDS clause .....	2-18
Example RECORDS clauses .....	2-18
Suppressing the last field terminator .....	2-19
Describing each data field in result records .....	2-19
Format of the USING clause .....	2-19
Omitting the USING clause .....	2-20
Guidelines for specifying a USING clause .....	2-21
Inserting text (constants) into result records .....	2-21
Specifying the data type of a data field .....	2-22
Converting data types .....	2-38
Specifying a character set for an individual data field .....	2-40
Specifying a delimiter for an individual data field .....	2-41
Specifying a remote host name for LOB data .....	2-42
Identifying the path to the LOB directory .....	2-43
Specifying a file name for LOB files .....	2-43
Overwriting LOB files .....	2-45
Specifying a user name and password .....	2-46
Specifying a BLOB or CLOB data type .....	2-46
Specifying the position of a data field in a result record .....	2-47
Storing a value for NULL data .....	2-48
Example USING clause .....	2-51
Selecting the StorHouse data to unload .....	2-52
Format of the SELECT statement .....	2-52
Example SELECT statement .....	2-52
Example UNLOAD statements .....	2-53
Creating fixed-length fields and records .....	2-54
Terminating data fields and records, adding keywords next to each data field .....	2-55

Creating INSERT statements with NULL keywords for NULL data .....	2-56
Creating fixed-length data fields with NULL flags at the start of records .....	2-57
Generating data fields with NULL flags .....	2-58
Adding keywords but omitting them for NULL data .....	2-58
Unloading LOB data to the result file .....	2-59

## **Chapter 3: Using FTP to unload data .....3-1**

FTP commands for unloading data .....	3-2
The put command .....	3-3
The get command .....	3-4
Guidelines for specifying put and get commands .....	3-5
Sample session .....	3-6
Secure FTP (FTPS) .....	3-7
Compression .....	3-8
GUI and client considerations .....	3-8
Opening a session at an alternate port .....	3-8
Specifying keywords for a remote file name .....	3-9
Listing the remote directory .....	3-9
Using automatic binary mode .....	3-9
Displaying remote server messages .....	3-10
Globbing remote file names .....	3-10
Avoiding data timeouts in clients .....	3-11
Logging into the StorHouse FTP server .....	3-12
Setting the transfer type .....	3-13
Transferring the control file .....	3-14
Getting the result data .....	3-15
Displaying help for StorHouse FTP server commands .....	3-17
Logging off the StorHouse FTP server .....	3-18



**Appendix A: University of California copyright, conditions, disclaimer ..... A-1**

**Index ..... Index-1**



**Contents**

---



# Welcome

The *FileTek® FTP Data Unloader* is a tool for extracting data from StorHouse® database tables. It does not remove the data but rather executes a SELECT statement, then formats and transfers the result data to a file on your computer. With the FileTek FTP Data Unloader, you use your standard File Transfer Protocol (FTP) client software to communicate with the FileTek customized FTP server on StorHouse.

*FTP* is an established standard for transferring files between two computers connected by a network. The FileTek FTP Data Unloader conforms to the standards defined by the protocol.

## The StorHouse family of products

*StorHouse* is the FileTek enterprise-wide solution for managing the capture, storage, movement, and access of gigabytes to petabytes of relational and non-relational detail data. StorHouse technology combines industry-leading, scalable storage devices and Open System processors with the FileTek specialized storage management and relational database management system (RDBMS) software components.

### **StorHouse/SM**

*StorHouse/SM*, the storage management component, controls a hierarchy of storage devices including cache, redundant array of independent disks (RAID), Advanced Technology Attached (ATA) disks, massive array of idle disks (MAID),



## Welcome

---

The StorHouse family of products

erasable and write-once-read-many (WORM) optical disk jukeboxes, and erasable and WORM automated tape libraries. StorHouse/SM is also responsible for automating critical system management tasks, like data migration, backup, and recovery.

## StorHouse/RM

*StorHouse/RM*, the RDBMS component, works in conjunction with StorHouse/SM to store and access relational data. StorHouse/RM provides row-level SQL access to high volumes of detail data on any layer in the StorHouse storage hierarchy, including tape. SQL access is available from different platforms through a variety of industry-standard protocols. StorHouse/RM runs on Sun™ Solaris™, Hewlett-Packard HP-UX, and IBM® AIX platforms.

## StorHouse/Control Center

*StorHouse/Control Center* (CC) is the FileTek Windows®-based network computing system for providing administrative control of the StorHouse family of products. StorHouse/Control Center works with StorHouse/SM release 4.2 and higher and consists of one or more StorHouse/Control Center servers that communicate with StorHouse/Control Center clients over a TCP/IP network. The *StorHouse/Control Center server*, which runs on Windows NT, XP Pro, and 2000 platforms, provides network connectivity to StorHouse. The *StorHouse/Control Center clients*, which run on Windows 95, 98, 2000, XP Pro, and NT platforms, consist of one or more graphical user interface (GUI) modules for performing StorHouse system and database administration tasks, configuring and managing StorHouse/Control Center servers, and analyzing and monitoring StorHouse activity and performance.

## Purpose of this document

The *FileTek FTP Data Unloader Manual* explains how to unload data from StorHouse databases using FTP. It describes the UNLOAD control statement you prepare to format result data, the SELECT statement you prepare to select the data to unload, and the subset of FTP commands you use to transfer control information and to receive result data.

## Audience

If you're responsible for unloading data from StorHouse, then this manual is for you. The *FileTek FTP Data Unloader Manual* assumes you understand FTP commands and protocol, Structured Query Language (SQL), StorHouse database fundamentals, and your host operating system.

## What's in this guide

This document is organized as follows:

- Chapter 1, "The Basics," describes the features and components of the FileTek FTP Data Unloader. It also defines concepts used throughout the manual.
- Chapter 2, "Preparing an UNLOAD statement," describes the control statement you include in a control file for each unload.
- Chapter 3, "Using FTP to unload data," describes the subset of FTP commands you use to start an FTP session and to unload data.



## Welcome

For more information

---

## For more information

The following publications contain information related to the FileTek FTP Data Unloader.

- For complete information about FTP, refer to the *File Transfer Protocol Request for Comments* (RFC) 959 publication, available on the Internet.
- For explanations of StorHouse messages that you will see when running the FileTek FTP Data Unloader, refer to the *StorHouse Messages and Codes Manual*, publication number 900032.
- To learn how to load data into StorHouse user tables by using FTP, refer to the *FileTek FTP Data Loader Manual*, publication number 900115.
- To learn the basics of StorHouse/RM, refer to *StorHouse/RM Concepts*, publication number 900132.
- To perform StorHouse database administration tasks like creating user tables and indexes, managing accounts and privileges, and defining user tablespaces, refer to the *StorHouse Database Administration Guide*, publication number 900108.
- To learn the concepts, structures, and functions of StorHouse, refer to the *StorHouse Concepts and Facilities Manual*, publication number 900026.

## For quick reference

This manual contains complete descriptions of the UNLOAD statement and the FTP commands used to unload data. Refer to the *StorHouse/RM SQL and Utility Quick Reference*, publication number 900122, for just the syntax of the FTP commands and the UNLOAD statement.

## Conventions

This book uses the following conventions:

Convention	Meaning
Helvetica font	Statement formats and examples, FTP commands and replies
Courier font	Result data examples
<i>b</i>	Blank character in result data
<i>Italics</i>	New terms, emphasized text, and publication titles
▼	Procedures

See page 2-2 for format conventions of the UNLOAD statement.



## **Welcome**

Conventions

---

# The basics

This chapter contains general information about the FileTek FTP Data Unloader. It explains:

- What an unload is
- How the Unloader works with FTP
- What you need to unload data
- The unload process
- What to do if an unload fails
- The types of replies
- Limits and locks for unloads
- What you should know about result data

## What's an unload?

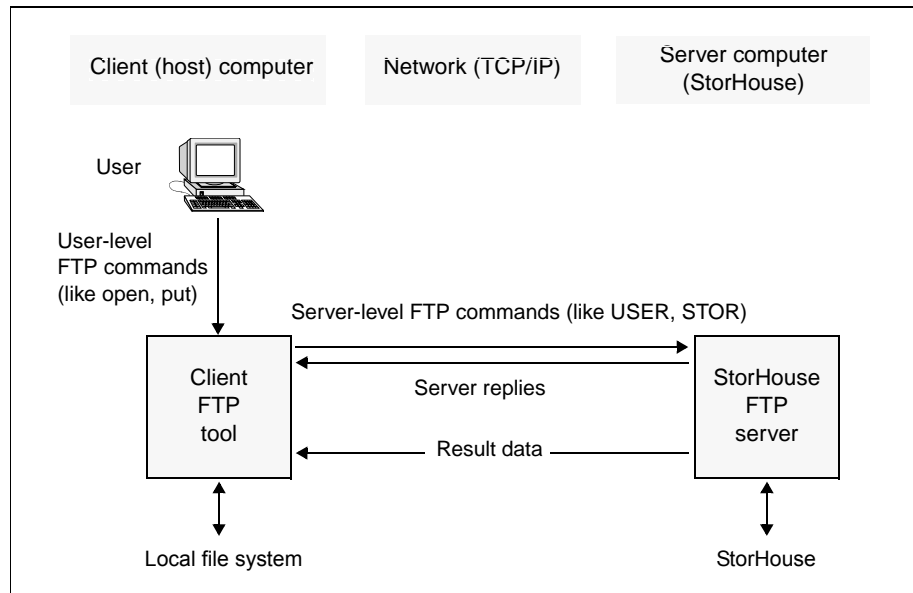
When you *unload* data, you select data from a StorHouse database and receive the result data in a sequential (or flat) file on your computer or in a Virtual Record Access Manager (VRAM™) file on StorHouse. You can receive large object (LOB) values in the result file or in separate files on a local or remote host. Depending on your host environment, you can pipe the output to another program such as a data loading utility. You can unload an entire table, specific columns or rows of a table, or even multiple tables (join) by issuing a SELECT statement.

**Note:** An unload does not remove data from StorHouse but rather executes a query, formatting the output as desired by a user-supplied control statement.

## How the Unloader works with FTP

FTP is layered on top of Transmission Control Protocol (TCP). Two programs communicate over a TCP/IP connection to transfer files between the local file system on your host and a remote file system on StorHouse. These programs are:

- *Client FTP software* (or tool) – interacts with you and your local file system. It sends server-level FTP commands and your control file to the StorHouse FTP server.
- *StorHouse FTP server* – interacts with the remote file system (StorHouse database tables), replies to your client's FTP commands, invokes a StorHouse process to execute the unload query, and sends the result data to your computer.





**Note:** Some of the source code for the StorHouse FTP server was derived from the source code used in the BSD (University of California at Berkeley) FTP tool. See Appendix A for copyright, conditions, and disclaimer information.

## An FTP session

You unload data during an FTP session. An *FTP session* starts when you log into the StorHouse FTP server and ends when you log off. During an FTP session, you can run one or multiple unload operations. You can also load data with the FileTek FTP Data Loader during the same session.

When you run multiple operations during the same FTP session, be sure each operation completes successfully before starting the next one. If you start an unload in the middle of a load (for example, after transferring the control file but before transferring the data file), an error will occur and you will have to restart or abort the load.

For example, the following graphic shows two loads and one unload during one FTP session. Each operation completes before the next one starts.

Operation	Command	Description
Start FTP session	ftp	Log into StorHouse FTP server
Load	put put put - confirm	Transfer control file Transfer data file Confirm successful load
Unload	put get	Transfer control file Get result data
Load	put put put - confirm	Transfer control file Transfer data file Confirm successful load
Stop FTP session	bye	Log off StorHouse FTP server

## Transfer types

When you send and receive files with FTP, you can set the *transfer type* (also called *transfer mode*) so that your client FTP tool and the StorHouse FTP server know how to interpret the data being transferred. There are two transfer types:

- ASCII type – the data is the ASCII character set and the end of line (EOL) is indicated by a carriage return followed by a line feed (CRLF).
- BINARY or IMAGE type – the data is transferred “as is” without any translation. This transfer type is commonly used between UNIX machines.

The FileTek FTP Data Unloader supports both types, but the transfer type you use depends on the record format you want. See page 1-9 for more information about record formats of result data.

## What you need

You need a client FTP tool and StorHouse account information to unload StorHouse data.

## Your client FTP tool

You run the FileTek FTP Data Unloader by submitting user-level FTP commands with your client FTP tool. This tool can have a command line interface or a Graphical User Interface (GUI). It must have these minimum capabilities:

- ASCII transfer type (format control=non print)
- STREAM transmission mode (data is transferred as a single stream of bytes)
- FILE data structure (consists of a continuous sequence of data bytes)
- Ability to connect to the StorHouse FTP server at an alternate port (1985)

- Support for the following server-level FTP commands:

- OPEN (connect to a server)
- QUIT (disconnect from a server)
- PORT (declare the data port to be used)
- TYPE (set the transfer type)
- MODE (select the transmission mode)
- STRU (select the data structure)
- RETR (retrieve (get) a file when unloading data)
- STOR (send (put) a file)

The default state of your client FTP tool should be as follows:

- Transfer type: ASCII (format control=non print)
- Transmission mode: STREAM
- Structure: FILE

See “GUI and client considerations” on page 3-8 for other issues about client FTP tools.

## StorHouse account information

When you log into the StorHouse FTP server, you provide a *StorHouse account ID* and *password*. That account ID must have the following StorHouse privileges to unload data:

- SELECT privilege on the user table(s) you’re unloading
- SCAN privilege (if the query results in a full table scan)
- SQLEXECUTE privilege to submit StorHouse SQL
- SQLCOMMAND privilege to invoke the unloader

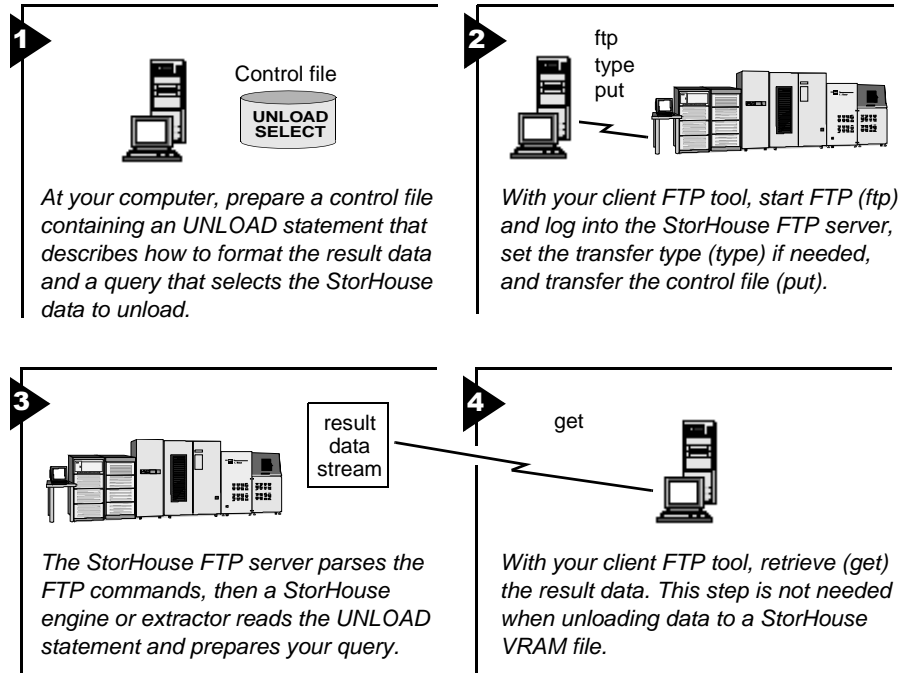
## 1

## The basics

## The unload process

## The unload process

To unload data from StorHouse user tables:



## If an unload fails

If an unload fails, you must correct the problem and start over. For example, if your control file contains syntax errors, correct those errors, then transfer the control file again. If the transfer fails while you're receiving the result data, you must start from the beginning (transfer the control file first, then get the result data). To abort an unload, press CtrlC.

## Replies

You receive messages—or replies—from the StorHouse FTP server in the standard reply format defined by FTP. For instance, replies that begin with 2 are successful or information replies and those that begin with 4 or 5 are error or warning replies. For example:

226 Transfer complete

501- %L-E-XLPUTSYNTAX, Syntax error in put keyword string

### Symbolic names in replies

The replies you receive after a put or get command originate from StorHouse. They're easy to identify because they start with a % followed by an alphabetic code (also called *symbolic name*). For example, the following StorHouse messages indicate a successful unload:

226- %L-I-XLDINFO,\09-MAY-2007:13:38:22 2163 total records processed\  
226- %WORLD-S-XWOK, A request was successfully completed.

The symbolic names (like XLDINFO) are listed in the StorHouse *Messages and Codes Manual*. These names correspond to numeric codes that you can reference to find more information about a message.

### Numeric SQL codes in replies

You may also see StorHouse/RM SQL codes in replies. SQL codes are 0 (successful) or a 5- or 6-digit code, which is usually negative. For example, the following reply contains SQL code 0, which indicates a successful unload:

226 \*\*\*\*\* Unload operation/request finished \*\*\*\*\* sqlcode=<0>

## Limits

A StorHouse engine or extractor executes the query in the UNLOAD statement. The StorHouse SQL\_SESSIONS system parameter specifies the maximum number of StorHouse engines and extractors that can run concurrently. The maximum number includes:

- Engines required for loads (one engine per load)
- Extractors required for queries (one extractor per query)
- Engines required for queries (one engine per query)

Requests beyond the SQL\_SESSIONS limit are rejected (the error code is XENORES).

## Locks

An unload places a shared (read) lock on the selected table(s). Multiple engines and extractors can have a shared lock on the same table.

## Result data

You can receive result data in a *result file* on your computer or in a VRAM file on StorHouse. If your host environment supports piping, you can also pipe the output to another program. If you're unloading BLOB or CLOB data, the FileTek FTP Data Unloader can create a *LOB file* on a local or a remote host for each LOB value or include LOB values in the result file with the other data. You can't pipe output to LOB files. This section defines terms and considerations for result data. See "Considerations for LOB data" on page 1-13 for more information about unloading LOB values.

## Data records and data fields

Result data is composed of data records. Typically, a *data record* corresponds to a row in a user table. Each data record contains *data fields* that usually correspond to values in columns of a user table.

## Data record formats

Result data can be in these record formats:

- Text
- Fixed-length
- Variable-length
- LOB-type

See “Considerations for LOB data” on page 1-13 for more information about LOB result data. You specify the record format you want during your FTP session on the FTP put or get command. Some considerations for choosing a text, fixed-length, or variable-length format are as follows:

- If you plan to use the result data as input to a data loader, you must choose a record format that your data loader accepts.
- The most efficient way to unload data is to use binary (unconverted) data types. This requires a BINARY transfer type and either fixed-length or variable-length record format.
- If the result data contains VAR-type fields, you should choose variable-length format.
- If your destination (data loader, other program, file) requires text (like ASCII), choose text format. Your result data then would need to consist of CHAR or EXTERNAL-type data with optional delimiters.

Note that if you're piping the result data to a program, your program has to know which record format you're using so that it knows how to unpack the records from the data stream.

## Text data

Text is the default record format.

### Text data specifications

Format	<ul style="list-style-type: none"><li>■ Each record ends with the appropriate EOL character, such as a UNIX newline (LF) or an ASCII carriage return and line feed (CRLF) pair.</li><li>■ You can receive text data in ASCII or BINARY mode, but if transferred in BINARY mode, the data will consist of text lines terminated by ASCII newlines (\n).</li><li>■ If you receive result data in ASCII mode and the character set of the local machine is not ASCII (for instance, EBCDIC), your client FTP tool will convert the data to the local character set.</li></ul>
Transfer type	ASCII or BINARY
To use	Omit the var and fixed keyword on the FTP put and get commands.

## Fixed-length data

Fixed-length data records have the same length.

### Fixed-length data specifications

Format	<ul style="list-style-type: none"><li>■ Each record is exactly the same length (in bytes).</li><li>■ If the result data is shorter than the fixed length, it's padded with blanks in the specified character set.</li><li>■ If the result data is longer than the fixed length, an error occurs.</li></ul>
Transfer type	BINARY
To use	Include the fixed keyword on the FTP put or get command, specifying the length in bytes, for example, fixed=80.



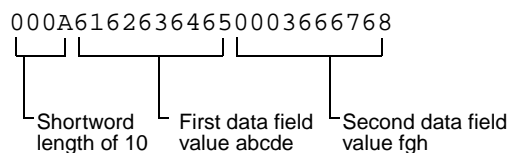
## Variable-length data

Variable-length data records are only as long as necessary to contain the data.

### Variable-length data specifications

Format	<ul style="list-style-type: none"><li>■ Each record is preceded by a 2-byte integer shortword indicating the length of the record. For example, a shortword of 000A (hex) indicates a record length of 10.</li><li>■ Shortword bytes are in native values key (nvk) format, that is, the first byte is the most significant byte (such as for SPARC) or the least significant byte (such as for PC).</li><li>■ Shortword bytes are not counted in the length of the data record.</li></ul>
Transfer type	BINARY
To use	Include the var keyword on the FTP put command.

For example, the following variable-length record is composed of two fields (in hex notation). The first data field is defined as CHAR(5). The second data field is defined as VARCHAR(5) but the actual length is 3 (as indicated by the 0003 preceding the data field).



## Considerations for binary result data

When you unload binary data, the FileTek FTP Data Unloader must know the client hardware type because different operating systems implement binary data fields—or *native data types*—differently. The FileTek FTP Data Unloader supports the following binary data types: BIGINT, BINARY (and synonyms RAW and BYTE), DECIMAL (and synonym NUMERIC), DOUBLE, FLOAT, INTEGER, SMALLINT, VARBINARY (and synonyms VARRAW and VARBYTE), and VARCHAR.

You identify your client hardware type with the `nvk` keyword on the FTP `put` command. The following table describes the different ways that different hosts interpret native data types.

**Native data type representation**

Host type	Byte order	Floating point	INT length
IBM S/370 mainframes	msb to lsb*	Proprietary S/370	4
SPARC (the default)	msb to lsb	IEEE	4
x86, DOS	lsb to msb+	IEEE	2
DEC VAX	lsb to msb	Proprietary VAX	4

*msb* is most significant byte and *lsb* is least significant byte

\* big-endian

+ little-endian

## Considerations for VAR-type result data

When you unload `VARBINARY` or `VARCHAR` data, each variable-length data field starts with a 2-byte `SMALLINT` field indicating the actual length of the data. For instance, the following data (hex, ASCII) has an actual length of 3:

```

0003414243
└─┬─┘ └─┬─┘
actual data
length (ABC)

```

The position of a VAR data field starts at the beginning of this 2-byte field containing the actual length of the data.

## Considerations for LOB data

With the FileTek FTP Data Unloader, you can unload LOB data to three locations:

- In the result file with the other result data
- In separate files (one LOB value per file) on your client computer
- In separate files (one LOB value per file) on a remote system

Even if you are unloading only LOB data to separate files, you will receive a result file on your local computer. The result file contains the names of the LOB files. The FileTek FTP Data Unloader always creates the result file on your local computer with the file name you provide on the FTP get command.

## Unloading LOB data with the other result data

You can include LOB data in the result file two ways:

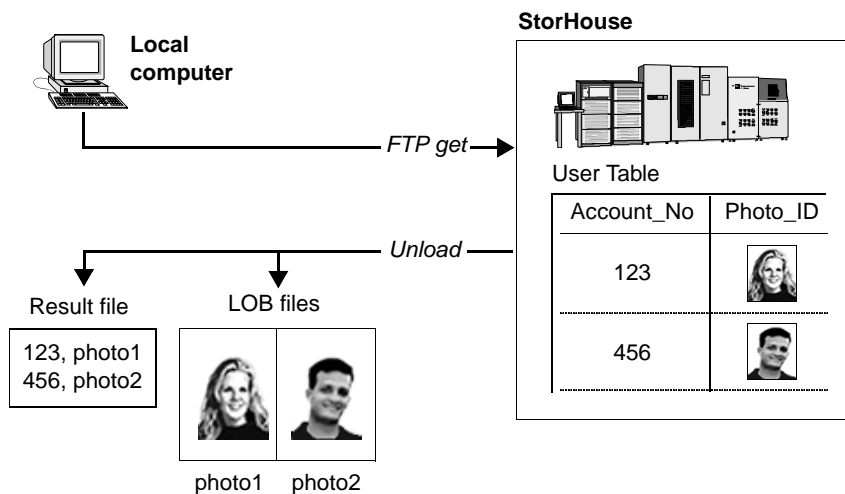
- When a LOB value fits within a result record (the LOB value and the result data record do not exceed 32 KB), you can define that LOB data field as an unloader data type (for instance, VARCHAR or VARBINARY). The following unloader data types (target) are valid for BLOB and CLOB column (source) data types:
  - BINARY
  - BINARY EXTERNAL
  - CHARACTER
  - VARBINARY
  - VARCHAR

For LOBs that fit in a result record, choose a transfer type and record format applicable for the data. For instance, if you unload a CLOB column as VARCHAR result data, specify the var keyword and use the BINARY transfer type.

- When a LOB value fits within or exceeds the result record length, you can define it as a BLOB or CLOB unloader data type. The LOB value then can span multiple records, but each record cannot exceed the maximum record length (32KB-1). When you unload BLOB or CLOB data, each result data field starts with a 64-bit field indicating the actual length of the data. See “Unloading LOB data to the result file” on page 2-59 for an example.

## Creating LOB files on your client computer

You can unload LOB data—one value per file—into files on your client computer. For instance, for the following user table, you can unload account numbers to the result file and photos to separate LOB files on your client computer. The result file contains the names of the LOB files. The LOB files contain data only (no preceding length).



To do this, you specify the directory path (where to create the LOB files) and file name with the BLOB\_FILE or CLOB\_FILE data type in the unload control file. You can also specify the user name and password required to create the files in the specified directory. See page 2-43 for more information about specifying a result file name for LOBs.

The transfer type and record formats you specify on the FTP put or get keywords do not apply to the separate LOB files. They apply to the control file and any result file containing other table data.

## **Creating LOB files on a remote system**

You can unload LOB data to files on a remote system. If you're also unloading non-LOB data, the FileTek FTP Data Unloader creates the result file on your local computer and the LOB files on the remote system. The result file contains the names of the LOB files. The LOB files contain data only (no preceding length).

To to this, in addition to specifying the directory path and file name on the BLOB\_FILE or CLOB\_FILE data type specification, you can include the following:

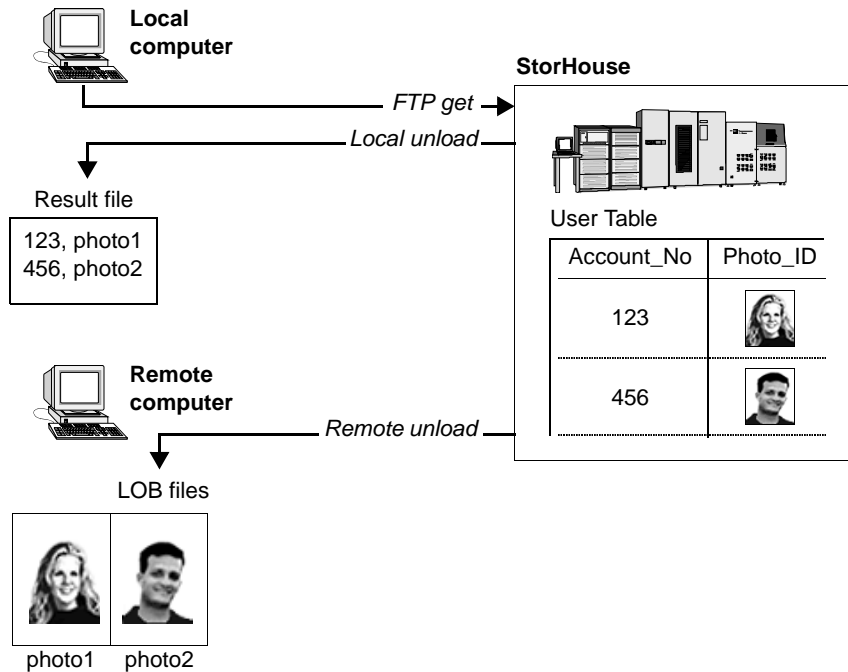
- Host name to identify the remote system (default is the host where FTP is being run)
- UNIX user name and password to log in to the specified host via FTP (default is the StorHouse account and password)

## 1

## The basics

## Result data

After you issue the FTP get command, the FileTek FTP Data Unloader establishes an FTP session with the remote host and creates the result files in the specified directory.



# Preparing an UNLOAD statement

For each unload, you must prepare an UNLOAD statement in a control file. This chapter explains the format of UNLOAD and provides examples.

## About the control file

A *control file* is a text file on your computer. To unload data, you must create a control file containing one SQL-like control statement called UNLOAD. This control statement:

- Specifies the character set of result data
- Describes result data formatting
- Specifies a character to append to the end of result records
- Describes each data field in result records
- Identifies an escape character
- Identifies host, directory path, and file name for LOB data
- Contains the query that selects the StorHouse data to unload

A control file can contain only one UNLOAD statement. No other statements are allowed. Note that the SELECT statement—the query that selects the StorHouse data to unload—is part of the UNLOAD statement.

## Control file conventions

Guidelines for entering an UNLOAD statement in a control file are as follows:

- The control statement can span more than one line, and any new line can begin with any keyword.
- Case is significant only in quoted strings, in the host name in a HOST clause, and in the user name in a USER clause. Otherwise, you can use uppercase and lowercase letters interchangeably.
- You can enclose literal strings in single (') or double (") quotes. In the SELECT statement, however, you must enclose literal strings in single quotes.
- You must include a semicolon (;) at the end of the UNLOAD statement (after the SELECT statement).
- You can mix comments throughout the control file by starting them with two dashes (--). Comments, however, cannot be on the same line as the final semicolon.

## Format conventions

The UNLOAD statement syntax has the same format conventions as StorHouse SQL statements. The SQL format conventions are:

Convention	Description
UPPERCASE	Uppercase terms indicate keywords that you type as shown.
lowercase	Lowercase terms refer to user-supplied values.
( ) , * : ; / + . ' - =	These characters are part of the syntax. Type them as shown.
{ }	Braces indicate required arguments.
[ ]	Brackets indicate optional arguments.



Convention	Description
	Vertical bars separate options. You can specify one option.
...	Ellipsis points indicate you can repeat the part of the statement preceding them any number of times.

## Format of UNLOAD

```
UNLOAD  
[unload_options]  
[ USING ( field_spec [ , field_spec ]... )  
query ;
```

where unload\_options include any of the following clauses in any order:

```
[ CHARACTERSET { cset_name | ccsid } ]  
[ OUTFILE { sth_file_name } [ /group ] ]  
[ FIELDS [ CHAR ] [ NULLFLAGS ] delimiter_spec ]  
[ ESCAPED BY [ DELIMITER | 'char' | NONE ] ]  
[ RECORDS TERMINATED [ BY ] { WHITESPACE | 'char' | X'hexbyte' } ]  
[ RECORDS NOT TERMINATED ]
```

Argument	Description
CHARACTERSET	Specifies the character set of the result data (see page 2-6)
cset_name	WE8EBCDIC500   WE8PC850   WE8ISO8859P1
ccsid	500   850   819
OUTFILE	Unloads data to a StorHouse VRAM file (see page 2-8)
sth_file_name	StorHouse VRAM file name
/group	File access group name
FIELDS	Formats result data (see page 2-9)
CHAR	Indicates all result data fields are to be generated in CHARACTER format

## 2

## Preparing an UNLOAD statement

Format of UNLOAD

Argument	Description
NULLFLAGS	Indicates NULL flags are to be inserted at the beginning of result records for data fields
delimiter_spec	[ TERMINATED [ BY ] { WHITESPACE   'char'   X'hexbyte' } ] [ ENCLOSED [ BY ] { 'char'   X'hexbyte' } [ AND { 'char'   X'hexbyte' } ] ]
ESCAPED BY	Identifies an escape character (see page 2-16)
DELIMITER	Keyword to use the enclosure delimiter as the escape character
'char'	Value of an escape character
NONE	Keyword to not use an escape character
RECORDS	Appends a character to the end of records (see page 2-17)
TERMINATED	Required keyword
BY	Optional keyword
WHITESPACE	Keyword to terminate a record with a space in the character set of the result data
'char'	Value of the record terminator, expressed in character format
X'hexbyte'	Value of the record terminator, expressed in hexadecimal format
RECORDS NOT TERMINATED	Suppresses the last field terminator (see page 2-19)

Argument	Description
USING	Describes each data field in result records (see page 2-19)
field_spec	position_spec   constant_spec
position_spec	datatype_spec [ POSITION ( position   * [ +num ] ) ] [ IFNULL field_assignment ]
datatype_spec	Name and length of the data field's data type (see page 2-22)
position	start_column [ { :   - } end_column ]
num	Unsigned integer optionally followed by K (x1024) or M (xKK)
field_assignment	[ :field_name   ( position ) ] = { any_string   BLANKS }
:field_name	Name of the data field in the result record
any_string	string   X string
constant_spec	[ :field_name ] CONSTANT any_value
any_value	any_string   num
query	Selects the StorHouse data to unload (see page 2-52)  SELECT [ALL   DISTINCT] { *   expr [column_alias] [, expr [column_alias] ]...} [FROM table_spec [, table_spec ]...] [WHERE condition] [GROUP BY column_name [,column_name]... [HAVING condition] ] [ {UNION   UNION ALL} SELECT ...] [ORDER BY {expr   position} [ASC   DESC] [, {expr   position} [ASC   DESC] ]... ] [FOR {FETCH   READ} ONLY]

The UNLOAD keyword and the SELECT statement are the only required arguments. Note the following:

- If you omit the USING and FIELDS CHAR clauses or just specify FIELDS NULLFLAGS, the FileTek FTP Data Unloader generates field\_specs, converting every expression (expr) in the SELECT statement select list to the corresponding unloader data type.

- If you omit the USING clause but include a FIELDS CHAR clause, the FileTek FTP Data Unloader generates field\_specs, converting every expression to CHARACTER (with any associated delimiter\_spec).

## Specifying the character set of result data

You can specify the character set of the result data. When you do, any character-based data and padding are converted to this character set. StorHouse supports ISO, EBCDIC, and PC character sets. You have four options for specifying the character set of the result data:

- Take the default (ISO 8859-1)
- Use the CHARSET keyword with a datatype\_spec for an individual data field of type CHAR or any EXTERNAL (see page 2-40)
- Use the CHARACTERSET clause in the UNLOAD statement
- Provide a value for the data\_ccsid keyword on the FTP put command (see page 3-3)

Note the following:

- If you specify the character set for *both* the CHARACTERSET clause and the data\_ccsid keyword, then the CHARACTERSET clause overrides the data\_ccsid keyword.
- If you don't include the CHARACTERSET clause, the CHARSET keyword, or the data\_ccsid keyword, then the default character set is ISO 8859-1.
- The CCSID value with a CHARSET keyword overrides both the CHARACTERSET clause and the data\_ccsid keyword for an individual data field.

- If you transfer the result data in ASCII mode (and therefore text format) and the target machine has a different native character set, your client FTP tool will convert the result data from ASCII to the native CCSID. You should, therefore, let the data character set default to ISO 8859-1. In other words, do not use CHARACTERSET, CHARSET, or data\_ccsid.

## Format of the CHARACTERSET clause

CHARACTERSET { cset\_name | ccsid }

Argument	Description
cset_name	(required if no ccsid is specified) Name of the result character set. Valid values: <ul style="list-style-type: none"><li>■ WE8EBCDIC500 for EBCDIC character set</li><li>■ WE8PC850 for PC character set</li><li>■ WE8ISO8859P1 for ISO 8859-1 character set</li></ul>
ccsid	(required if no cset_name is specified) CCSID of the result character set. Valid values: <ul style="list-style-type: none"><li>■ 819 for ISO 8859-1 character set</li><li>■ 500 for EBCDIC character set</li><li>■ 850 for PC character set</li></ul>

## Example CHARACTERSET clause

Assume the result data will be EBCDIC. You could include the following CHARACTERSET clause in the UNLOAD statement to specify an EBCDIC character set:

```
CHARACTERSET 500
```

## Unloading data to a StorHouse VRAM file

Use the OUTFILE clause to unload result data to a StorHouse VRAM file. Note the following:

- You have to create this VRAM file first on StorHouse.
- If the VRAM file contains data, the new data will be appended to the file.
- When you use OUTFILE, you don't issue an FTP get command.
- If an error occurs during the unload, you should delete the VRAM file and start over.

### Format of the OUTFILE clause

OUTFILE { sth\_file\_name } [ /group ]

Argument	Description
sth_file_name	(required) Name of the StorHouse VRAM file to contain the result data. You can optionally enclose the file name in double quotes (").
/group	(optional) Name of the StorHouse file access group to which the VRAM file belongs. If the VRAM file is not in the default group of the StorHouse account ID you use to log in the StorHouse FTP server, then you must specify the group name.

---

### Example OUTFILE clause

Assume you want to unload data to a StorHouse VRAM file called SEP2198 in a group called ATM. You'd use this OUTFILE clause if ATM is the default group of the StorHouse account you use to log in the StorHouse FTP server:

```
OUTFILE SEP2198
```

Or you'd use this OUTFILE clause if ATM is not your default group:

OUTFILE SEP2198/ATM

## Formatting result data

Use the FIELDS clause to:

- Delimit character-based data fields (CHARACTER, BLOB\_FILE, CLOB\_FILE, and all EXTERNAL data types) in a result record. See “Guidelines for delimiting data fields” on page 2-10 for more information.
- Generate field\_specs, formatting all data fields as CHARACTER. To do this, include the CHAR keyword with the FIELDS clause and omit the USING clause.
- Insert NULL flags (T for NULL or F for NOT NULL) at the beginning of each result record for each data field. To do this, include the NULLFLAGS keyword with the FIELDS clause and omit the USING clause.

Note the following:

- You can use the CHAR and NULLFLAGS keywords in a FIELDS clause only when you omit a USING clause.
- If you omit the USING and FIELDS clauses or just specify the FIELDS NULLFLAGS clause, the FileTek FTP Data Unloader generates field\_specs and converts each expression in the select list to the corresponding unloader data type.
- When you use FIELDS NULLFLAGS (and omit CHAR) and the result data contains binary data fields that are transferred “as is,” you should use var or fixed data record format (not text) to avoid any false newlines in the result data.

- If you omit the USING clause but include a FIELDS CHAR clause, the FileTek FTP Data Unloader converts every expression in the select list to CHARACTER (with any associated delimiter spec).
- When you use FIELDS CHAR, FileTek recommends you include a delimiter\_spec. Keep in mind that without a delimiter\_spec, the FileTek FTP Data Unloader pads VARBINARY and VARCHAR fields to the maximum length and uses a default of 1 for non-delimited, non-string CHAR data.
- If you specify the FIELDS CHAR clause and the result file contains LOB data fields, all fields must still fit in a legal (32K-1-max) record size.
- If you omit the field\_spec and the FIELDS CHAR clause, the FileTek FTP Data Unloader places LOB data fields at the end of the record. This means that the order of the generated field list for an unload may not match the order of expressions in the SELECT statement. The informational messages returned to you will indicate when this happens.

## Guidelines for delimiting data fields

Delimiters mark the boundaries of data fields. For BLOB\_FILE and CLOB\_FILE data types, delimiters mark the boundaries of the LOB file names in the result file. Note that you cannot delimit VARCHAR or any binary data fields.

You can *terminate* data fields with any single character or a blank, also called *whitespace*. For example, the following data fields are terminated by blanks (indicated by *b*):

```
2839bMcGuirebJack
2388bCornflakebSue
```



You can also *enclose* data fields with the same or different character. For example, the following data fields are enclosed with parentheses (two different characters):

```
( 2839 )(McGuire)(Jack)
( 2388 )(Cornflake)(Sue)
```

You can also terminate and enclose data fields. For example, the following data fields are enclosed with double quotes and terminated with blanks:

```
" 2839 "b"McGuire"b"Jack "
" 2388 "b"Cornflake"b"Sue "
```

If you don't delimit data fields, then they run together. For example:

```
2839McGuireJack
2388CornflakeSue
```

Note the following:

- The maximum length of a data field does not include the delimiter(s).
- A `delimiter_spec` in a `FIELDS` clause applies to or supplements all specified or generated `field_specs`. For example, you can include `TERMINATED BY` in a `FIELDS` clause and `ENCLOSED BY` in a `field_spec`.
- You can override the `delimiter_spec` in a `FIELDS` clause as needed for individual data fields. See page 2-41 for more information about specifying a different delimiter for an individual data field.
- A `delimiter_spec` in a `FIELDS` clause or in a `field_spec` in a `USING` clause does not apply to `CONSTANT field_specs` (see page 2-21) and `IFNULL` strings (see page 2-48). This means you must include any delimiters in the `CONSTANT` string or `IFNULL` string if you want them.
- If you use a `FIELDS` clause to specify a terminator delimiter and there's no `RECORDS` clause, the FileTek FTP Data Unloader uses the `FIELDS` delimiter

as the record terminator. For example, assume a FIELDS clause specifies ! as the terminator delimiter and there's no RECORDS clause.

**FIELDS TERMINATED BY '!'**

The unloader adds the ! delimiter after the last result data field like this:

```
2839!McGuire!Jack!  
2388!Cornflake!Sue!
```

- Any instances of the terminator or enclosure delimiter(s) in the data will be doubled. For example, if the data field is SAN ANTONIO, TX and the terminator delimiter is a comma, then the result data field would include two commas within the data field and one comma after the data field:

```
SAN  ANTONIO , , TX ,
```

- If a data field is enclosed by a single delimiter (like a comma or single quote) and that data field is NULL or zero-length VAR, adjacent delimiters may be interpreted as data. For example, the following data fields are enclosed by single quotes. The second data field, which is NULL, could be interpreted as data:

```
'2839'""'Jack'
```

To prevent this, you could use the IFNULL keyword to insert a blank (see page 2-48) or specify TERMINATED BY along with ENCLOSED BY.

- If a data field is terminated but not enclosed, no data is left after any trailing blanks are trimmed, and the data is NULL, the adjacent terminator delimiters

in the result data might be interpreted as data if you subsequently re-load the data. For instance:

```
2839 , , Jack ,
```

In this case, when re-loading the data, use enclosure delimiters to protect the data field.

- If a data field is terminated and optionally enclosed, no data is left after any trailing blanks are trimmed, and the data is not NULL, then the FileTek FTP Data Unloader omits the enclosure delimiter(s) and adds a blank before the terminator. This provides a way to distinguish zero-length VARCHAR (or all-blank CHAR) data from NULL data when the unload result is subsequently loaded (with the load clauses NULLIF EMPTY TERMINATED BY and OPTIONALLY ENCLOSED BY).

## Format of the FIELDS clause

**FIELDS** [ CHAR ] [ NULLFLAGS ] delimiter\_spec

where delimiter\_spec:

```
[ TERMINATED [ BY ] { WHITESPACE | 'char' | X'hexbyte' } ]  
[ [ OPTIONALLY ] ENCLOSED [ BY ] { 'char' | X'hexbyte' } [ AND { 'char' |  
X'hexbyte' } ] ]
```

## 2

## Preparing an UNLOAD statement

Formatting result data

Argument	Description
CHAR	(optional) Keyword for generating field_specs and formatting all result data fields as CHARACTER. If you include the CHAR keyword, a delimiter_spec is optional but if omitted, all data fields will run together and the default field lengths may be insufficient. An error occurs if you include FIELDS CHAR and a USING clause.
NULLFLAGS	(optional) Keyword for generating field_specs as the corresponding unloader data type (if CHAR is omitted) and inserting NULL flags (T for NULL and F for NOT NULL) at the beginning of each result record for each data field. An error occurs if you include FIELDS NULLFLAGS and a USING clause.
delimiter_spec	(optional, but recommended)
TERMINATED	Keyword for terminating data fields. Note that this terminator will not be added to the last data field, that is, the last position_spec in the USING clause if a RECORDS clause is used.
BY	Keyword for readability only.
WHITESPACE	Keyword for indicating that the delimiter is a blank character. WHITESPACE is equivalent to a space in the character set of the associated data field. You can use this keyword with the TERMINATED keyword, not the ENCLOSED keyword.
'char'	Value of the delimiter, specified in character format and enclosed in single or double quotes. A character delimiter consists of exactly one character.
X'hexbyte'	Value of the delimiter, specified in hexadecimal format. A hex byte consists of exactly two hex digits enclosed in single or double quotes.
OPTIONALLY	Keyword for indicating that some data fields may be enclosed with the specified delimiter(s). You can specify OPTIONALLY only when using TERMINATED. You can include both keywords in the FIELDS clause, or both in a datatype_spec, or one in a FIELDS clause and the other in a datatype_spec.

Argument	Description
ENCLOSED	Keyword for specifying enclosed data fields.
AND	Keyword for specifying data fields enclosed with different starting and ending delimiters, for example, '(' AND ')'. If omitted, the enclosure delimiters are the same.

## Example FIELDS clauses

The following examples show how to terminate and enclose data fields in result data as well as how to use CHAR and NULLFLAGS keywords. Each example shows sample result records. Remember that the FileTek FTP Data Unloader does not add a terminator delimiter after the last data field in a record.

- To terminate data fields with an exclamation point (a character):

**FIELDS TERMINATED BY '!'**

```
2839!McGuire!Jack!  
2388!Cornflake!Sue!
```

- To format all data fields as CHARACTER and to terminate them with blanks:

**FIELDS CHAR TERMINATED BY WHITESPACE**

```
2839bMcGuirebJackb  
2388bCornflakebSueb
```

- To format all data fields as CHARACTER, to terminate them with blanks, and to insert NULL flags at the beginning of each result record:

**FIELDS CHAR NULLFLAGS TERMINATED BY WHITESPACE**

```
FFF2839bMcGuirebJackb  
FFF2388bCornflakebSueb
```

## 2

### Preparing an UNLOAD statement

---

Identifying an escape character

- To terminate data fields with blanks and enclose them in double quotes:

FIELDS TERMINATED BY WHITESPACE ENCLOSED BY '"'

```
"2839"b"McGuire"b"Jack"
```

```
"2388"b"Cornflake"b"Sue"
```

- To enclose data fields in parentheses (different starting and ending delimiters):

FIELDS ENCLOSED BY '(' AND ')'

```
(2839)(McGuire)(Jack)
```

```
(2388)(Cornflake)(Sue)
```

## Identifying an escape character

Include the ESCAPED BY clause to specify an escape character for unloading delimited data from StorHouse. If a delimiter is found in the data that is being unloaded, the FileTek FTP Data Unloader adds the escape character before it outputs the data character.

## Format of ESCAPED BY clause

ESCAPED BY [ DELIMITER | 'char' | NONE ]

Argument	Description
DELIMITER	(optional) Keyword to insert the enclosure delimiter (for example, specified on the FIELDS clause) as the escape character. This is the default for ENCLOSED data. The FileTek FTP Data Unloader ensures that consecutive zero-length fields do not result in adjacent (doubled) terminators by inserting a blank in output data.
'char'	(optional) Value of the terminator or enclosure delimiter to use as the escape character, for instance, '\'. The 'char' value cannot be a whitespace character.
NONE	(optional) Keyword to not specify an escape character. This is the default for TERMINATED data. An error occurs if there are any delimiters in the data.

## Example ESCAPED BY clause

To unload StorHouse data which is to be loaded into Informix (that is, to emulate the Informix UNLOAD):

```
UNLOAD ESCAPED BY '\' FIELDS TERMINATED BY '|' SELECT ...
```

## Appending a character to the end of result records

Use the RECORDS clause to append a character—such as an end of line marker—to the end of a result record. This character is called a *record terminator*. Use a CONSTANT field\_spec (see page 2-21) to append more than one character. Note that if you receive result data in text format and ASCII transfer type, the proper record or line terminator for your host system is already included.

## 2

**Preparing an UNLOAD statement**

Appending a character to the end of result records

**Note:** You cannot use the RECORDS clause when unloading a LOB in the result file, that is, when specifying a BLOB or CLOB unloader data type.

## Format of the RECORDS clause

RECORDS TERMINATED [ BY ] { WHITESPACE | 'char' | X'hexbyte' }

Argument	Description
TERMINATED	(required) Keyword that must accompany RECORDS.
BY	(optional) Keyword for readability only.
WHITESPACE	Keyword for terminating a record with a blank in the character set of the result data.
'char'	Value of the record terminator, specified in character format and enclosed in single or double quotes. A character terminator consists of exactly one character.
X'hexbyte'	Value of the record terminator, specified in hexadecimal format. A hex byte consists of exactly two hex digits enclosed in single or double quotes.

## Example RECORDS clauses

The following examples show how to append a character to the end of result records.

- To terminate each result record with a semicolon:

RECORDS TERMINATED BY ';'

- To terminate each result record with a blank:

RECORDS TERMINATED WHITESPACE



## Suppressing the last field terminator

Include the RECORDS NOT TERMINATED clause to omit the last field terminator in each record. The FileTek FTP Data Unloader suppresses the last field's terminator character, leaving only the newline at the end of each record. The format of the clause is:

RECORDS NOT TERMINATED

## Describing each data field in result records

Include a USING clause to describe each data field in result records. A USING clause consists of field\_specs. There are two types of field\_specs:

- A *constant\_spec* inserts text in a result record. For example, you could insert the names of the data fields in the result records, or you could provide INSERT statement syntax if you plan to load the result data into another database.
- A *position\_spec* identifies the data type of each result data field. A position\_spec can optionally specify the location of the data field in the result record and a string to insert into the record if the associated SELECT expression is NULL.

A USING clause can contain a combination of constant\_specs and position\_specs. You can omit the USING clause or use a FIELDS clause to generate field\_specs using defaults. See "Formatting result data" on page 2-9 for more information about generating field\_specs.

## Format of the USING clause

USING ( field\_spec [ , field\_spec ]... )

## 2

**Preparing an UNLOAD statement**

Describing each data field in result records

where field\_spec:

position\_spec | constant\_spec

Argument	Format
position_spec	datatype_spec [ POSITION (position   * [ +num ] ) ] [ IFNULL field_assignment ]
datatype_spec	Name and length of the data field's data type (see page 2-22)
position	start_column [ { :   - } end_column ]
num	Unsigned integer optionally followed by K (x1024) or M (xKK)
field_assignment	[ :field_name   ( position ) ] = { any_string   BLANKS }
:field_name	Name of a data field in the result record
any_string	string   X string
constant_spec	[ :field_name ] CONSTANT any_value
any_value	any_string   num

## Omitting the USING clause

If you omit the optional USING clause, the FileTek FTP Data Unloader generates field\_specs as follows:

- If you provide a FIELDS clause with the CHAR keyword, all data fields default to CHARACTER (with any associated delimiter\_spec).
- If you omit a FIELDS clause or include a FIELDS NULLFLAGS clause (without the CHAR keyword), each expression in the SELECT statement is converted to its corresponding unloader data type.

## Guidelines for specifying a USING clause

Note the following:

- The number of position\_specs must match the number of SELECT expressions. For example, if you're unloading four columns in a user table, you must provide four position\_specs in a USING clause. The number of constant\_specs does not have to match the number of SELECT expressions.
- When specifying a position, any bytes added for the selected record format are not considered part of the result record. This includes the shortword for variable-length data (FTP var format) and the end-of-line character for text data.
- If a POSITION clause determines the length of the data field, this length does not include delimiters. The start position, however, will be the start position of an enclosure delimiter, so it's possible that the data could overflow the specified field. To avoid this, always specify a data type length (don't take the default) when using fixed position ranges with delimiter\_specs. Note also that the length of a delimited field depends on the data.
- Any gaps in a result record are set to blanks in the specified character set of the result data.
- The length of a result record is determined by the last data field or delimiter in the record, not by the maximum possible length.

## Inserting text (constants) into result records

Use a constant\_spec in a USING clause to insert text into a result record. You can also use a constant\_spec to append multiple characters to the end of a result

## 2

**Preparing an UNLOAD statement**

Describing each data field in result records

record (the RECORDS clause only appends one character). Note that a FIELDS clause does not apply to constant\_specs. The format is:

[ :field\_name ] CONSTANT any\_value

Argument	Description
:field_name	(optional) Name of the data field, preceded by a colon (:).
any_value	(required) Text to insert. Format: any_string   num
any_string	string   X string
string	Character string enclosed in any form of quote, such as 'CA'
X string	Hex digits, for example, x'01ff'
num	Unsigned integer optionally followed by K (x1024) or M (xKK)

For example, to insert data field names in result data (where CHAR is the datatype\_spec for that data field):

```
FIELDS TERMINATED BY ','
USING (CONSTANT 'Last_Name=', CHAR,
CONSTANT 'First_Name=', CHAR,
CONSTANT 'State=', CHAR)
```

Sample result data:

```
Last_Name=Tort,First_Name=Sarah,State=OH
Last_Name=Jones,First_Name=Alex,State=NE
```

See pages 2-55 through 2-58 for more examples of the constant\_spec.

## Specifying the data type of a data field

Include a datatype\_spec in a position\_spec to provide the data type name and length of a data field. The FileTek FTP Data Unloader uses the data type name and length to format and/or convert SELECT results in a result record. For LOB

file data, a datatype\_spec identifies host, directory path, file name, and user information. For some data types, you can also include a character set (see page 2-40) and delimiter(s) (see page 2-41) in a datatype\_spec. Note that:

- The data type name is always required.
- The length, character set, and delimiters are optional.
- If you omit the length, the FileTek FTP Data Unloader uses the default length for that data type.

The data type *must* be one of the data types supported by the FileTek FTP Data Unloader. Those data types are described in the following tables. Definitions of the data type specifications are as follows:

#### Definitions of data type specifications

UNLOAD syntax	Format of the data type in the unload control file
Result field size	Length, default length, length ranges of the data in the output stream
Result format	Format of the data field in the result record
Source types	Valid source types (from DESCRIBE of SELECT, usually a column's CREATE TABLE data type), in other words, the supported conversions. See page 2-39 for a summary of conversions.

#### BIGINT data type

UNLOAD syntax	BIGINT
Result field size	8
Result format	Signed integer (byte order depends on nvk value on FTP put command)
Source types	BIGINT, INTEGER, SMALLINT

## 2

**Preparing an UNLOAD statement**

---

Describing each data field in result records

**BINARY data type**

UNLOAD syntax	BINARY [(length)] RAW [(length)] BYTE [(length)] CHAR[ACTER] [(length)] CHARSET 65535
Result field size	Default length: DESCRIBE length if BINARY or CHAR, else 1 Length range: 1 to 32705
Result format	Fixed-length string (padded with binary zeroes if needed)
Source types	BINARY, BLOB, CHAR, CLOB, VARBINARY, VARCHAR
Notes	<ul style="list-style-type: none"><li>■ If the UNLOAD length is less than the DESCRIBE length, the excess characters are silently truncated if all zeroes, else an error occurs.</li><li>■ If the UNLOAD length is greater than the DESCRIBE length, binary zeroes are added.</li><li>■ When the source type is [VAR]CHAR, there are no conversions, that is, the output characters are copied directly.</li></ul>

---

**BINARY EXTERNAL data type**

UNLOAD syntax	BINARY EXTERNAL [(length)] [CHARSET ccsid ] [delimiter_spec] RAW EXTERNAL [(length)] [CHARSET ccsid ] [delimiter_spec] BYTE EXTERNAL [(length)] [CHARSET ccsid ] [delimiter_spec]
Result field size	Default length: 512 (with delimiter_spec) or else DESCRIBE length*2 if BINARY or CHAR, else 2  Length range: 1 to 32705
Result format	hexits (2 hexits = 1 byte) padded with 00 if needed
Source types	All except TIME, DATE, TIMESTAMP, and DECIMAL
Notes	<ul style="list-style-type: none"><li>■ If the UNLOAD length/2 is less than the data length, the excess characters are silently truncated if all zeroes, else an error occurs.</li><li>■ If the UNLOAD length/2 is greater than the data length, zeroes are added.</li><li>■ If you delimit a BINARY EXTERNAL data field, the actual length is based on the source data (that is, the DESCRIBE length for fixed-length data or the VAR data length).</li><li>■ If you delimit a BINARY EXTERNAL data field and omit the data type length, the default max length is 512 regardless of the source type.</li></ul>

## 2

**Preparing an UNLOAD statement**

---

Describing each data field in result records

**BLOB data type**

---

UNLOAD syntax	BLOB [(max_length [K M G])]
---------------	-----------------------------

Result field size	Contents of 64-bit length field + 8
-------------------	-------------------------------------

Default max\_length: DESCRIBE max\_length

Length range: 0+8 (minimum) to 2G-9+8 (maximum)

---

Result format	64-bit length (in nvk format) followed by data
---------------	--

Source type	BLOB, CLOB
-------------	------------

---

Notes	<ul style="list-style-type: none"><li>■ If a BLOB data value exceeds the max_length, the unload fails only if non-zeros are truncated.</li><li>■ When the source type is CLOB, there are no conversions, that is, the output characters are copied directly rather than converted from hexits or converted from any data character set.</li><li>■ See “Specifying a BLOB or CLOB data type” on page 2-46 for additional considerations.</li></ul>
-------	---

---



**BLOB\_FILE data type**

UNLOAD syntax	BLOB_FILE [(length)] [delimiter_spec] [HOST hostname] [PATH pathspec] FILENAME filename_spec [OVERWRITE] [USER username/password]
Result field size	Default length: 1024 Length range: 1 to 32705
Result format	Any characters that form a valid qualified file name
Output file size	Length of BLOB data in the column
Source type	BLOB, CLOB
Notes	<ul style="list-style-type: none"><li>■ The length is the maximum length of the filename_spec, not the length of the BLOB data in the result file.</li><li>■ If you omit the HOST clause, the default host is your local (client) host.</li><li>■ If you omit the PATH clause, then you must include the path on the FILENAME clause.</li><li>■ If you omit the OVERWRITE clause, an error occurs if files with the same file name exist in the specified path.</li><li>■ If you omit the USER clause, the default user name and password are the StorHouse account ID and password used to log in to the StorHouse FTP server.</li><li>■ See the following page for guidelines on using the corresponding clause:<ul style="list-style-type: none"><li>– HOST: page 2-42</li><li>– PATH: page 2-43</li><li>– FILENAME: page 2-43</li><li>– OVERWRITE: page 2-45</li><li>– USER: page 2-46</li></ul></li></ul>

## 2

## Preparing an UNLOAD statement

Describing each data field in result records

**CHARACTER data type**

UNLOAD syntax	CHAR[ACTER] [(length)] [CHARSET ccsid] [delimiter_spec]
Result field size	<p>Default length: MIN (CREATE TABLE length, 32705) if BLOB, CHAR, CLOB, BINARY, VARBINARY, or VARCHAR; or else 256 (with a delimiter_spec); or else 1</p> <p>Length range: 1 to 32705</p>
Result format	Fixed-length string (padded with blanks if needed)
Source types	All
Notes	<ul style="list-style-type: none"> <li>■ If ccsid is 65535, the data type is changed to BINARY and any delimiter_spec is ignored.</li> <li>■ If the UNLOAD length is less than the source data length, the excess characters are silently truncated if blanks, else an error occurs.</li> <li>■ If the UNLOAD length is greater than the source data length, blanks are added.</li> <li>■ If the source type is [VAR]BINARY, there is no conversion, that is, the output characters are copied directly rather than converted to hexits.</li> <li>■ If the source type is VAR and you include a delimiter_spec, trailing blanks are <i>not</i> trimmed. If the source type is not VAR and you include a delimiter_spec, trailing blanks <i>are</i> trimmed.</li> </ul>

**CLOB data type**

UNLOAD syntax	CLOB [(max_length [K M G])] [CHARSET ccsid]
---------------	---

Result field size	Contents of 64-bit length field + 8 Default max_length: DESCRIBE max_length Length range: 0+8 (minimum) to 2G-9+8 (maximum)
-------------------	---

Result format	64-bit length (in nvk format) followed by data
---------------	--

Source type	CLOB, BLOB
-------------	------------

Notes	<ul style="list-style-type: none"><li>■ If a CLOB data value exceeds the max_length, the unload fails only if non-blanks are truncated.</li><li>■ If the ccsid is 65535, the data type is changed to BLOB.</li><li>■ When the source type is BLOB, there are no conversions, that is, the output characters are copied directly rather than converted to hexits.</li><li>■ See “Specifying a BLOB or CLOB data type” on page 2-46 for additional considerations.</li></ul>
-------	--

## 2

**Preparing an UNLOAD statement**

Describing each data field in result records

**CLOB\_FILE data type**

UNLOAD syntax	CLOB_FILE [(length)] [CHARSET ccscid] [delimiter_spec] [HOST hostname] [PATH pathspec] FILENAME filename_spec [OVERWRITE] [USER username/password]
Result field size	Default length: 1024 Length range: 1 to 32705
Result field format	Any characters that form a valid qualified file name
Output file size	Length of CLOB data in the column
Source type	CLOB, BLOB
Notes	<ul style="list-style-type: none"> <li>■ The length is the maximum length of the filename_spec, not the length of the CLOB data in the result file.</li> <li>■ If you include the CHARSET clause, the character set applies to the LOB data, not the LOB file name.</li> <li>■ If you omit the HOST clause, the default host is your local (client) host.</li> <li>■ If you omit the PATH clause, then you must include the path on the FILENAME clause.</li> <li>■ If you omit the OVERWRITE clause, an error occurs if files with the same file name exist in the specified path.</li> <li>■ If you omit the USER clause, the default user name and password are the StorHouse account ID and password used to log in to the StorHouse FTP server.</li> <li>■ See the following page for guidelines on using the corresponding clause: <ul style="list-style-type: none"> <li>– CHARSET: page 2-40</li> <li>– HOST: page 2-42</li> <li>– PATH: page 2-43</li> <li>– FILENAME: page 2-43</li> <li>– OVERWRITE: page 2-45</li> <li>– USER: page 2-46</li> </ul> </li> </ul>

**DATE or DATE EXTERNAL data type**

UNLOAD syntax	DATE [EXTERNAL] [(length)] ["mask"] [CHARSET ccsid] [delimiter_spec]
Result field size	Default length: 256 (with delimiter_spec) or else 75 (with mask) or else 10  Length range: 1 to 32705
Result format	Default: MM/DD/YYYY (right padded with blanks if needed)
Source types	DATE, TIMESTAMP
Notes	<ul style="list-style-type: none"><li>■ The mask (if present) will be used as a format string. Refer to the <i>StorHouse SQL Reference Manual</i> for valid date format strings.</li><li>■ If you delimit a DATE EXTERNAL data field, any blanks are trimmed.</li><li>■ If the result length is greater than the UNLOAD length, an error occurs.</li></ul>

## 2

**Preparing an UNLOAD statement**

---

Describing each data field in result records

**DECIMAL data type**

UNLOAD syntax	DEC[IMAL] [PACKED] NUMERIC [PACKED] DEC[IMAL] (precision[, scale]) NUMERIC (precision[, scale])
Result field size	(precision + 2) / 2  Default precision: DESCRIBE precision and scale  Default scale (if precision supplied): 0  Precision range: 1 to 31  Scale range: 0 to precision
Result format	IBM S/370 packed decimal format converted (w/ rounding) from DESCRIBE precision and scale
Source types	DECIMAL
Notes	<ul style="list-style-type: none"><li>■ If you omit the precision, the FileTek FTP Data Unloader attempts to calculate it with the POSITION clause.</li><li>■ If you provide the precision or the FileTek FTP Data Unloader can calculate it but you omit the scale, the default scale is 0.</li></ul>

---

**DECIMAL EXTERNAL data type**

UNLOAD syntax	DEC[IMAL] EXTERNAL (length[, scale]) [CHARSET ccsid] [delimiter_spec]  NUMERIC EXTERNAL (length[, scale]) [CHARSET ccsid] [delimiter_spec]
Result field size	Default length: 256 (with delimiter_spec) or else DESCRIBE precision + 3  Default scale: DESCRIBE scale  Length range: 1 to 32705
Result format	Digits and decimal point, left-padded with blanks if needed
Source types	DECIMAL
Notes	<ul style="list-style-type: none"> <li>■ If the result length is greater than the UNLOAD length, an error occurs.</li> <li>■ If a DECIMAL EXTERNAL data field is delimited, the result length will vary.</li> <li>■ Any fractional digits past the scale are rounded.</li> </ul>

**DOUBLE data type**

UNLOAD syntax	FLOAT FLOAT (bits) DOUBLE [ PRECISION ]
Result field size	8
Result format	Double-precision float (format depends on nvk value on FTP put command)
Source types	REAL, DOUBLE
Note	bits = 22 to 53

## 2

**Preparing an UNLOAD statement**

Describing each data field in result records

**FLOAT (REAL) data type**

UNLOAD syntax	FLOAT (bits) REAL
Result field size	4
Result format	Single-precision float (format depends on nvk value on FTP put command)
Source types	REAL, DOUBLE
Note	bits = 1 to 21

**FLOAT EXTERNAL data type**

UNLOAD syntax	FLOAT EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Result field size	Default length: 24 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32705
Result format	± d.d.....dE±dd (left-padded with blanks if needed)
Source types	REAL, DOUBLE
Notes	<ul style="list-style-type: none"> <li>■ If the result length is greater than the UNLOAD length, an error occurs.</li> <li>■ If you delimit a FLOAT EXTERNAL data field, the result field size will vary.</li> </ul>

**INTEGER data type**

UNLOAD syntax	INT[EGER]
Result field size	2 if nvk=DOS, else 4
Result format	Signed integer (byte order depends on nvk value on FTP put command)
Source types	BIGINT, INTEGER, SMALLINT



**INTEGER EXTERNAL data type**

UNLOAD syntax	INT[EGER] EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Result field size	Default length: 11 (without delimiter_spec) or 256 (with delimiter_spec)  Length range: 1 to 32705
Result format	Left-padded with blanks if needed
Source types	BIGINT, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none"><li>■ If the result length is greater than the UNLOAD length, an error occurs.</li><li>■ If you delimit an INTEGER EXTERNAL data field, the result field size will vary.</li></ul>

**SMALLINT data type**

UNLOAD syntax	SMALLINT
Result field size	2
Result format	Signed short integer (byte order depends on nvk value on FTP put command)
Source types	BIGINT, INTEGER, SMALLINT
Note	If the source type is INTEGER and the result type is SMALLINT, an error occurs when a result data value is greater than 32767.

## 2

## Preparing an UNLOAD statement

Describing each data field in result records

**TIME EXTERNAL data type**

UNLOAD syntax	TIME EXTERNAL [(length)] ["mask"] [CHARSET ccscid] [delimiter_spec]
Result field size	Default length: 256 (with delimiter_spec) or else 75 (with mask) or else 12  Length range: 1 to 32705
Result format	Default mask: HH24:MI:SS.MLS (right padded with blanks if needed)
Source types	TIME, TIMESTAMP
Notes	<ul style="list-style-type: none"> <li>■ The mask (if present) is used as a format string. Refer to the <i>StorHouse SQL Reference Manual</i> for valid time format strings.</li> <li>■ If you use the default mask, and the MLS (milliseconds) is zero, and the data field is fixed-length, then the FileTek FTP Data Unloader inserts four blanks instead of 000 for the MLS portion.</li> <li>■ If the result length is greater than the UNLOAD length, an error occurs.</li> <li>■ If you delimit a TIME EXTERNAL data field, the result field size will vary.</li> </ul>

**TIMESTAMP EXTERNAL data type**

UNLOAD syntax	TIMESTAMP EXTERNAL [(length)] ["mask"] [CHARSET ccSID] [delimiter_spec]
Result field size	Default length: 256 (with delimiter_spec) or else 75 (with mask) or else 26  Length range: 1 to 32705
Result format	Default mask: MM/DD/YYYY HH24:MI:SS.MLS (right padded with blanks if needed)
Source types	TIMESTAMP
Notes	<ul style="list-style-type: none"> <li>■ The mask (if present) is used as a format string. Refer to the <i>StorHouse SQL Reference Manual</i> for valid format strings.</li> <li>■ If the result length is greater than the UNLOAD length, an error occurs.</li> <li>■ If you delimit a TIMESTAMP EXTERNAL data field, the result field size will vary.</li> </ul>

**VARBINARY data type**

UNLOAD syntax	VARBINARY [(max_length)] VARRAW [(max_length)] VARBYTE [(max_length)] VARCHAR [(max_length)] CHARSET 65535
Result field size	Contents of SMALLINT length field + 2  Default max_length: DESCRIBE max_length+2 if VARBINARY or VARCHAR, else 256+2  Length range: 0+2 (minimum) to 32705+2 (maximum)
Result format	Same as internal format except length bytes are in nvk byte order
Source types	BLOB, BINARY, CHAR, CLOB, VARBINARY, VARCHAR
Notes	<ul style="list-style-type: none"> <li>■ If the source length is larger than the UNLOAD max_length, the data is silently truncated if all zeroes, else an error occurs.</li> <li>■ When the source type is [VAR]CHAR, there are no conversions, that is, the output characters are copied directly.</li> </ul>

## 2

## Preparing an UNLOAD statement

Describing each data field in result records

**VARCHAR data type**

UNLOAD syntax	VARCHAR [(max_length)] [ CHARSET ccsid ]
Result field size	<p>Contents of SMALLINT length field + 2</p> <p>Default max_length: DESCRIBE max_length+2 if VARCHAR or VARBINARY, else 256+2</p> <p>Length range: 0+2 (minimum) to 32705+2 (maximum)</p>
Result format	Same as internal format except length bytes are in nvk byte order
Source types	All
Notes	<ul style="list-style-type: none"> <li>■ If the ccsid is 65535, the data type is changed to VARBINARY.</li> <li>■ If the data length is larger than UNLOAD max_length, the data is silently truncated if all blanks, else an error occurs.</li> <li>■ When the source type is [VAR]BINARY, there is no conversion, that is, the output characters are copied directly.</li> </ul>

**Converting data types**

The following table summarizes the data type conversions for unloading StorHouse data. In the table:

- The *Result type* is the data type of the result data (that is, the unloader type).
- The *Source type* is the data type of the expression being unloaded.

For instance, you can unload a BINARY, BLOB, CHARACTER, CLOB, VARBINARY, or VARCHAR column in a StorHouse table (Source type) to a BINARY data field in a result record (Result type). Synonyms are not listed but

are supported. For instance, BYTE, a synonym for BINARY, has the same conversions as BINARY.

### Data type conversions for unloading data

Result type	Source Type													
	BIGINT	BINARY	BLOB	CHARACTER	CLOB	DATE	DOUBLE PRECISION	INTEGER	NUMERIC (DECIMAL)	REAL	SMALLINT	TIME	TIMESTAMP	VARBINARY
BIGINT	X							X			X			
BINARY		X	X	X	X									X
BINARY EXTERNAL		X	X	X	X		X	X		X	X			X
BLOB			X		X									
BLOB_FILE			X		X									
CHARACTER	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CLOB			X		X									
CLOB_FILE			X		X									
DATE EXTERNAL						X							X	
DECIMAL									X					
DECIMAL EXTERNAL									X					
DOUBLE							X			X				
FLOAT(REAL)							X			X				
FLOAT EXTERNAL							X			X				
INTEGER	X							X			X			
INTEGER EXTERNAL	X							X			X			
SMALLINT	X							X			X			
TIME EXTERNAL												X	X	
TIMESTAMP EXTERNAL													X	

## 2

## Preparing an UNLOAD statement

Describing each data field in result records

Data type conversions for unloading data (continued)

Result type	Source Type												
	BIGINT	BINARY	BLOB	CHARACTER	CLOB	DATE	DOUBLE PRECISION	INTEGER	NUMERIC (DECIMAL)	REAL	SMALLINT	TIME	TIMESTAMP
VARBINARY		X	X	X	X								X
VARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X

## Specifying a character set for an individual data field

For CHAR, CLOB, CLOB\_FILE, VARCHAR, and EXTERNAL data fields, you can specify a character set for an individual data field by including the CHARSET keyword and CCSID value with a datatype\_spec. This character set overrides the character set in the CHARACTERSET clause of the UNLOAD statement and in the data\_ccsid keyword on the FTP put command. Use the CHARSET keyword only when you want to override one of these character sets (or the default character set) for an individual data field. For CLOB\_FILE, this character set refers to the LOB data, not the LOB file name.

Remember that when you receive result data using the ASCII transfer type and the target machine has a different native character set, your client FTP tool will convert the result data from ASCII to the native CCSID. Use the CHARSET feature only when receiving result data using the BINARY transfer type.

The CCSIDs you can specify with the CHARSET keyword are:

- 819 for ISO 8859-1
- 500 for EBCDIC
- 850 for PC

For CHAR and VARCHAR data fields, you can also specify CCSID 65535 with the CHARSET keyword. Specifying CCSID 65535 with the CHARSET keyword for these data fields is a synonym for BINARY and VARBINARY, respectively. Specifying CCSID 65535 with any of the EXTERNAL data types is invalid.

For example, assume the character set specified for the CHARACTERSET clause is EBCDIC, but the character set of an INT EXTERNAL data field in the result data is PC. In the datatype\_spec, you'd include this CHARSET keyword to specify a different character set for this data field only:

```
INT EXTERNAL(4) CHARSET 850
```

### **Specifying a delimiter for an individual data field**

For CHAR, BLOB\_FILE, CLOB\_FILE, and any of the EXTERNAL data fields, you can specify a delimiter or override the default delimiter in a FIELDS clause by including a delimiter\_spec with a datatype\_spec. See “Formatting result data” on page 2-9 for the format of the delimiter\_spec.

**Note:** If you specify a delimiter for a data field, any trailing blanks will be trimmed (except for VAR source types and BINARY EXTERNAL). Therefore, the actual length of the data in the result record can vary and any length specified by the data type or the POSITION keyword becomes a maximum length.

For example, assume you want to terminate most result data fields with a comma, but you want to terminate one data field (an INT EXTERNAL field that starts in position 12) with a colon. You'd include a FIELDS clause to set the default delimiter to a comma:

```
FIELDS TERMINATED BY ','
```

Then you'd include a delimiter\_spec with the datatype\_spec to override the FIELDS clause for the INT EXTERNAL data field only:

```
INT EXTERNAL(8) POSITION(12) TERMINATED BY ':'
```

In this example, all data fields in the result records will be terminated by a comma, but the data field that starts in position 12 will be terminated by a colon.

## **Specifying a remote host name for LOB data**

For BLOB\_FILE and CLOB\_FILE data types, include a HOST clause in the datatype\_spec to indicate the name of the remote machine where the LOB files are to be created. A HOST clause is not necessary to create LOB files on your client computer. Note the following:

- The host must be known to StorHouse, that is, the host name must be listed in the StorHouse /etc/hosts file so that the FileTek FTP Data Unloader can convert the host name to an IP address. Contact your FileTek customer support representative for more information about entering host names in the /etc/hosts file.
- The remote host must be FTP-enabled (capable of receiving an FTP connection).
- The host name is case sensitive, but you do not have to enclose it in single quotes.

For example, to create LOB files on a remote host named prod1, type:

```
BLOB_FILE HOST 'prod1' FILENAME '/lobs/photo%d.jpg'  
USER 'juliette'/romeo
```

See “Specifying a file name for LOB files” on page 2-43 for more information about providing a FILENAME clause in a datatype\_spec. If the user name and password required to access this host via FTP is different from the StorHouse account ID and password you use to log into the StorHouse FTP server, then you must also include a USER clause in the datatype\_spec. See “Specifying a user name and password” on page 2-46 for more information about providing a USER clause in a datatype\_spec.



## Identifying the path to the LOB directory

For BLOB\_FILE and CLOB\_FILE data types, include a PATH clause and path name to identify the directory to contain the LOB files. The path must be appropriate for your host operating system and can be a fully qualified path or a relative path.

You can omit the PATH clause and instead include the path name with the FILENAME keyword. If you include the PATH clause, the directory name does not appear in the result row. If you omit the PATH clause (and include with the FILENAME clause), the directory name appears in the result row. The file name always appears in the result row whether PATH is included or omitted.

Guidelines for specifying a path name is as follows:

- Enclose the path name in quotes.
- The ending slash (/) in a UNIX path is optional as the FileTek FTP Data Unloader provides it when omitted.
- The path name cannot exceed the maximum length (if specified) on the BLOB\_FILE and CLOB\_FILE data types. The largest possible length is 1024 bytes.

For example, to specify a path for a BLOB data field, type:

```
BLOB_FILE PATH '/home/user/blobs/' FILENAME 'photo%08dCoIA'
```

If the user name and password required to access this directory is different from the StorHouse account ID and password you use to log into the StorHouse FTP server, then you must also include a USER clause.

## Specifying a file name for LOB files

For BLOB\_FILE and CLOB\_FILE data types, specify the LOB file name with the FILENAME clause. You can include a format string (printf) in the file name to

generate a sequence number for each LOB file. The FILENAME clause is required and the filename\_spec must be enclosed in quotes. The FileTek FTP Data Unloader inserts the LOB file names in the result rows.

You can include the directory path with the FILENAME clause or use a PATH clause instead. If you include the path with FILENAME, then the directory name appears in the result row with the file name. If you omit the path with FILENAME, then just the file name appears in the result row.

Note the following:

- The file name must be in the same character set as the data (optionally specified on the data\_ccsid keyword for the FTP put command).
- The directory path must be appropriate for the host operating system and can be a fully qualified path or a relative path.
- The format string can contain any printf-style modifiers. Some common modifiers are listed below. Refer to a C language handbook for more information about printf.
  - %c – Single character
  - %d – Signed decimal integer
  - %ld – Signed long decimal integer
  - %f – Decimal floating-point number
  - %s – Character string
  - %u – Unsigned decimal integer
  - %lu – Unsigned long decimal integer

For instance, in the following example, the FileTek FTP Data Unloader generates file names photo1.jpg, photo2.jpg, photo3.jpg, and so on.

```
BLOB_FILE FILENAME '/home/user/blobs/photo%d.jpg'
```

In this example, the FileTek FTP Data Unloader generates file names photo00000001ColA, photo00000002ColA, photo00000003ColA, and so on.

```
BLOB_FILE PATH '/home/user/blobs/' FILENAME 'photo%08dColA'
```

- If you omit the format string, the FileTek FTP Data Unloader inserts a sequence number (%d) before the extension. In the following example, the FileTek FTP Data Unloader generates file names photo1.jpg, photo2.jpg, photo3.jpg, and so on.

```
BLOB_FILE FILENAME '/home/user/blobs/photo.jpg'
```

- If the file name doesn't have an extension and you omit the format string, the FileTek FTP Data Unloader inserts the sequence format at the end of the file name. In the following example, the FileTek FTP Data Unloader generates file names photo1, photo2, photo3, and so on.

```
BLOB_FILE FILENAME '/home/user/blobs/photo'
```

- If you're unloading from a table with more than one LOB column per row, the FileTek FTP Data Unloader increments the sequence number for each LOB column. In the following example, the FileTek FTP Data Unloader generates file names mylob1, yourlob1, mylob2, yourlob2, and so on (instead of mylob0, yourlob1, mylob2, yourlob3).

```
BLOB_FILE FILENAME 'mylob%d'  
BLOB_FILE FILENAME 'yourlob%d'
```

- The sequence number is incremented for each row, even if there are NULL LOB values in a row, for example, mylob1, yourlob1, mylob2, mylob4, yourlob4, mylob5, yourlob5.

## Overwriting LOB files

You can overwrite existing LOB files by including the OVERWRITE clause on the BLOB\_FILE or CLOB\_FILE unloader data type. If you omit the OVERWRITE

**2****Preparing an UNLOAD statement**

---

Describing each data field in result records

clause, an error occurs if files with the same file name exist in the specified path. For example, the FileTek FTP Data Unloader overwrites any files with the name photo1.jpg, photo2.jpg, and so on in the path home/user/blobs/:

```
BLOB_FILE PATH '/home/user/blobs/' FILENAME 'photo.jpg' OVERWRITE
```

**Specifying a user name and password**

For BLOB\_FILE and CLOB\_FILE data types, include a USER clause in a datatype\_spec if the user name and password required to access a remote host or the local directory are not the StorHouse account ID and password used to log into the StorHouse FTP server. The user name and password are case sensitive, but you do not have to enclose them in single quotes.

For example, to specify the user name tka and password tka for a local UNIX path, type:

```
BLOB_FILE PATH '/home/user/' FILENAME 'photo.jpg' USER 'tka'/'tka'
```

Or to specify the user name juliette and password romeo for a remote host named prod1, type:

```
BLOB_FILE HOST 'prod1' FILENAME '/lobs/photo%d.jpg'  
USER juliette/romeo
```

**Specifying a BLOB or CLOB data type**

For BLOB and CLOB data types, specify the field\_spec last in the field list. BLOB and CLOB data fields must be relatively positioned, that is, as POSITION(\*).

## Specifying the position of a data field in a result record

Use POSITION in a position\_spec to indicate the location of a data field in a result record. The first character of a result record is at position 1. You can specify:

- A *fixed position* with a starting (and optional ending) column number
- A *relative position* as a continuation or an offset from the previous data field

FileTek recommends you use relative positioning when the result data contains VAR-type data fields or delimited data fields. You must use relative positioning for BLOB and CLOB data fields. For VAR data fields, the position starts at the 2-byte field containing the actual length of the data field. For LOB data fields, the position starts at the 8-byte field containing the actual length of the data field. Any relatively positioned data fields (using \* or \*+num) *after* a VAR or LOB data field will start at varying positions based on the actual length of the data field in each result record. POSITION \* (relative position as a continuation) is the default.

The format is:

POSITION ( position | \* [ +num ] )

where position:

start\_column [ { : | - } end\_column ]

For example:

- To specify the starting column number of the data field:

POSITION (1)

In this example, the data field starts in position 1 of the result record. The length of the data field comes from the datatype\_spec.

- To specify the starting and ending column numbers of the data field:

POSITION (1:3)

In this example, the data field starts in position 1 and ends in position 3 of the result record. Another way to specify this is POSITION (1-3). The length of the datatype\_spec is ignored.

- To specify continuation from the previous data field:

POSITION (\*)

In this example, the data field is located one record position after the previous data field. Therefore, if the previous data field ends in position 15, then this data field starts in position 16 of the result record. The length of the data field comes from the datatype\_spec. POSITION (\*) is the default.

- To specify an offset from the previous data field:

POSITION (\*+5)

In this example, the data field is located 5 positions plus 1 position after the previous data field. Therefore, if the previous data field ends in position 10, then this data field starts in position 16 of the result record. POSITION (\*+0) is the same as POSITION (\*). The length of the data field comes from the datatype\_spec.

## Storing a value for NULL data

By default, nothing is stored in a result data field for NULL data. Use IFNULL in a position\_spec to indicate a string to store in a result record when the associated SELECT expression is NULL. The string can be one or more blanks, a character string (such as the word NULL), or a hex string. IFNULL must follow the datatype\_spec and delimiter\_spec (if used) in a position\_spec.

The format is:

IFNULL field\_assignment

where field\_assignment:

[ :field\_name | ( position ) ] = { any\_string | BLANKS }

Argument	Description
:field_name	(optional) Name of the data field, preceded by a colon.
( position )	(optional) Starting and optionally ending column in the result record. Format:  start_column [ { :   - } end_column ]  Example: (1:2) or (1-2) where 1 is the starting column and 2 is the ending column
any_string	(required if not using BLANKS) Value to store, specified as a character string or hex string.
string	Character string enclosed in any form of quote, such as 'NULL' or "NULL"
X string	Hex digits, for example, x'01ff'
BLANKS	(required if not using any_string) One or more blanks to store in the data field. The number of blanks is determined as follows: <ul style="list-style-type: none"><li>■ If IFNULL refers to a :field_name, the number of blanks equals the length of the constant_spec.</li><li>■ If you specify a starting and ending column in the position, this determines the number of blanks. For example, (6:7) stores two blanks.</li><li>■ If you omit the ending column in the position, for example (6), a single blank is used.</li><li>■ If IFNULL refers to the associated SELECT expression and the data field is delimited or the data type is VAR, then a single blank is used; otherwise, the length of the fixed-length data field determines the number of blanks.</li></ul>

Note the following:

- If you omit the (position) or :field\_name, the target data field is that of the associated SELECT expression.
- If you provide a :field\_name, the IFNULL result overrides the constant\_spec, that is, the IFNULL string is stored instead of the CONSTANT string. See “Creating fixed-length data fields with NULL flags at the start of records” on page 2-57 for an example.
- An IFNULL string affects the record length of the target data fields, even if fixed-length. See pages 2-56 and 2-57 for examples.
- You must include any desired delimiters in the IFNULL string. The FileTek FTP Data Unloader does not add delimiters to IFNULL strings even when there’s a delimiter\_spec for the data field and you omit the (position) and :field\_name.
- If you omit IFNULL and a fixed-length data field is NULL, the FileTek FTP Data Unloader inserts blanks in that position.

For example:

- To store blanks for NULL data (where CHAR is the datatype\_spec):  
  
CHAR IFNULL=BLANKS
- To store the word NULL (where INTEGER EXTERNAL is the datatype\_spec):  
  
INTEGER EXTERNAL IFNULL='NULL'



## Example USING clause

The following USING clause contains eight field\_specs. The first seven field\_specs are position\_specs. The last field\_spec is a constant\_spec.

```
USING (INT EXTERNAL(4) POSITION(1),  
CHAR(15) POSITION(5),  
CHAR(10) POSITION(20),  
CHAR(15) POSITION(30),  
CHAR(10) POSITION(45),  
CHAR POSITION(55) IFNULL=BLANKS,  
SMALLINT,  
:STATE CONSTANT 'TX')
```

Note the following:

- Commas separate each field\_spec. Parentheses enclose all field\_specs.
- The data type name in a position\_spec is always required. The data type length is optional and if omitted, the default length is used. For instance, the length was omitted on the last CHAR position\_spec, so the default length (for CHAR source type) is the DESCRIBE length for the corresponding SELECT expression.
- The SMALLINT field doesn't have a POSITION clause. The remaining position\_specs contain the starting column number only. In both cases, the length of the data type determines the length of the data field. For instance, the length of SMALLINT is 2 bytes, so the SMALLINT data field has a length of 2 bytes.

## Selecting the StorHouse data to unload

Use a SELECT statement to choose the data to unload. Note this:

- The number of position\_specs in a USING clause must match the number of expressions (expr) in the SELECT expression list.
- The maximum size of a result record is 32,767. An error occurs when a result record is longer than this.

### Format of the SELECT statement

```
SELECT [ALL | DISTINCT]
{* | expr [column_alias] [, expr [column_alias] ]...}
[FROM table_spec [, table_spec ]...]
[WHERE condition]
[GROUP BY column_name [,column_name]... [HAVING condition] ]
[ {UNION | UNION ALL} SELECT ...]
[ORDER BY {expr | position} [ASC | DESC] [, {expr | position}
[ASC | DESC] ]... ]
[FOR {FETCH | READ} ONLY]
```

Refer to the *StorHouse SQL Reference Manual* for a complete description of the SELECT statement arguments.

### Example SELECT statement

The following SELECT statement contains six expressions in the expression list. If you included a USING clause, you'd provide six position\_specs.

```
SELECT ID, NAME, ADDRESS, STATE, ZIP, PHONE
FROM CUSTOMER
```



## Example UNLOAD statements

This section shows example UNLOAD statements that unload data from a table called INVENTORY. The first and last examples illustrate how to unload LOB data. The CREATE TABLE statement for the INVENTORY table is as follows:

```
CREATE TABLE INVENTORY  
(ITEM CHAR(10),  
QUANTITY INTEGER,  
STATUS VARCHAR(10),  
PHOTO BLOB)
```

The table below contains sample data. A – indicates NULL data.

**INVENTORY table**

ITEM	QUANTITY	STATUS	PHOTO
plug	12	–	
cable	–	out	

## Creating fixed-length fields and records

**UNLOAD statement**      UNLOAD  
 USING (CHAR(10),  
 INTEGER EXTERNAL(10),  
 CHAR(10),  
 BLOB\_FILE(12) PATH '/home/tka/' FILENAME 'photo%d.jpg' USER 'tka/tka' )  
 SELECT \* FROM INVENTORY;

**Result file**      plugbbbbbbbbbbbbbb12bbbbbbbbbbphoto1.jpgbb  
 cablebbbbbbbbbbbbbboutbbbbbbphoto2.jpgbb

Notes:

- Each data field has a fixed length of 10 bytes.
- *b* represents blanks.
- In fixed-length data, INTEGER EXTERNAL data is right-justified (12 is right-justified in the second field) and CHAR data is left-justified (out is left-justified in the third field).
- The FileTek FTP Data Unloader inserted the generated LOB file names in the result data, for instance, photo1.jpg and photo2.jpg.
- Blanks are added to values shorter than the fixed length of fields, for example, plugbbbbbb, bbbbbbbb12, cablebbbbbb, outbbbbbb, photo1.jpgbb, and photo2.jpgbb.
- Each record contains a NULL data field (third field in the first record and second field in the second record). When a fixed-length data field is NULL, the FileTek FTP Data Unloader inserts blanks in that position. So in the first record, the Unloader inserted 10 blanks in the third field and in the second record the Unloader inserted 10 blanks in the second field.

**LOB files**      The FileTek FTP Data Unloader placed each photo in a file in the /home/tka/ client directory using the permissions of the user tka. The name of the first file is photo1.jpg and the name of the second file is photo2.jpg.

## **Terminating data fields and records, adding keywords next to each data field**

**UNLOAD statement**      UNLOAD  
FIELDS TERMINATED BY ','  
RECORDS TERMINATED BY ';'   
USING (CONSTANT 'Item=', CHAR,  
CONSTANT 'Quantity=', CHAR,  
CONSTANT 'Status=', CHAR)  
SELECT ITEM, QUANTITY, STATUS FROM INVENTORY;

**Result file**      Item=plug,Quantity=12,Status=;  
Item=cable,Quantity=,Status=out;

Notes:

- Keywords (Item=, Quantity=, or Status=) are inserted before each data field (constant\_specs).
- Each data field (except the last) is terminated by a comma (FIELDS clause).
- Each record is terminated by a semicolon (RECORDS clause).

## Creating INSERT statements with NULL keywords for NULL data

### UNLOAD statement

```
UNLOAD
FIELDS TERMINATED BY ','
RECORDS TERMINATED BY ';'
USING (CONSTANT 'insert into otbl values (',
CHAR ENCLOSED BY '"' IFNULL='NULL',
INTEGER EXTERNAL IFNULL='NULL',
CHAR ENCLOSED BY '"' IFNULL='NULL',
CONSTANT ')')
SELECT ITEM, QUANTITY, STATUS FROM INVENTORY;
```

### Result file

```
insert into otbl values ('plug',12,NULL);
insert into otbl values ('cable',NULL,'out');
```

### Notes:

- The constant `insert into otbl values (` is placed before the data fields and the constant `)` is placed after the data fields (constant\_specs).
- All data fields (except the last) are terminated by commas (FIELDS clause).
- Each record is terminated by a semicolon (RECORDS clause).
- The first and third data fields are enclosed by single quotes (') because their datatype\_specs contain a delimiter\_spec. A delimiter\_spec in a datatype\_spec overrides and/or supplements a FIELDS clause.
- The NULL keyword in the second record is terminated by a comma because the IFNULL string contains the delimiter. Delimiters are not added to an IFNULL string, so you must include them.

## Creating fixed-length data fields with NULL flags at the start of records

### UNLOAD statement

```
UNLOAD
RECORDS TERMINATED BY ';'
USING (:n1 CONSTANT 'F',
:n2 CONSTANT 'F',
:n3 CONSTANT 'F',
CHAR(10) IFNULL :n1 = 'T',
INTEGER EXTERNAL(10) IFNULL :n2 = 'T',
CHAR(10) IFNULL :n3 = 'T')
SELECT ITEM, QUANTITY, STATUS FROM INVENTORY;
```

### Result file

```
FFTplugbbbbbbbbbbbbbb12bbbbbbbbbbb;
FTFcablebbbbbbbbbbbbbboutbbbbbbb;
```

### Notes:

- The NULL flags (FFT and FTF) identify which data fields are NULL. T indicates a NULL field, so in the first record the third data field is NULL and in the second record the second data field is NULL.
- When using :field\_names (such as :n1), the IFNULL string (such as IFNULL :n1='T') is stored instead of the CONSTANT string (such as :n1 CONSTANT 'F').
- In fixed-length data, INTEGER EXTERNAL data is right-justified (12 is right-justified in the second field) and CHAR data is left-justified (out is left-justified in the third field).
- If a fixed-length data field is NULL, the FileTek FTP Data Unloader inserts blanks in that position. So in the first record, the Unloader inserted 10 blanks in the third field and in the second record the Unloader inserted 10 blanks in the second field.

## Generating data fields with NULL flags

**UNLOAD statement**     UNLOAD  
                               FIELDS CHAR NULLFLAGS TERMINATED BY ','  
                               RECORDS TERMINATED BY ';'   
                               SELECT ITEM, QUANTITY, STATUS FROM INVENTORY;

**Result file**         FFTplug,12,;  
                               FTFcable,,out;

Note:

The FileTek FTP Data Unloader converted all data fields to CHAR and inserted the NULL flags (T for NULL and F for NOT NULL) at the beginning of each result record.

## Adding keywords but omitting them for NULL data

**UNLOAD statement**     UNLOAD  
                               FIELDS TERMINATED BY ','  
                               RECORDS TERMINATED BY ';'   
                               USING (:k1 CONSTANT 'Item=', CHAR IFNULL :k1 = "",  
                               :k2 CONSTANT 'Quantity=', CHAR IFNULL :k2 = "",  
                               :k3 CONSTANT 'Status=', CHAR IFNULL :k3 = "",  
                               BLOB\_FILE HOST 'prod1' PATH '/unload/' FILENAME 'photo%d.jpg'  
                               USER 'juliette/romeo')  
                               SELECT \* FROM INVENTORY;

**Result file**         Item=plug,Quantity=12,,photo1.jpg;  
                               Item=cable,,Status=out,photo2.jpg;



Notes:

- The keywords `Item=`, `Quantity=`, and `Status=` are inserted before data fields that are not NULL.
- The IFNULL strings contain empty double quotes (""), so the keywords (`constant_specs`) are omitted for NULL data.
- The adjacent termination delimiters for the NULL data could be misinterpreted in a subsequent load. Enclosing the data with enclosure delimiters would resolve this.
- The FileTek FTP Data Unloader inserted the generated LOB file names in the result data, for instance, `photo1.jpg` and `photo2.jpg`.

#### LOB files

The FileTek FTP Data Unloader placed each photo in a LOB file in the unload directory on a remote host named `prod1` using the user name `juliette` and the password `romeo`. The name of the first file is `photo1.jpg` and the name of the second file is `photo2.jpg`.

## Unloading LOB data to the result file

#### UNLOAD statement

```
UNLOAD
FIELDS NULLFLAGS
SELECT ITEM, PHOTO FROM INVENTORY;
```

#### Result file

Assume each photo is 64K, which is too long to fit into a single 32K record. The sample data stream, in FTP var format, shows how each BLOB is organized into multiple 32K-1byte records [hex representations in brackets]. The *b* represents blanks and *LOB data* . . . represents the BLOB.

```
[000c]FFplugbbbbbb
[05cc][000000000000005c4]LOB data . . .
[000c]FFcablebbbbbb
[035c][00000000000000354]LOB data . . .
```

## 2

### **Preparing an UNLOAD statement**

---

Example UNLOAD statements

## Using FTP to unload data

This chapter describes the FTP commands you use to:

- Log in the StorHouse FTP server
- Set the transfer type
- Transfer the control file
- Get the result data
- Display help for StorHouse FTP server commands
- Log off the StorHouse FTP server

The commands in this chapter represent a SunOS implementation and command line client FTP tool. Your command formats may differ depending on your host environment. Refer to the documentation provided by your client FTP tool for command usage. Also, if your client FTP tool has a GUI interface, be sure to read the GUI considerations on page 3-8 for other guidelines about submitting FTP commands to the StorHouse FTP server.

If your client supports it, FTPS can be used for secure, encrypted communication with the StorHouse server.

## 3

## Using FTP to unload data

FTP commands for unloading data

## FTP commands for unloading data

You enter the following user-level FTP commands with your client FTP tool to unload data. If your implementation differs from SunOS, see the corresponding server-level FTP command in the table to determine the correct user-level FTP command for your implementation.

User-level	Server-level	Description
ascii	TYPE A	Sets the transfer type to ASCII. You can also use the type ascii command.
binary	TYPE I	Sets the transfer type to BINARY. You can also use the type image command.
bye	QUIT	Logs off the StorHouse FTP server and quits FTP.
close	QUIT	Logs off the StorHouse FTP server but doesn't quit FTP.
get	PORT RETR	Retrieves the result data, placing it in a file on your computer or piping it to another program.
open	USER PASS	Logs into the StorHouse FTP server at alternate port 1985.
put	PORT STOR ALLO	Starts an unload and transfers the control file.
quit	QUIT	Logs off the StorHouse FTP server and quits FTP.
remotehelp	HELP	Provides a list of server-level FTP commands, or when followed by a command name, a definition for a specific command.
type ascii	TYPE A	Sets the transfer type to ASCII.
type image	TYPE I	Sets the transfer type to BINARY.

## The put command

Use the FTP put command to start an unload and transfer the control file. The format is:

put local-file keyword=value[,keyword=value]...

put argument	Description
put	(required) Command verb.
local-file	(required) Name of the control file.
keyword=value	(required) Customized keywords and values necessary to unload data from StorHouse. This series of keywords and values replaces the remote file name argument on the FTP put command.
[load_type=]	(required) Type of operation; transfers the control file. Valid value: unload
dbn[ame]=	(required) Name of the StorHouse database that contains the data you're unloading. The database name is case sensitive.
sql_ccsid=	(optional) Character set of the control statements in the control file. Valid values: <ul style="list-style-type: none"><li>■ 819 for ISO 8859-1 (default)</li><li>■ 500 for EBCDIC</li><li>■ 850 for PC</li></ul>
data_ccsid=	(optional) Default character set of the result data. See "Specifying the character set of result data" on page 2-6 to override this value. Valid values: <ul style="list-style-type: none"><li>■ 819 for ISO 8859-1 (default)</li><li>■ 500 for EBCDIC</li><li>■ 850 for PC</li></ul>
fixed=	(optional) Result data is to be composed of fixed-length records with the specified record length (in bytes), for example, fixed=80. The maximum record length is 32767. If you omit the fixed or var keyword, the FileTek FTP Data Unloader interprets the result data as text records. If you include the fixed keyword, you must set the transfer type to BINARY.

## 3

## Using FTP to unload data

FTP commands for unloading data

put argument	Description
var	(optional) Result data is to be composed of variable-length binary records. Each data record will be preceded by the shortword length (in nvk format). If you omit the var or fixed keyword, the FileTek FTP Data Unloader interprets the result data as text records. If you include the var keyword, you must set the transfer type to BINARY.
nvk=	(optional) Host hardware type for interpreting lengths and values of binary data. See "Considerations for binary result data" on page 1-11 for more information about the native values key. Valid values: <ul style="list-style-type: none"> <li>■ SPARC (default)</li> <li>■ IBM</li> <li>■ DOS</li> <li>■ VAX</li> </ul>

## The get command

Use the FTP get command to receive result data in a file on your computer or to pipe the output to another program. The format is:

```
get keyword=value[,keyword=value]... local-file
```

get argument	Description
get	(required) Command verb.
keyword=value	(required) Customized keywords and values necessary to unload data from StorHouse. This series of keywords and values replaces the remote file name argument on the FTP get command.
[load_type=]	(required) Type of operation; requests the result data. Valid value: data
fixed=	(optional) Result data is to be composed of fixed-length records with the specified record length (in bytes), for example, fixed=32767. The maximum record length is 32767. If you omit the fixed or var keyword, the FileTek FTP Data Unloader interprets the data as text records. If you include the fixed keyword, you must set the transfer type to BINARY.

get argument	Description
var	(optional) Result data is to be composed of variable-length binary records. Each data record is preceded by the shortword length (in nvk format). If you omit the var or fixed keyword, the FileTek FTP Data Unloader interprets the data as text records. If you include the var keyword, you must set the transfer type to BINARY.
local-file	(required) Name of the result file to contain result data and any LOB file names. If you're piping the output to another program, however, specify the program name and any program arguments here.

Note the following:

- The var and fixed keywords are allowed on both the put and get commands, but you only need to specify one keyword on one command, not both. If you specify different keywords on both commands (for example, var on put and fixed on get), then the value specified for get overrides the value specified for put.
- The var and fixed keywords apply to the local-file, not to any LOB files.

## Guidelines for specifying put and get commands

When specifying a put or get command, note the following:

- Type the command verb—put or get—according to the case conventions of your client FTP tool. Most tools require that you enter FTP command verbs in lowercase characters.
- Type keywords in any order and in any case because they are not case sensitive. For instance, you can specify nvk, NVK or Nvk.

**3**

## Using FTP to unload data

---

### Sample session

- You can omit the part of the keyword shown in brackets [ ]. For instance, you can omit the [load\_type=] keyword entirely, and you can specify dbn for dbn[ame].
- No spaces are allowed between the equal sign and the keyword or between the equal sign and the value. For example: dbn=database1 (correct), but nvk = IBM (incorrect, contains spaces).
- You can type values in any case with one exception: the database name is case sensitive.
- Separate each keyword and value pair with a comma and no intervening spaces, and be sure the entire put or get command fits on one line. For example: get load\_type=data,var (correct), but get load\_type = data, var (incorrect, contains spaces).
- Use a least one space to separate the keyword=value sets from the local file name.
- If a value contains special characters, enclose it in single quotes. For example, if a database name is data\$, enclose the value in single quotes like this: dbn='data\$'

## Sample session

The following session illustrates the user-level FTP commands you enter to log in, transfer a control file, get the result data, and log off. It also shows some of the replies you receive during an unload.

Start FTP and log in. → fltksun.1> ftp sth1 1985

Connected to sth1.

220 sth1 LD/FTP server ... ready. Enter StorHouse account.



Enter your → Name (jjw:juliette): juliette  
StorHouse account  
ID. 331 Password required for StorHouse account juliette.

Enter your → Password:  
StorHouse  
password. 230 StorHouse User (account) juliette logged in.

Set the transfer type. → ftp> binary

200 Type set to I.

Transfer the control → ftp> put control.inp unload,dbn=database1  
file.

200 PORT command successful.  
150 Opening BINARY mode data connection.  
226 \*\*\*\*\* OK, now get data\*\*\*\*\*

Get the result data. → ftp> get data,var file.out

200 PORT command successful.  
150 Opening BINARY mode data connection.  
226- %L-I-XLDINFO, \09-MAY-2000:13:38:22 2163 total records processed\  
226- %WORLD-S-XWOK, A request was successfully completed.  
226 \*\*\*\*\* Unload operation/request finished \*\*\*\*\* sqlcode=<0>

Log off. → ftp> quit

221 Goodbye.

## Secure FTP (FTPS)

StorHouse/RM supports FTPS (i.e., explicit FTPS as per RFC 4217). FTPS can be used to encrypt all communication between the client and server machines, including the login. The server-level commands are AUTH SSL, AUTH TLS, PBSZ, and PROT.

FTPS is usually selected via a GUI menu. To use the data loader FTPS functionality, you must set up a server certificate. The certificate can be self-signed, using a toolset like OpenSSL, or it can be established by a certificate authority such as Verisign, depending on your requirements and your security environment.

## Compression

StorHouse/RM supports MODE Z compression, which is usually selected using a GUI checkbox. FileTek recommends using MODE Z compression for all FTPS transfers to reduce the CPU load for encryption.

## GUI and client considerations

If your client FTP tool has a GUI interface, then you complete dialog boxes or click buttons rather than enter commands at a command line. This section contains guidelines for using a GUI client FTP tool to unload data. It also describes other issues about client FTP tools.

### Opening a session at an alternate port

The StorHouse FTP server listens at alternate port 1985. Your client FTP tool must provide a way for you to specify an alternate port. Be sure to start an FTP session at the alternate port instead of the default port.

## Specifying keywords for a remote file name

When your GUI client FTP tool asks for a remote file name, enter the put and get keywords and values instead. Note this:

- Don't type the command verb (put or get); simply supply its keywords and values.
- Type a semicolon at the end of your entry if your client FTP tool lets you specify a directory instead of a file name.
- Use all other put and get command conventions described on page 3-5.

Below is an example:

**Enter remote file name:**

## Listing the remote directory

Some GUI tools automatically list the remote directory (using the PWD, NLST, and LIST server-level FTP commands) before and after a file transfer. The FileTek FTP Data Unloader lets your tool do this, but a remote directory and listing do not apply to this system so responses are fabricated.

## Using automatic binary mode

Some client FTP tools automatically set the transfer type to BINARY (after issuing a SYST server-level FTP command) when connecting to a remote computer. The FileTek FTP Data Unloader supports BINARY mode, so this shouldn't be a problem and you don't have to worry about setting the transfer type with the binary or type image commands.

## Displaying remote server messages

If your client FTP tool provides an option for displaying server messages, be sure to use that option. This is the only way to receive messages from the StorHouse FTP server and to know whether your unload succeeds or fails.

## Globbering remote file names

Some UNIX client FTP tools preprocess (glob) file names in put or get commands before sending the command to the server. If your tool globs remote file names, you may have to use quotes or escape characters if the put keyword string contains the special characters listed in the following table. Unless specified, quotes can be single quotes or double quotes but not back quotes.

Special character	UNIX meaning/usage	Action needed if used in a keyword string
\$	Take value of variable	Enclose string in single quotes
!	History substitution	Use escape (\) to disable substitution
;	End of command	Don't use; reserved character
\	Escape character	Use another escape character or quotes
&	Miscellaneous	Use escape character or quotes
< >	Redirection	Use escape character or quotes
()	Grouping	Use escape character or quotes
^	History substitution	Use escape character or quotes
@	Execute shell file	Use escape character or quotes
~	Home directory	Use escape character or quotes
`	Execute command script	Use escape character or quotes
%	Regular expression symbol	Use escape character or quotes
?	Regular expression symbol	Use escape character or quotes

Special character	UNIX meaning/usage	Action needed if used in a keyword string
[ ]	Regular expression symbols	Use escape character or quotes
{ }	Regular expression symbols	Use escape character or quotes
*	Regular expression symbol	Use escape character or quotes

While the globbing that's done by the client FTP tool may not be identical to that done by a UNIX shell, it should be similar (at least for UNIX implementations). To see if a keyword string will be preserved correctly, you could try testing the string with the echo command, which simply prints whatever text was entered after doing any globbing dictated by the shell.

To avoid globbing altogether, you could turn off globbing with your client FTP tool. Another option is to enclose the keyword values in single quotes. This would protect all special characters except for the exclamation point (!) and an embedded single quote.

## Avoiding data timeouts in clients

Ping functionality compensates for data channel timeouts enforced by some FTP clients once an unload has been started and no data is seen for a period of time (for example, five minutes for AIX 5.3). StorHouse/RM provides ping functionality with the FTP server SITE PINGINT and SITE PINGBUF commands.

Command	Description
SITE PINGINT [n]	Sets the ping interval in seconds. If n is omitted, the command reports the current ping interval. A value of 0 disables pinging.
SITE PINGBUF [n]	Sets the size (in bytes) of the ping buffer. If n is omitted, the command reports the current ping buffer size. A value of 0 disables pinging.

## Logging into the StorHouse FTP server

Use the UNIX `ftp` command to log into the StorHouse FTP server. If your client FTP tool does not let you specify a server name and alternate port on the `ftp` command, then use the FTP open command. In either case, you need the name of your StorHouse system and your StorHouse account ID and password to log in. This StorHouse account must have the `SQLEXECUTE` privilege and the `SELECT` privilege for the table(s) being unloaded.

### ▼ To log in with the `ftp` command

1. At the UNIX prompt, type the following (replace `sth_name` with the name of your StorHouse system) and press `Enter`:

```
ftp sth_name 1985
```

For example (where `sth_name` is `alpha1`):

```
ftp alpha1 1985
```

Note that 1985 is the alternate port at which the StorHouse FTP server listens.

2. At `Name:`, type your StorHouse account ID and press `Enter`.
3. At `Password:`, type your StorHouse account password and press `Enter`.

### ▼ To log in with the `open` command

1. At the UNIX prompt, type `ftp` and press `Enter`.
2. At the `ftp` prompt, type the following (replace `sth_name` with the name of your StorHouse system) and press `Enter`:

```
open sth_name 1985
```

For example (where `sth_name` is `alpha1`):

open alpha1 1985

3. At Name:, type your StorHouse account ID and press **Enter**.
4. At Password:, type your StorHouse account password and press **Enter**.

## Setting the transfer type

Before transferring a control file, set the transfer type to ASCII or BINARY. Note the following:

- You can transfer the control file in ASCII or BINARY mode (if on a UNIX host).
- You *must* receive fixed-length data and variable-length data in BINARY mode.
- You can receive text data in ASCII or BINARY mode; but in BINARY mode, text lines will be terminated by ASCII newlines (\n).
- When you receive text data in ASCII mode, the proper record/line terminator for your host will be included with the result data.
- When transferring in ASCII mode from a non-ASCII machine (for example, EBCDIC), your client FTP tool will convert the data to the local character set. Don't specify an alternate character set using the data\_ccsid keyword on the FTP put command or the CHARACTERSET clause or the CHARSET keyword on the UNLOAD statement.
- If you are unloading LOB values, choose a transfer type applicable for the data. For instance, if you unload a CLOB column as VARCHAR result data, specify the var keyword and use the BINARY transfer type.

**3****Using FTP to unload data**

---

## Transferring the control file

- If you are unloading LOB data to separate LOB files, the transfer type does not apply those files. The transfer type applies to the control file and any result file containing other table data and LOB file names.

The default transfer type is ASCII, so if that's the one you want, you don't have to do this procedure. But keep in mind that if your previous transfer type was BINARY and you used the close and open commands, your client FTP tool doesn't reset the transfer type to the default ASCII. You should always set the transfer type after logging in the StorHouse FTP server.

▼ **To set the transfer type to ASCII**

At ftp>, enter ascii or type ascii and press .

▼ **To set the transfer type to BINARY**

At ftp>, enter binary or type image and press .

## Transferring the control file

Use the FTP put command to start an unload and transfer the control file. You must supply the load\_type (unload) and the database name. If you omit all other keywords and values, and if you omit the fixed or var keyword on the get command, then the FileTek FTP Data Unloader assumes you're unloading text data and uses the following defaults:

- sql\_ccsid=819 (for ISO 8859-1)
- data\_ccsid=819 (for ISO 8859-1)
- nvk=SPARC

If you don't want to use the listed defaults, be sure to include the appropriate keywords and values on the put or get command.



### ▼ To transfer the control file

1. At ftp>, type the following and press **Enter**:
  - put
  - followed by the name of the control file you're transferring
  - followed by the load\_type value unload
  - followed by dbn= and the name of the StorHouse database
  - followed by any other appropriate put keywords and values

For example, to specify a control file called control.inp and a database called database1, type:

```
ftp> put control.inp unload,dbn=database1
```

2. Your next step depends on the result of the control file transfer. If:
  - You receive an error reply (one that starts with a 4 or 5), then fix the error and start over.
  - The transfer completed successfully (you receive message 226- %WORLD-S-XWOK, A request was successfully completed) and the result data is in a StorHouse VRAM file (your UNLOAD statement contains an OUTFILE clause), then the unload has completed.
  - The transfer completed successfully and you received message 226 OK, now get data, then issue the get command to receive the result data.

## Getting the result data

After transferring the control file, use the FTP get command to receive the result data. You do not issue a get command when you unload data to a StorHouse VRAM file. When issuing a get command, note the following:

**3**

**Using FTP to unload data**

---

Getting the result data

- You must specify the `load_type`, which is `data`.
- If you omit the `fixed` and `var` keywords on the `put` and `get` commands, then the FileTek FTP Data Unloader assumes you're unloading text data.
- If you specify the `fixed` or `var` keyword on both the `put` and `get` commands, then the keyword and value specified on `get` overrides the one on `put`.
- If you include the `fixed` or `var` keyword, the transfer type must be `BINARY`.
- The `fixed` and `var` keywords and transfer types do not apply to LOB files.
- If a result file already contains data, your client FTP tool may overwrite the existing data, but the action is implementation-dependent.

▼ **To get the result data**

1. At `ftp>`, type the following and press `Enter`:

- `get`
- followed by the `load_type` value `data`
- if appropriate, followed by the `fixed` keyword and value or the `var` keyword
- followed by the name of the file or program to receive the result data

For example, to specify a result file called `file.out` to contain fixed-length data with a record length of 50, type:

```
ftp> get data,fixed=50 file.out
```

Or, to pipe the output to a program called `myprog.sh`, type:

```
ftp> get data,fixed=50 "| myprog.sh"
```

2. Your next step depends on the result of the file transfer. If:

- You receive an error message, then start over from the beginning (transfer the control using put, then receive the data using get) after correcting the error.
- The transfer completed successfully, you can log off the StorHouse FTP server, or perform another unload, or load data with the FileTek FTP Data Loader.

## Displaying help for StorHouse FTP server commands

Use the `remotehelp` command to display a list of server-level FTP commands as well as a definition for a specific command.

### ▼ To list all StorHouse FTP server commands

At `ftp>`, type `remotehelp` and press `Enter`.

The system displays a list of all server-level FTP commands. The FileTek FTP Data Unloader does not support those commands marked with an asterisk.

### ▼ To display help for a specific command

At `ftp>`, type `remotehelp` followed by the server-level FTP command name and press `Enter`.

Some of the server-level commands are: `ALLO`, `HELP`, `PASS`, `PORT`, `QUIT`, `RETR`, `SITE DEBUG`, `SITE PINGINT`, `SITE PINGBUE`, `STOR`, `STAT`, `SYST`, `TYPE A`, `TYPE I`, `USER`. For example, to display help for the `put` command, type `remotehelp stor`.

## Logging off the StorHouse FTP server

Use the bye or quit command to log off the StorHouse FTP server and quit FTP. Use the close command to log off the StorHouse FTP server but stay in FTP. If you use the close command and want to unload more data, be sure to set the appropriate transfer type before you transfer the control file.

▼ **To log off the StorHouse FTP server and quit FTP**

At ftp>, type bye or quit and press .

▼ **To log off the StorHouse FTP server and stay in FTP**

At ftp>, type close and press .

# University of California copyright, conditions, disclaimer

Some of the source code for the StorHouse FTP server is derived from the source code used in the University of California, Berkeley FTP tool.

Copyright (c) 1985, 1988, 1993, 1994 Regents of the University of California.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products

derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Index

## Numerics

65535 CCSID 2-41

## A

aborting an un load 1-6

account ID, StorHouse 1-5

ALLO FTP command 3-2

alternate port 3-8

AND keyword 2-15

appending a character to a record 2-17

ascii FTP command 3-2, 3-14

ASCII transfer type 1-4

AUTH SSL secure ftp command 3-7

AUTH TLS secure ftp command 3-7

automatic binary mode 3-9

avoiding data timeouts in clients 3-11

## B

BIGINT data type 2-23

BINARY data type 2-24, 2-41

binary data, interpreting 1-11

BINARY EXTERNAL data type 2-25

binary FTP command 3-2, 3-14

binary mode, automatic 3-9

BINARY transfer type 1-4

blanks, number of 2-49

BLOB data type 2-26

BLOB\_FILE data type 2-27

BY keyword 2-14, 2-18

bye FTP command 3-2

byte order 1-12

## C

case in control file 2-2

CCSID 65535 2-41

## Index

### C

- CCSIDs 2-7
- CHAR keyword 2-14
- CHARACTER data type 2-28
- character set names 2-7
- CHARACTERSET clause 2-7
- CHARSET keyword 2-40
- clauses
  - CHARACTERSET 2-6
  - CHARSET 2-40
  - CONSTANT 2-21
  - ESCAPED BY 2-17
  - FIELDS 2-9
  - FILENAME 2-43
  - HOST 2-42
  - IFNULL 2-48
  - OUTFILE 2-8
  - OVERWRITE 2-30
  - PATH 2-45
  - POSITION 2-47
  - RECORDS 2-17
  - RECORDS NOT TERMINATED 2-19
  - USER 2-46
  - USING 2-19
- client FTP software 1-2, 1-4
- CLOB data type 2-29
- CLOB\_FILE data type 2-30
- close FTP command 3-2
- commands
  - FTP server
    - ALLO 3-2
    - HELP 3-2
    - LST 3-9
    - NLST 3-9
    - PASS 3-2
  - FTP user
    - ascii 3-2
    - binary 3-2
    - bye 3-2
    - close 3-2
    - get 3-2
    - open 3-2
    - put 3-2
    - quit 3-2
    - remotehelp 3-2
    - type 3-2
  - comments in control file 2-2
  - compression 3-8
  - Concepts and Facilities Manual, StorHouse xii
  - constant\_spec, definition 2-19
  - control file
    - conventions 2-2
    - definition 2-1
    - transferring 3-14
  - conventions, control file 2-2
  - conversions, data type 2-38
  - CRLF 1-4
  - PORT 3-2
  - PWD 3-9
  - QUIT 3-2
  - RETR 3-2
  - STOR 3-2
  - SYST 3-9
  - TYPE A 3-2
  - TYPE I 3-2
  - USER 3-2



## D

- data field, description 1-9
- data record 1-9
- data type specification 2-22
- data types
  - BIGINT 2-23
  - BINARY 2-24
  - BINARY EXTERNAL 2-25
  - BLOB 2-26
  - BLOB\_FILE 2-27
  - CHARACTER 2-28
  - CLOB 2-29
  - CLOB\_FILE 2-30
  - conversions 2-38
  - DATE EXTERNAL 2-31
  - DECIMAL 2-32
  - DECIMAL EXTERNAL 2-33
  - DOUBLE 2-33
  - FLOAT EXTERNAL 2-34
  - FLOAT or REAL 2-34
  - INTEGER 2-34
  - INTEGER EXTERNAL 2-35
  - SMALLINT 2-35
  - TIME EXTERNAL 2-36
  - TIMESTAMP EXTERNAL 2-37
  - VARBINARY 2-37
  - VARCHAR 2-38
- DATE EXTERNAL data type 2-31
- dbname keyword 3-3
- DECIMAL data type 2-32
- DECIMAL EXTERNAL data type 2-33
- definitions
  - constant\_spec 2-19
  - control file 2-1
  - data field 1-9
  - data record 1-9
  - enclosed data 2-11
  - fixed position 2-47
  - LOB file 1-8
  - native data type 1-11
  - position\_spec 2-19
  - record formats 1-9
  - record terminator 2-17
  - relative position 2-47
  - result data type 2-38
  - result file 1-8
  - source data types 2-38
  - StorHouse FTP server 1-2
  - terminated data 2-10
  - transfer types 1-4
  - unload 1-1
- DELIMITER keyword 2-17
- delimiter specification
  - AND keyword 2-15
  - BY keyword 2-14
  - ENCLOSED keyword 2-15
  - hexadecimal delimiter 2-14
  - with a data type specification 2-41
- describing data fields 2-17, 2-19
- directory path 1-14
- displaying help for FTP server commands 3-17
- displaying remote server messages 3-10
- DOUBLE data type 2-33

## E

- EBCDIC character set 2-7
- enclosed data 2-11

## Index

### F

ENCLOSED keyword in FIELDS clause 2-15

engine 1-8

examples

- CHARACTERSET clause 2-7

- CHARSET keyword 2-41

- CONSTANT clause 2-22

- control statements 2-53

- delimiter specification for a data field 2-41

- ESCAPED BY 2-17

- FIELDS clause 2-15, 2-16

- FILENAME clause 2-43

- get command 3-16

- HOST clause 2-42

- IFNULL keyword 2-50

- OUTFILE clause 2-8

- POSITION keyword 2-47

- put command 3-15

- RECORDS clause 2-18

- sample unload session 3-6

- SELECT statement 2-52

- USER clause 2-46

- USING clause 2-51

expression, SELECT 2-52

extractor 1-8

## F

field name 2-22

field specification 2-20

FIELDS clause

- BY keyword 2-14

- ENCLOSED keyword 2-15

- TERMINATED keyword 2-14

file name, LOB 1-14, 2-43

File Transfer Protocol (FTP) ix

FILENAME clause 2-43

fixed keyword 3-3, 3-4

fixed-length record format 1-10

FLOAT (REAL) data type 2-34

FLOAT EXTERNAL data type 2-34

format conventions, SQL 2-2

formats

- CHARACTERSET clause 2-7

- CHARSET keyword 2-40

- CONSTANT clause 2-22

- ESCAPED BY clause 2-17

- FIELDS clause 2-13

- get command 3-4

- IFNULL keyword 2-49

- OUTFILE clause 2-8

- POSITION keyword 2-47

- put command 3-3

- RECORDS clause 2-18

- RECORDS NOT TERMINATED clause 2-19

- SELECT statement 2-52

- UNLOAD statement 2-3

- USING clause 2-19

formatting result data 2-9

ftp command 3-12

FTP commands

- server

  - ALLO 3-2

  - HELP 3-2

  - LIST 3-9

  - NLST 3-9

  - PASS 3-2

  - PORT 3-2

  - PWD 3-9

- QUIT 3-2
- RETR 3-2
- SITE PINGBUF 3-11
- SITE PINGINT 3-11
- STOR 3-2
- SYST 3-9
- TYPE A 3-2
- TYPE I 3-2
- USER 3-2

**user**

- ascii 3-14
- binary 3-2
- bye 3-2
- close 3-2
- get 3-2, 3-4
- open 3-2
- put 3-3
- quit 3-2
- remotehelp 3-2
- type 3-2

FTP session 1-3

## G

**get command**

- conventions 3-5
- description 3-2
- format 3-4

globbing remote file names 3-10

**GUI considerations**

- globbing remote file names 3-10
- listing the remote directory 3-9
- opening a session at an alternate port 3-8
- specifying keywords for a remote file name 3-9
- using automatic binary mode 3-9

## H

HELP FTP command 3-2

HOST clause 2-42

host variables 2-52

## I

IFNULL keyword 2-48

IMAGE transfer type 1-4

INTEGER data type 2-34

INTEGER EXTERNAL data type 2-35

IP address 2-42

ISO 8859-1 character set 2-7

## K

**keywords, get command**

- fixed 3-4
- load\_type 3-4
- var 3-5

**keywords, put command**

- data\_ccsid 3-3
- dbname 3-3
- fixed 3-3
- load\_type 3-3
- nvk 3-4
- sql\_ccsid 3-3
- var 3-4

## L

- large objects (LOBs)
  - BLOB data type 2-26
  - BLOB\_FILE data type 2-27
  - CLOB data type 2-29
  - CLOB\_FILE data type 2-30
  - considerations 1-13
  - creating LOB files on a remote system 1-15
  - creating LOB files on your client computer 1-14
  - in-line LOB 1-13
  - overwriting LOB files 2-45
  - result file 1-8
  - specifying a file name for LOB files 2-43
  - specifying a remote host name for LOB data 2-42
  - specifying a user name and password 2-45
  - transfer type 1-13, 3-13
  - unloading with other result data 1-13
- limits 1-8
- line terminator 2-17
- listing the remote directory 3-9
- literal strings in control file 2-2
- LOAD DATA statement
  - syntax
    - ESCAPED BY 2-17
- load\_type keyword 3-3, 3-4
- locks 1-8
- logging into the StorHouse FTP server 3-12
- logging off the StorHouse FTP server 3-18
- LST FTP command 3-9

## M

- messages 1-7
- Messages and Codes Manual, StorHouse xii
- MODE Z compression 3-8

## N

- native data types 1-11
- native values key 1-12
- NLST FTP command 3-9
- NONE keyword 2-17
- NULL values 2-48
- NULLFLAGS keyword 2-14
- num 2-22
- nvk keyword 3-4

## O

- open FTP command 3-2, 3-12
- opening a session at an alternate port 3-8
- OVERWRITE clause 2-30
- overwriting LOB files 2-45

## P

- PASS FTP command 3-2

password, StorHouse 1-5  
PBSZ secure ftp command 3-7  
PC character set 2-7  
piping output 1-10, 3-16  
PORT FTP command 3-2  
POSITION keyword 2-47  
position\_spec, definition 2-19  
printf format 2-43, 2-44  
privileges required to unload 1-5  
PROT secure ftp command 3-7  
put command  
    conventions 3-5  
    description 3-2  
    example 3-15  
    format 3-3  
PWD FTP command 3-9

## Q

query 2-52  
QUIT FTP command 3-2  
quit FTP command 3-2

## R

record formats  
    fixed-length 1-10  
    text 1-9  
    variable-length 1-11

record terminator 2-17  
RECORDS clause 2-18  
RECORDS NOT TERMINATED clause 2-19  
remote directory 3-9  
remote file name 3-9  
remote host 1-15, 2-42  
remotehelp FTP command 3-2  
replies, FTP 1-7  
reply code, StorHouse 1-7  
requirements, client FTP tool 1-4  
result data 1-8  
result data types 2-38  
result file 1-8  
RETR FTP command 3-2  
RFC 959 xii

## S

sample unload session 3-6  
SCAN privilege 1-5  
secure ftp commands  
    AUTH SSL 3-7  
    AUTH TLS 3-7  
    PBSZ 3-7  
    PROT 3-7  
SELECT privilege 1-5  
SELECT statement 2-52  
semicolon in control file 2-2

## Index

---

### T

session, FTP 1-3  
setting the transfer type 3-13  
SITE PINGBUF FTP command 3-11  
SITE PINGINT FTP command 3-11  
SMALLINT data type 2-35  
SMALLINT in VAR-type data fields 1-12  
source data types 2-38

specifying  
    a constant value 2-21  
    keywords for a remote file name 3-9  
    the character set of result data 2-6

SQL format conventions 2-2

sql\_ccsid keyword 3-3

SQL\_SESSIONS parameter 1-8

SQLEXECUTE privilege 1-5

status code 1-7

STOR FTP command 3-2

StorHouse  
    account information 1-5  
    description ix  
    messages 1-7  
    VRAM file 2-8

StorHouse Database Administration Guide xii

StorHouse documentation  
    Concepts and Facilities Manual xii  
    Messages and Codes Manual xii

StorHouse FTP server  
    description 1-2  
    logging into 3-12  
    logging off 3-18

StorHouse/Control Center, description x

StorHouse/RM, description x

StorHouse/SM, description ix

string 2-22

symbolic name 1-7

system parameter for unloading 1-8

### T

tables  
    ESCAPED BY clause 2-17

TCP/IP 1-2

terminated data 2-10

TERMINATED keyword in FIELDS clause 2-14

terminator, line 2-17

text record format 1-10

TIME EXTERNAL data type 2-36

TIMESTAMP EXTERNAL data type 2-37

transfer modes 1-4

transfer types 1-4, 3-13

transferring the control file 3-14

Transmission Control Protocol (TCP) 1-2

TYPE A FTP command 3-2

type ascii FTP command 3-14

type FTP command 3-2

TYPE I FTP command 3-2

type image FTP command 3-14

## U

### UNLOAD

examples 2-53

format 2-3

unload, description 1-1

USER clause 2-46

USER FTP command 3-2

user name and password 2-46

## V

var keyword 3-4, 3-5

VARBINARY data type 2-37, 2-41

VARCHAR data type

actual length 1-12

description 2-38

starting column 2-47

variable-length record format 1-11

VAR-type data type considerations 1-12

VRAM file 2-8

## W

WHITESPACE keyword 2-14, 2-18