



FileTek FTP  
Data Loader Manual  
StorHouse/RM Release 3.2

Publication Number  
900115 Rev. J

August 19, 2002

---

**FileTek**



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.

This publication Copyright © 1996-2002 by FileTek, Inc., Rockville, MD  
Publication Number: 900115 Rev. J

**NOTE: U.S. GOVERNMENT USERS**

**Restricted Rights Legend**

Use, duplication or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or the Commercial Computer Software - Restricted Rights clause at 48 CFR 52.227-19, as applicable. Unpublished-rights reserved under the copyright laws of the United States. 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. Other patents pending.

# Contents

<b>Welcome .....</b>	<b>xiii</b>
StorHouse family of products .....	xiii
StorHouse/SM .....	xiii
StorHouse/RM .....	xiv
Control Center .....	xiv
Purpose of this document .....	xiv
Audience .....	xv
What's in this guide .....	xv
Conventions .....	xvi
For more information .....	xvi
For quick reference .....	xvii
 <b>Chapter 1: The basics .....</b>	 <b>1-1</b>
Loading features .....	1-1
How it works with FTP .....	1-3
FTP session .....	1-5
Transfer types .....	1-5
What you need .....	1-6
Your client FTP tool .....	1-6
StorHouse account information .....	1-7
Before loading .....	1-8

The load data process .....	1-10
Loads and segments .....	1-11
Segment size .....	1-13
Segment replacement .....	1-14
Segment merge .....	1-14
Loads and subspaces .....	1-14
Default selection of subspaces .....	1-15
When there is one subspace for all component types .....	1-16
When there is one subspace for each component type .....	1-16
When there are multiple subspaces for each component type .....	1-16
When indexes or LOB columns are assigned to multiple user tablespaces ..	1-18
Explicit selection of subspaces .....	1-19
When loading one segment .....	1-19
When loading multiple segments .....	1-20
When there are multiple indexes of the same type .....	1-22
When indexes are assigned to different user tablespaces .....	1-24
Rotation among subspaces .....	1-27
When there are multiple subspaces for each component type .....	1-28
When component types share a subspace .....	1-29
When indexes or LOB columns are assigned to different user tablespaces ..	1-31
Loads and indexes .....	1-34
Load parallelism .....	1-35
Loading different tables in one load .....	1-36
Loading multiple segments of a table in one load .....	1-37
Loading multiple segments of multiple tables in one load .....	1-37
Loading different tables in multiple loads .....	1-37
Loading the same table in multiple loads .....	1-38
Loading multiple segments of multiple tables in multiple loads .....	1-38
Querying a table while it's being loaded .....	1-39
Locking during loads .....	1-39
If an operation fails .....	1-39
Restart .....	1-39
Abort .....	1-40

Status reporting .....	1-40
Symbolic names in replies .....	1-41
Numeric SQL codes in replies .....	1-42
System table updates .....	1-42
Metadata updates for a data load operation .....	1-42
Metadata updates for a replace operation .....	1-43
Metadata updates for an index load operation .....	1-43
Metadata updates for a merge operation .....	1-44

## **Chapter 2: The data file ..... 2-1**

Data file .....	2-1
Data records and data fields .....	2-1
Data in a control file .....	2-1
Record formats .....	2-2
Text data .....	2-2
Variable-length data .....	2-3
Fixed-length data .....	2-4
Data type considerations .....	2-4
Native data types .....	2-4
VAR-type data .....	2-5
Difference between a column and a field in a data record .....	2-6
Difference between a logical record and a physical record .....	2-7
Delimited data .....	2-7
Terminated data .....	2-8
Enclosed data .....	2-8
Blank characters .....	2-9
LOB data .....	2-9
LOB data in the data file .....	2-9
LOB data files on your client computer .....	2-10
LOB path and file name in the data file .....	2-11

LOB path in the control file and file name in the data file .....	2-11
Multiple paths to LOB data files .....	2-12
LOB data files on a remote system .....	2-13
Guidelines for specifying a LOB file name in a data file .....	2-14

## **Chapter 3: The control file .....3-1**

About the control file .....	3-1
Control file guidelines .....	3-2
Format conventions .....	3-3
SQL identifiers .....	3-3
Control statement formats .....	3-5
LOAD DATA .....	3-5
General notes about the LOAD DATA statement .....	3-7
Description of the LOAD DATA clauses .....	3-7
LOAD INDEX .....	3-9
MERGE .....	3-10
Loading data already on StorHouse .....	3-10
Format of INFILE clause .....	3-11
Example INFILE clauses .....	3-12
To load data in a VRAM file .....	3-12
To load data from multiple VRAM files .....	3-13
To load discarded records .....	3-13
To load data from a data file and collect discarded records .....	3-14
Collecting discarded records in a discard file .....	3-15
Format of DISCARDFILE clause .....	3-16
Example DISCARDFILE clause .....	3-17
Limiting the number of discarded records .....	3-17
Format of DISCARDS clause .....	3-18
Example DISCARDS clause .....	3-18
Specifying the character set of the input data .....	3-18
Format of CHARACTERSET clause .....	3-20
Example CHARACTERSET clause .....	3-20

Concatenating a fixed number of physical records into a logical record .....	3-20
Format of CONCATENATE clause .....	3-21
Example CONCATENATE clause .....	3-21
Combining a varied number of physical records into a logical record .....	3-22
Format of CONTINUEIF clause .....	3-22
Example CONTINUEIF clauses .....	3-24
To combine the current physical record with the next one .....	3-24
To combine the next physical record with the previous one .....	3-25
To use the last non-blank data column as the comparison value .....	3-26
To specify the starting column number of a continuation field .....	3-27
To specify the starting and ending column numbers of a continuation field .....	3-28
To use a character string as a comparison value .....	3-29
To use a hex string as a comparison value .....	3-29
To use blank characters as a comparison value .....	3-30
To use a not equal comparison operator .....	3-31
Preserving blanks in input data .....	3-31
Format of PRESERVE BLANKS clause .....	3-32
Example PRESERVE BLANKS clause .....	3-32
Rotating among subspaces .....	3-33
Format of SUBSPACE ROTATE clause .....	3-35
Example SUBSPACE ROTATE clauses .....	3-35
To rotate among subspaces in a user tablespace .....	3-35
To rotate among subspaces in multiple user tablespaces .....	3-36
Identifying the user table to load .....	3-39
Format of INTO TABLE clause .....	3-40
Example INTO TABLE clause .....	3-40
Choosing which records to load .....	3-40
Format of WHEN clause .....	3-42
Example WHEN clauses .....	3-43
To specify the starting column number of the selection criteria .....	3-43
To specify starting and ending column numbers of the selection criteria .....	3-43
To use a column name to identify the selection criteria .....	3-44
To use a field name to identify the selection criteria .....	3-45
To use a character string as selection criteria .....	3-46

To use a hexadecimal string as selection criteria .....	3-46
To test blanks .....	3-47
To test multiple values (using AND) .....	3-47
To test one value or another (using OR) .....	3-47
To test one value or another and multiple values (using OR and AND) ...	3-48
Generating field_specs, identifying NULL flags, specifying default delimiters and other defaults .....	3-48
Guidelines for specifying a default delimiter .....	3-50
Format of FIELDS clause .....	3-52
Example FIELDS clauses .....	3-54
To describe data fields terminated by a character .....	3-54
To describe data fields terminated by a blank .....	3-55
To describe data fields enclosed by the same delimiter .....	3-55
To describe data fields enclosed by different delimiters .....	3-56
To describe data fields that are terminated and enclosed .....	3-56
To generate CHAR field_specs .....	3-56
To identify NULL flags in input data records .....	3-57
To load NULL values for empty data fields .....	3-57
Loading missing data fields with null values .....	3-58
Format of TRAILING NULLCOLS clause .....	3-59
Example TRAILING NULLCOLS clause .....	3-59
Loading one or more segments .....	3-60
Format of SAME and DIFFERENT SEGMENT clauses .....	3-61
Example SAME and DIFFERENT SEGMENT clauses .....	3-61
To load multiple segments of the same user table .....	3-61
To load multiple segments of different user tables .....	3-62
Naming a segment .....	3-63
Format of SEGMENT clause .....	3-64
Example SEGMENT clauses .....	3-64
To use the load ID as the segment tag .....	3-64
To assign different segment tags for multiple segments of the same user table ..	3-65
Replacing a segment .....	3-65
Format of REPLACE SEGMENT clause .....	3-67



Example REPLACE SEGMENT clause .....	3-67
Selecting subspaces .....	3-68
Format of SUBSPACE number clause .....	3-69
Example SUBSPACE number clauses .....	3-69
To select subspaces when loading one segment .....	3-70
To select subspaces when loading multiple segments .....	3-71
To select subspaces in multiple user tablespaces .....	3-72
Describing data fields .....	3-74
Format of field_spec list .....	3-75
Providing a field name .....	3-77
Providing a column name .....	3-77
Loading a record number into a column .....	3-78
Generating a sequence of values .....	3-78
Loading the current date into a column .....	3-79
Loading a constant value into a column .....	3-79
Specifying the position of a data field .....	3-80
Specifying the data type .....	3-83
Converting data types .....	3-101
Calculating the length of a data field .....	3-103
Specifying a character set for an individual data field .....	3-105
Specifying a delimiter for an individual data field .....	3-106
Specifying a host name for LOB data files .....	3-107
Specifying a path name for LOB data files .....	3-107
Specifying a user name and password to access LOB data files .....	3-108
Specifying a BLOB or CLOB data type .....	3-109
Setting a column to a null value .....	3-111
Setting a column to the default value .....	3-111
Using multiple into_table_specs .....	3-112
Creating multiple logical records from one physical record .....	3-113
Using the same data file to load multiple user tables .....	3-114
Including SQL statements in a control file .....	3-115
Including data in a control file .....	3-115
Example LOAD DATA statements .....	3-117
Example 1: Loading all records into one user table .....	3-118

Example 2: Combining a fixed number of records and loading some of them into one user table .....	3-119
Example 3: Loading delimited data into multiple user tables .....	3-120
Example 4: Combining a variable number of records and loading null values .....	3-121
Example 5: Loading SMALLINT, DECIMAL, and VARCHAR data .....	3-123
Example 6: Using relative positioning to load delimited data into multiple user tables .....	3-125
Example 7: Using multiple selection criteria and including the data in the control file .....	3-129
Example 8: Selecting subspaces for a component type .....	3-132
Example 9: Loading LOB data from local LOB files .....	3-135
Example 10: Loading LOB data in the input data file and from LOB files .....	3-136
Example 11: Loading LOB data fields using a default field list and NULLFLAGS .....	3-138
Loading a deferred index .....	3-140
Format of LOAD INDEX statement .....	3-141
Example LOAD INDEX statements .....	3-142
Merging segments of a table .....	3-143
Format of MERGE statement .....	3-144
Example MERGE statements .....	3-146
<b>Chapter 4: The FTP commands .....</b>	<b>4-1</b>
FTP commands for loading data .....	4-2
The put command .....	4-3
Keywords for the put command .....	4-4
Guidelines for entering a put command .....	4-6
When to use a keyword .....	4-7
Sample session .....	4-8
GUI and client FTP tool considerations .....	4-9
Opening a session at an alternate port .....	4-9
Specifying keywords for a remote file name or target directory .....	4-10
Listing the remote directory .....	4-10

Using automatic binary transfer type .....	4-11
Displaying remote server messages .....	4-11
Resetting the transfer type to ASCII .....	4-11
Globbering remote file names .....	4-11
Using CtrlC to abort a transfer .....	4-13
Handling client-side errors .....	4-13
Setting a timer to drain the data channel .....	4-14
Automating command entry .....	4-15
Using FTP auto login .....	4-15
Creating macros for the put command .....	4-16
Macro for loading data that's in the control file .....	4-16
Macro for loading data in a separate data file .....	4-17
Macro for confirming a load during the same FTP session .....	4-17
Macro for restarting a load .....	4-17
Macro for aborting a load .....	4-18
Logging into the StorHouse FTP server .....	4-18
Setting the transfer type .....	4-20
Validating the data and control file before loading .....	4-21
Transferring the control file .....	4-22
Transferring the data file .....	4-24
Piping the data .....	4-25
Checking the status of an operation .....	4-26
Restarting an operation .....	4-28
Confirming an operation .....	4-32
Aborting an operation .....	4-33
Displaying help for StorHouse FTP server commands .....	4-34
Logging off the StorHouse FTP server .....	4-35

<b>Chapter 5: Examples .....</b>	<b>5-1</b>
Submitting an SQL statement .....	5-1
Loading text data, transferring one file .....	5-2
Loading fixed-length data, transferring two files .....	5-3
Loading variable-length data, transferring two files .....	5-5
Replacing segments without loading .....	5-6
Loading a deferred index for all segments .....	5-8
Merging all segments of a table .....	5-9
 <b>Appendix A: University of California copyright, conditions, disclaimer .....</b>	 <b>A-1</b>

## **Index**

# Welcome

The *FileTek® FTP Data Loader* is a tool for loading data into StorHouse® user tables. With the FileTek FTP Data Loader, 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 Loader follows the standards defined by the protocol.

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

## Welcome

Purpose of this document

## 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<sup>TM</sup> Solaris<sup>TM</sup> and Hewlett-Packard HP-UX platforms.

## Control Center

StorHouse *Control Center* (CC) is the FileTek Windows<sup>®</sup>-based network computing system for providing administrative control of the StorHouse family of products. Control Center works with StorHouse/SM release 4.2 and higher and consists of one or more Control Center servers that communicate with Control Center clients over a TCP/IP network. The *Control Center server*, which runs on Windows NT, XP Pro, and 2000 platforms, provides network connectivity to StorHouse. The *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 Control Center servers, and analyzing and monitoring StorHouse activity and performance.

## Purpose of this document

The *FileTek FTP Data Loader Manual* explains how to run the FileTek FTP Data Loader. It describes the formats of data you can load, explains how to create a LOAD DATA statement that defines the data, and describes the FTP commands you use to start a load and if necessary to restart or stop a load. This manual also explains how to load deferred indexes and merge segments with the FileTek FTP Data Loader.



## Audience

If you manage, schedule, or run the FileTek FTP Data Loader at your organization, then this manual is for you. The *FileTek FTP Data Loader Manual* assumes you're familiar with the data loading concepts and processes of your local database. It also assumes you understand Structured Query Language (SQL), StorHouse database fundamentals, FTP commands and protocol, Transmission Control Protocol (TCP), and your host operating system.

## What's in this guide

This document is organized as follows:

- Chapter 1, “The basics,” describes the features, concepts, and components of the FileTek FTP Data Loader.
- Chapter 2, “The data file,” explains the record formats that the FileTek FTP Data Loader accepts and defines input data concepts.
- Chapter 3, “The control file,” describes the SQL-like LOAD DATA, LOAD INDEX, and MERGE statements.
- Chapter 4, “The FTP commands,” describes the subset of FTP commands you use to load data.
- Chapter 5, “Examples,” contains examples of different loads.

## Conventions

This book uses the following conventions:

Convention	Meaning
Helvetica font	Control statement formats and examples, FTP commands, and FTP replies
<i>Italics</i>	Defined terms, emphasized text, and publication titles
▼	Procedures

See page 3-3 for format conventions of the LOAD DATA, LOAD INDEX, and MERGE statements.

## For more information

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

- 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 you receive when running the FileTek FTP Data Loader, refer to the StorHouse *Messages and Codes Manual*, publication number 900032.
- To learn how to unload data from StorHouse user tables by using FTP, refer to the *FileTek FTP Data Unloader Manual*, publication number 900137.
- 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 comprehensive descriptions of the FTP commands and the LOAD DATA, LOAD INDEX, and MERGE statements. Refer to the two quick reference cards that accompany this manual for just the syntax of the FTP commands and the LOAD DATA statement.



## **Welcome**

---

For quick reference

# The basics

This chapter contains general information about the FileTek FTP Data Loader. It describes:

- Loading features
- How it works with FTP
- What you need to load data and indexes
- What to do before loading
- The load data process
- Loads and segments
- Loads and subspaces
- Loads and indexes
- Load parallelism
- Locking during loads
- The restart and abort process
- Status reporting
- System table updates

## Loading features

The FileTek FTP Data Loader is a bulk data loading system designed to load large volumes of data from your host into user tables on StorHouse. Those user tables may be empty or they may contain data from a previous load. You also use the

FileTek FTP Data Loader to load deferred indexes and to merge existing segments. Key features of the FileTek FTP Data Loader are as follows.

**Portability.** Users with different UNIX<sup>®</sup> platforms can load data without installing or porting special client software. That's because you use your standard client FTP software to load data. The FileTek FTP Data Loader supports a subset of standard FTP commands that are common to most UNIX hosts and other non-UNIX environments like PCs.

**Parallelism.** StorHouse segmentation technology and read-only databases enable you to load the same table in parallel. You can also query and load the same table at the same time. See “Loads and segments” on page 1-11 for information about segmentation. See “Load parallelism” on page 1-35 for ways to load in parallel.

**Multiple record formats.** The FileTek FTP Data Loader accepts input data in fixed-length, variable-length, text, and large object (LOB) type formats. It also accepts files containing character large objects (CLOBs) and binary large objects (BLOBs), also referred to as large objects (LOBs). See Chapter 2 for more information about input data.

**Conversion.** The FileTek FTP Data Loader supports a wide range of data types and automatically converts the data types of your input data to the data types of the user table. For example, you can load input data defined as INTEGER EXTERNAL into a column of a user table defined as CHARACTER, INTEGER, SMALLINT, or VARCHAR. See “Converting data types” on page 3-101 for the supported conversions.

**Compatibility.** The control information that you supply is similar and compatible with the control information accepted by Oracle<sup>®</sup> and DB2<sup>®</sup> load utilities. The FileTek FTP Data Loader accepts Oracle and DB2 clauses that are not part of the StorHouse syntax but ignores those that do not apply to StorHouse.

**Exception processing.** You can collect discarded records that do not meet your load criteria and then load those discarded records as required. See “Collecting discarded records in a discard file” on page 3-15 for more information about exception processing.

**Status reporting.** During an operation, the FileTek FTP Data Loader provides informational and error messages that describe the input data, explain errors requiring user action, and report the status of each operation. See “Status reporting” on page 1-40 for more information about the replies and status codes you receive during loads.

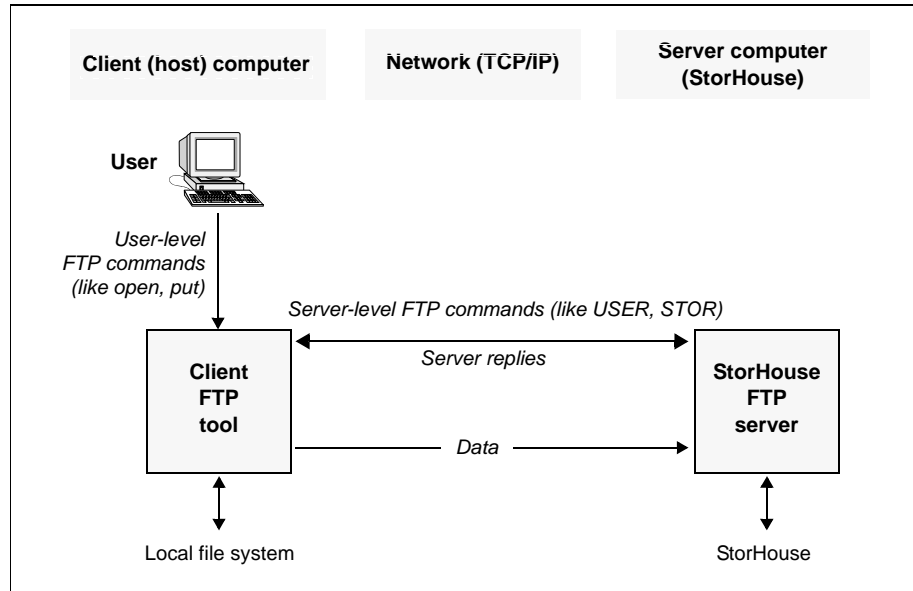
**Restart capability.** You can restart an operation from the point of failure. You can also abort an operation and then start from the beginning. See “If an operation fails” on page 1-39 for more information about restart options.

**SQL tool.** You can use the FileTek FTP Data Loader to submit StorHouse SQL statements. For example, before you load data you can create the user table and indexes by issuing CREATE TABLE and CREATE INDEX statements. The FileTek FTP Data Loader commits each SQL statement when it completes. See “Submitting an SQL statement” on page 5-1 for an example.

## How it works with FTP

With the FileTek FTP Data Loader, you use your standard FTP client software to communicate with the FileTek StorHouse FTP server. These two programs

communicate over a TCP/IP connection to transfer files from your local file system on your host to a remote file system on StorHouse.



Your *client FTP software* (or tool) interacts with you and your local file system. It sends server-level FTP commands and your control and input data to the StorHouse FTP server. The *StorHouse FTP server* interacts with the remote file system (StorHouse user tables and indexes) and replies to your client's FTP commands.

**Note:** Some of the source code for the StorHouse FTP server is 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.

## 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 loads. You can even unload data with the FileTek FTP Data Unloader 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 try to start a load in the middle of another load (for example, after transferring the control information but before transferring the data), an error will occur and you will have to restart or abort the load.

## Transfer types

Before you transfer files with FTP, you can set the *transfer type* so that the FileTek FTP Data Loader knows how to interpret data and end of line (EOL) characters. The FileTek FTP Data Loader supports ASCII and BINARY transfer types.

**ASCII type.** The data is text in the ASCII character set and the EOL is indicated by a carriage return (CR) followed by a line feed (LF), or <CRLF>. The C notation is \r\n. This transfer type is intended primarily for transferring text files. Note that when transferring control statements and input data in ASCII transfer type from a non-ASCII machine, your client FTP tool will convert the control statements and data to ASCII.

**BINARY or IMAGE type.** The data is transferred “as is” without any translation. This transfer type is the most efficient and commonly used between UNIX machines. You must use the BINARY transfer type for fixed-length and variable-length data. See page 2-2 for more information about these record formats.

## What you need

You need a client FTP tool and StorHouse account information to use the FileTek FTP Data Loader.

### Your client FTP tool

You run the FileTek FTP Data Loader 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 sent as a single stream of bytes and the end of file (EOF) is indicated by the client FTP tool closing the data channel)
- 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)
  - 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 FTP tool considerations” on page 4-9 for other issues about client FTP tools.

## **StorHouse account information**

When logging into the StorHouse FTP server, you provide a *StorHouse account ID* and *password*. The StorHouse account ID must have the following StorHouse privileges to load data, load indexes, and merge segments.

- Database component privilege:
  - INSERT privilege for the user table(s) you’re loading
- Access privileges:
  - SQLCOMMAND
  - SQLEXECUTE
- Command privileges:
  - ATF
  - DELETE
  - GET
  - PUT
  - RECORD
  - SETGROUP
  - VTF

If you submit StorHouse SQL statements with the FileTek FTP Data Loader, the account ID must have the appropriate privilege for the statement. For example, to submit CREATE TABLE and CREATE INDEX statements with the FileTek FTP Data Loader, the account ID used to log into the StorHouse FTP server must have the DBA or RESOURCE database privilege.

Refer to the *StorHouse Database Administration Guide* for more information about database component and access privileges. Refer to the *StorHouse Command Language Reference Manual* for more information about command privileges.

## Before loading

The StorHouse system or database administrator must complete specific tasks before loading. The following table lists these tasks and who is typically responsible for performing them.

StorHouse task	Command/Statement	Person responsible
Create the StorHouse account and password used to log into the StorHouse FTP server. The account ID must have the access and command privileges listed on page 1-7.	CREATE ACCOUNT	StorHouse system administrator
Create the volume sets and file sets for the VRAM™ files, discard files, user tables, and indexes, if needed.	CREATE VSET CREATE FSET	StorHouse system administrator
Create the user tablespace associated with the user table to be loaded.	CREATE TABLE SPACE	StorHouse database administrator
Create the user table to be loaded.	CREATE TABLE	StorHouse database administrator

<b>StorHouse task</b>	<b>Command/Statement</b>	<b>Person responsible</b>
Create indexes for the user table. An administrator can create deferred indexes after the table is loaded.	CREATE INDEX	StorHouse database administrator
Grant INSERT privilege to the StorHouse signon account ID for the user table to be loaded.	GRANT	StorHouse database administrator
Create discard files to collect discarded records that do not meet selection criteria. This file is VRAM file.	CREATE FILE	StorHouse system administrator

## The load data process

To load data into StorHouse:

1

### Prepare the input

At your host, prepare a control file and a data file.

- A *control file* contains a *LOAD DATA* statement that identifies the user table and describes the data you're loading.
- A *data file* contains the data records you're loading. You can place data records in a separate data file or in the control file. If your client FTP tool supports piping, you can also feed the data from a program instead of including data in a file.

2

### Transfer the files

With your client FTP tool:

- Start FTP and log into the StorHouse FTP server.
- If needed, set the transfer type using the FTP type command.
- Transfer the control file (or combined control/data file) using the FTP put command.
- Transfer the data file (if the data is in a separate file) using the FTP put command.

The FileTek FTP Data Loader formats your input into a *data stream*. A StorHouse engine checks your StorHouse privileges and obtains the metadata to store the table data, indexes, and LOB data. The FileTek FTP Data Loader then loads the StorHouse user table(s) and builds and stores the indexes.

3

### Confirm the load

When the load completes and you receive the final reply, confirm the load with the FTP confirm or end command. Confirming a load updates the metadata and removes checkpoint information needed to restart a load. If you don't know whether the load completed OK, you can check the status with the FTP status command.

## Loads and segments

When the FileTek FTP Data Loader “loads” a user table and “stores” an index, it actually writes the table data, LOB data, and index entries to files on the StorHouse storage hierarchy. This set of files is called a *segment*. When you load data into a table for the first time, the FileTek FTP Data Loader creates a segment on StorHouse. When you load more data into that user table, the FileTek FTP Data Loader creates another segment. Each load into a table creates a segment.

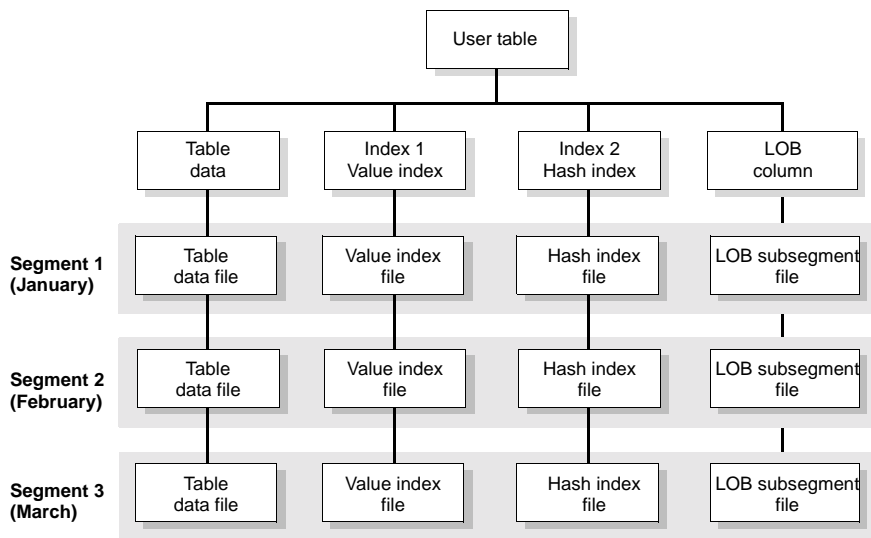
Each segment consists of:

- One table data file
- One index file for each value index
- One index file for each hash index
- One or more LOB subsegment files for LOB columns

A range index applies to all segments of the user table. Depending on the actual size or by user request, a LOB value may be stored in the table data file, or multiple LOB values in different columns may be stored in the same LOB subsegment file, or LOB values in the same column may be stored in multiple LOB subsegment files. An *in-line LOB* is a LOB value stored in the table data file. An *out-of-line LOB* is a LOB value stored in a LOB subsegment file. Refer to the *StorHouse Database Administration Guide* for more information about LOB storage options.

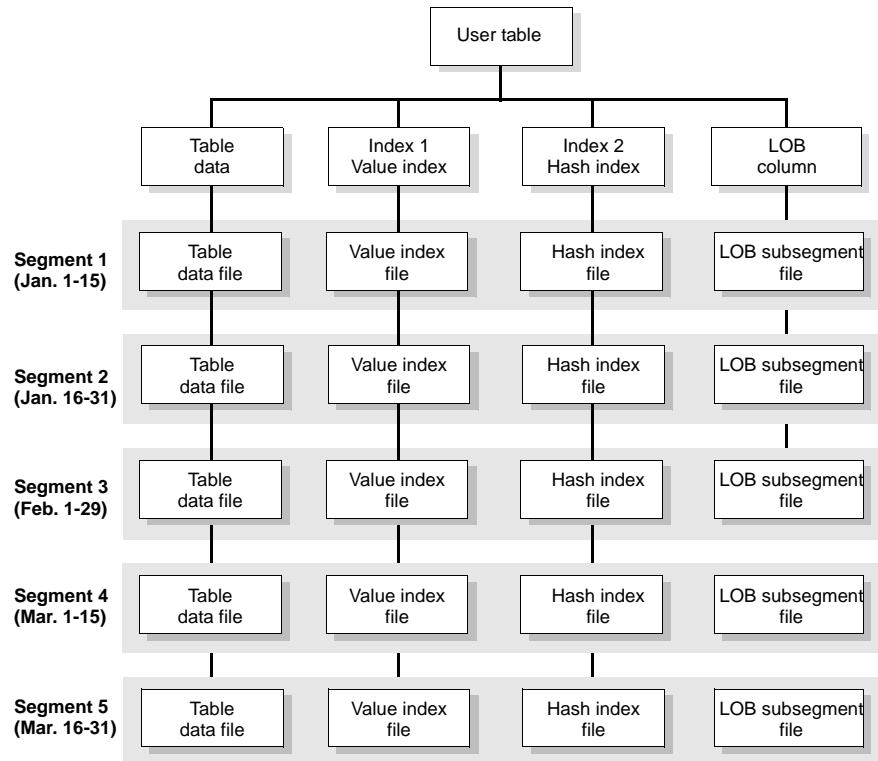
For example, suppose a user table consists of one value index, one hash index, and one LOB column. Assume you loaded data into the table in January, February, and March. After the March load, the table consists of three segments.

Each segment contains one table data file, one value index file, one hash index file, and one LOB subsegment file.



By default, the FileTek FTP Data Loader creates *one* segment for each load. You also have the option of creating *multiple* segments during one load. For example,

in January you could create two segments, in February you could create one segment, and in March you could create two segments.



## Segment size

The maximum size of a table data file is approximately 100 GB. If your input data exceeds 100 GB, you must do one of the following:

- Load the data into multiple segments, ensuring that each segment does not exceed the maximum size. See page 3-60 for more information about loading data into multiple segments during one load.

- Split the data into multiple data files and run multiple loads. Each load writes to a different segment.

## Segment replacement

You can replace an existing segment. Replacing a segment invalidates the segments files, making them inaccessible. A *replaced segment* continues to reside on StorHouse and the segment files are system-managed according to the user tablespace parameters. You can delete and remove a replaced segment or re-validate it later if users need to access it. The range index entries for a replaced segment remain in the system tables.

## Segment merge

You can merge existing segments of a table with the FileTek FTP Data Loader. A *merge*, or *coalesce*, *operation* consolidates selected segments into one segment or more segments depending on your merge criteria. You perform a merge operation separately from a load operation. See “Merging segments of a table” on page 3-143 for more information about performing a merge operation.

## Loads and subspaces

A user tablespace consists of one or more *subspaces* that specify where and how segments are stored on StorHouse. When creating a user table, you assign it to a user tablespace. If the user table contains LOB columns, you can assign those LOB columns to the same user tablespace as the table or to different user tablespaces. And when creating an index for a user table, you can assign it to the same user tablespace as the table or to a different user tablespace. You can even assign different indexes of a table to different user tablespaces.

When you load data, the FileTek FTP Data Loader determines the user tablespace(s) and can store each component in a default subspace. The data



loader uses the value of the `OBJECT_TYPE` parameter (specified on a `CREATE TABLE SPACE` or `ALTER TABLE SPACE` statement) to determine the type of component allowed in a subspace. The `OBJECT_TYPE` values are:

- Blank for all components
- T for table data
- H for hash indexes
- V for value indexes
- L for LOB data

You can also explicitly select subspaces or rotate among subspaces that are valid for a component type. This section describes default selection, explicit selection, and rotation of subspaces. Refer to the *StorHouse Database Administration Guide* for more information about user tablespaces and subspaces.

**Note:** LOB subspaces and tablespaces are not used for in-line LOBs. The FileTek FTP Data Loader uses the table data subspace and the user table tablespace for LOB values that fit within a row.

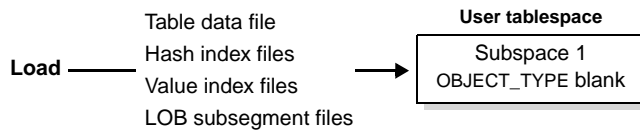
## Default selection of subspaces

A *default subspace* is the lowest-numbered subspace that allows a component type. If you do not explicitly select subspaces or request rotation during a load, the data loader selects the default. This section describes the default selection of subspaces when:

- There is one subspace for all component types
- There is one subspace for each component type
- There are multiple subspaces for each component type
- Indexes or LOB columns are assigned to multiple user tablespaces

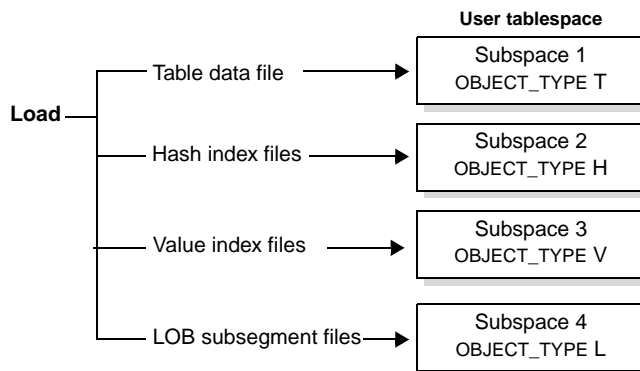
## When there is one subspace for all component types

When a user tablespace contains one subspace for all component types, and the user table, indexes and LOB columns are assigned to the same user tablespace, the data loader automatically selects that subspace during a load.



## When there is one subspace for each component type

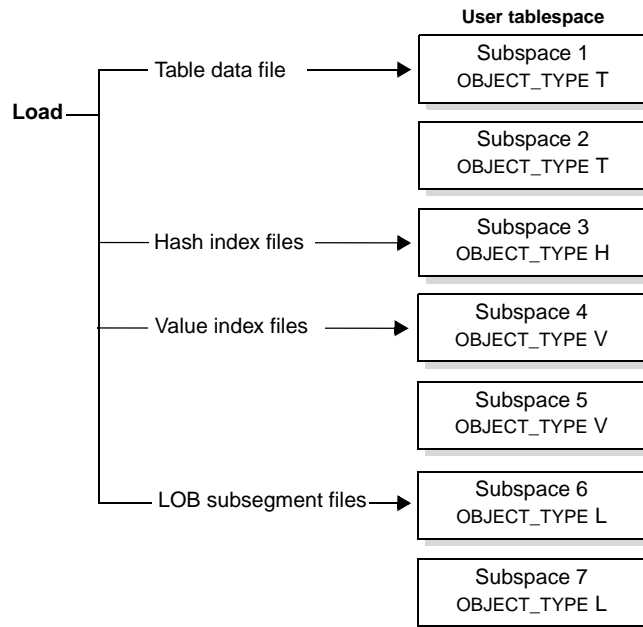
When a user tablespace contains one subspace for each component type, and the user table, indexes, and LOB columns are assigned to the same user tablespace, the data loader automatically selects the appropriate subspace for each component type.



## When there are multiple subspaces for each component type

When a user tablespace contains multiple subspaces for the same component type or multiple subspaces with no component type restrictions (OBJECT\_TYPE is blank), and the user table, indexes, and LOB columns are assigned to the same

user tablespace, the data loader automatically selects the lowest-numbered subspace for the component type.



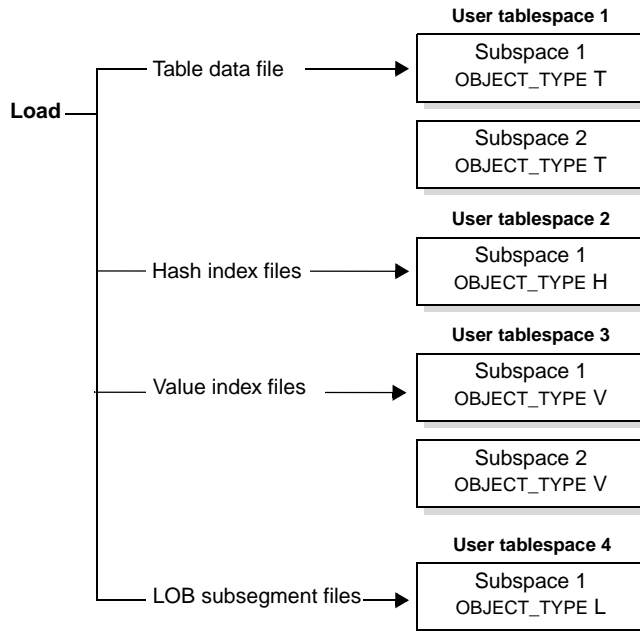
In this example, the data loader selects:

- Subspace 1 for the table data file (because it's the lowest-numbered subspace that allows table data)
- Subspace 3 for hash index files (because it's the only subspace that allows hash indexes)
- Subspace 4 for value index files (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 6 for LOB subsegment files (because it's the lowest-numbered subspace that allows LOB data)

The data loader does not use subspaces 2, 5, and 7 unless you explicitly select them.

## When indexes or LOB columns are assigned to multiple user tablespaces

When any indexes or LOB columns for a user table are assigned to different user tablespaces, the data loader automatically selects the lowest-numbered subspace in the appropriate user tablespace for the component type. For instance, assume a user table has one hash index, one value index, and one LOB column. The user table is assigned to user tablespace 1, the hash index is assigned to user tablespace 2, the value index is assigned to user tablespace 3, and the LOB column is assigned to user tablespace 4. In the following example, the data loader selects subspace 1 as the default for each component type in each user tablespace.



## Explicit selection of subspaces

If a user tablespace contains multiple subspaces for the same component type and you want to use a subspace other than the default, you can explicitly select that subspace during a data load, index load, or merge operation. You can select a subspace for each component type. This section describes explicit selection when:

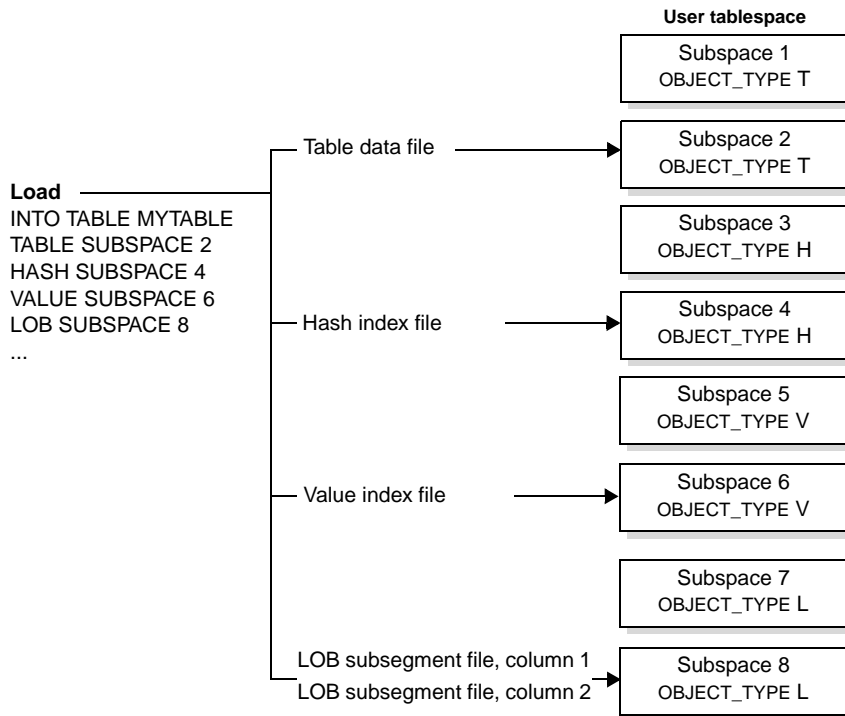
- Loading one segment
- Loading multiple segments
- There are multiple indexes of the same type
- Indexes are assigned to different user tablespaces

You include a SUBSPACE number clause on a LOAD DATA, LOAD INDEX, or MERGE statement to select subspaces. See “Selecting subspaces” on page 3-68 for the format, examples, and more information about this clause. This section also contains examples.

### When loading one segment

When loading one segment into a user tablespace with multiple subspaces for component types, you can select a subspace for each component type. For example, assume the user table has one hash index, one value index, and two LOB columns assigned to the same user tablespace as the table. You’re loading one segment into a user tablespace that contains two subspaces for each component type. Remember that if you don’t explicitly select subspaces, the data loader automatically selects the subspace with the lowest subspace number for the applicable component type.

In the following example, to use subspace 2 for the table data, subspace 4 for the hash index, subspace 6 for the value index, and subspace 8 for the LOB data, you must explicitly select those subspaces for each component type.

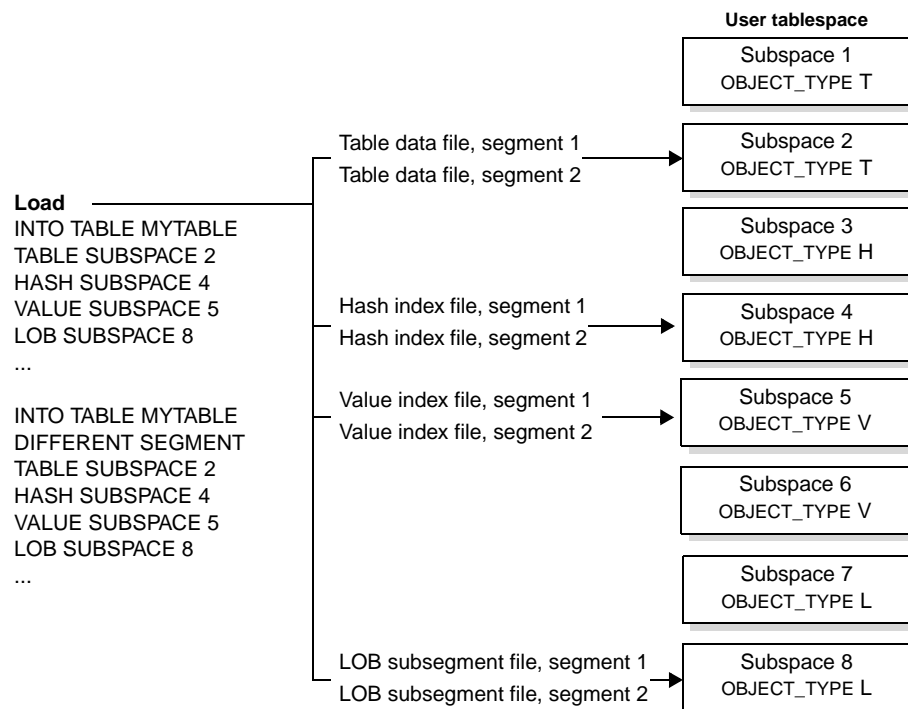


## When loading multiple segments

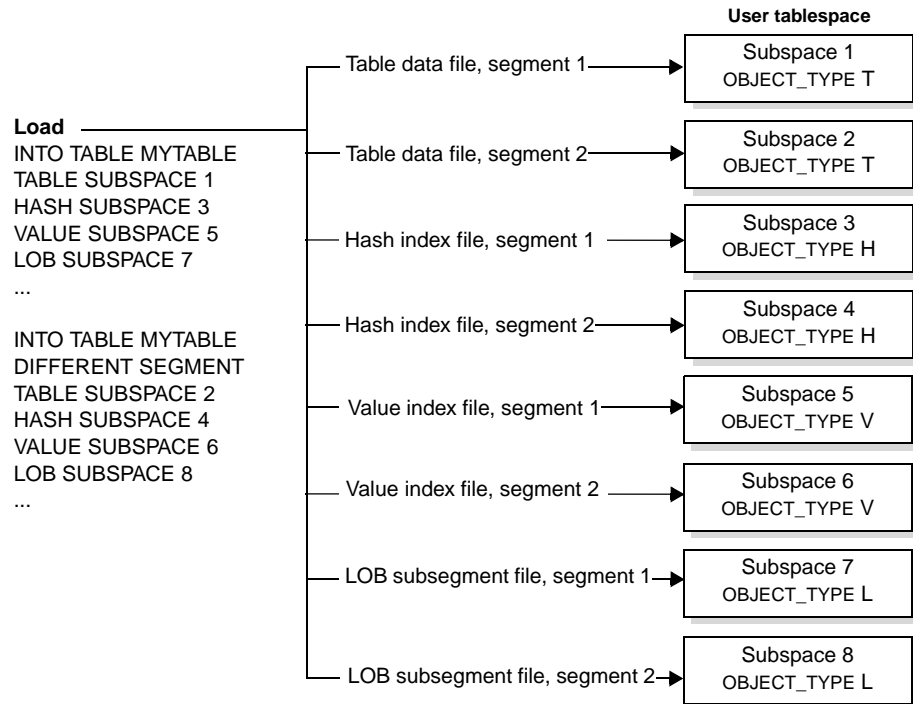
When loading multiple segments into a user tablespace with multiple subspaces for component types, you can select the same subspaces for all segments or different subspaces for each segment.

For example, assume a user table has one hash index, one value index, and one LOB column assigned to the same user tablespace as the table. You're loading two segments into a user tablespace that contains two subspaces for each component type. You can select the same subspace for each segment and component type, for

instance, subspace 2 for both table data files, subspace 4 for both hash index files, subspace 5 for both value index files, and subspace 8 for both LOB subsegment files.



Or you can select different subspaces for each segment and component type.

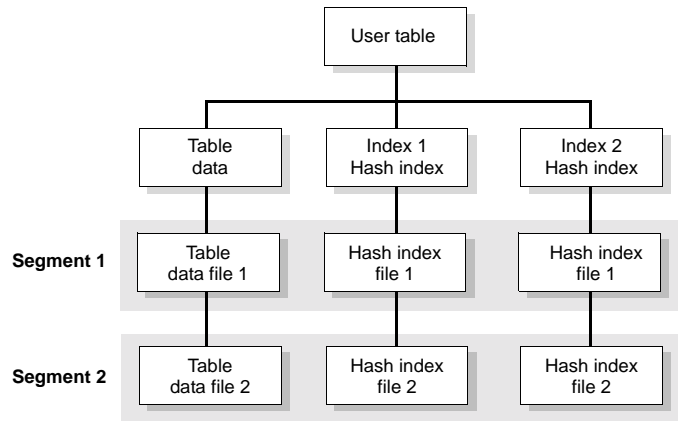


### When there are multiple indexes of the same type

Remember that for each hash index and each value index on a user table, there's one hash index file and one value index file for each table data file. So if you're

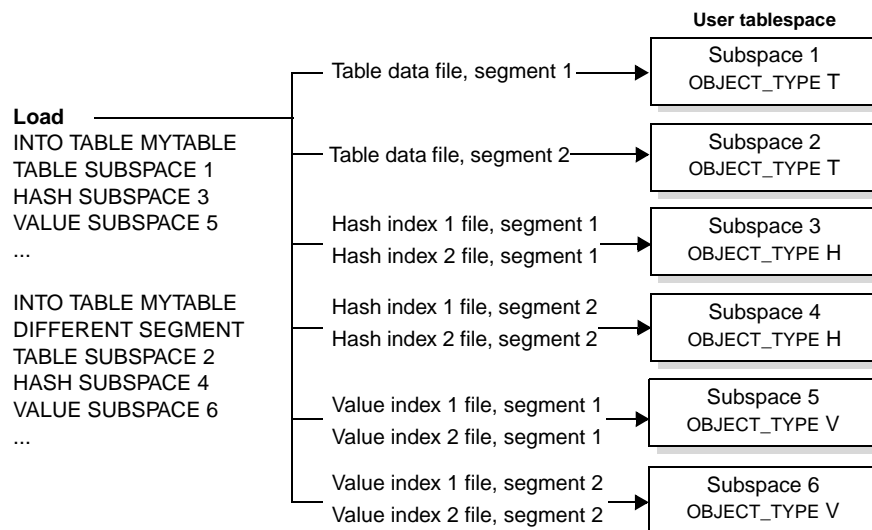


loading two segments, and the user table has two hash indexes, then a load creates four hash index files (two files in each segment).



When you select subspaces for indexes, all same-type (hash or value) index files that correspond to the same segment use the same subspace. For example, assume you're loading two segments. The user table has two hash indexes and two value indexes assigned to the same user tablespace as the table. For the first segment, you could select subspace 3 for both hash index files and subspace 5 for both

value index files. For the second segment, you could select subspace 4 for both hash index files and subspace 6 for both value index files.

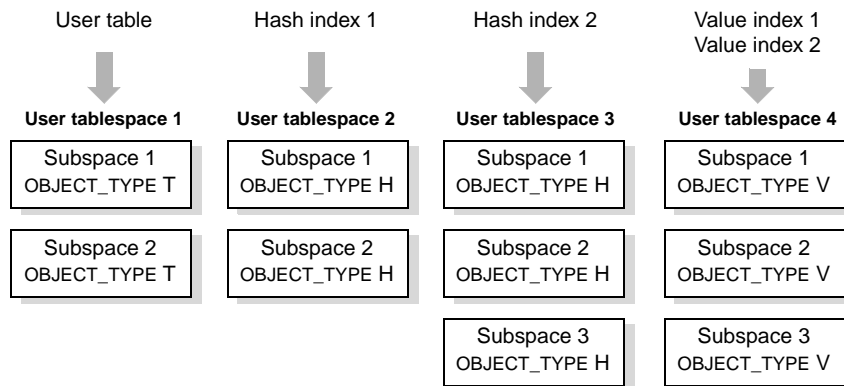


You don't select subspaces for certain indexes (like hash index 1 or value index 2). You select subspaces for component types (like hash indexes or value indexes). All hash index files that correspond to the same segment use the same subspace. All value index files that correspond to the same segment use the same subspace.

## When indexes are assigned to different user tablespaces

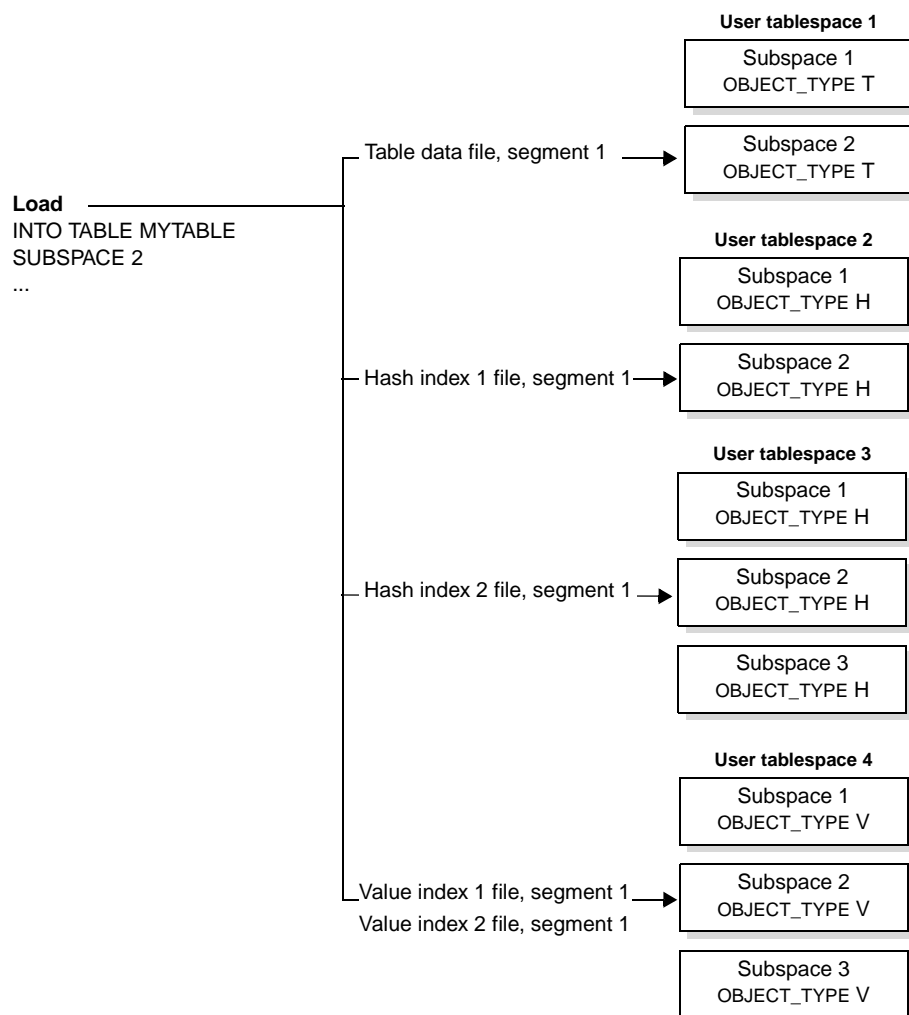
You can select a specific subspace for different component types, even when same-type indexes are assigned to different user tablespaces. The subspace number, however, must exist in all applicable user tablespaces and each subspace must allow that component type. For example, assume you're loading a user table with two hash indexes and two value indexes. The user table is assigned to user tablespace 1, the first hash index is assigned to user tablespace 2, the second hash

index is assigned to user tablespace 3, and both value indexes are assigned to user tablespace 4.



Notice that user tablespace 2 contains two subspaces for hash indexes and user tablespace 3 contains three subspaces for hash indexes. In this case, you cannot select subspace 3 for hash indexes because user tablespace 2 does not contain a subspace 3.

You can also select a specific subspace for all component types. For instance, you can simply select subspace 2. The data loader then uses subspace 2 in each user tablespace.



## Rotation among subspaces

If you're loading multiple segments into a user tablespace with multiple subspaces for component types, you can rotate among the applicable subspaces. You can also rotate among subspaces for index load and merge operations. Rotation occurs independently for each component type. That is:

- For the table data file in the first segment, the data loader starts with the lowest-numbered subspace that allows table data and then uses the next subspace that allows table data for the next segment.
- For hash index files in the first segment, the data loader starts with the lowest-numbered subspace that allows hash indexes and then uses the next subspace that allows hash indexes for the next segment.
- For value index files and LOB subsegment files, the same rotation occurs.

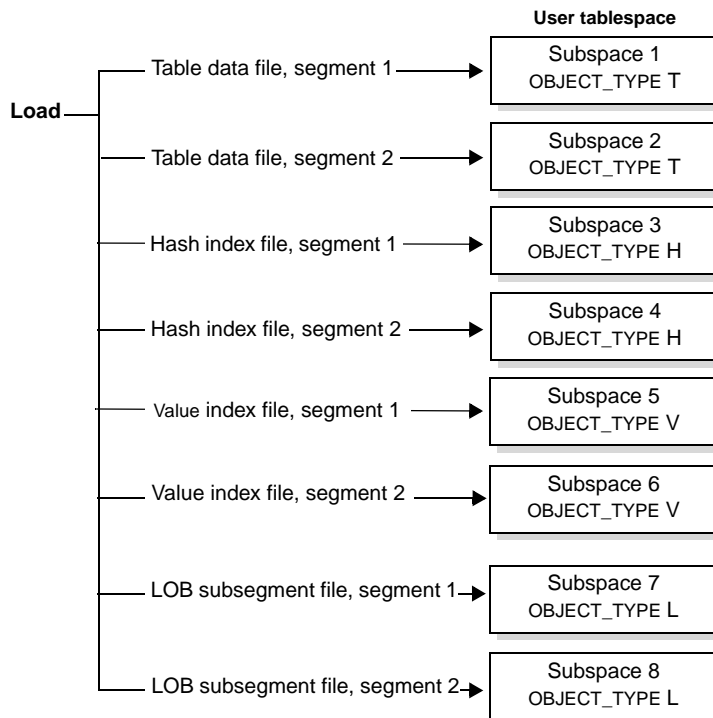
This section describes rotation when:

- There are multiple subspaces for each component type
- Component types share a subspace
- Indexes or LOB columns are assigned to different user tablespaces

You include a SUBSPACE ROTATE clause in a LOAD DATA, LOAD INDEX, or MERGE statement to rotate among subspaces. See “Rotating among subspaces” on page 3-33 for more information and additional examples.

## When there are multiple subspaces for each component type

A user tablespace must contain multiple subspaces for rotation to occur. For example, assume a user table has one hash index, one value index, and one LOB column assigned to the same user tablespace as the table. The user tablespace contains two subspaces for each component type. You're loading two segments and request to rotate among subspaces. The data loader rotates among subspaces as follows:



In this example, the data loader uses:

- Subspace 1 for the table data file in segment 1 (because it's the lowest-numbered subspace that allows table data)
- Subspace 2 for the table data file in segment 2 (because it's the next subspace that allows table data)
- Subspace 3 for the hash index file in segment 1 (because it's the lowest-numbered subspace that allows hash indexes)
- Subspace 4 for the hash index file in segment 2 (because it's the next subspace that allows hash indexes)
- Subspace 5 for the value index file in segment 1 (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 6 for the value index file in segment 2 (because it's the next subspace that allows value indexes)
- Subspace 7 for the LOB subsegment file in segment 1 (because it's the lowest-numbered subspace that allows LOB data)
- Subspace 8 for the LOB subsegment file in segment 2 (because it's the next subspace that allows LOB data)

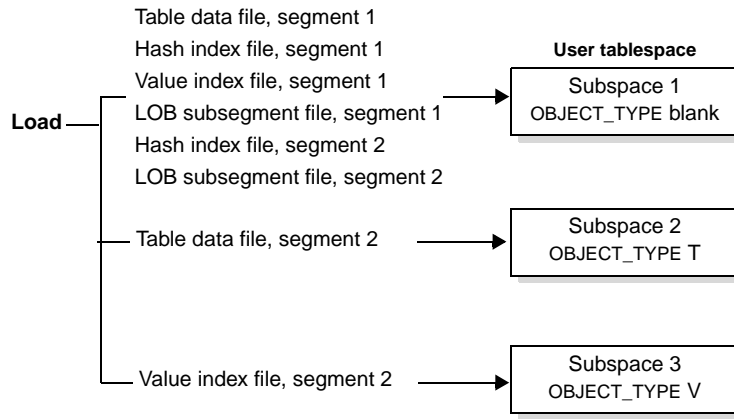
If you were loading a third segment, the data loader would use subspace 1 for the table data file, subspace 3 for the hash index file, subspace 5 for the value index file, and subspace 7 for the LOB subsegment file.

## **When component types share a subspace**

A user tablespace does not have to contain a subspace for each component type. You can also define subspaces that allow all components types (OBJECT\_TYPE is blank). For instance, assume a user tablespace contains three subspaces: one that allows all component types, a second for table data files only, and a third for value

index files only. In this example, subspace 1 is the only subspace that allows hash index files and LOB subsegment files, but it also allows table data files and value index files.

Assume you're loading two segments and request to rotate among the applicable subspaces for each component type. The user table has one hash index, one value index, and one LOB column assigned to the same user tablespace as the table. The data loader rotates among subspaces as follows:



In this example, the data loader uses:

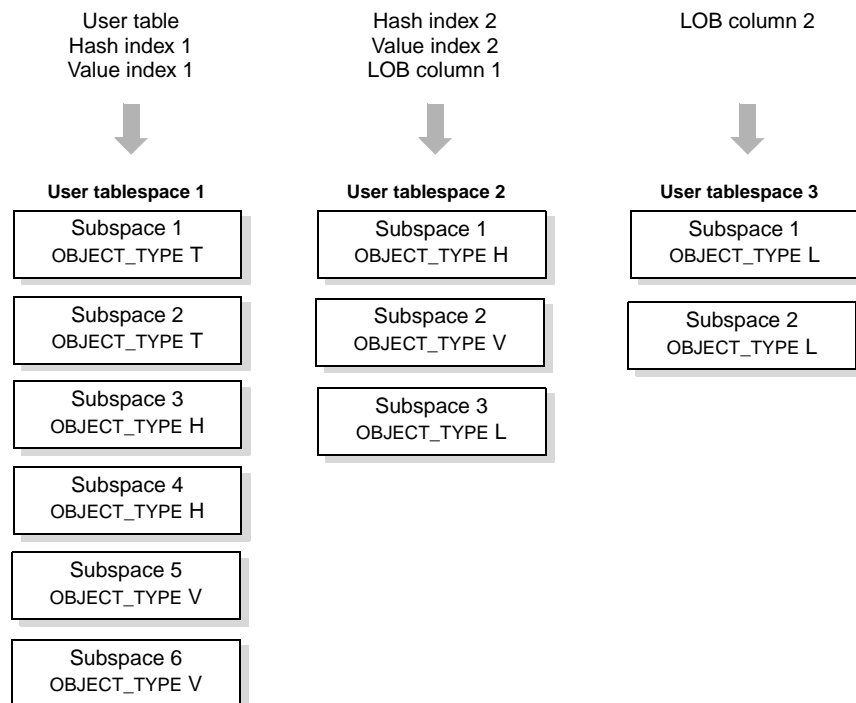
- Subspace 1 for the table data file, hash index file, value index file, and LOB subsegment file for segment 1 (because it's the lowest-numbered subspace that's valid for all component types)
- Subspace 2 for the table data file in segment 2 (because it's the next subspace that allows table data)
- Subspace 3 for the value index file in segment 2 (because it's the next subspace that allows value indexes)



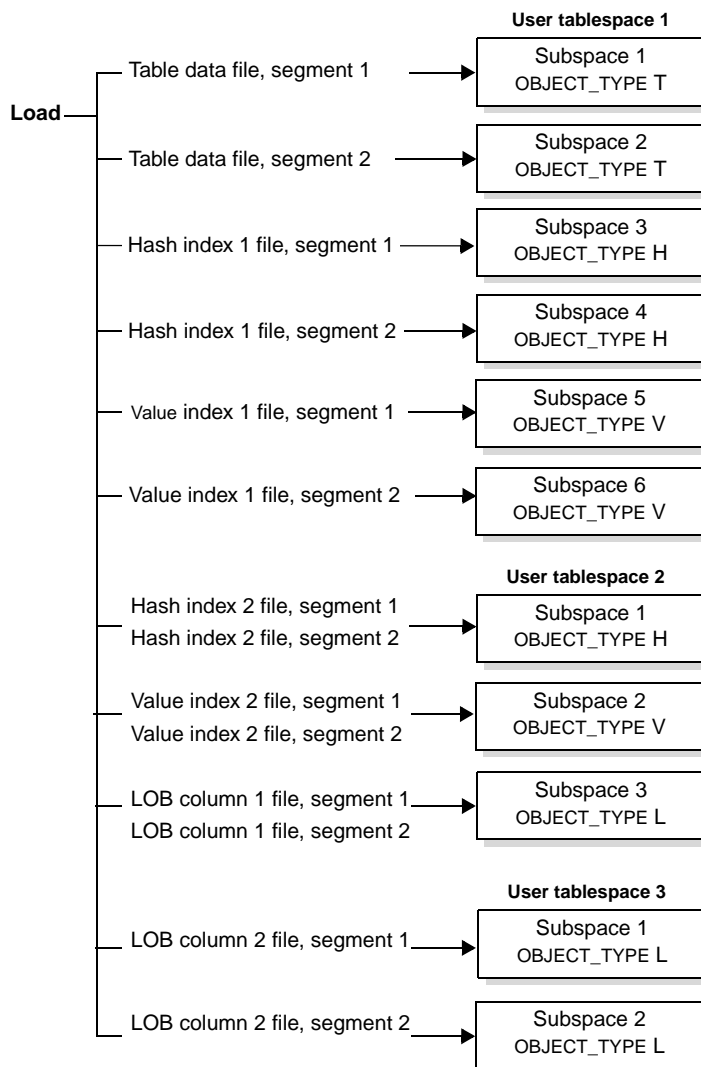
- Subspace 1 for the hash index file and LOB subsegment file in segment 2 (because it's the only subspace that allows hash indexes and LOB subsegment files)

## When indexes or LOB columns are assigned to different user tablespaces

When any indexes or LOB columns for a user table are assigned to different user tablespaces, the data loader rotates among applicable subspaces in the applicable user tablespaces. For example, assume the user table has two hash indexes, two value indexes, and two LOB columns. The user table, hash index 1, and value index 1 are assigned to user tablespace 1; hash index 2, value index 2, and LOB column 1 are assigned to user tablespace 2; and LOB column 2 is assigned to user tablespace 3.



In the following example, the data loader rotates among subspaces in the user tablespaces as follows:



In this example, the data loader uses the following subspaces for table data files:

- Subspace 1 in user tablespace 1 for the table data file in segment 1 (because it's the lowest-numbered subspace that allows table data)
- Subspace 2 in user tablespace 1 for the table data file in segment 2 (because it's the next subspace that allows table data)

The data loader uses the following subspaces for hash index files:

- Subspace 3 in user tablespace 1 for the hash index file of hash index 1 in segment 1 (because it's the lowest-numbered subspace that allows hash indexes)
- Subspace 4 in user tablespace 1 for the hash index file of hash index 1 in segment 2 (because it's the next subspace that allows hash indexes)
- Subspace 1 in user tablespace 2 for both hash index files of hash index 2 in both segments (because it's the only subspace that allows hash indexes)

The data loader uses the following subspaces for value index files:

- Subspace 5 in user tablespace 1 for the value index file of value index 1 in segment 1 (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 6 in user tablespace 1 for the value index file of value index 1 in segment 2 (because it's the next subspace that allows value indexes)
- Subspace 2 in user tablespace 2 for both value index files of value index 2 in both segments (because it's the only subspace that allows value indexes)

The data loader uses the following subspaces for LOB subsegment files:

- Subspace 3 in user tablespace 2 for LOB column 1 subsegment files for both segments (because it's the only subspace that allows LOB data)

- Subspace 1 in user tablespace 3 for the LOB column 2 subsegment file for segment 1 (because it's the lowest-numbered subspace that allows LOB data)
- Subspace 2 in user tablespace 3 for the LOB column 2 subsegment file for segment 2 (because it's the next subspace that allows LOB data)

## Loads and indexes

You can create indexes before or after a table has been loaded. A *deferred index* is an index created after a table has been loaded. You must perform an *index load operation* to load a deferred index. This operation is different from a data load. Here's what you do:

---

### 1 Prepare the LOAD INDEX statement

At your host, prepare a control file with a *LOAD INDEX* statement that identifies the indexes and optionally the segments to be loaded.

---

### 2 Transfer the control file

With your client FTP tool:

- Start FTP and log into the StorHouse FTP server.
- Transfer the control file using the FTP put command.

A StorHouse engine checks your StorHouse privileges and then the FileTek FTP Data Loader creates the index entries for the specified indexes and segments.

---

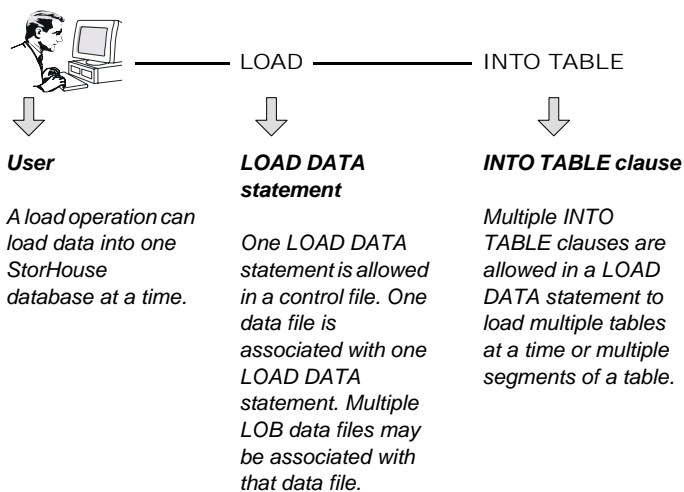
### 3 Confirm the index load operation

When the load completes and you receive the final reply, confirm the operation with the FTP confirm or end command. Confirming an index load operation removes checkpoint information needed to restart the operation.

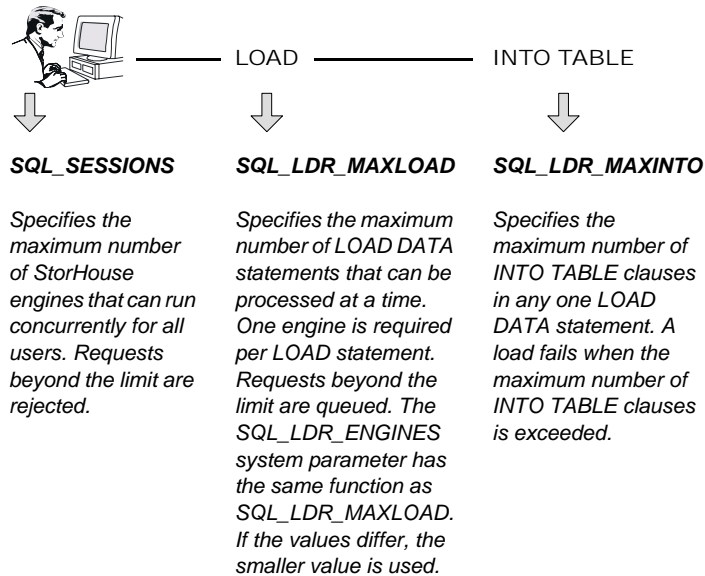
See “Loading a deferred index” on page 3-140 for more information about performing an index load operation.

## Load parallelism

This section illustrates the different ways you can load data in parallel. The graphics on the following pages contain these components:

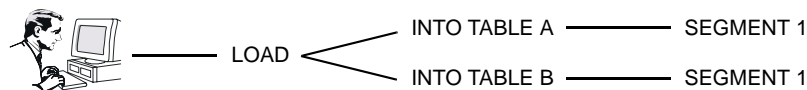


Three tunable StorHouse *system parameters* help control parallelism:



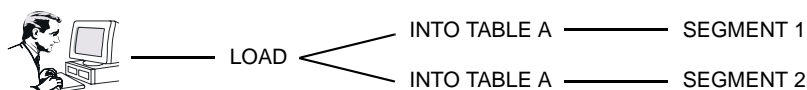
## Loading different tables in one load

A user can load different user tables in one load. By default, a load writes to one segment.



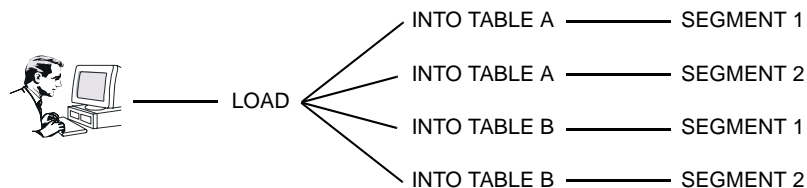
## Loading multiple segments of a table in one load

A user can load one or more segments of the same user table in one load.



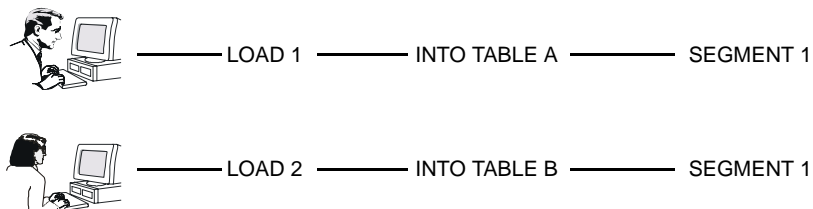
## Loading multiple segments of multiple tables in one load

A user can load one or more segments of multiple user tables in one load.



## Loading different tables in multiple loads

Multiple users can load different user tables concurrently. By default, a load writes to one segment.



## Loading the same table in multiple loads

Multiple users can load the same user table concurrently. Each load writes to a different segment of that user table.



————— LOAD 1 ————— INTO TABLE A ————— SEGMENT 1



————— LOAD 2 ————— INTO TABLE A ————— SEGMENT 2

## Loading multiple segments of multiple tables in multiple loads

Multiple users can load one or more segments of multiple user tables. Each load writes to a different segment of a table.



————— LOAD 1 —————  
————— INTO TABLE A ————— SEGMENT 1  
————— INTO TABLE B ————— SEGMENT 1



————— LOAD 2 —————  
————— INTO TABLE A ————— SEGMENT 2  
————— INTO TABLE B ————— SEGMENT 2



## Querying a table while it's being loaded

A user can query confirmed segments while another user loads new data into that table. New segments can be accessed *after* the load is confirmed.



LOAD INTO TABLE A SEGMENT 2



SELECT FROM TABLE A SEGMENT 1

## Locking during loads

A load acquires and releases an *exclusive lock* on the SYSTABLES system table at load start and on the SYSSTHSEGMENTS system table at load end. No locks are held during the middle of a load. A load acquires a *shared lock* on the user table during load commit processing to prevent user table changes during this operation.

## If an operation fails

For data load, index load, and segment merge operations, the FileTek FTP Data Loader maintains *checkpoint files* with enough information about the operation to restart it if necessary. If an operation fails, you can restart it or abort it. When loading LOB data, however, restart is not supported. You must abort the load and start over.

## Restart

When you *restart* a data load, your load continues from the last checkpoint, which is the last data extent written. For instance, if you're loading a user table and a load fails when half of the extents have been created, the FileTek FTP Data Loader does not re-create the completed extents. You do, however, have to re-send all of the data on a restart. The FileTek FTP Data Loader skips the data it does not need. When you restart an index load or a segment merge operation, the FileTek FTP Data Loader aborts the operation and then starts over, restarting after the last completed segment.

See “Restarting an operation” on page 4-28 for restart guidelines.

## Abort

When you *abort* a data load, the FileTek FTP Data Loader removes all checkpoint information for that load and deletes and removes any partially written segment files on StorHouse. If you want to load the table again, all work starts at the beginning, just like a new load.

When you abort an index load, the FileTek FTP Data Loader deletes and removes the last in-progress segment. Any completed segments have already been committed.

When you abort a segment merge, the FileTek FTP Data Loader deletes and removes any partially created result segment. You cannot abort a merge operation that completed successfully because the FileTek FTP Data Loader automatically commits each result segment after creating it.

See “Aborting an operation” on page 4-33 for guidelines.

## Status reporting

You receive messages—or *replies*—from the StorHouse FTP server in the standard message format defined by FTP. Server replies consist of a three-digit number optionally followed by message text. The digits in the number indicate the type of reply. For instance, replies that begin with 2 are successful or informational 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 an FTP put 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 load:

226- %L-I-XLLOADST, Table loading started on server \at record 1\  
226- %L-I-XLDINFO, \09-MAR-2002:13:36:35 12500 records processed...\  
226- %L-I-XLDINFO, \09-MAR-2002:13:36:43 25000 records processed...\  
226- %L-I-XLDINFO,\09-MAR-2002:13:38:22 37500 total records processed\  
226- %WORLD-S-XWOK, A request was successfully completed.

Replies that start with XLDINFO provide helpful diagnostic information. These informational messages, logged at the beginning of or during a load, describe your input data, including any columns omitted in the control information and any defaults used.

Some symbolic names, such as XRSFREEPOOL, indicate StorHouse/SM errors:

550- %L-W-XRSFREEPOOL Not enough empty volumes in library device  
free pool

Refer to the StorHouse *Messages and Codes Manual* for a list and explanation of StorHouse/SM symbolic names.

## 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 load:

```
226 ***** Load operation/request finished ***** sqlcode=<0>
```

The following reply contains SQL code -20232, which indicates an error:

```
550- %L-W-XLDSQLERR, SQL Error \sqlcode=<-20232> Invalid number  
string\
```

Refer to Appendix A in the *StorHouse SQL Reference Manual* for a list of SQL codes.

## System table updates

A StorHouse engine updates metadata during loading operations. The updates differ depending on the type of operation.

### Metadata updates for a data load operation

During a data load, a StorHouse engine obtains and increments the segment ID in the SYSTABLES system table. After you confirm a load, a StorHouse engine inserts index entries into applicable range index system tables, inserts rows into

the SYSSTHFILES system table for each segment file, and inserts the following into the SYSSTHSEGMENTS system table:

- Table ID of the user table
- Segment ID of the segment
- Number of logical records in a table data file
- Average number of logical records per page in a table data file
- Date and time the load was committed
- Segment tag

You must explicitly confirm a data load to enable access to new segments.

## Metadata updates for a replace operation

For replaced, or invalidated, segments, a StorHouse engine inserts the following into the SYSSTHSEGMENTS system table:

- Flag indicating that the segment was replaced
- Date and time the segment was replaced

You must explicitly confirm a replace operation to commit these updates.

## Metadata updates for an index load operation

For an index load, a StorHouse engine:

- Inserts rows into the SYSSTHFILES system table for each hash index file and value index file
- Inserts index entries into applicable range index system tables

- Updates the IDXCOMPRESS column in the SYSINDEXES system table to N for normal (index complete) only if all index entries for all segments of the table are created

These updates occur before you confirm the operation. Confirming an index load operation simply removes the checkpoint.

## Metadata updates for a merge operation

For a segment merge operation, a StorHouse engine:

- Updates the segment ID and subsegment ID in the SYSSTHFILES system table for all LOB subsegments
- Performs the same updates as a data load for the result segment
- Performs the same updates as a replace operation for the input segments

These updates occur before you confirm the operation. Confirming a merge operation simply removes the checkpoint.

# The data file

This chapter describes concepts and considerations of input data.

## Data file

A *data file* contains the input data records you're loading. You can load data in a file on your host or in a VRAM file on StorHouse. You can also feed the data from a program, if your client FTP tool allows piping. Table data must reside in one data file (it cannot span multiple files). LOB data may reside in the data file or in separate *LOB data files*, one LOB value per file. See "LOB data" on page 2-9 for more information about LOB input data.

## Data records and data fields

Typically, a *data record* corresponds to a row in a user table, but it's possible that multiple data records make up one row in a user table. Each data record is composed of *data fields* that often correspond to values in columns of a user table. The maximum allowable length of an input data record is 32,767 bytes.

## Data in a control file

You can include input data records in the same file as the control file. A *data delimiter record* separates the control statements from the data records. The data

can be in any record format—text, variable-length, or fixed-length. Typically you would use a separate file for variable-length or fixed-length data because the control statements are text format. The FileTek FTP Data Loader, however, can handle a record format change in a file. See “Including data in a control file” on page 3-115 for guidelines.

## Record formats

The FileTek FTP Data Loader accepts input data in these formats: text, variable-length, fixed-length, and LOB-type. See “LOB data” on page 2-9 for more information about LOB-type formats. You specify the record format during your FTP session on the FTP put command.

### Text data

Text is the default record format.

#### Text data specifications

Format	<ul style="list-style-type: none"><li>■ Each record must end with the appropriate end of line (EOL) character, such as a UNIX newline (LF) or an ASCII carriage return and line feed (CRLF) pair.</li><li>■ Records cannot contain any bytes that would prematurely signal the EOL.</li><li>■ The last record must have an EOL character at the end of file (EOF). The FileTek FTP Data Loader will terminate the load if the last text record doesn't end with an EOL.</li><li>■ If the character set of the text data is not ASCII, be sure that newline characters exist <i>only</i> at the end of the lines.</li></ul>
Transfer type	ASCII or BINARY
To use	Omit the var and fixed keyword on the FTP put command.



## Variable-length data

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

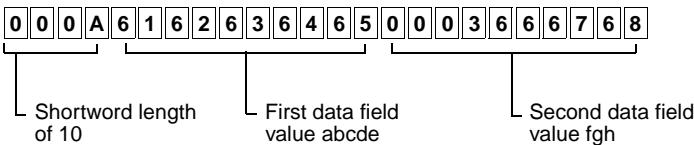
### Variable-length data specifications

- Format
- Each record must be preceded by a 2-byte SMALLINT *shortword* indicating the length of the record. For example, a shortword of 000A (hex) indicates a record length of 10.
  - Shortword bytes are in native values key (nvk) format, that is, the first byte is either the most significant byte (such as for SPARC) or the least significant byte (such as for PC).
  - Shortword bytes are not counted in the length of the data record.
  - Shortword bytes are not part of a logical record, so they are not loaded into a user table.
  - No gaps are allowed between records.
  - The data file must end cleanly, that is, the end of the file is at the end of a record (indicated by the record length).

Transfer type      BINARY

To use              Include the var keyword on the FTP put command.

For example, the following variable-length data 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).



## 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 must have the same length (in bytes).</li><li>■ The last byte in the data file must exactly finish a fixed-length record or the FileTek FTP Data Loader will terminate the load.</li></ul>
Transfer type	BINARY
To use	Include the fixed keyword on the FTP put command, specifying the length in bytes, for example, fixed=80.

## Data type considerations

This section describes considerations for native data types and VAR-type data. See the data type specifications starting on page 3-83 for more information about the supported input data types.

## Native data types

When you load binary data, the FileTek FTP Data Loader must know the client hardware type because different operating systems implement binary data fields—or *native data types*—differently. The FileTek FTP Data Loader supports the following native data types: 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` (native values key) keyword on the FTP `put` command. The following table describes the different ways that different hosts interpret native data types.

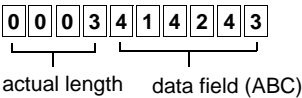
Native data type representation

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

## VAR-type data

When loading `VARBINARY`, `VARBYTE`, `VARCHAR`, or `VARRAW` data, each variable-length data field must be preceded by a two-byte `SMALLINT` field indicating the actual length of the data field. For instance, the following data field (hex, ASCII) has an actual length of 3:



## Difference between a column and a field in a data record

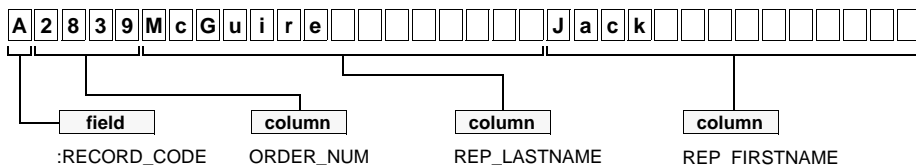
A data field in an input data record can be a column or a field.

- *Column* – contiguous data bytes that you load into a column in the target user table.
- *Field* – contiguous data bytes that you don't load into a user table. You use a field to assign a name to a portion of a record to be used in a condition, for instance, to indicate which records to load. You distinguish field names from column names by preceding them with a colon in a field\_spec.

For example, assume you're loading data into a user table that you created with the following CREATE TABLE statement:

```
CREATE TABLE JACK.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS95
```

Now assume you're loading this data record into that user table:



The first data field—A (:RECORD\_CODE)—is a field. You could use this field to select and load records that begin with a certain record code, like A. Fields are not loaded into user tables.

## Difference between a logical record and a physical record

One data record in a data file is a *physical record*. You can assemble a *logical record* from one or more physical records. Logical records can contain both fields and columns.

In the following example, each physical record corresponds to one logical record:

Physical records			Logical records		
A	2839	McGuire	Jack	A	2839 McGuire Jack
B	2388	Cornflake	Sue	B	2388 Cornflake Sue

In this example, two physical records make up one logical record:

Physical records				Logical records				
A	2839	McGuire	Jack	C	A	2839	McGuire	Jack 120.00
120.00					B	2388	Cornflake	Sue 528.45
B	2388	Cornflake	Sue	C				
528.45								

You can combine a fixed number of physical records into one logical record, or you can combine a varied number when physical records contain a *continuation field* with a *comparison value*. In the previous example, the C is the comparison value. In most cases, the FileTek FTP Data Loader removes continuation fields from physical records when assembling the logical record. LOB records are physical records that are automatically included in the logical record.

## Delimited data

One way to separate data fields in a data record is to use *delimiters*—markers that follow data fields or enclose them. The FileTek FTP Data Loader trims delimiters from data records; it doesn't load delimiters into user tables. Data records with

delimiters are called *delimited data*. There are two types of delimited data: terminated and enclosed.

**Note:** You can't use delimiters for native data types: BINARY (and synonyms RAW and BYTE), DECIMAL (and synonym NUMERIC), DOUBLE, FLOAT, INTEGER, SMALLINT, VARBINARY (and synonyms VARRAW and VARBYTE), and VARCHAR.

## Terminated data

*Terminated data* are data fields followed by a *termination delimiter*—any single character or a blank. For example, a termination delimiter could be a comma:

2	8	3	9	,	M	c	G	u	i	r	e	,	J	a	c	k	,	
2	3	8	8	,	C	o	r	n	f	l	a	k	e	,	S	u	e	,

Terminated data is read from the starting position of the data field up to, but not including, the termination delimiter. The end of the data record will serve as a delimiter (if needed).

## Enclosed data

*Enclosed data* are data fields preceded and followed by *enclosure delimiters*, such as parentheses:

(	2	8	3	9	)	(	M	c	G	u	i	r	e	)	(	J	a	c	k	)	
(	2	3	8	8	)	(	C	o	r	n	f	l	a	k	e	)	(	S	u	e	)

This type of data is read by skipping any characters until the first enclosure delimiter, then reading data until the second enclosure delimiter.

## Blank characters

Character-type data fields can contain blank characters, also called *spaces*, *whitespace*, or just *blanks*. *Leading blanks* are blanks at the beginning of a data field. *Trailing blanks* are blanks at the end of a data fields. The FileTek FTP Data Loader:

- Does not trim blanks that are part of a data field that's enclosed by enclosure delimiters
- May trim leading blanks from a character-type data field when optional enclosure delimiters are not present
- Does not trim trailing blanks from character-type data fields
- Does not trim leading and trailing blanks in VARCHAR data fields

## LOB data

LOB input data can originate from three locations:

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

## LOB data in the data file

You can include LOB data in a data file two ways:

- When a LOB value fits in an input data record (the LOB value or the input data record does not exceed 32 KB), you can define that LOB value as a loader data type (for instance, VARCHAR or VARBINARY) and include it with the

other input data in the data file. In this case, the following loader data types (source) are valid for BLOB and CLOB column (target) data types: BINARY, BINARY EXTERNAL, CHARACTER, VARBINARY, and VARCHAR.

- When a LOB value fits within or exceeds the record length, you can define it as a BLOB or CLOB data type. The LOB value then can span multiple records, but each record cannot exceed the maximum record length (32KB-1). A *LOB record* is a BLOB or CLOB data field that consists of one or more physical records in the input data file. LOB records are automatically considered part of the logical record.

Note the following guidelines for including LOB data fields in the input data file with other non-LOB data:

- Each LOB data field must start with a 64-bit length field followed by the data. The native values key affects the interpretation of the length.
- A LOB data field does not have to start at the beginning of a record. It can start in the same record as the non-LOB data.
- The last LOB data field must end at the end of a record.
- All other non-LOB data fields must still fit into a single record.
- LOB data fields must be placed after all non-LOB fields in the record.

See “Specifying a BLOB or CLOB data type” on page 3-109 for additional considerations about including LOB data in the input data file.

## **LOB data files on your client computer**

You can load LOB data from LOB data files on your client computer. Each LOB data file contains a value for a specific LOB column in a specific row. In this case, you must include a BLOB\_FILE or CLOB\_FILE data type specification in the



control file. You must also specify the directory path (where to access the LOB data file) and the LOB data file name. You can do this two ways:

- Specify both the directory path and the file name in the data file
- Specify the directory path in the control file and the file name in the data file

## LOB path and file name in the data file

You can specify directory paths and LOB data file names in the load data file. For example, assume you're loading photos from the `/home/tka/lobs/` directory on the client computer into a BLOB column. The control file LOAD DATA statement contains a BLOB\_FILE data type specification, and the data file contains the directory path and file names of the photos. You can include a BLOB\_FILE USER clause to specify the user name and password required to access those files on your client computer. If you omit the user information, the default is the StorHouse loader account and password.

### Control file

```
PHOTO BLOB_FILE USER 'tka'/tka
```

### Data file

```
/home/tka/lobs/lobfile1.jpg  
/home/tka/lobs/lobfile2.jpg  
/home/tka/lobs/lobfile3.jpg
```

### Client computer



```
/home/tka/lobs/  
lobfile1.jpg  
lobfile2.jpg  
lobfile3.jpg
```

In this example, the FileTek FTP Data Loader loads the contents of `lobfile1.jpg` into row 1, the contents of `lobfile2.jpg` into row 2, and so on.

## LOB path in the control file and file name in the data file

You can specify directory paths in the control file and LOB data file names in the data file. You use the PATH clause of the BLOB\_FILE and CLOB\_FILE data types on a LOAD DATA statement to specify a directory path in the control file. Additionally, you can include a USER clause to specify the user name and password required to access those files on your client computer.

In the following example, the control file contains the name of the directory path (home/tka/lobs) and the data file contains the names of the LOB data files (lobfile1.jpg, lobfile2.jpg, and lobfile3.jpg).

**Control file**

PHOTO BLOB\_FILE PATH '/home/tka/lobs/' USER 'tka'/tka

**Client computer****Data file**

lobfile1.jpg  
lobfile2.jpg  
lobfile3.jpg

/home/tka/lobs/  
lobfile1.jpg  
lobfile2.jpg  
lobfile3.jpg

## Multiple paths to LOB data files

You can specify multiple directory paths in a control file. The FileTek FTP Data Loader searches the list in the given order for each file name in the data file. For instance, assume:

- The /home/tka/blobs1/ directory contains the lobfile1.jpg and lobfile3.jpg files
- The /home/tka/blobs2/ directory contains the lobfile2.jpg file

For lobfile1.jpg, the FileTek FTP Data Loader searches the first path listed (home/tka/blobs1) in the control file and finds the file. For lobfile2.jpg, the FileTek FTP Data Loader searches the first path again, doesn't find the file, searches the second path (home/tka/blobs2), and finds the file.

**Control file**

PHOTO BLOB\_FILE PATH '/home/tka/blobs1/':'/home/tka/blobs2/'

**Client computer****Data file**

lobfile1.jpg  
lobfile2.jpg  
lobfile3.jpg

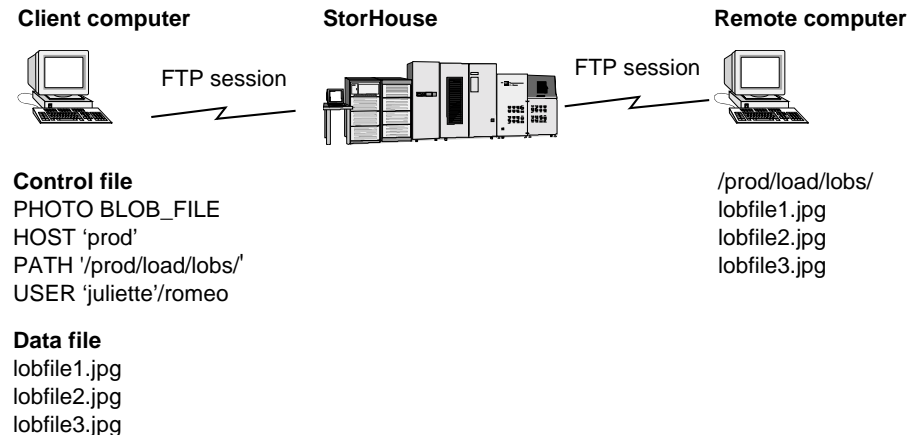
/home/tka/blobs1/  
lobfile1.jpg  
lobfile3.jpg  
  
/home/tka/blobs2/  
lobfile2.jpg

## LOB data files on a remote system

You can load LOB data from LOB data files on a remote system. The FileTek FTP Data Loader accesses the files on the remote host using a separate FTP connection. In this case, you must specify the following:

- Host name in the control file (with the HOST clause on the BLOB\_FILE or CLOB\_FILE data type) to identify the remote system.
- Directory paths in the control file (with the PATH clause on the BLOB\_FILE or CLOB\_FILE data type) or in the data file to identify the directories containing the LOB data files.
- User name and password in the control file (with the USER clause on the BLOB\_FILE or CLOB\_FILE data type) to establish an FTP session with the remote host.
- File names in the data file to identify the LOB data files.

In the following example, the FileTek FTP Data Loader establishes an FTP session with a remote host name prod using user name juliette and password romeo:



## **Guidelines for specifying a LOB file name in a data file**

Note the following for specifying a LOB file name in a data file:

- If the file name is a full UNIX path name, the FileTek FTP Data Loader ignores any path specification in the control file.
- The file name must be the same character set as the data (that is, in the `data_ccsid` optionally specified in the FTP put keyword).
- The file name cannot contain a host name.

# The control file

This chapter explains the format and provides examples of the LOAD DATA, LOAD INDEX, and MERGE statements. You include these statements in a control file.

## About the control file

A *control file* is a text file on your computer. To load data, you must create a control file containing one LOAD DATA statement that identifies the user table(s) you're loading and describes the input data. This control file can also contain the following:

- An Oracle OPTIONS clause with runtime options such as SKIP, ERRORS, ROWS, BINDSIZE, and so on. The FileTek FTP Data Loader does not process these options but allows them for compatibility.
- One or more SQL statements. The SELECT statement is the only SQL statement not allowed in a control file. See “Including SQL statements in a control file” on page 3-115 for more information.
- The data records, preceded by a data delimiter record (BEGINDATA). See “Including data in a control file” on page 3-115 for more information.

To load a deferred index, a control file can contain one LOAD INDEX statement and any SQL statements. To merge segments, a control file can contain one MERGE statement. The term *control statement* refers to LOAD DATA, LOAD INDEX, MERGE, and SQL statements.

## Control file guidelines

Guidelines for entering control statements in a control file are as follows:

- Only one LOAD DATA or LOAD INDEX or MERGE statement is allowed in a control file, and it must follow any StorHouse SQL statements.
- For compatibility, a LOAD DATA statement can include Oracle and DB2 clauses (for example, Oracle OPTIONS clause) that are not defined in the StorHouse syntax. The FileTek FTP Data Loader accepts but does not process these clauses.
- Control statements can span more than one line, and any new line can begin with any keyword.
- Character strings can be enclosed in single or double quotes, for example, 'A' or "A".
- Hexadecimal strings are preceded with X and enclosed in single or double quotes, for example, X'0001' or X"0001".
- Case is significant only in quoted strings and in HOST and USER clauses.
- Spaces are significant only in quoted strings. For example, 'A' is a different value from ' A '.
- You can include comments by starting them with two dashes (--).
- Do not place comments after the last control statement (before the BEGINDATA record).
- Control statements must end with a semicolon. The semicolon must appear at the end of a line after any comment.

## Format conventions

The control statement syntax has the same format conventions as StorHouse SQL. Those conventions are:

Convention	Description
UPPERCASE	Uppercase terms indicate keywords that are part of the syntax. Type keywords in any case.
lowercase	Lowercase terms refer to grammar elements (like <code>field_spec</code> ) and user-supplied values (like <code>segment_tag</code> ). When supplying values, only quoted strings are case sensitive.
<code>() , / * - ; : . + ' </code>	These characters are part of the syntax. Type them as shown.
<code>{ }</code>	Braces indicate that the item is required. When a list of items is enclosed in braces and separated by a vertical bar, you must choose one item.
<code>[ ]</code>	Brackets indicate that the item is optional.
<code> </code>	Vertical bar separates alternatives. You can specify one of the alternatives shown.
<code>...</code>	Ellipsis points indicate that you can repeat the part of the statement preceding them any number of times.

## SQL identifiers

The SQL identifiers that you supply on a control statement—like unquoted table, column, and field names—must follow these rules:

- Start with a letter
- Cannot exceed 32 characters
- Must be a contiguous string of characters (no blanks)
- Can consist of the characters a–z, A–Z, 0–9, `_` (underscore)
- Cannot be an SQL reserved word
- Are case insensitive (you can type them in any case) unless delimited

You must delimit SQL identifiers that don't follow the rules. The delimiters are double quotes. For example:

- "95orders" (doesn't start with a letter)
- SUE."ORDER DETAILS" (contains a blank)
- "july\$" (contains a special character)
- "GROUP" (is a reserved word)

Note that:

- Period(s) used for qualified table names must be outside the quotes, like in  
  
SUE."ORDER DETAILS"
- The preceding colon for a delimited field name in a field\_spec must be outside the quotes. For example:  
  
:"RECORD CODE" POSITION (1) CHAR



## Control statement formats

This section contains the syntax of the LOAD DATA, LOAD INDEX, and MERGE statements.

### LOAD DATA

```
LOAD [ DATA ]
[ CHARACTERSET { cset_name | ccsid } ]
[ { INFILE | INDDN } { [ NOENVIRON ] (infile_list) | * | - } ]
[ load_options ]
{ into_table_spec }...
```

Argument	Format
cset_name	WE8EBCDIC500   WE8PC850   WE8ISO8859P1
ccsid	500   850   819
(infile_list)	sm_file_name [ /group ] [ NOENVIRON ] [, sm_file_name [ /group ] [ NOENVIRON ] ]...
load_options	[ { DISCARDFILE   DISCARDN } sth_file_name [ /group ] ] [ { DISCARDS   DISCARDMAX } num_discards ] [ CONCATENATE num_lines ] [ CONTINUEIF continueif_condition ] [ PRESERVE BLANKS ] [ SUBSPACE ROTATE ]
continueif_condition	{ [ THIS   NEXT ] ( position )   LAST } operator { any_string   BLANKS }
position	start_column [ { :   - } end_column ]
operator	=   !=   ^=   <>   <   >   <=   >=
any_string	string   Xstring

Argument	Format
into_table_spec	INTO TABLE [ owner. ] table_name [ WHEN field_condition [ { AND   OR } field_condition ]... ] [ FIELDS fields_specs ] [ TRAILING [ NULLCOLS ] ] [ SAME SEGMENT ] [ DIFF[ERENT] SEGMENT ] [ SEGMENT segment_tag ] [ REPLACE SEGMENT [ [owner.] table_name.] segment_tag ] [ [ TABLE   VALUE   HASH   LOB ] SUBSPACE number ]... [ ( field_spec [ ,field_spec ]... ) ]
field_condition	{ (position)   column_name   field_name } operator { any_string   BLANKS }
fields_specs	[ CHAR ] [ NULLFLAGS ] [ delimiter_spec ] [ NULLIF ( EMPTY   BLANK ) ] [ DEFAULTIF ( EMPTY   BLANK ) ]
delimiter_spec	[ TERMINATED [ BY ] { WHITESPACE   'char'   X'hexbyte' } ] [ [ OPTIONALLY ] ENCLOSED [BY] { 'char'   X'hexbyte' } ] [ AND { 'char'   X'hexbyte' } ] ]
field_spec	{ :field_name   column_name } data_spec
data_spec	RECNUM   SEQUENCE ( start_num [ ,increment ] )   SYSDATE   CONSTANT any_value   position_spec
any_value	any_string   identifier   n
position_spec	[ POSITION ( position   * [ +num ] ) ] [ datatype_spec ] [ NULLIF field_condition ] [ DEFAULTIF field_condition ]
datatype_spec	See page 3-83 for loader data types.

## General notes about the LOAD DATA statement

Note the following:

- The LOAD keyword and INTO TABLE clause are the only required arguments.
- You can specify load\_options, into\_table\_spec, fields\_specs, delimiter\_spec, and position\_spec clauses in any order.
- If you omit a field\_spec list and a FIELDS clause, the FileTek FTP Data Loader generates a field\_spec for every column in the named table using the CREATE TABLE data types and lengths. You can do this only when the input data is relatively positioned and has the same order and same data types as the CREATE TABLE definition.
- If you omit a field\_spec list but include a FIELDS NULLFLAGS clause, the FileTek FTP Data Loader generates a field\_spec for every column in the named table using the CREATE TABLE data types and lengths.
- If you omit a field\_spec list but include a FIELDS CHAR clause, the FileTek FTP Data Loader generates a field\_spec for every column in the named table using the CHARACTER loader data type (with any associated delimiter\_spec).

## Description of the LOAD DATA clauses

The following table lists the function of each clause and keyword in the LOAD DATA statement. See the listed page for more information.

To	Use	See page
Load data already on StorHouse	INFILE or INDDN	3-10
Collect discarded records	DISCARDFILE or DISCARDN	3-15
Limit the number of discarded records	DISCARDS or DISCARDMAX	3-17

To	Use	See page
Specify the character set of the input data	CHARACTERSET	3-18
Combine a fixed number of physical records into one logical record	CONCATENATE	3-20
Combine a varied number of physical records into one logical record	CONTINUEIF	3-22
Retain blank characters in input data	PRESERVE BLANKS	3-31
Rotate among subspaces	SUBSPACE ROTATE	3-33
Identify the name of the user table	INTO TABLE	3-39
Choose which records to load	WHEN	3-40
Generate field_specs, identify NULL flags, specify default delimiters and other defaults	FIELDS	3-48
Load missing data fields with null values	TRAILING NULLCOLS	3-58
Load one or more segments at a time	SAME SEGMENT and DIFFERENT SEGMENT	3-60
Name a segment that may be replaced later	SEGMENT	3-63
Replace a segment	REPLACE SEGMENT	3-65
Select subspaces	SUBSPACE number	3-68
Load a record number into a column	RECNUM	3-78
Generate a sequence of values	SEQUENCE	3-78
Load the current date into a column	SYSDATE	3-79
Load a constant value into a column	CONSTANT	3-79
Specify the position of a data field	POSITION	3-80
Specify a character set for a data field	CHARSET	3-105
Identify a host containing a LOB data file	HOST	3-107
Specify a directory path to a LOB data file	PATH	3-107

To	Use	See page
Specify a user name and password to access a LOB data file	USER	3-108
Set a column to a null value	NULLIF	3-111
Set a column to the default value	DEFAULTIF	3-111

## LOAD INDEX

LOAD INDEX[ES] index\_name [ ,index\_name ]... [subspace\_clause]  
[SEGMENTS segment\_list]

Argument	Format
subspace_clause	SUBSPACE ROTATE   [ VALUE   HASH ] SUBSPACE number
segment_list	segment_list_item [ , segment_list_item ]...
segment_list_item	segment_range   segment
segment_range	first_segment - last_segment

## MERGE

```
{MERGE | COALESCE} INTO TABLE table_name [subspace_clause]
[SEGMENT segment_tag] [SEGMENTS segment_list]
[EXCLUDE segment_list] [MAXINSIZE n] [MINOUTSIZE n]
```

Argument	Format
subclause_clause	SUBSPACE ROTATE   [ VALUE   HASH   TABLE ] SUBSPACE number
segment_list	IDS segment_list_item [ , segment_list_item ]...   TAGS segment_tag [ , segment_tag ]...
segment_list_item	segment_range   segment
segment_range	first_segment - last_segment

## Loading data already on StorHouse

You can load data that's already on StorHouse, for instance:

- Discarded records in a discard file (which is a VRAM file)
- Data in one or more StorHouse VRAM files

Specify the INFILE (synonym INDDN) clause with a VRAM file name or a list of file names to load data already on StorHouse. If the input data is in a data file or combined control/data file on your host or if you're piping the data from a program, you can omit the INFILE clause or include INFILE \* or INDDN - to indicate that the data originates on your host.

You can use the INFILE clause in combination with the DISCARDFILE clause to collect discarded records while loading data in a host data file or in a VRAM file. See "Collecting discarded records in a discard file" on page 3-15 for more information about using the DISCARDFILE clause.

If you specify multiple files names and an error occurs during the load, the FileTek FTP Data Loader provides information about the location of the error within a specific file. For example, the following messages indicate that an error occurred in data record number 50363 of file KK.CDWDATA.3. The (336.96) means that this is the 97th blocked record in actual record number 336.

```
500- %L-I-XLDINFO, \08-MAR-2002:18:20:36 200000 data records
processed...\
```

```
500- %L-E-XLDSQLERR, SQL error \sqlcode=<-60017>\
```

```
500- %L-I-XLCONT, \data_record#343935 (KK.CDWDATA.3 record 50363
(336.96))\
```

```
500- %L-I-XLCONT, \table=KK_CDW field=FLAG data='L'\
```

```
500- %L-W-XLDSQLERR, SQL error
```

When you restart the load, the file being read at the checkpoint position will be opened at the proper record.

## Format of INFILE clause

```
{ INFILE | INDDN } { [ NOENVIRON ] (infile_list) | * | - }
```

where infile\_list is:

```
sm_file_name [ /group ] [ NOENVIRON ] [, sm_file_name [ /group]
[ NOENVIRON ] ]...
```

Argument	Description
NOENVIRON	(optional) Keyword to use when loading discarded records or data in a VRAM file created by another application program. If specifying a list of files, you can place this keyword before the list if it applies to all files in the list. Otherwise, place the keyword after any applicable file name and group.
sm_file_name	(required) Name of the StorHouse VRAM file that contains the data you are loading. If specifying a list of files, enclose the list in parentheses and use a comma to separate each file name and group. Parentheses are not required when specifying a single file.
/group	(optional) Name of the StorHouse file access group to which the VRAM file belongs. You can omit the group name if it is the default group of the StorHouse account ID you use to log in the StorHouse FTP server.
* or -	(optional) Argument for loading data in a file or from a program on your host instead of a VRAM file on StorHouse. This is the default if you omit the INFILE clause.

## Example INFILE clauses

This section contains example INFILE clauses.

### To load data in a VRAM file

You can load data from a VRAM file created by any application program interface (API) program that can write to StorHouse. In this case, however, you must use the NOENVIRON keyword in the INFILE clause. Additionally, 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 StorHouse group name.



For example, assume you're loading data in a StorHouse VRAM file called SEP2195 in a group called ATM. You would use this INFILE clause if ATM is your default group:

```
INFILE SEP2195 NOENVIRON
```

Or you would use this INFILE clause if ATM is not your default group:

```
INFILE SEP2195/ATM NOENVIRON
```

### **To load data from multiple VRAM files**

You can specify a list of VRAM files to load data from multiple files into a table. Enclose the list in parentheses and use a comma to separate each file name. You can place the NOENVIRON keyword before the file list if it applies to all files. For example:

```
LOAD INFILE NOENVIRON (FILE1/ATM, FILE2/ATM, FILE3/ATM)
```

Otherwise, you can place the NOENVIRON keyword after any specific file in the list if it applies to that file only.

```
LOAD INFILE (FILE1/ATM NOENVIRON, FILE2/ATM, FILE3/ATM)
```

### **To load discarded records**

You can load discarded records in a discard file by using the INFILE clause with the NOENVIRON keyword and by specifying the name of the discard file. If the discard file is not in the default group of the StorHouse account ID you use to log in the StorHouse FTP server, then you must also specify the StorHouse group name after the discard file name. See “Collecting discarded records in a discard file” on page 3-15 for more information about how to specify a discard file name as well as to limit the number of discarded records.

For instance, to load discarded records in a discard file called FILE1 in group STH, you would use this INFILE clause if STH is your default group:

```
INFILE FILE1 NOENVIRON
```

Or you would use this INFILE clause if STH is not your default group:

```
INFILE FILE1/STH NOENVIRON
```

Now if you also wanted to collect any discarded records generated while loading these discarded records, you would use the DISCARDFILE clause and specify the name of the discard file to collect any new discarded records. For example, if you wanted to load discarded records in FILE1 and collect discarded records in FILE2, you would specify these clauses:

```
INFILE FILE1 NOENVIRON  
DISCARDFILE FILE2
```

### **To load data from a data file and collect discarded records**

You can load data from a data file and collect discarded records by using INFILE \* and specifying a DISCARDFILE clause. See “Collecting discarded records in a discard file” on page 3-15 for more information about the DISCARDFILE clause.

For example, assume you’re loading data from a data file, and you want to collect discarded records in a discard file called FILE1 in group STH. You would use these clauses if STH is your default group:

```
INFILE *  
DISCARDFILE FILE1
```

Or you would specify these clauses if STH is not your default group:

```
INFILE *  
DISCARDFILE FILE1/STH
```

## Collecting discarded records in a discard file

*Discarded records* are logical records that do not meet the selection criteria in a WHEN clause. If you're using a WHEN clause and you want to collect discarded records in a discard file, then use the DISCARDFILE (synonym DISCARDN) clause.

If you're loading data from a file on your host or piping the data and want to collect discarded records, then use INFILE \* or INDDN - with the DISCARDFILE clause. If you're loading data from a VRAM file on StorHouse, then specify INFILE with the VRAM file name, followed by the DISCARDFILE clause. See "Identifying the user table to load" on page 3-39 for more information about the INFILE clause.

Note the following:

- A discard file is a VRAM file on StorHouse. Someone must create this file on StorHouse before you can collect discarded records. A load fails if the discard file does not exist on StorHouse.
- You must use your own API program to view discarded records in a discard file or to determine which records were discarded.
- You discard logical records, but the records in the discard file appear as a set of physical records (the original physical records that constituted the discarded logical record).
- If the input data is loaded via FTP, the discarded records are in blocked format. If the input data is in a StorHouse VRAM file, the discarded records are blocked if the VRAM file data was blocked.
- If you forget to include a DISCARDFILE clause and any logical records do not satisfy the selection criteria of a WHEN clause, you'll receive a warning message and the discarded records will not be saved.

- If you specify the same discard file name for different loads, then new discarded records are appended to existing discarded records in that discard file. Discarded records are not overwritten.
- A LOAD DATA statement contains only one DISCARDFILE clause; therefore, all discarded records are placed into one discard file no matter how many INTO TABLE clauses or WHEN clauses you use.
- Whenever discarded records are created, the FileTek FTP Data Loader issues an informational message indicating the number of records discarded.
- If you want to set a maximum number of discarded records, use the DISCARDS clause (see page 3-17).
- LOB records are not written to a discard file.

**Caution:** If a load fails and you restart it, the discard results won't be reliable because the discard file may contain duplicate records.

## Format of DISCARDFILE clause

{ DISCARDFILE | DISCARDDN } sth\_file\_name [ /group ]

Argument	Description
sth_file_name	(required) Name of the StorHouse VRAM file to contain discarded records during the load.
/group	(optional) Name of the StorHouse group to which the discard file belongs. If the discard file is not in the default group of the StorHouse account ID you use to log in the StorHouse FTP server, then you must also specify the StorHouse group name.

## Example DISCARDFILE clause

To collect discarded records in a discard file called FILE1 and a group called STH, you would use one of these clauses if STH is your default group:

```
DISCARDFILE FILE1  
DISCARDN FILE1
```

Or you would specify one of these clauses if STH is not your default group:

```
DISCARDFILE FILE1/STH  
DISCARDN FILE1/STH
```

## Limiting the number of discarded records

You can limit the number of discarded records by using the DISCARDS (synonym DISCARDMAX) clause. The number of discarded records as well as the records themselves are logical records. Note the following:

- A load fails when the number of discarded records exceeds this limit.
- If you include a DISCARDFILE clause but omit a DISCARDS clause, then there's no limit to the number of discarded records.
- If you include a DISCARDS clause but omit a DISCARDFILE clause, then discarded records will be limited but not saved.
- The meaning of DISCARDS 0 or DISCARDMAX 0 depends on the db\_ref value on the FTP put command.
  - If db\_ref is DB2, there's no limit to the number of discarded records.
  - If db\_ref is Oracle or ANSI, no discarded records are allowed.

## Format of DISCARDS clause

{ DISCARDS | DISCARDMAX } num\_discards

Argument	Description
num_discards	(required) Maximum number of discarded records for each data file. Valid values are 0 (none allowed for Oracle or ANSI or no limit for DB2) through 2147483646.

---

## Example DISCARDS clause

To limit the number of discarded records to 500, you would specify one of these clauses:

DISCARDS 500  
DISCARDMAX 500

## Specifying the character set of the input data

You can specify the character set of the input data. When you do, any character-based data and padding are converted to this character set. StorHouse supports ISO, EBCDIC, and PC character sets. The default is ISO.

**Caution:** If you transfer the data with the BINARY transfer type and the character set is *not* ISO 8859-1, you *should* specify an alternate character set. If you transfer the data with the ASCII transfer type from a non-ASCII machine, your client FTP tool will convert the data to ASCII. In this case, you should *not* specify an alternate data character set.

You have four options for specifying the character set of the input data you're loading:

- Take the default (ISO 8859-1)
- Provide a value for the `data_ccsid` keyword on the FTP put command (see page 4-5)
- Use the `CHARACTERSET` clause in a `LOAD DATA` statement
- Use the `CHARSET` keyword in a `datatype_spec` for an individual data field of type `CHAR`, `CLOB`, `CLOB_FILE`, or `EXTERNAL` (see page 3-105)

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.

## Format of CHARACTERSET clause

CHARACTERSET { cset\_name | ccsid }

Argument	Description
cset_name	(required if no ccsid is specified) Name of the character set. Valid values: <ul style="list-style-type: none"><li>■ WE8ISO8859P1 for ISO 8859-1 character set</li><li>■ WE8EBCDIC500 for EBCDIC character set</li><li>■ WE8PC850 for PC character set</li></ul>
ccsid	(required if no cset_name is specified) CCSID of the 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 input data is the PC character set. You would specify one of these CHARACTERSET clauses:

```
CHARACTERSET 850
CHARACTERSET WE8PC850
```

## Concatenating a fixed number of physical records into a logical record

You can create logical records from the *same* number of physical records by using the CONCATENATE clause. For example, when every pair of data records corresponds to a row in a user table, you could use this clause to combine the first



pair of physical records into one logical record, the second pair of physical records into another logical record, and so on. If the number of physical records to be combined varies, then use the CONTINUEIF clause described on page 3-22 instead of the CONCATENATE clause.

**Note:** If the LOAD DATA statement contains multiple into\_table\_specs, then the CONCATENATE clause applies to *all* user tables being loaded. The CONCATENATE clause does not apply to LOB records in the data file.

## Format of CONCATENATE clause

CONCATENATE num\_lines

Argument	Description
num_lines	(required) Number of physical records to combine. This number must be greater than 1.

## Example CONCATENATE clause

Assume that two physical records in a data file make up one row in a user table. You would use this CONCATENATE clause to combine records 1 and 2, records 3 and 4, records 5 and 6, and so on:

CONCATENATE 2

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4

Then these are the logical records:

1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4

## Combining a varied number of physical records into a logical record

Use the CONTINUEIF clause when the number of physical records to be combined varies, or to combine physical records that satisfy a certain *condition*. A condition is true when a continuation field contains or does not contain a *comparison value* in a specified *location*.

For example, with a CONTINUEIF clause you can combine physical records that contain an \* (comparison value) in column 80 (location). Or you can combine physical records that don't contain blanks (comparison value) in columns 1 and 2 (location).

**Note:** The CONTINUEIF clause applies to *all/into\_table\_specs* in a LOAD DATA statement. The CONTINUEIF clause does not apply to LOB records in the data file.

### Format of CONTINUEIF clause

CONTINUEIF continueif\_condition

where continueif\_condition:

---

Combining a varied number of physical records into a logical record

{ [ THIS | NEXT ] ( position ) | LAST } operator { any\_string | BLANKS }

Argument	Description
THIS	(optional and the default) If the condition is true in the current record, then append the next physical record to it, continuing until the condition is false. If the condition is false in the current record, then it is the last physical record of the current logical record. The continuation field is removed from the physical record.
NEXT	(optional) If the condition is true in the next record, then append it to the previous (current) physical record. If the condition is false in the next record, then the current physical record is the last physical record of the current logical record. The continuation field is removed from the physical record.
(position)	<p>Starting (required) and ending (optional) column numbers of the continuation field in the physical record. The format is:</p> <p>start_column [ { :   - } end_column ]</p> <p>For example (1:2) or (1-2). The position is required if you specify THIS or NEXT. The first position in a record is 1. This range of columns is removed from every physical record when the logical records are assembled. This is the only place where you'll specify column numbers in the physical record instead of the logical record.</p>
LAST	(required if you omit (position)) If the condition is true in the last non-blank character(s) in the current physical record, then append the next physical record to it, continuing until the condition is false. If the condition is false in the current record, then the current record is the last physical record of the current logical record. The continuation field is <i>not</i> removed from the physical record; therefore, it remains in the logical record.
operator	<p>(required) Comparison operator. Specify one of the following:</p> <p>= (equal), != or ^= or &lt;&gt; (not equal), &lt; (less than), &gt; (greater than), &lt;= (less than or equal), &gt;= (greater than or equal)</p>

## 3

**The control file**


---

Combining a varied number of physical records into a logical record

Argument	Description
any_string	(required if you omit BLANKS) Comparison value. The format is: string   Xstring
string	String of characters enclosed in single or double quotes. Use string for continuation fields containing character data. This character string must match the case of the input data.
Xstring	String of hex digits preceded with X and enclosed in single or double quotes, for example, x'01ff'. Use X string for continuation fields containing binary data.
BLANKS	(required if you omit any_string) Keyword to specify a blank as the comparison value.

**Note:** For compatibility with Oracle, parentheses are allowed, but optional, from (position) to the end of the continueif\_condition. For example:

```
CONTINUEIF NEXT ( (5:7) = 'ABC' )
```

## Example CONTINUEIF clauses

This section contains example CONTINUEIF clauses.

### To combine the current physical record with the next one

You can combine the current physical record with the next one by using the THIS keyword in a CONTINUEIF clause. When you use CONTINUEIF THIS, the continuation field is removed from every physical record before the logical records are assembled. Note that THIS is the default if you do not specify NEXT, LAST, or THIS.

For example, assume that the comparison value is an \* located in column 1. You would use one of these CONTINUEIF clauses:

CONTINUEIF THIS (1) = '\*'

or

CONTINUEIF (1) = "

If record 1 contains an asterisk in column 1, then record 2 will be combined with record 1. If record 2 also contains an asterisk in column 1, then record 3 will be combined with records 1 and 2. If record 2 doesn't contain an asterisk in column 1, it is still combined with record 1, but record 3 starts a new logical record.

So if these are the physical records:

*	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2
*	3	3	3	3	3	3	3	3	3	3
*	4	4	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5

Then these are the assembled logical records:

1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2								
3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5

Notice that the continuation field (column 1) is removed from all physical records. The \* and blanks are not part of the logical records.

## To combine the next physical record with the previous one

You can combine the next physical record with the previous one by using the NEXT keyword in a CONTINUEIF clause. When you use CONTINUEIF NEXT, the continuation field is removed from every physical record before the logical records are assembled.

## 3

## The control file

---

Combining a varied number of physical records into a logical record

For example, assume that the comparison value is ABC located in columns 5, 6, and 7. You would use this CONTINUEIF NEXT clause:

```
CONTINUEIF NEXT (5:7) = 'ABC'
```

If record 2 contains ABC in columns 5, 6, and 7, then record 2 will be appended to record 1. If record 2 does not contain the comparison value in the indicated columns, then it begins a new logical record.

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	A	B	C	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	A	B	C	5	5	5

Then these are the assembled logical records:

1	1	1	1	1	1	1	1						
2	2	2	2	2	2	2	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	5	5	5	5	5	5

Notice that the continuation field (columns 5 through 7) is removed from all physical records.

### To use the last non-blank data column as the comparison value

The LAST keyword in a CONTINUEIF clause is similar to the THIS keyword in that you use it to combine the current physical record with the next one. LAST differs from THIS as follows:

- The test is *always* made against the last non-blank character(s).

- You don't specify starting and ending column numbers to identify the location of the continuation field.
- The continuation field is not removed from the physical records.

For instance, assume that the comparison value is a comma located in the last non-blank data column. You would use this CONTINUEIF LAST clause:

CONTINUEIF LAST = ','

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1	,
2	2	2	2	2	2	2	2	2	2	
3	3	3	3	3	3	3	3	3	3	,
4	4	4	4	4	4	4	4	4	4	,
5	5	5	5	5	5	5	5	5	5	

Then these are the assembled logical records:

1	1	1	1	1	1	1	1	1	1	,	2	2	2	2	2	2	2	2	2										
3	3	3	3	3	3	3	3	3	3	,	4	4	4	4	4	4	4	4	4	,	5	5	5	5	5	5	5	5	

Notice that the continuation field is *not* removed from the physical records. The commas and blanks are part of the logical records.

### To specify the starting column number of a continuation field

You must specify a starting column number with the THIS or NEXT keyword to identify the starting location of the continuation field in a physical record. Enclose the column number in parentheses. Note that column numbers start with 1. When you specify only a starting column number, the length of the continuation field is equal to the length of the comparison value that you provide.

For example, assume the comparison value is the letter C, located in column 79. You would use this CONTINUEIF clause:

```
CONTINUEIF THIS (79) = 'C'
```

In this example, the length of the continuation field is 1 because the comparison value is the letter C. If the current physical record contains the letter C in column 79, then the next physical record is combined with it.

### **To specify the starting and ending column numbers of a continuation field**

You can specify both the starting and ending column numbers with the THIS or NEXT keyword to identify the location of the continuation field in the physical record. Enclose the column numbers in parentheses, and use a colon or dash to separate the starting column number from the ending column number.

For example, assume the comparison value is ABC located in columns 5, 6, and 7. You want to combine the next physical record with the previous one, so you would specify the following CONTINUEIF clause:

```
CONTINUEIF NEXT (5:7) = 'ABC'
```

If the comparison value is shorter than the length defined by the starting and ending column numbers, then the FileTek FTP Data Loader pads the value on the right with blanks (if a character string) or zeros (if hex digits). For example, if

```
CONTINUEIF NEXT (5:7) = 'AB'
```

then the actual selection criteria is 'AB ' (blank padded on the right).

If the comparison value is longer than the length defined by the starting and ending column numbers, then the FileTek FTP Data Loader trims the value on the right. For instance, if

```
CONTINUEIF NEXT (5:7) = 'ABCD'
```



then the D is trimmed and not considered part of the comparison value. You will receive a warning message when any non-blank characters or non-zero bytes are trimmed.

### **To use a character string as a comparison value**

If the type of data in the continuation field is character, then specify the comparison value as a character string enclosed in single or double quotes. The character string *must* match the case (UPPER, lower, or Mixed) of the input data.

For instance, if the comparison value is the letter c, located in column 79 of the physical record, you would specify the comparison value as a character string:

```
CONTINUEIF (79) = 'c'
```

### **To use a hex string as a comparison value**

If the type of data in the continuation field is binary, then specify the comparison value as an even number of hex digits preceded with X and enclosed in single or double quotes.

For example, if the comparison value is the number 1, data type SMALLINT, located in column 1 of the physical record, you would specify this comparison value as a string of hex digits:

```
CONTINUEIF (1:2) = X'0001'
```

**Note:** Be sure to use the correct byte order when specifying a hex string. For instance, if your host operating system is SPARC, then specify the most significant byte first. See “Native data types” on page 2-4 for more information about byte order.

## To use blank characters as a comparison value

You can specify one or more blank characters as a comparison value by using the **BLANKS** keyword in the **CONTINUEIF** clause. You can use **BLANKS** with the **THIS** and **NEXT** keywords; **LAST** tests non-blank data only. The continuation field is removed from every physical record before the logical records are assembled.

**Note:** The default length of **BLANKS** is 1; so, if you specify a starting column only, the length of the continuation field is one blank.

For example, to combine the next physical record with the current one whenever column 10 and column 11 contain a blank character, you would use this **CONTINUEIF** clause:

**CONTINUEIF THIS (10:11) = BLANKS**

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1	
2	2	2	2	2	2	2	2	2		
3	3	3	3	3	3	3	3	3		
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5		
6	6	6	6	6	6	6	6	6	6	6

Then these are the assembled logical records:

1	1	1	1	1	1	1	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Notice that the continuation field (columns 10 and 11) is removed from all physical records. Also note that record 1 does not contain a blank character in column 10; therefore, the condition is false and record 2 is not appended to record 1.

## To use a not equal comparison operator

You can specify that a continuation field *not equal* a specific comparison value by using any one of these “not equal” comparison operators in a CONTINUEIF clause: != or ^= or <>

For instance, to combine the next physical record with the current one whenever column 10 *does not* contain a blank character, you would use this CONTINUEIF clause:

```
CONTINUEIF THIS (10) <> BLANKS
```

So if these are the physical records:

1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	
3	3	3	3	3	3	3	3	3	
4	4	4	4	4	4	4	4	4	
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	

Then these are the assembled logical records:

1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3								
4	4	4	4	4	4	4	4	4								
5	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6

## Preserving blanks in input data

The FileTek FTP Data Loader automatically trims leading blanks from a delimited data field that’s supposed to be enclosed but isn’t. If the previous data

field is not TERMINATED BY WHITESPACE, you can retain the leading blanks by using the PRESERVE BLANKS clause.

**Note:** Blanks within enclosure delimiters are always preserved.

Before reading this section, it may be helpful to have a basic understanding of delimited data and blanks. Consider reading the following sections first:

- “Delimited data” (see page 2-7)
- “Blank characters” (see page 2-9)
- “Generating field\_specs, identifying NULL flags, specifying default delimiters and other defaults” (see page 3-48)

## Format of PRESERVE BLANKS clause

PRESERVE BLANKS

## Example PRESERVE BLANKS clause

Suppose you specified this FIELDS clause:

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'

But the optional enclosure delimiter (double quote) isn't present in the first two data fields of this data record.

	2	8	3	9	,		M	c	G	u	i	r	e	,		"	J	a	c	k		"	,
--	---	---	---	---	---	--	---	---	---	---	---	---	---	---	--	---	---	---	---	---	--	---	---

Without the PRESERVE BLANKS clause, the leading blanks would be trimmed and the following data fields would be loaded:

2	8	3	9	M	c	G	u	i	r	e	J	a	c	k	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

data field 1      data field 2      data field 3

With the PRESERVE BLANKS clause, the leading blanks would be retained and the following data fields would be loaded:

	2	8	3	9		M	c	G	u	i	r	e	J	a	c	k	
--	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---	---	--

data field 1      data field 2      data field 3

## Rotating among subspaces

You can rotate user table components (table data, value indexes, hash indexes, and LOB data) among subspaces by using the SUBSPACE ROTATE clause. You might do this if you're loading multiple segments of a table at a time and do not need to select certain subspaces for components. Rotation is not necessary when loading one segment or when a user tablespace contains one subspace for all component types or one subspace for each component type. You use SUBSPACE ROTATE with multiple INTO TABLE clauses and DIFFERENT SEGMENT clauses. You can also use SUBSPACE ROTATE to rotate among subspaces for successive result segments of a LOAD INDEX or MERGE operation.

Note the following:

- You must specify the SUBSPACE ROTATE clause before the INTO TABLE clauses of the LOAD DATA and MERGE statements.
- You cannot use both a SUBSPACE ROTATE and a SUBSPACE number clause in a LOAD DATA, LOAD INDEX, or MERGE statement. Only one of these clauses is allowed.

- If you omit both the SUBSPACE ROTATE and the SUBSPACE number clause, then the FileTek FTP Data Loader uses the lowest-numbered subspace that allows the component type.
- Rotation of subspaces occurs independently for each component type, that is, the data loader rotates among applicable subspaces for table data, for hash indexes, for value indexes, and for LOB data.
- If an INTO TABLE clause specifies (or defaults to) SAME SEGMENT, it shares the subspace of the most recent INTO TABLE clause for the same table.

For example, in the following LOAD DATA statement, the second INTO TABLE clause shares the same subspace as the first INTO TABLE clause, but the third INTO TABLE clause uses the next subspace.

```
LOAD
SUBSPACE ROTATE

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE= '1/1/2000'

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '1/31/2000'

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '2/1/2000'
DIFF SEGMENT;
```

- If a user table has multiple indexes (for instance, two hash indexes), and those indexes are assigned to the same user tablespace, each *index* does not rotate among subspaces but rather both index *files* (for instance, both hash index files) for an INTO TABLE clause use the same subspace.
- When indexes for a user table are assigned to different user tablespaces, each user tablespace can contain a different number of subspaces for each index type. For example, one user tablespace may contain two subspaces for hash

indexes and three subspaces for value indexes, while the other user tablespace may contain four subspaces for hash indexes and five subspaces for value indexes.

## Format of SUBSPACE ROTATE clause

SUBSPACE ROTATE

## Example SUBSPACE ROTATE clauses

This section contains example SUBSPACE ROTATE clauses.

### To rotate among subspaces in a user tablespace

Assume you're loading two segments. The user table has a hash index and a value index assigned to the same user tablespace as the table. The user tablespace contains the following subspaces:

Subspace number	OBJECT_TYPE
1	blank (all component types)
2	T (table data only)
3	V (value indexes only)

The following LOAD DATA statement creates two table data files, two hash index files, and two value index files for the ATM.TRANSACTIONS table:

```
LOAD
SUBSPACE ROTATE

INTO TABLE ATM.TRANSACTIONS
WHEN (1) = 'A'

INTO TABLE ATM.TRANSACTIONS
WHEN (1) = 'B'
DIFF SEGMENT;
```

For the first INTO TABLE clause (segment 1), the FileTek FTP Data Loader uses subspace 1 for the table data file, hash index file, and value index file (because subspace 1 is the lowest-numbered subspace that allows all component types).

For the second INTO TABLE clause (segment 2), the FileTek FTP Data Loader uses the following subspaces:

- Subspace 2 for the table data file (because it's the next subspace that allows table data)
- Subspace 1 for the hash index file (because it's the only subspace that allows hash indexes)
- Subspace 3 for the value index file (because it's the next subspace that allows value indexes)

### **To rotate among subspaces in multiple user tablespaces**

Assume you're loading two segments. The user table has two hash indexes, two value indexes, and two LOB columns (to be stored out-of-line). The table, hash index 1, and value index 1 are assigned to user tablespace 1. Hash index 2, value index 2, and both LOB columns are assigned to user tablespace 2.



User tablespace 1 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)
3	H (hash indexes only)
4	H (hash indexes only)
5	V (value indexes only)
6	V (value indexes only)

User tablespace 2 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	H (hash indexes only)
2	V (value indexes only)
3	L (LOB data only)
4	L (LOB data only)

The following LOAD DATA statement creates two table data files, four hash index files, four value index files, and two LOB subsegment files for the POS.TRANSACTIONS table:

```
LOAD
SUBSPACE ROTATE

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE= '1/1/2000'
```

```
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE = '2/1/2000'  
DIFF SEGMENT;
```

For the first INTO TABLE clause (segment 1), the FileTek FTP Data Loader uses the following subspaces:

- Subspace 1 in user tablespace 1 for the table data file (because it's the lowest-numbered subspace that allows table data)
- Subspace 3 in user tablespace 1 for the hash index file of hash index 1 (because it's the lowest-numbered subspace that allows hash indexes)
- Subspace 5 in user tablespace 1 for the value index file of value index 1 (because it's the lowest-numbered subspace that allows value indexes)
- Subspace 1 in user tablespace 2 for the hash index file of hash index 2 (because it's the only subspace that allows hash indexes)
- Subspace 2 in user tablespace 2 for the value index file of value index 2 (because it's the only subspace that allows value indexes)
- Subspace 3 in user tablespace 2 for both LOB subsegment files (because it's the lowest-numbered subspace that allows LOB data)

For the second INTO TABLE clause (segment 2), the FileTek FTP Data Loader uses the following subspaces:

- Subspace 2 in user tablespace 1 for the table data file (because it's the next subspace that allows table data)
- Subspace 4 in user tablespace 1 for the hash index file of hash index 1 (because it's the next subspace that allows hash indexes)
- Subspace 6 in user tablespace 1 for the value index file of value index 1 (because it's the next subspace that allows value indexes)

- Subspace 1 in user tablespace 2 for the hash index file of hash index 2 (because it's the only subspace that allows hash indexes)
- Subspace 2 in user tablespace 2 for the value index file of value index 2 (because it's the only subspace that allows value indexes)
- Subspace 4 in user tablespace 2 for both LOB subsegment files (because it's the next subspace that allows LOB data)

## Identifying the user table to load

An INTO TABLE clause specifies the name of the user table you are loading. You can provide the fully qualified table name (with the owner name), or you can omit the owner name if the account ID you use to log into the StorHouse FTP server is the owner of the user table. The table name can be a synonym but it cannot be a view.

One INTO TABLE clause is required for each user table. For instance, to load two user tables with one data file, you would specify two INTO TABLE clauses. The maximum number of INTO TABLE clauses you can include in a LOAD DATA statement depends on the value of the StorHouse SQL\_LDR\_MAXINTO system parameter set for your organization. For more information about this system parameter, refer to the *StorHouse Database Administration Guide*.

**Caution:** Your load will fail if the number of INTO TABLE clauses exceeds the maximum number.

## Format of INTO TABLE clause

INTO TABLE [ owner. ] table\_name

Argument	Description
owner.	(optional) StorHouse account ID of the owner of the user table. You can omit the owner name if the account ID you use to log into the StorHouse FTP server is the owner.
table_name	(required) Name assigned to the user table on the CREATE TABLE or CREATE SYNONYM statement.

## Example INTO TABLE clause

Assume the account ID you use to log into the StorHouse FTP server is JULIETTE, and you're loading a user table called ORDERS owned by JACK. In this case, you must include the owner name:

INTO TABLE JACK.ORDERS

## Choosing which records to load

You can choose to load or discard a record by using the WHEN clause. This clause tests one or more conditions in a logical record. If you omit the WHEN clause, which you can because it's optional, then *all* logical records are loaded into the specified user table. If you include a WHEN clause, enter it after the INTO TABLE clause.

The WHEN clause gives you a number of options for selecting the data that you want to load, whether you're loading one user table or multiple user tables. With one data file you can:

- Load logical records that meet one or more conditions, that is, where certain data fields contain certain values.
- Load *all* logical records into multiple user tables. In other words, you can load different user tables with the same data records.
- Load *all* logical records into some user tables and *some* logical records into other user tables.
- Load *some* logical records into some user tables and *some* logical records into other user tables.

With the WHEN clause, you specify criteria for selecting a logical record. This selection criteria includes the location of a value you want tested as well as the value itself. Logical records that satisfy the selection criteria are loaded.

Note the following:

- You cannot use a WHEN clause for delimited data or relatively positioned data fields.
- When specifying multiple criteria (AND and OR), ANDs have a higher precedence than ORs. You can use parentheses to alter or force precedence.
- For clarity, you can use parentheses to enclose a field condition.
- You can use the DISCARDFILE clause to collect logical records that don't satisfy the selection criteria in any WHEN clause. See "Collecting discarded records in a discard file" on page 3-15 for more information.

## Format of WHEN clause

WHEN field\_condition [ { AND | OR } field\_condition ]...

Argument	Description
field_condition	{ ( position )   column_name   field_name } operator { any_string   BLANKS }
(position)	<p>(required if not using column_name or field_name) Starting (required) and ending (optional) column numbers of the column or field that contains the criteria you are testing. The format is:</p> <p>start_column [ { :   - } end_column ]</p> <p>For example: (1:3) or (1-3) indicates the criteria starts in column 1 and ends in column 3.</p> <p>Note that when specifying a (position) with a character (not a hex) string, the data is assumed to be in the default data CCSID.</p>
column_name	(required if not using (position) or field_name) Name of a column that contains the criteria you are testing.
field_name	(required if not using (position) or column_name) Name of a field that contains the criteria you are testing.
operator	<p>(required) Comparison operator. Specify one of the following:</p> <p>= (equal), != or ^= or &lt;&gt; (not equal), &lt; (less than), &gt; (greater than), &lt;= (less than or equal), &gt;= (greater than or equal)</p>
any_string	<p>(required if not using BLANKS) Criteria you are testing. Specify one of the following, depending on the type of data in the data field:</p> <ul style="list-style-type: none"> <li>■ string – string of characters enclosed in single or double quotes. Use string for data fields containing character data. This character string must match the case (UPPER, lower, or Mixed) of the input data.</li> <li>■ Xstring – string of hexadecimal digits preceded with X and enclosed in single or double quotes. Use Xstring for data fields containing binary data.</li> </ul>
BLANKS	(required if not using any_string) Keyword to test one or more blanks.

Argument	Description
AND	(optional) Keyword for testing multiple values. A logical record is loaded when all conditions specified for AND are true.
OR	(optional) Keyword for testing multiple values. A logical record is loaded when at least one condition specified for OR is true.

## Example WHEN clauses

This section contains example WHEN clauses.

### To specify the starting column number of the selection criteria

You can specify a starting column number to identify the starting location of the value you want tested as selection criteria. Enclose the column number in parentheses. When you specify a starting column number only, the length of the selection criteria is derived from the comparison string.

For example, assume the selection criteria is the number 3, data type SMALLINT, located at column 1 of the logical record. You would use this WHEN clause:

```
WHEN (1) = X'0003'
```

In this example, because you didn't specify an ending column number, the length is derived from the comparison value, which is 2.

### To specify starting and ending column numbers of the selection criteria

You can specify both the starting and ending column numbers to identify the location of the value you want tested as selection criteria. Enclose the column numbers in parentheses, and use a colon or a dash to separate the starting column number from the ending column number.

For example, assume the value you want tested is the character string ABC located in columns 1 through 3. You would specify this WHEN clause:

```
WHEN (1:3) = 'ABC'
```

If the value is shorter than the length defined by the starting and ending column numbers, then the FileTek FTP Data Loader pads that value on the right with blanks (if a character string) or zeros (if hexadecimal digits). For example, if

```
WHEN (1:3) = 'AB'
```

then the actual selection criteria is 'AB ' (blank padded on the right).

If the value is longer than the length defined by the starting and ending column numbers, then the FileTek FTP Data Loader trims that value on the right. For instance, if

```
WHEN (1:3) = 'ABCD'
```

then the D is trimmed and not considered part of the selection criteria. You'll receive a warning message when any non-blank characters or non-zero bytes are trimmed.

### **To use a column name to identify the selection criteria**

You can use a column name instead of the starting and ending column numbers or a field name to identify the value you want tested as selection criteria. This column name must *exactly* match the column name in the table's CREATE TABLE statement. You can delimit column names that do not adhere to the conventions of StorHouse SQL identifiers (see page 3-3 for conventions).

When you use a column name in a WHEN clause, the field\_spec for that column identifies the location and length of the value. For example, assume the name of



the column that contains the value you want tested is CUSTOMER\_NUMBER.  
Here's the WHEN clause:

```
WHEN CUSTOMER_NUMBER = '3392003900'
```

Here's the field\_spec for the CUSTOMER\_NUMBER column:

```
CUSTOMER_NUMBER POSITION (6) INTEGER EXTERNAL(10)
```

In this example, the WHEN clause identifies the value that you are testing and the field\_spec identifies the location and length of that value. All logical records with the integer 3392003900 starting in position 6 will be loaded.

### **To use a field name to identify the selection criteria**

You can use a field name instead of the starting and ending column numbers or a column name to identify the value you want tested as selection criteria. You must delimit field names that do not follow the conventions of StorHouse SQL identifiers (see page 3-3 for conventions).

When you use a field name in a WHEN clause, the field\_spec for that field identifies the location and length of the value. For example, assume the name of the field that contains the value you want tested is RECORD\_CODE and the value you are testing is B. Here's the WHEN clause:

```
WHEN RECORD_CODE = 'B'
```

Here's the field\_spec for the RECORD\_CODE field:

```
:RECORD_CODE POSITION (1) CHAR
```

In this example, the WHEN clause identifies the value that you are testing and the field\_spec identifies the location and length of that value. Field names in a WHEN clause must *not* be preceded by a colon. Field names in a field\_spec must be preceded by a colon. All logical records with the letter B in position 1 will be loaded.

### **To use a character string as selection criteria**

When the value you are testing as selection criteria is character data, enclose that value in single or double quotes. Note the following:

- The character string *must* match the case (UPPER, lower, or Mixed) of the input data.
- If the selection column or field is of VARCHAR type, then the comparison is made against the character data only; the length part (first 2 bytes) is not included. If the actual length of the data field differs from the length of the tested value, the test can be true only if non-significant (blank) bytes are trimmed from the larger value.
- If the value of the data\_ccsid keyword on the FTP put command differs from the character set of the control statements, then the FileTek FTP Data Loader converts all comparison values that are character strings to the CCSID of the compared data field.

For example, if the selection criteria is the letter A, data type CHAR, located in column 1 of the logical record, then you would specify this selection criteria as a character string enclosed in quotes:

```
WHEN (1) = 'A'
```

### **To use a hexadecimal string as selection criteria**

When the value you are testing as selection criteria is binary data, precede the value with X and then enclose the value in single or double quotes. Be sure to use the correct byte order when specifying a hexadecimal string. For instance, if your host operating system is SPARC, then specify the most significant byte first. See “Data type considerations” on page 2-4 for more information about byte order.

For example, assume the selection criteria is the number 1, data type SMALLINT, located at column 1 of the logical record. You would specify this selection criteria

as a string of hexadecimal digits preceded with X and enclosed in single or double quotes:

```
WHEN (1:2) = X'0001'
```

### **To test blanks**

You can test one or more blanks as selection criteria by using the **BLANKS** keyword in a **WHEN** clause. The default length of **BLANKS** is 1; therefore, if you specify a starting column only, the selection criteria is one blank.

For instance, you would specify this **WHEN** clause to load all logical records that contain a blank character in column 80:

```
WHEN (80) = BLANKS
```

This **WHEN** clause tests for two blanks (in columns 79 and 80):

```
WHEN (79:80) = BLANKS
```

### **To test multiple values (using AND)**

You can test multiple values by using the **AND** keyword in a **WHEN** clause. A logical record is loaded when *all* conditions are true. For example:

```
WHEN (1) = 'A' AND STOCK_NUMBER = '1023992'
```

In this example, all records with the letter A in column 1 and where the column **STOCK\_NUMBER** contains a value of 1023992 will be loaded.

### **To test one value or another (using OR)**

You can test one value or another by using the **OR** keyword in a **WHEN** clause. A logical record is loaded when at least one of the conditions is true. For example:

```
WHEN (11:13) = '800' OR (11:13) = '888'
```

In this example, a logical record with either 800 or 888 in columns 11 through 13 will be loaded.

### **To test one value or another and multiple values (using OR and AND)**

You can specify multiple selection criteria by using both the AND and OR keywords in a WHEN clause. A logical record is loaded when *all* AND conditions are true and when at least one of the OR conditions is true. You can use parentheses to enforce precedence. For example:

```
WHEN ((1:3)= '301' OR (1:3)= '410') AND ((11:13)= '301' OR (11:13)= '410')
```

In this example, a logical record with either 301 or 410 in columns 1 through 3 *and* with either 301 or 410 in columns 11 through 13 will be loaded.

## **Generating field\_specs, identifying NULL flags, specifying default delimiters and other defaults**

You can use a FIELDS clause to:

- Generate a field\_spec as the CHARACTER loader data type for every column in the named user table. To do this, include the CHAR keyword with the FIELDS clause and omit the field\_spec for all data fields.
- Indicate each input data record starts with a sequence of 1-byte NULL flags (T for NULL or F for NOT NULL). To do this, include the NULLFLAGS keyword with the FIELDS clause and omit the field\_spec for all data fields.
- Specify a default delimiter for character-based data fields (CHARACTER, BLOB\_FILE, CLOB\_FILE, and all EXTERNAL data types). See "Guidelines for specifying a default delimiter" on page 3-50 for more information. You can also specify a delimiter in a field\_spec for an individual data field.

- Load a NULL value for any data field that is empty or contains blanks. An *empty data field* consists of two adjacent delimiters. To do this, include the NULLIF clause with the FIELDS clause. You can also include a NULLIF clause with a field\_spec for an individual data field.
- Load a column's default value for any data field that is empty or contains blanks. To do this, include a DEFAULTIF clause with the FIELDS clause. You can also include a DEFAULTIF clause with a field\_spec for an individual data field.

Note the following:

- You can include the CHAR and NULLFLAGS keywords in a FIELDS clause only when you omit a field\_spec for all data fields.
- If you omit a field\_spec for all data fields and you omit the FIELDS clause, the FileTek FTP Data Loader generates a field\_spec for every column in the named table using the corresponding the CREATE TABLE data type.
- When you use the NULLFLAGS keyword, the FileTek FTP Data Loader appends a NULLIF clause to each generated field\_spec when the NULL flag is T (for NULL). This loads a NULL into the corresponding table column.
- When you use FIELDS CHAR, FileTek recommends you include a delimiter\_spec.
- NULLIF and/or DEFAULTIF clauses in the FIELDS clause apply to any generated field\_specs or to any data fields that do not have NULLIF and/or DEFAULTIF clauses in their field\_specs.
- If you specify FIELDS CHAR and the input data file contains LOB data fields, all fields must still fit in a legal (32K-1-max) record size.
- If you omit the field\_specs and the FIELDS CHAR clause, the FileTek FTP Data Loader and the FileTek FTP Data Unloader place any LOB data fields at the end of the record in CREATE TABLE order. This means that the order of

the generated field list for a load may not match the order of expressions in the SELECT statement. The informational messages returned to you will indicate when this happens.

## **Guidelines for specifying a default delimiter**

You can specify a *default delimiter* for character data fields (CHARACTER, BLOB\_FILE, CLOB\_FILE, and all of the EXTERNAL data types) by using a FIELDS clause in a LOAD DATA statement. This default delimiter applies to all user-specified field\_specs or to all loader-generated field\_specs. You can override or supplement this default delimiter as needed for an individual data field in a datatype\_spec.

Note the following:

- A delimiter can be only one character.
- The maximum length of a data field does not include the delimiter(s).
- You cannot use delimited data fields in a WHEN clause.
- If a data field contains a character that is also used as an enclosure delimiter, you can double that character in the data to protect it. The FileTek FTP Data Loader converts the doubled character to a single character.

- If a data field contains a character that is also used as a termination delimiter, you cannot double that character in the data; but you can use an enclosure delimiter. For instance, if the data field is:

SAN ANTONIO, TX

and the termination delimiter is a comma:

SAN ANTONIO,TX,

then use an enclose delimiter (such as parentheses) to protect the data field:

(SAN ANTONIO,TX),

- If data fields are enclosed with a single delimiter (like a single quote) and are not separated by a terminator or blanks, adjacent delimiters would be interpreted as data. For example, the following data fields are enclosed by single quotes. The end delimiter of the first data field and the start delimiter of the second data field would be interpreted as data:

'2839"Jack'

- You can use the **OPTIONALLY** keyword only when using **TERMINATED** keyword. You can include both keywords in the **FIELDS** clause, or both in a **field\_spec**, or one in a **FIELDS** clause and the other in a **field\_spec** (for example, **OPTIONALLY ENCLOSED** in a **FIELDS** clause and **TERMINATED** in a **field\_spec**).

See “Delimited data” on page 2-7 for basic information about delimited data and types of delimiters. See “Specifying a delimiter for an individual data field” on page 3-106 for more information about overriding or supplementing the default delimiter.

## Format of FIELDS clause

FIELDS fields\_specs

where fields\_specs:

```
[ CHAR ]
[ NULLFLAGS ]
[ delimiter_spec ]
[ NULLIF ( EMPTY | BLANK ) ]
[ DEFAULTIF ( EMPTY | BLANK ) ]
```

and where delimiter\_spec:

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

Argument	Description
CHAR	(optional) Keyword for generating a field_spec as the CHARACTER loader data type for all columns in the named table. A delimiter_spec is highly suggested. An error occurs if you include FIELDS CHAR and a field_spec list in a LOAD DATA statement. The lengths are determined by the default length rules for a CHAR field_spec.
NULLFLAGS	(optional) Keyword for generating a field_spec for all columns as the corresponding CREATE TABLE data type (if CHAR is omitted) and for identifying NULL flags (T for NULL and F for NOT NULL) at the beginning of each input data record for each data field. An error occurs if you include FIELDS NULLFLAGS and a field_spec list in a LOAD DATA statement.
delimiter_spec	(optional but recommended) Keywords for describing delimited data.
TERMINATED	Keyword for describing terminated data fields.
BY	Keyword for readability only.



---

 Generating field\_specs, identifying NULL flags, specifying default delimiters and other defaults

Argument	Description
WHITESPACE	Keyword for indicating that the delimiter is one or more blank characters. You can use this keyword with the TERMINATED keyword, not the ENCLOSED keyword.
'char'	Value of a character delimiter, consisting of exactly one character and enclosed in single or double quotes. This value must match the case (UPPER or lower) of the input data.
X'hexbyte'	Value of a hexadecimal delimiter, consisting of exactly two hex digits.
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 field_spec, or one in a FIELDS clause and the other in a field_spec.
ENCLOSED	Keyword for describing enclosed data fields.
AND	Keyword for describing data fields enclosed with different starting and ending delimiters. If omitted, the enclosure delimiters are the same. The AND keyword must follow the ENCLOSED keyword.
NULLIF	(optional) Option to load a NULL value for any data field that is empty or contains blanks.
EMPTY	<p>Keyword for indicating an empty condition, that is, two adjacent delimiters. Using the EMPTY keyword is equivalent to specifying an empty pair of delimiters in the NULLIF clause.</p> <p>If dbref=Oracle, an empty condition is defined as follows:</p> <ul style="list-style-type: none"> <li>■ The data field consists of just two adjacent delimiters</li> <li>■ The data field is TERMINATED BY WHITESPACE and there are just blanks</li> <li>■ The data field is TERMINATED and OPTIONALLY ENCLOSED BY and there are just blanks and a terminator or end of record acting as a terminator</li> </ul>

## 3

**The control file**

---

Generating field\_specs, identifying NULL flags, specifying default delimiters and other defaults

---

Argument	Description
BLANK	Keyword for indicating a data field contains zero or more blanks.
DEFAULTIF	(optional) Option to load a column's default value for any data field that is empty or contains blanks. See the preceding EMPTY and BLANK keywords for more information.

---

## Example FIELDS clauses

This section contains example FIELDS clauses.

### To describe data fields terminated by a character

When data fields are terminated by a single character, use the TERMINATED keyword and specify the value of the delimiter in a FIELDS clause. You can specify a delimiter's value in character or hexadecimal format. Enclose the value in quotes. The end of a logical record always serves as a termination delimiter for the last data field if no delimiter is present.

For example, to specify the delimiter for these data fields:

2	8	3	9	!	M	c	G	u	i	r	e	!	J	a	c	k	!	
2	3	8	8	!	C	o	r	n	f	l	a	k	e	!	S	u	e	!

include this FIELDS clause:

FIELDS TERMINATED BY '!'

## To describe data fields terminated by a blank

When data fields are terminated by a blank, use the TERMINATED keyword with the WHITESPACE keyword in a FIELDS clause. For example, to specify the delimiter for these data fields:

2	8	3	9		M	c	G	u	i	r	e		J	a	c	k		
2	3	8	8		C	o	r	n	f	l	a	k	e		S	u	e	

include this FIELDS clause:

FIELDS TERMINATED BY WHITESPACE

**Note:** The next field's data will not include any of the whitespace from the previous data field unless the next data field has a fixed-start position.

## To describe data fields enclosed by the same delimiter

When data fields are enclosed by the same delimiter, use the ENCLOSED keyword and specify the value of the enclosure delimiter. For example, to specify that data fields are enclosed by double quotes and terminated with blanks:

"	2	8	3	9	"		"	M	c	G	u	i	r	e	"		"	J	a	c	k	"		
"	2	3	8	8	"		"	C	o	r	n	f	l	a	k	e	"		"	S	u	e	"	

include this FIELDS clause:

FIELDS TERMINATED BY WHITESPACE ENCLOSED BY '"'

### To describe data fields enclosed by different delimiters

When data fields are enclosed by different enclosure delimiters, use the ENCLOSED keyword and specify the delimiter characters with the AND keyword. For example, to specify that data fields are enclosed by parentheses:

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

include this FIELDS clause:

```
FIELDS ENCLOSED BY '(' AND ')'
```

### To describe data fields that are terminated and enclosed

When data fields are both terminated and enclosed, use the TERMINATED keyword followed by the ENCLOSED keyword in a FIELDS clause. You can also use the OPTIONALLY keyword when some data fields are enclosed. For example, to specify that all data fields are terminated by a comma and some are enclosed by parentheses:

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

include this FIELDS clause:

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '(' AND ')'
```

### To generate CHAR field\_specs

You can generate field\_specs as the CHARACTER loader data type by including the CHAR keyword with the FIELDS clause and by omitting field\_specs from the

LOAD DATA statement. For example, to generate a field\_spec for each of these data fields:

2	8	3	9		M	c	G	u	i	r	e		J	a	c	k		
2	3	8	8		C	o	r	n	f	l	a	k	e		S	u	e	

include this FIELDS clause:

FIELDS CHAR TERMINATED BY WHITESPACE

## To identify NULL flags in input data records

You can identify NULL flags in input data records by using the NULLFLAGS keyword in a FIELDS clause and by omitting field\_specs in a LOAD DATA statement. In the following example, the FileTek FTP Data Loader generates field\_specs using the CHARACTER data type.

For example, to describe these data fields:

F	F	F	2	8	3	9		M	c	G	u	i	r	e		J	a	c	k		
F	F	F	2	3	8	8		C	o	r	n	f	l	a	k	e		S	u	e	

include this FIELDS clause:

FIELDS CHAR NULLFLAGS TERMINATED BY WHITESPACE

## To load NULL values for empty data fields

You can load a NULL value for an empty data field (two adjacent delimiters) by including the NULLIF clause with the EMPTY keyword. For example, to load a

NULL value for the empty data field in the first record below and to generate field\_specs using the CHARACTER data type:

```
( 2 8 3 9 ) ( M c G u i r e ) ( J a c k ) ( )
( 2 3 8 8 ) ( C o r n f l a k e ) ( S u e ) ( L )
```

include this FIELDS clause:

FIELDS CHAR ENCLOSED BY '(' AND ')' NULLIF EMPTY

## Loading missing data fields with null values

A record that's missing a data field is called a *short record*. By using the TRAILING NULLCOLS clause, you can load a null value for a data field that's defined in a field\_spec but missing in a record. If you omit this clause, then the FileTek FTP Data Loader generates an error message when a record is missing a data field.

Note the following:

- The column in the user table must allow null values. The load will fail if you try to load a null value into a column defined as NOT NULL.
- If the field\_spec refers to a field (not a column), no error is generated and no null value is loaded. Fields are not loaded into user tables.
- If the db\_ref is Oracle and the record ends before a data field's fixed start position, then a null value is loaded.
- If a record ends before the start of a data field and you specify a TRAILING NULLCOLS clause, then a null value is loaded.
- If a record ends in the middle of a character-type data field and you specify the TRAILING NULLCOLS clause, then the record is padded.

If a data field is *not* missing but you want to replace the data value with a null value, then you can use the NULLIF clause in a position\_spec. See “Setting a column to a null value” on page 3-111 for more information about the NULLIF clause.

## Format of TRAILING NULLCOLS clause

TRAILING [ NULLCOLS ]

The NULLCOLS keyword is optional.

## Example TRAILING NULLCOLS clause

Assume that you are loading a user table with three columns. The `field_specs` are as follows:

```
(ORDER_NUM POSITION (1) INT EXTERNAL(4),
REP_LASTNAME POSITION (5) CHAR(15),
REP_FIRSTNAME POSITION (20) CHAR(15))
```

Here are some of the data records. Record 4 doesn't contain the representative's first name.

[illegible]

If you included the TRAILING NULLCOLS clause and if the user table's column allowed null values, then the REP\_FIRSTNAME column would be loaded with a null value. If you omitted the TRAILING NULLCOLS clause (and the database being emulated isn't Oracle), then the FileTek FTP Data Loader would generate

an error message because data's missing. The load would also fail if the REP\_FIRSTNAME column in the user table was defined as NOT NULL.

## Loading one or more segments

You can load data into multiple segments of a user table by including multiple INTO TABLE clauses and specifying the SAME SEGMENT and DIFFERENT SEGMENT clauses. You can also load multiple segments of different user tables during one load. See “Loads and segments” on page 1-11 for more information about segments.

SAME SEGMENT is the default, which means that the FileTek FTP Data Loader loads one segment. Simply omit the SAME SEGMENT and DIFFERENT SEGMENT clauses when you want to load data into one new segment.

Note the following:

- The value of the StorHouse SQL\_LDR\_MAXINTO system parameter must be greater than 1 to load multiple segments during a load.
- When there are multiple INTO TABLE clauses, then SAME SEGMENT loads data into the same segment as the most recent INTO TABLE clause for the same user table.
- You can name segments if you need or plan to replace them later. See page 3-63 for more information about using the SEGMENT clause to name a segment.
- You cannot load LOB records into multiple segments.



## **Format of SAME and DIFFERENT SEGMENT clauses**

SAME SEGMENT

DIFF[ERENT] SEGMENT

## **Example SAME and DIFFERENT SEGMENT clauses**

This section contains examples that show how to load data into multiple segments of the same user table as well as into different user tables.

### **To load multiple segments of the same user table**

The following example loads two segments in the ATM.TRANSACTIONS table. Records containing A in column 1 will be loaded into one segment and those containing B in column 1 will be loaded into a different segment.

```
LOAD  
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'A'
```

```
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'B'  
DIFF SEGMENT;
```

### **To load multiple segments of different user tables**

The following example loads one segment of one user table and multiple segments of another user table.

```
LOAD
INTO TABLE ATM.TRANSACTIONS

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE= '5/1/1999'

INTO TABLE POS.TRANSACTIONS
WHEN TRANS_DATE = '6/1/1999'
DIFF SEGMENT;
```

Note the following:

- All records will be loaded into one segment of the ATM.TRANSACTIONS user table. SAME SEGMENT (omitted from the into\_table\_spec) is the default.
- Records containing 5/1/1999 in the TRANS\_DATE column will be loaded into one segment of the POS.TRANSACTIONS table. SAME SEGMENT (omitted from the into\_table\_spec) is the default.
- Records containing 6/1/1999 in the TRANS\_DATE column will be loaded into a different segment of the POS.TRANSACTIONS table.

## Naming a segment

You can name the current segment you're loading in the event you need to replace it in the future. This name is called a *segment tag*. You specify a segment tag with the SEGMENT clause. Note the following:

- If you omit the SEGMENT clause from the LOAD DATA statement, the default segment tag is the load ID supplied for the loadident keyword on the FTP put command.

**Note:** StorHouse/RM does not normalize load IDs to uppercase. If you specify a load ID in lower or mixed case and subsequently supply that load ID as the segment tag on a REPLACE clause, you must use the same case and delimit (with double quotes) the load ID on the REPLACE clause.

- Segment tags need not be unique, but this could result in several segments being invalidated by a subsequent replace operation.
- If there are multiple INTO TABLE clauses for the same segment, you can specify only one SEGMENT clause for one of the INTO TABLE clauses. An error will occur if you specify different segment tags for the same segment.
- Segment tags are stored in the SYSSTHSEGMENTS system table.
- StorHouse/RM normalizes the segment tag specified on a SEGMENT clause to uppercase. This means you can use any case when providing a segment tag on a REPLACE clause. Segment tags are case sensitive only when delimited.

## Format of SEGMENT clause

SEGMENT segment\_tag

Argument	Description
segment_tag	(required) Name of the segment that you're loading into the user table specified on the INTO TABLE clause. This name cannot exceed 40 characters and must follow SQL identifier conventions (see page 3-3) or be quoted.

---

## Example SEGMENT clauses

This section contains example SEGMENT clauses.

### To use the load ID as the segment tag

Omit the SEGMENT clause to use the load ID provided on the loadident keyword on the FTP put command as the segment tag. For example, there's no SEGMENT clause for either INTO TABLE clause below, so the load ID is the segment tag for both segments.

```
LOAD  
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'A'
```

```
INTO TABLE ATM.TRANSACTIONS  
WHEN (1) = 'B'  
DIFF SEGMENT;
```

## To assign different segment tags for multiple segments of the same user table

In the following example, the segment tag in the first INTO TABLE clause is MAYSEGMENT. The segment tag in the second INTO TABLE clause is JUNESEGMENT.

```
LOAD  
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE= '5/1/1999'  
SEGMENT MAYSEGMENT
```

```
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE = '6/1/1999'  
DIFF SEGMENT  
SEGMENT JUNESEGMENT;
```

## Replacing a segment

You can invalidate an existing segment by using the REPLACE SEGMENT clause. When you replace a segment, a StorHouse engine updates the SYSSTHSEGMENTS system table, setting the INVALID\_FLAG column to Y and the INVALID\_TIME column to the current time for all segments with the given segment tag, owner, and table name.

Note the following:

- You can obtain the segment tag from the SYSSTHSEGMENTS system table.
- If you named a segment with the SEGMENT clause, you can use any case when providing the segment tag on the REPLACE clause. StorHouse/RM normalizes these segment tags to uppercase. For example:

```
SEGMENTcallsload  
REPLACE SEGMENT CALLSLOAD or REPLACE SEGMENT Callsload
```

- If you delimited the segment tag on the SEGMENT clause, you must delimit it and use the same case on the REPLACE clause. For example:

```
SEGMENT "1seg"  
REPLACE SEGMENT "1seg"
```

- If you omitted a SEGMENT clause and specified a lower or mixed case load ID on the FTP put command or a load ID that doesn't conform to SQL identifier conventions, then you must use the same case and delimit the load ID on the REPLACE clause. Examples:

```
ftp> put control.inp load,dbn=database1,loadid=callsload  
REPLACE SEGMENT "callsload"
```

or

```
ftp> put control.inp load,dbn=database1,loadid=1load  
REPLACE SEGMENT "1load"
```

- If you omit the owner and table name, the default is the owner and table name specified on the INTO TABLE clause.
- If you include both the SEGMENT and REPLACE SEGMENT clauses and specify the same segment tag for both, the current (new) segment will not be invalidated.
- If there are multiple INTO TABLE clauses for the same segment, you can specify only one REPLACE SEGMENT clause for one of the INTO TABLE clauses. An error will occur if you specify different segment tags for the same segment.
- No error will occur if there are no segments with the specified segment tag.
- The FileTek FTP Data Loader will generate a reply indicating the number of segments that were replaced.

## Format of REPLACE SEGMENT clause

REPLACE SEGMENT [ [owner.] table\_name.] segment\_tag

Argument	Description
owner.	(optional) StorHouse account ID of the owner of the user table containing the segment you are replacing. If you omit the owner name, the FileTek FTP Data Loader uses the owner name on the corresponding INTO TABLE clause.
table_name.	(optional) Name of the user table containing the segment you are replacing. If you omit the table name, the FileTek FTP Data Loader uses the table name on the corresponding INTO TABLE clause.
segment_tag	(required) Name assigned to the segment (on the SEGMENT clause) when you loaded the user table. If you omitted the SEGMENT clause, the load ID is the segment tag.

## Example REPLACE SEGMENT clause

The following example loads data into a new segment called JULYSEGMENT and invalidates the segment called JUNESEGMENT. The FileTek FTP Data Loader will replace all segments named JUNESEGMENT in the POS.TRANSACTIONS table.

```
LOAD  
INTO TABLE POS.TRANSACTIONS  
WHEN TRANS_DATE= '7/1/1999'  
SEGMENT JULYSEGMENT  
REPLACE JUNESEGMENT;
```

See page 5-6 for an example that simply replaces a segment (doesn't load data).

## Selecting subspaces

You can select specific subspaces for user table components (table data, value indexes, hash indexes, and LOB data) by including one or more SUBSPACE number clauses after an INTO TABLE clause of a LOAD DATA or MERGE statement or on a LOAD INDEX statement. Note the following:

- You cannot use both a SUBSPACE ROTATE and a SUBSPACE number clause in a LOAD DATA, LOAD INDEX, or MERGE statement. Only one of these clauses is allowed.
- If you omit both the SUBSPACE ROTATE and the SUBSPACE number clause, the FileTek FTP Data Loader uses the lowest-numbered subspace that allows the component type.
- You can include a SUBSPACE number clause for any of the component types. For instance, for a data load, you can select a subspace for just table data and use the default selection (lowest-numbered subspace) for indexes and LOB data. For an index load, you can select different subspaces for value and hash indexes. And for a merge operation, you can select different subspaces for table data, value indexes, and hash indexes. LOB data is not reprocessed in a merge operation, so it remains in its current location.
- If you omit the component type (for example, TABLE or VALUE), the clause applies to all component types (unless one or more is overridden by a later SUBSPACE number clause). For example, if you specify SUBSPACE 2, then the data loader uses subspace 2 for table data, hash indexes, value indexes, and LOB data.
- If you specify multiple SUBSPACE number clauses in the same INTO TABLE clause, later ones that specify the same component type(s) supersede earlier ones.
- Subspace(s) must exist for the applicable component type. For example, you can select subspace 3 for table data if the user tablespace contains a subspace 3 defined with OBJECT\_TYPE T (for table data) or blank (for all component



types). An error occurs if you select a subspace that does not exist or is not valid for a component type.

- The FileTek FTP Data Loader uses the table data subspace and the user table tablespace for LOB values that fit within a row, that is, for in-line LOBs.

## Format of SUBSPACE number clause

[ [ TABLE | VALUE | HASH | LOB ] SUBSPACE number ]...

Argument	Description
TABLE	(optional) Keyword to select a subspace for table data.
VALUE	(optional) Keyword to select a subspace for value indexes.
HASH	(optional) Keyword to select a subspace for hash indexes.
LOB	(optional) Keyword to select a subspace for LOB data. StorHouse/RM uses LOB subspaces only for out-of-line LOBs.
number	(required) Subspace number. If you omit TABLE, VALUE, HASH or LOB, all component types use the same subspace number.

See page 3-141 for the SUBSPACE number clause of the LOAD INDEX statement and page 3-144 for the SUBSPACE number clause of the MERGE statement.

## Example SUBSPACE number clauses

This section contains example SUBSPACE number clauses.

## To select subspaces when loading one segment

You can select a specific subspace for each component type. For example, assume you're loading one segment. The user table has three hash indexes, two value indexes, and two LOB columns assigned to the same user tablespace as the table. The user tablespace contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)
3	H (hash indexes only)
4	H (hash indexes only)
5	V (value indexes only)
6	V (value indexes only)
7	L (LOB data only)
8	L (LOB data only)

With the following LOAD DATA statement, the FileTek FTP Data Loader uses the following subspaces:

- Subspace 2 for the table data file
- Subspace 4 for all three hash index files
- Subspace 6 for both value index files
- Subspace 8 for both LOB subsegment files

```
LOAD
INTO TABLE TOLLFREE
TABLE SUBSPACE 2
HASH SUBSPACE 4
VALUE SUBSPACE 6
LOB SUBSPACE 8
(FROMNUM POSITION(1) BINARY EXTERNAL(10),
TONUM BINARY EXTERNAL(10),
ACCOUNT CHAR(12),
PHOTO_ID BLOB_FILE PATH '/home/tka/sth3/' USER 'tka'/tka,
STATEMENT BLOB_FILE PATH '/home/tka/sth3/' USER 'tka'/tka)
WHEN (11:13) = '800' OR (11:13) = '888' ;
```

## To select subspaces when loading multiple segments

When loading multiple segments, you can use the same subspace or different subspaces for each component type. For example, assume you're loading two segments. The user table has one hash index and one value index assigned to the same user tablespace as the table.

The user tablespace contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)
3	T (table data only)
4	H (hash indexes only)
5	H (hash indexes only)
6	H (hash indexes only)
7	V (value indexes only)
8	V (value indexes only)
9	V (value indexes only)

Assume you want to use a different subspace for each table data file but the same subspace for both hash index files and the same subspace for both value index files. With the following LOAD DATA statement, the FileTek FTP Data Loader uses the following subspaces:

- Subspace 1 for the table data file in segment 1
- Subspace 2 for the table data file in segment 2
- Subspace 5 for the hash index files in segments 1 and 2
- Subspace 9 for the value index files in segments 1 and 2

```
LOAD
  INTO TABLE MARCHCALLS
  TABLE SUBSPACE 1
  HASH SUBSPACE 5
  VALUE SUBSPACE 9
  (FROMNUM POSITION(1) BINARY EXTERNAL(10),
  TONUM BINARY EXTERNAL(10),
  ACCOUNT CHAR(12))
  WHEN (11:13) = '800' OR (11:13) = '888'
```

```
  INTO TABLE MARCHCALLS
  DIFFERENT SEGMENT
  TABLE SUBSPACE 2
  HASH SUBSPACE 5
  VALUE SUBSPACE 9
  (FROMNUM POSITION(1) BINARY EXTERNAL(10),
  TONUM BINARY EXTERNAL(10),
  ACCOUNT CHAR(12))
  WHEN (11:13) != '800' AND (11:13) != '888';
```

### **To select subspaces in multiple user tablespaces**

If any indexes or LOB columns for a user table are assigned to different user tablespaces, you can select different subspaces in different user tablespaces. If the subspace number is the same in all user tablespaces, then you can omit the TABLE | HASH | VALUE | LOB keywords and just specify the SUBSPACE keyword with the subspace number.

For example, assume you're loading one segment. The user table, assigned to user tablespace 1, has two hash indexes, two value indexes, and two LOB columns. The hash indexes are assigned to user tablespace 2, the value indexes are assigned to user tablespace 3, and the LOB columns are assigned to user tablespace 4.

User tablespace 1 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	T (table data only)
2	T (table data only)

User tablespace 2 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	H (hash indexes only)
2	H (hash indexes only)

User tablespace 3 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	V (value indexes only)
2	V (value indexes only)

User tablespace 4 contains the following subspaces:

Subspace number	OBJECT_TYPE
1	L (LOB data only)
2	L (LOB data only)

With the following LOAD DATA statement, the FileTek FTP Data Loader uses the following subspaces:

- Subspace 2 in user tablespace 1 for the table data file
- Subspace 2 in user tablespace 2 for both hash index files
- Subspace 2 in user tablespace 3 for both value index files
- Subspace 2 in user tablespace 4 for both LOB subsegment files

```
LOAD
INTO TABLE CALLSDBA.BILLSUMMARY
SUBSPACE 2
(BILL_ACCOUNT CHAR(10),
BILL_DATE DATE EXTERNAL(10),
BILL_CATEGORY CHAR(1),
NUMBER_CALLS INT EXTERNAL(3),
PHOTO_ID BLOB_FILE PATH '/home/tka/sth3/' USER 'tka'/tka,
BILL_IMAGE BLOB_FILE PATH '/home/tka/sth3/' USER 'tka'/tka);
```

## Describing data fields

A *field\_spec* describes a data field in a logical record. Include a field\_spec to:

- Describe a field in a WHEN clause.

You must provide a field\_spec for any field name specified in a WHEN clause. Remember that fields aren't loaded into user tables. You use fields only to assign a name to a portion of a logical record to be used in a condition.

- Generate a value to be loaded into a column. You can generate:
  - the record number of a logical record (RECNUM)
  - a sequence of values (SEQUENCE)
  - the current date (SYSDATE)
  - a constant value (CONSTANT)

- Describe a data field to be loaded into a column. You can describe:
  - the position of the data field in the logical record (position\_spec)
  - the name and length of the data type (datatype\_spec)
  - the character set for an individual data field (CHARSET)
  - a delimiter for a CHAR or EXTERNAL data field (delimiter\_spec)
  - a condition that causes a column to be loaded with a null value (NULLIF) or a default value (DEFAULTIF)
- Load user table columns with input values in LOB files.

You can omit all or some field\_specs when describing data fields to be loaded into columns.

- If you include a FIELDS CHAR clause, you must omit all field\_specs. The FileTek FTP Data Loader generates a field\_spec for every column with the CHARACTER loader data type and determines the length of the data fields using the default-length rules for CHARACTER.
- You can omit a FIELDS CHAR clause and all field\_specs if the input data is in the same order as the CREATE TABLE definition and all data fields are relatively positioned and of the same data type. The FileTek FTP Data Loader uses the CREATE TABLE definition (data types and lengths) to determine how to interpret the input data.
- If you omit a FIELDS CHAR clause and *some* field\_specs, the FileTek FTP Data Loader loads the omitted columns with the default value. If no default value was assigned when the user table was created, a null value is loaded.

## Format of field\_spec list

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

where field\_spec:

{ :field\_name | column\_name } data\_spec

Argument	Description
:field_name	(required if using a field name in a condition) Name of the field you are using in a condition. This name must adhere to the conventions of StorHouse SQL identifiers (see page 3-3) or be quoted. Precede this field name with a colon (:).
column_name	(required to generate data or to describe a column to be loaded) Name of the column in the input data. This name must match the column name specified in the CREATE TABLE statement.
data_spec	(optional) Value to be generated or description of the data field. The format is: RECNUM   SEQUENCE ( start_num [ ,increment ] )   SYSDATE   CONSTANT any_value   position_spec
position_spec	(optional) Position of the data field in a logical record, name and length of the data type, condition that loads a null value or a default value into a column. You can specify position_spec clauses in any order. The format is: [ POSITION ( position   * [ +num ] ) ] [ datatype_spec ] [ NULLIF field_condition ] [ DEFAULTIF field_condition ]
datatype_spec	(optional) Name of the data type and length of the column or field in the logical record. Depending on the data type, you can also specify a character set or a delimiter for an individual data field. For LOB files, a datatype_spec may specify the host, path, and user information necessary to access the LOB files.



## Providing a field name

If you used a field name in a condition (like in a WHEN clause), then you must provide a field\_spec for that field. For instance, for the following WHEN clause:

```
WHEN RECORD_CODE = 'A'
```

you would provide this field\_spec:

```
:RECORD_CODE POSITION (1) CHAR
```

Remember that:

- A field name in a WHEN clause does *not* start with a colon.
- A field name in a field\_spec *must* start with a colon.
- Fields aren't loaded, only columns are loaded into user tables.
- If a field name is delimited, the preceding colon (in the field\_spec) must be outside the quotes, for example, : "RECORD CODE" POSITION (1) CHAR

## Providing a column name

When generating data or describing data fields to be loaded, you must provide a column name in a field\_spec. Note the following:

- Column names must match the names assigned on the CREATE TABLE statement. For instance, if a column name in the CREATE TABLE statement is ORDER\_NUM, then specify ORDER\_NUM (not ORDER\_NUMBER or ORDERNUMBER) in the field\_spec.
- Column names are not case sensitive unless quoted.

## Loading a record number into a column

Use the RECNUM keyword after a column name to load a column with the record number of the logical record from which that row was loaded. Record numbers start at 1 and increment for each logical record, including discarded records. You can load this record number into a table column of data type SMALLINT or INTEGER.

For example, to generate record numbers for a column called RECORD\_NUMBER, specify:

```
RECORD_NUMBER RECNUM
```

## Generating a sequence of values

Use the SEQUENCE clause after a column name to generate a sequence of unique values for each logical record that is not discarded. You can load these values into a table column of data type SMALLINT or INTEGER.

The format of SEQUENCE is:

```
SEQUENCE ( start_num [ ,increment ] )
```

Argument	Description
start_num	(required) Starting value, which must be a positive integer or 0.
increment	(optional) Number to increment subsequent logical records that are not discarded. If you omit the increment, the default is 1.

For example, to generate unique values for a column named CUSTOMER\_NUM, starting with 1 for the first logical record not discarded and incrementing by 1 for each subsequent logical record not discarded, type:

```
CUSTOMER_NUM SEQUENCE (1,1)
```

## Loading the current date into a column

Use the SYSDATE keyword after a column name to load the current date into a column, specifically, the date that the load or restart operation *began* (even if the load continues past midnight). The table column must be of data type CHAR, CLOB, DATE, or VARCHAR.

For example, to load the current date into a column called ORDER\_DATE, specify:

```
ORDER_DATE SYSDATE
```

## Loading a constant value into a column

Use the CONSTANT clause after a column name to load a constant value into a column. Note the following:

- The constant value can be a quoted character string or hexadecimal string, an unquoted identifier, or a number.
- The table column can be any data type.
- The FileTek FTP Data Loader treats the constant value as a string and converts it to the data type of the table column.
- If you specify a character string and the data field has a fixed position (specified by a POSITION clause), the FileTek FTP Data Loader pads or truncates that string if it is shorter or longer than the fixed position.

The format of CONSTANT is:

CONSTANT any\_value

Argument	Description
any_value	Value to load into the column. The format is: any_string   identifier   num
any_string	Character string enclosed in single or double quotes or hexadecimal string preceded with X and enclosed in quotes
identifier	Unquoted string
n	Unsigned integer optionally followed by K (x1024) or M (xKK)

For example, to load the value CA into a column called STATE, specify:

STATE CONSTANT 'CA'

## Specifying the position of a data field

To load a column or use a field, the FileTek FTP Data Loader must know where to locate it in a logical record. You specify the location of a data field by including a POSITION clause in a field\_spec. 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

The POSITION clause is optional. If you omit it, the default is POSITION (\*) or relative position as a continuation from the previous data field. Typically, you omit the POSITION clause when the input data contains VAR-type or delimited data fields. You must use relative positioning (the default POSITION (\*)) for BLOB and CLOB data fields.

For VARCHAR or VARBINARY data fields, the position must include the 2-byte field containing the actual length of the data field. Any relatively positioned data

fields (using \* or +num) *after* a VAR-type data field will start at varying positions based on the actual length of the data field in each logical record.

The format of POSITION is:

POSITION ( position | [ +num ] )

Argument	Description
position	Starting (required) and ending (optional) column numbers of the data field in the logical record. The format is: start_column [ { :   - } end_column ]
start_column	Starting position of the data field in the logical record. The first position in a logical record is 1.
: or -	Character that separates the starting column from the ending column. Either character is valid.
end_column	Ending position of the data field in the logical record. If you omit the ending column, the length comes from the datatype_spec of the data field. If you include the ending position, the length depends on the db_ref value.
*	The data field immediately follows the previous data field. When you include *, the length comes from the datatype_spec of the data field.
+num	An offset from the previous data field, where num is the number of positions. Use +num with *.

For example:

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

POSITION (1)

In this example, the data field starts in position 1 of the logical record. The length comes from the datatype\_spec of the data field. For instance, if the data type is SMALLINT, which has a length of 2 bytes, then the data field starts in position 1 and ends in position 2.

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

POSITION (1:3)

In this example, the data field starts in position 1 and ends in position 3 of the logical record. Another way to specify this is POSITION (1-3). If you also specify the length of the data type, the length is either ignored (if db\_ref is DB2) or supersedes the POSITION length.

- 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. So if the previous data field ends in position 15, then this data field starts in position 16 of the logical record. The length of the data field comes from the datatype\_spec. POSITION (\*) is the default.

**Note:** When a LOAD DATA statement contains multiple INTO TABLE clauses, the position used when the first field\_spec of an INTO TABLE clause is relative is:

- column 1 of the logical record if no prior INTO TABLE clauses have been selected for loading (as determined by their WHEN clauses),
- or the column following the last data field in the last INTO TABLE clause that was selected for loading.

See page 3-125 for an example of relative positioning with multiple INTO TABLE clauses.

- 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 logical record. POSITION (\*+0) is the same as POSITION (\*). The length of the data field comes from the datatype\_spec.

## Specifying the data type

A datatype\_spec provides the data type name and length of a data field in a logical record. The FileTek FTP Data Loader uses this information to interpret a data field. For some data types, you can also specify a character set and a delimiter. For LOB data in separate files, a datatype\_spec provides the host, path, and user information necessary to access LOB data files. Note that:

- The data type name is always required in a datatype\_spec.
- The length, character set, and delimiters are optional in a datatype\_spec
- For LOB data in separate LOB files:
  - The host name is optional if the LOB data files are on your client computer.
  - The path name is optional if the data file contains the path name.
  - The user information is optional if the StorHouse account ID you use to log into the StorHouse FTP server is the same user information required to access the LOB data files on your client or a remote computer.

See page 3-103 for more information about how the FileTek FTP Data Loader calculates the length of a data field if you omit the length in a datatype\_spec.

The data type of an input data field does not have to match the data type of the column in the user table, but those data types must be compatible. The FileTek FTP Data Loader automatically performs any necessary conversions. See page 3-102 for a summary of compatible data types.

In a datatype\_spec, the data type must be one of the data types supported by the FileTek FTP Data Loader. Those data types are described in the following tables. Definitions of the data type specifications are as follows:

**Definitions of data type specifications**

LOAD syntax	Format of the data type in a LOAD datatype_spec
Input field size	Length, default length, length ranges of the input data
Input field format	Acceptable format and/or contents of the input data
Range	Minimum and maximum data values allowed for the data type
Storage size	Amount of space when stored on StorHouse. Note that null data takes no space (in the corresponding CREATE TABLE data type).
Target types	Valid CREATE TABLE data types for this input type, in other words, the supported conversions. See page 3-102 for a summary of conversions.

---



**BINARY data type**

LOAD syntax	BINARY [(length)] RAW [(length)] BYTE [(length)] CHAR[ACTER] [(length)] CHARSET 65535
Input field size	Default length: CREATE TABLE length if BINARY or CHAR, else 1 Length range: 1 to 32765
Input field format	Any bytes
Storage size	CREATE TABLE length (1 to 256)
Target types	BINARY, BLOB, CHAR, CLOB, VARBINARY, VARCHAR
Notes	<ul style="list-style-type: none"> <li>■ If the LOAD length is greater than the CREATE TABLE length, the excess characters are silently truncated, that is, the FileTek FTP Data Loader does not report or log whether any data is trimmed.</li> <li>■ The input data is copied directly. There is no conversion, even when the target type is CHAR, CLOB, or VARCHAR.</li> </ul>

**BINARY EXTERNAL data type**

LOAD syntax	BINARY EXTERNAL [(length)] [CHARSET ccsid ] [delimiter_spec] RAW EXTERNAL [(length)] [CHARSET ccsid ] [delimiter_spec] BYTE EXTERNAL [(length)] [CHARSET ccsid ] [delimiter_spec]
Input field size	Default length: 512 (with delimiter_spec) or else CREATE TABLE length2 if BINARY or CHAR, else 2 Length range: 1 to 32765
Input field format	Hexits (0-9, a-f, A-F), where 2 hexits=1 byte
Target types	All except TIME, DATE, TIMESTAMP, and DECIMAL
Notes	<ul style="list-style-type: none"> <li>■ If an odd number of hexits is supplied, a leading 0 is added.</li> <li>■ If the LOAD length/2 is greater than the CREATE TABLE length, the excess characters are silently truncated.</li> <li>■ BINARY EXTERNAL does not consider nvk. For example, if you loaded a BINARY EXTERNAL(4) value into an INTEGER column, the FileTek FTP Data Loader would interpret it as a big-endian value and would pad it on the left with zeroes.</li> </ul>

**BLOB data type**

LOAD syntax	BLOB [(max_length [K M G])] CLOB [(max_length [K M G])] CHARSET 65535
Input field size	Contents of 64-bit length field Length range: 0+8 (minimum) to 2G-9+8 (maximum) Default max_length = CREATE TABLE max_length + 8
Input field format	64-bit length field followed by the data
Storage size	In-line LOB: Size of BLOB plus 4 bytes Out-of-line LOB: Size of BLOB in the LOB subsegment file plus 22 bytes for the object identifier (OID) in the table data file
Target type	BLOB, CLOB
Notes	<ul style="list-style-type: none"> <li>■ The length of the data is read from a 64-bit field preceding the data. The native values key affects the interpretation of these bytes.</li> <li>■ If a BLOB data value exceeds the LOAD DATA max_length, the load fails.</li> <li>■ If a BLOB data value exceeds the CREATE TABLE length, the data is silently truncated.</li> <li>■ When the target type is CLOB, there are no conversions, that is, the input characters are copied directly rather than converted to hexits or from any data character set.</li> <li>■ See “Specifying a BLOB or CLOB data type” on page 3-109 for additional considerations.</li> </ul>

---

**BLOB\_FILE data type**

LOAD syntax	BLOB_FILE [(length)] [delimiter_spec] [HOST hostname] [PATH path_spec] [USER username/password]
Input field size	Default length: 1024 Length range: 1 to 32765
Input field format	Any characters that form a valid qualified file name
Storage size	In-line LOB: Size of BLOB plus 4 bytes Out-of-line LOB: Size of BLOB in the LOB subsegment file plus 22 bytes for the object identifier (OID) in the table data file
Target type	BLOB
Notes	<ul style="list-style-type: none"><li>■ The length is the maximum length of the path and file name(s), not the length of the BLOB data in the LOB data file.</li><li>■ If a BLOB data value exceeds the CREATE TABLE length, the data is silently truncated.</li><li>■ See page 3-107 for guidelines on specifying the optional HOST clause, page 3-107 for the PATH clause, and page 3-108 for the USER clause.</li></ul>

## 3

## The control file

Describing data fields

**CHARACTER data type**

LOAD syntax	CHAR[ACTER] [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: MIN (CREATE TABLE length, 32705) if BINARY, BLOB, CHAR, CLOB, VARBINARY, or VARCHAR; or else 256 (with a delimiter_spec) or else 1 Length range: 1 to 32765
Input field format	Any characters
Storage size	CREATE TABLE length (1 to 256)
Target types	All
Notes	<ul style="list-style-type: none"> <li>■ If the ccsid is not 819, then the data will be converted.</li> <li>■ If ccsid=65535, the data type is changed to BINARY and any delimiter_spec is ignored.</li> <li>■ The CHAR length is not enforced on the LOAD DATA statement. This is helpful if you need to load delimited data with a size greater than 256 into a VARCHAR table column. In this case, you must use a CHAR data type to define that delimited data.</li> <li>■ If the LOAD length is greater than the CREATE TABLE length, the excess characters are silently truncated.</li> <li>■ If the target type is [VAR]BINARY, there is no conversion, that is, the input characters are copied directly.</li> <li>■ When the target is a numeric type, a data field of only blanks is an error.</li> </ul>

**CLOB data type**

LOAD syntax	CLOB [(max_length [K M G])] [CHARSET ccsid]
Input field size	Contents of 64-bit length field Length range: 0+8 (minimum) to 2G-9+8 (maximum) Default max_length = CREATE TABLE max_length + 8
Input field format	64-bit length field followed by the data
Storage size	In-line LOB: Size of CLOB plus 4 bytes Out-of-line LOB: Size of CLOB in the LOB subsegment file plus 22 bytes for the object identifier (OID) in the table data file
Target type	CLOB, BLOB
Notes	<ul style="list-style-type: none"><li>■ If a CLOB data value exceeds the LOAD DATA max_length, the load fails.</li><li>■ If a CLOB data value exceeds the CREATE TABLE length, the value is silently truncated.</li><li>■ If the ccsid is 65535, the data type is changed to BLOB.</li><li>■ When the target type is BLOB, there is no conversion, that is, the input characters are copied directly rather than interpreted as hexits.</li><li>■ See "Specifying a BLOB or CLOB data type" on page 3-109 for additional considerations.</li></ul>

## 3

## The control file

Describing data fields

**CLOB\_FILE data type**

LOAD syntax	CLOB_FILE [(length)] [CHARSET ccsid] [delimiter_spec] [HOST hostname] [PATH path_spec] [USER username/password]
Input field size	Default length: 1024 Length range: 1 to 32765
Input field format	Any characters that form a valid qualified file name
Storage size	In-line LOB: Size of CLOB plus 4 bytes Out-of-line LOB: Size of CLOB in the LOB subsegment file plus 22 bytes for the OID in the table data file
Target type	CLOB
Notes	<ul style="list-style-type: none"> <li>■ The CHARSET is the character set of the CLOB data, not the file name(s).</li> <li>■ The length is the maximum length of the path and file name(s), not the length of the CLOB data in the LOB data file.</li> <li>■ If a CLOB data value exceeds the CREATE TABLE length, the data is silently truncated.</li> <li>■ See page 3-107 for guidelines on specifying the optional HOST clause, page 3-107 for the PATH clause, and page 3-108 for the USER clause.</li> </ul>

**DATE or DATE EXTERNAL data type**

LOAD syntax	DATE [EXTERNAL] [(length)] ["mask"] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 10 (without delimiter_spec) or 256 (with delimiter_spec)  Length range: 1 to 32765
Input field format	Character representation of a date. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for valid input formats for dates.
Storage size	4
Range	1/1/0001 to 12/31/9999
Target types	DATE
Note	The mask (its length or content) is currently not used.

## 3

## The control file

Describing data fields

**DECIMAL or NUMERIC data type**

LOAD syntax	DEC[IMAL] [PACKED] NUMERIC [PACKED] DEC[IMAL] (precision[, scale]) NUMERIC (precision[, scale])
Input field size	(precision + 2) / 2 Default precision and scale: CREATE TABLE precision and scale Default scale (if precision supplied): 0 Precision range: 1 to 31 Scale range: 0 to precision
Input field format	Number in the form <i>ddd...ds</i> , where <i>d</i> is a decimal digit represented by 4 bits and <i>s</i> is a 4-byte sign value (where a plus sign (+) is represented by A, C, E, or F and a minus sign (-) is represented by B or D)
Storage size	(precision + 2) / 2
Target types	DECIMAL, INTEGER, SMALLINT
Note	If you omit the precision but include a starting and ending column in the POSITION clause for the data field, the FileTek FTP Data Loader will calculate the precision from the POSITION clause.



**DECIMAL EXTERNAL data type**

LOAD syntax	DEC[IMAL] EXTERNAL (length[, scale]) [CHARSET ccoid] [delimiter_spec] NUMERIC EXTERNAL (length[, scale]) [CHARSET ccoid] [delimiter_spec]
Input field size	Default length: 256 (with delimiter_spec) or else CREATE TABLE precision + 3 if DECIMAL, else 11 Default scale: 0 Length range: 1 to 32765
Input field format	String of characters that represents a number, with or without a sign or a decimal point. <i>E-notation</i> (floating-point literal) is also accepted. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for E-notation format.
Target types	DECIMAL, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none"><li>■ If no decimal point exists in the input and E-notation is not used, there is an implied decimal point at the position indicated by the scale (after any trailing blanks are trimmed).</li><li>■ A data field of only blanks is an error.</li></ul>

## 3

## The control file

Describing data fields

---

**DOUBLE data type**

LOAD syntax	FLOAT FLOAT (bits) DOUBLE [ PRECISION ]
Input field size	8
Input field format	Interpretation of the input value depends on the nvk value supplied on the FTP put command. See "Native data types" on page 2-4 for more information.
Storage size	8
Range	IEEE limits
Target types	REAL, DOUBLE, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none"><li>■ If the db_ref value on the FTP put command is DB2 or ANSI, FLOAT is interpreted as the DOUBLE data type.</li><li>■ If the db_ref value is Oracle, FLOAT is interpreted as the FLOAT (REAL) data type (see page 3-95).</li><li>■ bits = 22 to 53</li></ul>

---

**FLOAT (REAL) data type**

LOAD syntax	FLOAT FLOAT (bits) REAL
Input field size	4
Input field format	Interpretation of the input value depends on the nvk value supplied on the FTP put command. See “Native data types” on page 2-4 for more information.
Storage size	4
Range	IEEE limits
Target types	REAL, DOUBLE, INTEGER, SMALLINT
Notes	<ul style="list-style-type: none"> <li>■ If the db_ref value on the FTP put command is DB2 or ANSI, FLOAT is interpreted as the DOUBLE data type.</li> <li>■ If the db_ref value is Oracle, FLOAT is interpreted as the FLOAT (REAL) data type.</li> <li>■ bits = 1 to 21</li> </ul>

**FLOAT EXTERNAL data type**

LOAD syntax	FLOAT EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 24 (without delimiter_spec) or 256 (with delimiter_spec)  Length range: 1 to 32765
Input field format	Integer value, decimal value, or character representation of a floating-point number (E-notation). Refer to the <i>StorHouse SQL Reference Manual</i> for E-notation format. Leading and trailing blanks are allowed.
Target types	REAL, DOUBLE, DECIMAL, INTEGER, SMALLINT
Note	A data field of only blanks is an error.

## 3

## The control file

Describing data fields

**INTEGER data type**

LOAD syntax	INT[EGER]
Input field size	2 if nvk=DOS, else 4
Input field format	Signed binary integer
Storage size	4
Range	-2147483648 to 2147483647
Target types	INTEGER, SMALLINT
Note	Interpretation of the byte order depends on the nvk value supplied on the FTP put command. See "Native data types" on page 2-4 for more information.

**INTEGER EXTERNAL data type**

LOAD syntax	INT[EGER] EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 11 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32765
Input field format	Array of characters that represents a number, with or without a sign, without a decimal point. Leading and trailing blanks are allowed.
Target types	INTEGER, SMALLINT
Note	A data field of only blanks is an error.

**SMALLINT data type**

LOAD syntax	SMALLINT
Input field size	2
Input field format	Signed binary integer
Storage size	2
Range	-32768 to 32767
Target types	INTEGER, SMALLINT
Note	Interpretation of the byte order depends on the nvk value supplied on the FTP put command.

**TIME EXTERNAL data type**

LOAD syntax	TIME EXTERNAL [(length)] [CHARSET ccsid] [delimiter_spec]
Input field size	Default length: 12 (without delimiter_spec) or 256 (with delimiter_spec) Length range: 1 to 32765
Input field format	Character representation of a time. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for valid input formats for time.
Storage size	4
Range	0:0:0.000 through 24:0:0.000 (leap seconds (up to 62 seconds) are also allowed)
Target types	TIME
Note	The FileTek FTP Data Loader accepts a time data field without milliseconds.

## 3

## The control file

Describing data fields

**TIMESTAMP EXTERNAL data type**

LOAD syntax	TIMESTAMP EXTERNAL [(length)] [CHARSET ccSID] [delimiter_spec]
Input field size	Default length: 26 (without delimiter_spec) or 256 (with delimiter_spec)  Length range: 1 to 32765
Input field format	Character representation of a timestamp. Leading and trailing blanks are allowed. Refer to the <i>StorHouse SQL Reference Manual</i> for valid input formats for timestamp.
Storage size	8
Range	See TIME and DATE, except TIMESTAMP contains microseconds
Target types	TIMESTAMP

**VARBINARY data type**

LOAD syntax	VARBINARY [(max_length)] VARRAW [(max_length)] VARBYTE [(max_length)] VARCHAR [(max_length)] CHARSET 65535
Input field size	Contents of SMALLINT length field + 2 Default max_length: CREATE TABLE max_length if VARBINARY or VARCHAR, else 256 Length range: 1 to 32765
Input field format	SMALLINT field followed by any bytes
Storage size	Length of data + 2 (unless compressed)
Target types	BINARY, BLOB, CHAR, CLOB, VARBINARY, VARCHAR
Notes	<ul style="list-style-type: none"><li>■ The length of the data is read from the SMALLINT field preceding the data.</li><li>■ Interpretation of the SMALLINT length field depends on the nvk value supplied on the FTP put command.</li><li>■ If the data exceeds the LOAD max_length or runs off the end of the record, an error is generated and the load is terminated.</li><li>■ If the data exceeds the CREATE TABLE length, the data is silently truncated.</li><li>■ If the CREATE TABLE length is greater than 4096, the data is compressed before being written.</li><li>■ If the target type is [VAR]CHAR, there is no conversion, that is, the input characters are copied directly.</li></ul>

## 3

## The control file

Describing data fields

---

**VARCHAR data type**

LOAD syntax	VARCHAR [(max_length)] [ CHARSET ccsid ]
Input field size	Contents of SMALLINT length field + 2  Default max_length: CREATE TABLE max_length if VARCHAR or VARBINARY, else 256  Length range: 1 to 32765
Input field format	SMALLINT field followed by any characters
Storage size	Length of data + 2 (unless compressed)
Target types	All
Notes	<ul style="list-style-type: none"> <li>■ The length of the data is read from the SMALLINT field preceding the data.</li> <li>■ Interpretation of the SMALLINT length field depends on the nvk value supplied on the FTP put command.</li> <li>■ If the data exceeds the LOAD max_length or runs off the end of the record, an error is generated and the load is terminated.</li> <li>■ If the data exceeds the CREATE TABLE length, the data is silently truncated.</li> <li>■ If CHARSET=65535, the data type is changed to VARBINARY.</li> <li>■ If the CREATE TABLE length is greater than 4096, the data is compressed before being written.</li> <li>■ If the target type is [VAR]BINARY, there is no conversion, that is, the input characters are copied directly.</li> <li>■ When the target is a numeric type, a data field of only blanks is an error.</li> </ul>

---



## Converting data types

The data type in a `datatype_spec` does not have to match the data type in the column definition of a `CREATE TABLE` statement, but the data types must be compatible. The FileTek FTP Data Loader automatically performs any necessary conversions, but you need to be sure that the data type you provide in a `datatype_spec` can be converted to the data type of the user table.

For instance, if the data type of the input data is `SMALLINT`, but the data type of the column in the user table is `INTEGER`, then the FileTek FTP Data Loader converts the input data from `SMALLINT` to `INTEGER`.

The FileTek FTP Data Loader trims input data fields that are longer than the length provided in a column definition of a `CREATE TABLE` statement. The FileTek FTP Data Loader also rescales `DECIMAL` or `NUMERIC` columns. For example, if the input data is `DECIMAL(7,3)` but the column definition for the user table is `DECIMAL(7,2)`, then the FileTek FTP Data Loader rescales the input data and stores it as `DECIMAL(7,2)` in the user table.

The following table identifies the allowable data type conversions. In the table:

- The *Input type* is the data type of the input data (LOAD data type).
- The *Target type* is the data type of the table being loaded (`CREATE TABLE` data 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 loading

Input type	Target type													
	BINARY	BLOB	CHARACTER	CLOB	DATE	DOUBLE PRECISION	INTEGER	NUMERIC (DECIMAL)	REAL	SMALLINT	TIME	TIMESTAMP	VARBINARY	VARCHAR
BINARY	X	X	X	X									X	X
BINARY EXTERNAL	X	X	X	X		X	X		X	X			X	X
BLOB		X		X										
BLOB_FILE		X												
CHARACTER	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CLOB		X		X										
CLOB_FILE				X										
DATE					X									
DECIMAL							X	X		X				
DECIMAL EXTERNAL							X	X		X				
DOUBLE						X	X		X	X				
FLOAT(REAL)						X	X		X	X				
FLOAT EXTERNAL						X	X	X	X	X				
INTEGER							X			X				
INTEGER EXTERNAL							X			X				
SMALLINT							X			X				
TIME EXTERNAL											X			
TIMESTAMP EXTERNAL												X		
VARBINARY	X	X	X	X									X	X
VARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X

## Calculating the length of a data field

In a datatype\_spec, you can optionally provide *explicit lengths* for most of the data types. For example, CHAR(10) describes a data field that consists of 10 bytes. Specifying CHAR without a length is also valid.

The data types DOUBLE, INTEGER, REAL, and SMALLINT have *implied lengths*. You don't specify lengths for these data types. Your host determines the size of these implied lengths. For example, INTEGER for a SPARC implementation has an implied length of 4 bytes; for a DOS implementation, 2 bytes.

A length on the BLOB\_FILE and CLOB\_FILE data types sets the maximum length of the path name. The default length is 1024 bytes.

The FileTek FTP Data Loader calculates the length of a data field:

- With the starting and ending column numbers in the POSITION clause of a field\_spec
- With the explicit or implied length of the data type in a datatype\_spec

Because you can provide both a POSITION clause and a datatype\_spec, it's possible that those lengths can conflict. You will receive a warning message when there is a conflict with the length of a data field. The FileTek FTP Data Loader determines the length of a data field in one of the following ways, *in the order listed*:

- For FLOAT, DOUBLE, INTEGER, REAL, and SMALLINT, the implied length is used. For instance, the implied length of SMALLINT is 2.
- If you specify a length for VARCHAR, VARBINARY, VARBYTE, or VARRAW, for instance, VARCHAR(25), then that length is the maximum number of characters or bytes in the data field; but, the actual length is in the 2-byte SMALLINT field preceding the data. The total length of a data field in a logical record is the actual data length plus 2. If you specify starting and

ending column numbers in a POSITION clause, be sure to include the 2-byte SMALLINT field containing the actual length.

- For delimited data, if you specify a length for the data type or if you specify starting and ending column numbers, then that length is the maximum length of the data field. The actual length may vary based on the presence of the delimiter, but it cannot exceed the maximum length.
- If you don't specify an ending column number in the POSITION clause, then the length of the data type is used.

For example, the length of the following data field is 18 characters:

```
REP_LASTNAME POSITION (4) CHAR(18)
```

- If you specify starting and ending column numbers in the POSITION clause *and* the length of a data type, then the data type length is used when the db\_ref is ANSI or Oracle and the starting and ending column numbers are used when the db\_ref is DB2.

For instance, the length of the following data field is 18 characters for ANSI and Oracle or 15 characters for DB2:

```
REP_LASTNAME POSITION (4:18) CHAR(18)
```

- If you don't specify an ending column number in the POSITION clause, and you specify a data type name but not an explicit length, then depending on the data type either the CREATE TABLE length or the *default length* of the loader data type determines the length of the data field. See the data type specifications to determine when the CREATE TABLE or default length is used.

For instance, the default length of the INTEGER EXTERNAL loader data type—which is 11—would determine the length of the following data field:

ORDER\_NUM POSITION (4) INT EXTERNAL

The CREATE TABLE length would determine the length of the following data field if the column data type is CHAR or BINARY:

REP\_LASTNAME POSITION (4) CHAR

- If you omit both the POSITION clause and the datatype\_spec, the CREATE TABLE data type determines the data type and length.

**Caution:** If the length of a data field in a datatype\_spec is greater than the length of the data type defined in the CREATE TABLE statement, then that data field will be trimmed. For example, if a column is defined as CHAR(10) in a CREATE TABLE statement, but the length of a data type in a datatype\_spec is 11 characters, then the last character of the data field will not be loaded.

## 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 using the CHARSET keyword with a CCSID value in a datatype\_spec. This character set overrides the character set in the CHARACTERSET clause of the LOAD DATA 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 (ASCII) for an individual data field. For CLOB\_FILE, this character set is for the LOB data, not the file name.

The CCSIDs you can specify with the CHARSET keyword are as follows:

- 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 default character set is 819 (ISO 8859-1), but the character set of an ORDER\_NUM data field in the input data is EBCDIC. In the datatype\_spec, you would include this CHARSET keyword to specify a different character set for this data field only:

```
ORDER_NUM POSITION (1) INT EXTERNAL(4) CHARSET 500
```

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

For CHARACTER, BLOB\_FILE, CLOB\_FILE, and any of the EXTERNAL data fields, you can include a delimiter\_spec in a datatype\_spec to:

- Specify a delimiter for an individual data field
- Override the default delimiter in a FIELDS clause for an individual data field
- Supplement the FIELDS clause for a particular data field

See page 3-6 for the format of the delimiter\_spec.

For example, assume that the data file contains terminated data. The default delimiter in the FIELDS clause is a comma, but an INTEGER EXTERNAL column called TRANSNUMBER is terminated by a colon instead of a comma. In this case, you would include a delimiter\_spec in the datatype\_spec to override the default delimiter in the FIELDS clause. Here's the FIELDS clause that sets the default delimiter to a comma:

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

Here's the field\_spec that overrides the FIELDS clause for the TRANSNUMBER column only:

```
TRANSNUMBER INT EXTERNAL(8) TERMINATED BY ':'
```

Now assume the FIELDS clause is the same (comma as a terminator), but the TRANSNUMBER data field is also enclosed by double quotes. In this case, the FIELDS clause would specify the terminator delimiter and the delimiter\_spec in the datatype\_spec would specify the enclosure delimiter. Here's the field\_spec that supplements the FIELDS clause for the TRANSNUMBER column only:

```
TRANSNUMBER INT EXTERNAL(8) ENCLOSED BY '"'
```

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

For BLOB\_FILE and CLOB\_FILE data types, include a HOST keyword to indicate the name of the remote machine where the LOB data files reside. A HOST keyword is not necessary when the LOB data files reside on your local computer. The host name is case sensitive whether or not you enclose it in single quotes.

For example, to specify a host named prod1, type:

```
BILL_IMAGE BLOB_FILE HOST 'prod1'
```

See “Specifying a user name and password to access LOB data files” on page 3-108 if the user name and password required to access this host is different from the StorHouse account ID and password you use to log into the StorHouse FTP server.

### **Specifying a path name for LOB data files**

For BLOB\_FILE and CLOB\_FILE data types, include a PATH keyword for LOB data files that are on a remote host or when the data file contains the LOB file name only (not the full path). The path must be appropriate for your host operating system and can be a fully qualified path or a relative path.

You can specify multiple paths if the LOB data files reside in different directories for a data field. The FileTek FTP Data Loader searches the path list in the given order for each file name in the data file.

Guidelines for specifying path names are as follows:

- Enclose each path name in quotes.
- Separate multiple paths with colons. Place the colon outside of the quotes. For example:

```
'/home/tka/blobdir1/': '/home/tka/blobdir2/'
```

- The ending slash (/) in a UNIX path is optional as the FileTek FTP Data Loader 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 single path for a LOB data field, type:

```
PHOTO_ID BLOB_FILE PATH '/home/tka/sth3/'
```

To specify multiple paths for a LOB data field, type:

```
PHOTO_ID BLOB_FILE PATH '/home/tka/blobdir1/': '/home/tka/blobdir2/'
```

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 user name and password to access LOB data files**

For BLOB\_FILE and CLOB\_FILE data types, include a USER clause 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 whether or not you enclose them in single quotes.



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

```
PHOTO_ID BLOB_FILE PATH '/home/tka/sth3/' USER tka/tka
```

## **Specifying a BLOB or CLOB data type**

When specifying a BLOB or CLOB data type to describe LOB data fields in the input data file, note the following guidelines:

- LOB data fields must be specified last in a field specification list of a LOAD DATA statement.
- Only BLOB and CLOB column (target) data types are valid for the BLOB and CLOB loader data types.
- LOB data fields must be relatively positioned, that is, POSITION(\*).
- The CONTINUEIF and CONCATENATE clauses are supported if LOB data records are present, but these clauses do not apply to LOB records.
- LOB records are not collected in discard files.
- LOB data fields cannot be sent to multiple output segments, that is you can't specify a fixed start position with multiple INTO TABLE clauses for different tables. This also applies to using the same table name with the DIFFERENT SEGMENT clause. For example, if you have LOB data in the input data file, you can't specify the following:

```
LOAD  
INTO TABLE table_1 (f1 POSITION(1) char(5), lobcol CLOB(20M))  
...  
INTO TABLE table_2 (f1POSITION(1) char(5), anotherlobcol CLOB(20M))  
...
```

- You can include the BLOB/CLOB data types and the BLOB\_FILE/CLOB\_FILE data types in the same LOAD DATA statement. In other words, you can load LOB data from LOB files as well as from the input data file. For example:

```
LOAD DATA
INTO TABLE x WHEN (1:4) = 'FILE' (lobcol CLOB_FILE(40))
...

INTO TABLE x WHEN (1:4) = 'INLI' (lobcol CLOB(2M) );
```

Note, however, that LOB data fields cannot be skipped due to a WHEN condition. So in the preceding example, if a record is skipped by the condition in the second INTO TABLE clause, it's because the record doesn't contain a LOB value.

- You cannot use the NULLIF and DEFAULTIF clauses to specify a condition for LOB data fields. For instance, you cannot specify the following:

```
INTO TABLE x
( not_a_lob POSITION(5) CHAR(1),
lobcol CLOB(4M) NULLIF lobcol = 'abc',
...
```

But you can include a NULLIF or DEFAULTIF clause for a LOB data field if the condition specifies a non-LOB data field or a NULL flag. For example, you can specify the following:

```
INTO TABLE x
( not_a_lob POSITION(5) CHAR(1),
lobcol CLOB(4M) NULLIF not_a_lob = 'N',
...
```

## Setting a column to a null value

You can set a column's value to a null value by using the NULLIF clause and specifying a field condition. A column is loaded with a null value if the condition is true. The column in the user table must be defined as NULL, otherwise the load will fail.

The field condition that you specify with NULLIF is the same as the one described at “Format of WHEN clause” on page 3-42. For clarity, you can optionally enclose the field condition in parentheses.

**Note:** The NULLIF clause differs from the TRAILING NULLCOLS clause described on page 3-58. Use NULLIF to replace the value of a data field that's in a logical record. Use TRAILING NULLCOLS when that data field is missing. A NULLIF clause in a FIELDS clause applies to any data field that doesn't have its own NULLIF clause in a field\_spec.

For example, suppose you want to load a null value into the INITIAL column of a user table whenever a data field consists of blanks. You could specify the column name and the BLANKS keyword with the NULLIF clause:

```
INITIAL POSITION (15) CHAR NULLIF (INITIAL=BLANKS)
```

Or you could specify the record position and the BLANKS keyword with the NULLIF clause:

```
INITIAL POSITION (15) CHAR NULLIF (15)=BLANKS
```

## Setting a column to the default value

When creating a user table, you can optionally define a default value for each column. If no default value was specified, the default is a NULL value. A default value can be:

- A literal—string, numeric, or binary—for a column of like data type

- A NULL value for a column of any data type
- The current date
- The current time
- The StorHouse account ID used to load the data for CHAR or VARCHAR columns

The FileTek FTP Data Loader loads a column's default value when you use the DEFAULTIF clause in a FIELDS clause or in a field\_spec or when you omit the field\_spec for the data field. A DEFAULTIF clause in a field\_spec describes a field condition that when true, sets the column to the default value. The field condition is the same as the one described at “Format of WHEN clause” on page 3-42. For clarity, you can enclose the DEFAULTIF field condition in parentheses.

For example, suppose you want to load a default value into a STATE column whenever a data field consists of blanks. You could specify the column name and the BLANKS keyword with the DEFAULTIF clause:

```
STATE POSITION (20:21) CHAR DEFAULTIF (STATE=BLANKS)
```

Or you could specify the column numbers and the BLANKS keyword with the DEFAULTIF clause:

```
STATE POSITION (20:21) CHAR DEFAULTIF (20:21)=BLANKS
```

## Using multiple into\_table\_specs

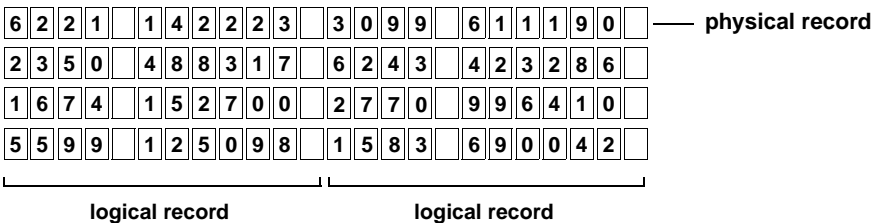
By using multiple into\_table\_specs in a LOAD DATA statement, you can:

- Create multiple logical records from one physical record
- Load data from the same data file into different user tables
- Load different segments of the same user table or multiple user tables (see page 3-60)

See pages 3-120, 3-125, and 3-129 for additional examples that show multiple into\_table\_specs.

## Creating multiple logical records from one physical record

You can create multiple logical records from one physical record by using multiple into\_table\_specs in a LOAD DATA statement. For instance, suppose you want to split each physical record into two logical records. Each logical record will contain a TRANSNUMBER column and an ACCOUNT\_NUMBER column.



You could use the following into\_table\_specs to split each physical record into two logical records and load them into one user table called ATM.TRANSACTIONS:

```
LOAD
INTO TABLE ATM.TRANSACTIONS
(TRANSNUMBER POSITION (1) INT EXTERNAL(4),
ACCOUNT_NUMBER POSITION (6) INT EXTERNAL(6))

INTO TABLE ATM.TRANSACTIONS
(TRANSNUMBER POSITION (13) INT EXTERNAL(4),
ACCOUNT_NUMBER POSITION (18) INT EXTERNAL(6));
```

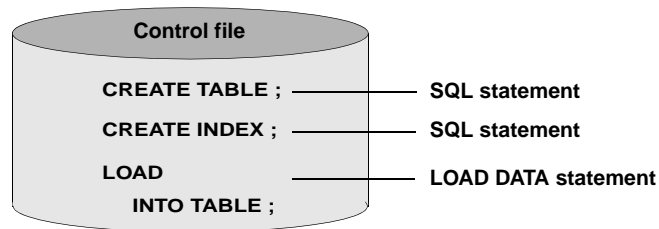


Note the following:

- The WHEN clause contains the field name RECORD\_CODE. The field\_spec describes the :RECORD\_CODE field.
- The starting and ending column numbers in the POSITION clause determine the lengths of the data fields.

## Including SQL statements in a control file

You can include StorHouse SQL statements in a control file. Those statements must precede the control statement and must end with a semicolon. For example:



The SELECT statement is the only SQL statement you cannot include in a control file. Note that the FileTek FTP Data Loader commits each SQL statement when it completes. See “Submitting an SQL statement” on page 5-1 for an example.

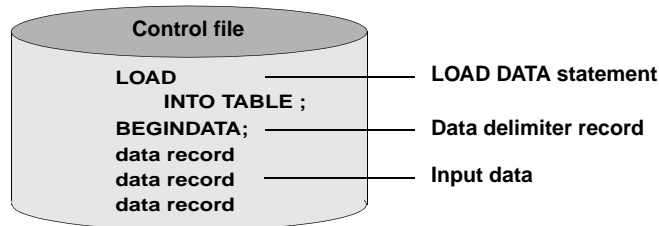
## Including data in a control file

For a data load, you can include the data records in the control file instead of a separate data file. When you do, then you must include the data delimiter record

**3****The control file**Including data in a control file

---

after the LOAD DATA statement and before the data records. This record—BEGINDATA—separates the control statements from the data.



The *only* time you include the data delimiter record is when the LOAD DATA statement and the data are in the same file. In all other cases, the FileTek FTP Data Loader includes the data delimiter record for you.

Guidelines for writing a data delimiter record are as follows:

- Start in the first character position of the record.
- Use any case (in the same character set used for the control statements).
- Type a semicolon after the keyword BEGINDATA, for example, BEGINDATA;
- Do not include comments after the record or they will be interpreted as data.
- Do not type spaces or other characters on the same line as the data delimiter record.

See page 3-129 for an example of a combined control/data file.



## Example LOAD DATA statements

This section contains examples of the LOAD DATA statement.

- Example 1 describes how to load all data records into one user table.
- Example 2 describes how to combine a fixed number of records and then load some of them into one user table.
- Example 3 describes how to load delimited data into two different user tables.
- Example 4 explains how to combine a variable number of records and then load all of them into one user table.
- Example 5 illustrates how to load SMALLINT, DECIMAL, and VARCHAR data.
- Example 6 describes how to use relative positioning to load delimited data into multiple user tables.
- Example 7 illustrates how to use multiple selection criteria to load specific records into multiple user tables.
- Example 8 describes how to load two segments of the same user table, selecting subspaces for each component type.
- Example 9 shows how to load a table with LOB data from separate LOB files.
- Example 10 describes how to load LOB data from separate LOB files as well as in the data file. This example uses two INTO TABLE clauses to specify two WHEN conditions for loading LOB data.
- Example 11 shows how to load LOB data fields using a default field list and specifying the NULLFLAGS keyword.



## Example 2: Combining a fixed number of records and loading some of them into one user table

Assume you're loading data into a user table called SUE.ORDER\_DETAILS. Each pair of physical records must be combined. You'll load only those records that contain a REP\_LASTNAME of Cornflake.

### CREATE TABLE statement

```
CREATE TABLE SUE.ORDER_DETAILS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(10) NOT NULL,
 BUYER_LASTNAME CHAR(15) NOT NULL,
 BUYER_FIRSTNAME CHAR(10) NOT NULL,
 STATE CHAR(2) NOT NULL)
TABLE SPACE ORDERS2000
```

### Data records

Two physical records make up one logical record.

2	8	3	9	M	c	G	u	i	r	e							J	a	c	k							
W	h	i	t	e													S	a	n	d	y					C	A
2	3	8	8	C	o	r	n	f	l	a	k	e						S	u	e							
H	i	l	i														R	i	c	h	a	r	d			N	M
1	4	3	9	M	c	G	u	i	r	e							J	a	c	k							
B	a	y	l	i	g	h	t										M	a	u	r	e	e	n			P	A

### LOAD DATA statement

```
LOAD
CONCATENATE 2
INTO TABLE SUE.ORDER_DETAILS
WHEN REP_LASTNAME = 'Cornflake'
(ORDER_NUM POSITION (1) INT EXTERNAL(4),
 REP_LASTNAME POSITION (5) CHAR(15),
 REP_FIRSTNAME POSITION (20) CHAR(10),
 BUYER_LASTNAME POSITION (30) CHAR(15),
 BUYER_FIRSTNAME POSITION (45) CHAR(10),
 STATE POSITION (55) CHAR(2));
```

## Example 3: Loading delimited data into multiple user tables

Assume you're loading terminated data into two user tables: JACK.ORDERS and SUE.ORDERS. You'll load all records into both tables.

### CREATE TABLE statements

```
CREATE TABLE JACK.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS2000
```

```
CREATE TABLE SUE.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS2000
```

### Data records

The data fields are terminated by a comma.

```
2839,McGuire,Jack
2388,Cornflake,Sue
1439,McGuire,Jack
3095,Cornflake,Sue
```

### LOAD DATA statement

```
LOAD
INTO TABLE JACK.ORDERS
FIELDS TERMINATED BY ','
(ORDER_NUM INT EXTERNAL(4),
 REP_LASTNAME CHAR(15),
 REP_FIRSTNAME CHAR(15))

INTO TABLE SUE.ORDERS
FIELDS TERMINATED BY ','
(ORDER_NUM POSITION(1) INT EXTERNAL(4),
 REP_LASTNAME CHAR(15),
 REP_FIRSTNAME CHAR(15));
```

## Example 4: Combining a variable number of records and loading null values

Assume that you're loading all data records into a user table called SUE.ACCOUNTS.

**CREATE TABLE statement**

```
CREATE TABLE SUE.ACCOUNTS
  (ACCOUNT_NUM SMALLINT NOT NULL,
  LASTNAME CHAR(15) NOT NULL,
  FIRSTNAME CHAR(10) NOT NULL,
  INITIAL CHAR(1),
  ADDRESS CHAR(20) NOT NULL,
  CITY CHAR(8) NOT NULL,
  STATE CHAR(2),
  FIRST_ORDER DATE,
  LAST_ORDER DATE)
TABLE SPACE ORDERS2000
```

**Data records** A variable number of physical records make up one logical record. Some data fields in logical records 1 and 3 contain blanks. Some data fields in logical record 2 are missing.

logical record 1	3	1	3	3	S	n	y	d	e	r							Q	u	a	y					*					
		2	3	5		F	o	x		H	o	l	l	o	w		D	r			F	a	i	r	l	a	w	n	*	
		O	H	1	2	/	1	0	/	1	9	9	4	1	0	/	1	2	/	1	9	9	5							
logical record 2	5	2	8	1	E	w	e	i	n	g							K	r	i	s	t	o	f	f			*			
		D	4	2		F	r	e	n	c	h	t	o	n		P	l	a	c	e			D	e	n	v	e	r		
logical record 3	1	0	0	3	H	u	r	i	w	i	n	d						O	p	a	i						*			
		H	6	2	2	9		S	o	u	t	h	e	r	n		B	l	v	d			D	e	s	t	i	n		*
		F	L	1	0	/	0	4	/	1	9	9	5																	

<b>LOAD DATA statement</b>	<b>LOAD</b> CONTINUEIF (30) = '**' INTO TABLE SUE.ACCOUNTS TRAILING NULLCOLS (ACCOUNT_NUM POSITION (1) INT EXTERNAL(4), LASTNAME POSITION (5) CHAR(15), FIRSTNAME POSITION (20) CHAR(10), INITIAL POSITION (30) CHAR(1) NULLIF INITIAL=BLANKS, ADDRESS POSITION (31) CHAR(20), CITY POSITION (51) CHAR(8), STATE POSITION (59) CHAR(2), FIRST_ORDER POSITION (61) DATE EXTERNAL(10) NULLIF FIRST_ORDER=BLANKS, LAST_ORDER POSITION (71) DATE EXTERNAL(10) NULLIF LAST_ORDER=BLANKS);
--------------------------------	--

In this example:

- CONTINUEIF THIS is the default, which means that the current physical record will be combined with the next one.
- The INITIAL data field in logical record 1 is blank; but a null value will be loaded because the NULLIF condition is true.
- Logical record 2 has three missing data fields—STATE, FIRST\_ORDER, LAST\_ORDER—but null values will be loaded because of the TRAILING NULLCOLS clause.
- The LAST\_ORDER data field in logical record 3 is blank; but a null value will be loaded because the NULLIF condition is true.

## Example 5: Loading SMALLINT, DECIMAL, and VARCHAR data

Suppose you're loading all data records into a user table called SALES\_BY\_LOCATION.

<b>CREATE TABLE statement</b>	<pre>CREATE TABLE SALES_BY_LOCATION (LOCATION_ID SMALLINT, COST DECIMAL(5,3), LAST_NAME VARCHAR(64), FIRST_NAME CHAR(12)) TABLE SPACE FY2000</pre>
-------------------------------	--

<b>LOAD DATA statement</b>	<pre>LOAD DATA INTO TABLE SALES_BY_LOCATION (LOCATION_ID POSITION(1) SMALLINT, COST POSITION(3) DECIMAL(4,2), LAST_NAME POSITION(6) VARCHAR(60), FIRST_NAME POSITION(*+2) VARCHAR(12));</pre>
----------------------------	---

Note these differences between the CREATE TABLE statement and the LOAD DATA statement:

- In CREATE TABLE, COST is data type DECIMAL(5,3).  
In LOAD, COST is data type DECIMAL(4,2).

The FileTek FTP Data Loader will rescale the length of the input data to DECIMAL(5,3).

- In CREATE TABLE, LAST\_NAME is data type VARCHAR(64).  
In LOAD, LAST\_NAME is data type VARCHAR(60).

A length in a LOAD DATA statement can be less than the length in a CREATE TABLE statement.

3

The control file

Example LOAD DATA statements

- In CREATE TABLE, FIRST\_NAME is data type CHAR(12).  
In LOAD, FIRST\_NAME is data type VARCHAR(12).

You can load a VARCHAR data field into a CHAR column as long as the data field isn't longer than the target column.

**Data records** Data is represented in hex, two hexits per column. Character data is represented in ASCII coding. For simplicity, only the characters A (x'41'), b (x'62'), c (x'63') and D (x'44') are used. All data fields are small to simplify the example.

column numbers	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
record 1	0	4	F	2	0	1	2	3	4	C	0	0	0	2	4
record 2	0	0	1	2	0	5	6	7	8	C	0	0	0	4	4

Below are descriptions of the data fields in record 1:

Column/field name	Position	Input value	Loaded value
LOCATION_ID	1 and 2	04F2	1266
COST	3 through 5	01234C	+12.340
LAST_NAME	6 through 9	00024162	Ab
*	10 and 11	2020	not loaded
FIRST_NAME	12 through 14	000163	c



Below are descriptions of the data fields in record 2:

Column/field name	Position	Input value	Loaded value
LOCATION_ID	1 and 2	0012	18
COST	3 through 5	05678C	+56.780
LAST_NAME	6 through 11	000444636262	Dcbb
*	12 and 13	2040	not loaded
FIRST_NAME	14 through 15	00024141	AA

Notice that the COST columns are rescaled to DECIMAL(5,3). Also notice the relative positioning of FIRST\_NAME. In record 1, LAST\_NAME ends in column 9; therefore, the relative position field \* starts in column 10. So \*+2 is column 12. The length field of FIRST\_NAME (hex 0001) begins in column 12. In record 2, LAST\_NAME ends in column 11; therefore, the length field of FIRST\_NAME (hex 0002) begins in column 14.

## Example 6: Using relative positioning to load delimited data into multiple user tables

Assume you're loading data into two user tables: EMPLOYEE and PROJECT.

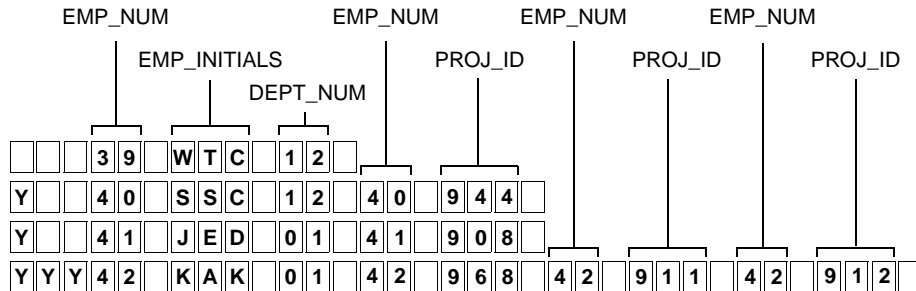
### CREATE TABLE statements

```
CREATE TABLE EMPLOYEE
(EMP_NUM SMALLINT NOT NULL,
EMP_INITIALS CHAR(3) NOT NULL,
DEPT_NUM SMALLINT NOT NULL)
TABLE SPACE EMPLOYEE
```

```
CREATE TABLE PROJECT
(EMP_NUM SMALLINT NOT NULL,
PROJ_ID SMALLINT NOT NULL)
TABLE SPACE EMPLOYEE
```

## Data records

The following example contains four logical data records.



In this example:

- Every record contains an employee definition (EMP\_NUM, EMP\_INITIALS, and DEPT\_NUM).
- A record may contain up to three project definitions (EMP\_NUM and PROJ\_ID) related to the employee.
- The first three fields contain flags indicating if the associated project definition is present. So a record with the maximum number of project definitions starts with YYY.
- A project definition follows the employee definition or the previous project definition.
- Each data field is terminated by whitespace.

## LOAD DATA statement

The LOAD DATA statement contains four into\_table\_specs. The first into\_table\_spec loads employee data (EMP\_NUM, EMP\_INITIALS, and DEPT\_NUM) into the EMPLOYEE table. The remaining into\_table\_specs load project data (EMP\_NUM and PROJ\_ID) into the PROJECT table.

The first `into_table_spec` contains a fixed-position `field_spec` (`EMP_NUM` starts at `POSITION(4)`). The remaining `into_table_specs` don't contain `POSITION` clauses; therefore, they are relatively positioned.

```
LOAD
  INTO TABLE EMPLOYEE
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM POSITION(4) INTEGER EXTERNAL(2),
  EMP_INITIALS CHAR(3),
  DEPT_NUM INTEGER EXTERNAL(2))

  INTO TABLE PROJECT
  WHEN (1:1) = 'Y'
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM INTEGER EXTERNAL(2),
  PROJ_ID INTEGER EXTERNAL(3))

  INTO TABLE PROJECT
  WHEN (2:2) = 'Y'
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM INTEGER EXTERNAL(2),
  PROJ_ID INTEGER EXTERNAL(3))

  INTO TABLE PROJECT
  WHEN (3:3) = 'Y'
  FIELDS TERMINATED BY WHITESPACE
  (EMP_NUM INTEGER EXTERNAL(2),
  PROJ_ID INTEGER EXTERNAL(3));
```

The FileTek FTP Data Loader checks each `into_table_spec` for each logical data record. If the `WHEN` clause is true or there is no `WHEN` clause, data is loaded into the table named in the `INTO TABLE` clause. The Loader then sets the relative position to the column number that follows the last field. The next selected `into_table_spec` will (if relative) start at this position.

In this example:

1. The FileTek FTP Data Loader processes the first data record.
  - a. The Loader checks the first `into_table_spec`, which doesn't contain a `WHEN` clause. Starting at column 4, the Loader then loads the `EMP_NUM`

(value of 39), the EMP\_INITIALS (value of WTC), and the DEPT\_NUM (value of 12) into the EMPLOYEE table.

b. The Loader checks the remaining three into\_table\_specs, whose WHEN clauses are false (the record doesn't contain the Y flag in at least one of the first three characters).

2. The Loader then processes the second data record.

a. The Loader checks the first into\_table\_spec, which doesn't contain a WHEN clause. Starting at column 4, the Loader then loads the EMP\_NUM (value of 40), the EMP\_INITIALS (value of SSC), and the DEPT\_NUM (value of 12) into the EMPLOYEE table.

b. The Loader checks the second into\_table\_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a DEPT\_NUM value of 12), the Loader then loads the EMP\_NUM (value of 40) and the PROJ\_ID (value of 944) into the PROJECT table.

c. The Loader checks the third and fourth into\_table\_specs, whose WHEN clauses are both false.

3. The Loader then processes the third data record.

a. The Loader checks the first into\_table\_spec, which doesn't contain a WHEN clause. Starting at column 4, the Loader then loads the EMP\_NUM (value of 41), the EMP\_INITIALS (value of JED), and the DEPT\_NUM (value of 01) into the EMPLOYEE table.

b. The Loader checks the second into\_table\_spec, whose WHEN clause is true. Starting after the last field that was previously loaded (which was a DEPT\_NUM value of 01), the Loader then loads the EMP\_NUM (value of 41) and the PROJ\_ID (value of 908) into the PROJECT table.

c. The Loader checks the third and fourth into\_table\_specs, whose WHEN clauses are both false.

4. The Loader finally processes the fourth record.
  - a. The Loader checks the first `into_table_spec`, which doesn't contain a `WHEN` clause. Starting at column 4, the Loader then loads the `EMP_NUM` (value of 42), the `EMP_INITIALS` (value of KAK), and the `DEPT_NUM` (value of 01) into the `EMPLOYEE` table.
  - b. The Loader checks the second `into_table_spec`, whose `WHEN` clause is true. Starting after the last field that was previously loaded (which was a `DEPT_NUM` value of 01), the Loader then loads the `EMP_NUM` (value of 42) and the `PROJ_ID` (value of 968) into the `PROJECT` table.
  - c. The Loader checks the third `into_table_spec`, whose `WHEN` clause is true. Starting after the last field that was previously loaded (which was a `PROJ_ID` value of 968), the Loader then loads the `EMP_NUM` (value of 42) and the `PROJ_ID` (value of 911) into the `PROJECT` table.
  - d. The Loader checks the fourth `into_table_spec`, whose `WHEN` clause is true. Starting after the last field that was previously loaded (which was a `PROJ_ID` value of 911), the Loader then loads the `EMP_NUM` (value of 42) and the `PROJ_ID` value of 912) into the `PROJECT` table.

## Example 7: Using multiple selection criteria and including the data in the control file

Assume you're loading data into three user tables: `TOLLFREE`, `TOLLCALL`, and `JUSTLOCALS`. The data is in the same file as the `LOAD DATA` statement, so the control file also contains the data delimiter record (`BEGINDATA`).

### CREATE TABLE statements

```
CREATE TABLE TOLLFREE
(FROMNUM BINARY(5) NOT NULL,
TONUM BINARY(5) NOT NULL,
ACCOUNT CHAR(12) NOT NULL)
TABLE SPACE MONTHLYCALLS
```

**3****The control file**Example LOAD DATA statements

---

```
CREATE TABLE TOLLCALL  
(FROMNUM BINARY(5) NOT NULL,  
TONUM BINARY(5) NOT NULL,  
ACCOUNT CHAR(12) NOT NULL)  
TABLE SPACE MONTHLYCALLS
```

```
CREATE TABLE JUSTLOCALS  
(FROMNUM BINARY(5) NOT NULL,  
TONUM BINARY(5) NOT NULL,  
ACCOUNT CHAR(12) NOT NULL)  
TABLE SPACE MONTHLYCALLS
```

**Combined  
control/data file**

This load data statement contains three into\_table\_specs. You can use any case (except in quoted strings) in a control statement. The example shows lowercase.

```
load  
into table tollfree  
(fromnum position(1) binary external(10),  
tonum binary external(10),  
account char(12))  
when (11:13) = '800' or (11:13) = '888'
```

```
into table tollcall  
(fromnum position(1) binary external(10),  
tonum binary external(10),  
account char(12))  
when (11:13) != '800' and (11:13) != '888'
```

```
into table justlocals  
(fromnum position(1) binary external(10),  
tonum binary external(10),  
account char(12))  
when ((1:3) = '301' or (1:3) = '410') and ((11:13) = '301' or (11:13) = '410');
```

```
begindata;  
30155512348005551212acct1234567  
30125110004105554321acct12345679  
60755598768881234567acct98765432  
30150512346075154321acct76456732
```

In this example:

- The first `into_table_spec` specifies that a record containing 800 or 888 in positions 11 through 13 be loaded into the `tollfree` table.
- The second `into_table_spec` specifies that a record that *does not* contain 800 or 888 in positions 11 through 13 be loaded into the `tollcall` table.
- The third `into_table_spec` specifies that a record containing either 301 or 410 in positions 1 through 3 *and* containing either 301 or 410 in positions 11 through 13 be loaded into the `justlocals` table.

**Load results**

The FileTek FTP Data Loader loads the following data into the three tables.

Table	FROMNUM	TONUM	ACCOUNT
tollfree	3015551234	8005551212	acct1234567
	6075559876	8881234567	acct98765432
tollcall	3012511000	4105554321	acct12345679
	3015051234	6075154321	acct76456732
justlocals	3012511000	4105554321	acct12345679

## Example 8: Selecting subspaces for a component type

Assume you're loading two segments of a user table called CALLSDBA.BILLSUMMARY. The data is in the same file as the LOAD DATA statement, so the control file also contains the data delimiter record.

### CREATE TABLE SPACE statement

The user tablespace contains six subspaces: two for table data, two for hash indexes, and two for value indexes.

```
CREATE TABLE SPACE BILLING
(SUBSPACE 1 VSET TCAT1 FSET TCAT1 OBJECT_TYPE T,
SUBSPACE 2 VSET HCAT1 FSET HCAT1 OBJECT_TYPE H,
SUBSPACE 3 VSET VCAT1 FSET VCAT1 OBJECT_TYPE V,
SUBSPACE 4 VSET TCAT2 FSET TCAT2 OBJECT_TYPE T,
SUBSPACE 5 VSET HCAT2 FSET HCAT2 OBJECT_TYPE H,
SUBSPACE 6 VSET VCAT2 FSET VCAT2 OBJECT_TYPE V)
```

### CREATE TABLE statement

The user table is assigned to the BILLING user tablespace.

```
CREATE TABLE CALLSDBA.BILLSUMMARY
(BILL_ACCOUNT BINARY(5) NOT NULL,
BILL_DATE DATE NOT NULL,
BILL_CATEGORY CHAR(1) NOT NULL,
NUMBER_CALLS INTEGER)
TABLE SPACE BILLING
```

### CREATE INDEX statements

The indexes are assigned to the same user tablespace (BILLING) as the user table (no TABLE SPACE clause on CREATE INDEX).

```
CREATE HASH INDEX BILLACCOUNT
ON CALLSDBA.BILLSUMMARY (BILL_ACCOUNT)
```

```
CREATE HASH INDEX BILLCATEGORY
ON CALLSDBA.BILLSUMMARY (BILL_CATEGORY)
```



**Combined  
control/data file**

```
CREATE VALUE INDEX BILLDATE
ON CALLSDBA.BILLSUMMARY (BILL_DATE)
```

This LOAD DATA statement creates two table data files, four hash index files, and two value index files. Data records containing a BILL\_CATEGORY of 1 are loaded into the first segment and those containing a BILL\_CATEGORY of 2 are loaded into the second segment.

```
LOAD DATA
INTO TABLE CALLSDBA.BILLSUMMARY
TABLE SUBSPACE 1
HASH SUBSPACE 2
VALUE SUBSPACE 3
(BILL_ACCOUNT BINARY EXTERNAL(10),
BILL_DATE DATE EXTERNAL(10),
BILL_CATEGORY CHAR(1),
NUMBER_CALLS INT EXTERNAL(3))
WHEN BILL_CATEGORY='1'

INTO TABLE CALLSDBA.BILLSUMMARY
DIFFERENT SEGMENT
TABLE SUBSPACE 4
HASH SUBSPACE 5
VALUE SUBSPACE 6
(BILL_ACCOUNT POSITION(1) BINARY EXTERNAL(10),
BILL_DATE DATE EXTERNAL(10),
BILL_CATEGORY CHAR(1),
NUMBER_CALLS INT EXTERNAL(3))
WHEN BILL_CATEGORY='2';
```

```
BEGIN DATA;
301792833901/31/20001004,
703274028301/31/20001002,
703872283901/31/20002001,
301339439201/31/20001003,
301340920301/31/20001002,
703419408301/31/20002003,
703229383901/31/20002001,
```

## 3

**The control file**Example LOAD DATA statements

---

In this example, the FileTek FTP Data Loader uses the following subspaces:

- Subspace 1 for the table data file in segment 1
- Subspace 2 for the two hash index files in segment 1
- Subspace 3 for the value index file in segment 1
- Subspace 4 for the table data file in segment 2
- Subspace 5 for the two hash index files in segment 2
- Subspace 6 for the value index file in segment 2

**Note:** In this example, SUBSPACE ROTATE has the same effect.

**Load results**

The FileTek FTP Data Loader loads the following data into each segment:

Segment	BILL_ ACCOUNT	BILL_ DATE	BILL_ CATEGORY	NUMBER_ CALLS
1	3017928339	01/31/2000	1	004
	7032740283	01/31/2000	1	002
	3013394392	01/31/2000	1	003
	3013409203	01/31/2000	1	002
2	7038722839	01/31/2000	2	001
	7034194083	01/31/2000	2	003
	7032293839	01/31/2000	2	001

## Example 9: Loading LOB data from local LOB files

This example loads an account number and a photo into a user table called photoids. The photos are in separate LOB data files on the client computer.

### CREATE TABLE statement

When space permits, the photos are to be stored in the table data file (in-line) with the account numbers.

```
create table photoids
(account_num smallint,
photo blob inline)
tablespace customercare
```

### CREATE INDEX statement

The index is assigned to the same user tablespace (customercare) as the user table (no tablespace clause on create index).

```
create range index by_account
on photoids (account_num)
```

### CREATE TABLE SPACE statement

The user tablespace contains two subspaces. The FileTek FTP Data Loader uses subspace 2 only when a photo must be stored out-of-line, that is, the photo or a row exceeds 32 KB.

```
CREATE TABLE SPACE customercare
(SUBSPACE 1 VSET T1 FSET T1 OBJECT_TYPE T,
SUBSPACE 2 VSET L1 FSET L1 OBJECT_TYPE L)
```

**Data file** Each input data record consists of two data fields: an account number (terminated by a comma) and the name of the LOB data file containing the photo.

```
1029,ssc_photo.jpg,  
4938,ggb_photo.jpg,  
3019,jde_photo.jpg,  
2267,slc_photo.jpg,  
3221,sgm_photo.jpg,  
2602,kak_photo.jpg,  
3677,sla_photo.jpg,  
1082,pkk_photo.jpg,
```

**LOAD DATA statement** The field specification for the BLOB data field contains the path and user information necessary to access the LOB data files.

```
LOAD  
INTO TABLE photoids  
(account_num int external(4) terminated by ',',  
photo blob_file PATH '/home/tka/photos/' USER 'tka'/'tka' terminated by ',');
```

## Example 10: Loading LOB data in the input data file and from LOB files

This example shows how to load LOB data in the input data file with the other non-LOB data as well as from separate LOB files.

**CREATE TABLE statement** The user table contains one CHAR column and one CLOB column.

```
create table sometimesinstreamlob  
(f1 char(5),  
f2 clob (1M) default 'the default')  
table space sm;
```

**Data file** The data file is in var format. Binary data is displayed in hex and enclosed in brackets.

```
[0013]FLOB01,lob1file.lob,
[001b]LLOB02[000000000000000d]lob2 contents
[0009]FLOB03, ,
[0006]LLOB04
[0018][0000000000000010]data for 4th LOB
[0013]FLOB05,lob5file.lob,
```

Note the following:

- The data in the first record is in a separate LOB file.
- The data in the second record is a LOB value that fits in the record, that is, a LOB record.
- The data in the third record is a NULL value.
- The data in the fourth and fifth record is a LOB value that spans two records. The fifth record is a continuation from the fourth record.
- The data in the sixth record is in a separate LOB file.

#### LOAD DATA statement

The LOAD DATA statement contains two INTO TABLE clauses for the same table. The first INTO TABLE clause specifies a WHEN condition for loading data in separate LOB data files. The second INTO TABLE clause specifies a WHEN condition for loading data in the input data file.

```
load data
into table sometimesinstreamlob
when (1) = 'F'
(f1 position(2) char terminated by ',',
f2 clob_file USER 'me'/'my_passwd' PATH '/home/me/mylobs' terminated by ',
defaultif f2 = BLANKS)
```

```
into table sometimesinstreamlob
when (1) = 'L'
(f1 position(2) char(5),
f2 clob);
```

**Load results** This load produces the following table:

**sometimesinstreamlob table**

f1	f2
LOB01	<lob1file.lob contents>
LOB02	lob2 contents
LOB03	the default
LOB04	data for 4th LOB
LOB05	<lob5file.lob contents>

## Example 11: Loading LOB data fields using a default field list and NULLFLAGS

This example shows how to load LOB data fields in the input data file using all the LOAD DATA statement defaults and specifying the NULLFLAGS keyword.

### CREATE TABLE statement

The user table contains two CLOB columns and one CHAR column.

```
create table insteamlobs
(f1 clob, f2 clob, f3 char(5))
table space sm;
```

### Data file

The data file is in var format. Binary data is displayed in hex and enclosed in brackets.

```
[002d]FFFline1[000000000000000f]f1lob1
contents[00000000000000006]f2lob1
[0008]FFTline2
[0008][000000000000000f]
[0006]f1lob2
[0011] contents[0000000000000000]
[001e]TTF [0000000000000000][0000000000000006]f2row3
```

Note the following:

- The first record contains all of the data to be loaded into the first row of the table. All values fit within the record.
- The data to be loaded into the second row of the table is spread across four records. The first record contains the non-LOB field. The second, third, and fourth records contain a LOB value. The fourth record also contains a placeholder [0000000000000000] for a NULL value.
- The fifth record contains the data to be loaded into the third row of the table. Two data fields are NULL. The record contains data (blanks and [0000000000000000]) as a placeholder for those fields.

**LOAD DATA  
statement**

The LOAD DATA statement simply contains the FIELDS clause with the NULLFLAGS keyword.

load data  
into table instreamlobs  
fields nullflags;

**Load results**

Here's the content of the loaded table. All blanks indicate NULL data.

**instreamlobs table**

f1	f2	f3
f1lob1contents	f2lob1	line1
f1lob2contents		line2
	2row3	

## Loading a deferred index

After creating a deferred index, you can load index entries for some or all of the segments of the table. You use the `LOAD INDEX` statement in a control file to perform an index load operation. The default is to load all segments, after which each loaded index is marked as complete or no longer deferred. Keep in mind that some queries may not use incomplete or partial indexes.

Note the following guidelines:

- You can load multiple indexes for the same table in one operation.
- You can specify a range or list of segments to load.
- All metadata inserts and updates are performed after each segment is processed, rather than when the operation is confirmed.
- The validate feature (to check the syntax of the control file) is not available for an index load operation.
- Only one `LOAD INDEX` statement is allowed in a control file.
- Other non-load statements, such as `CREATE INDEX`, are allowed in a control file but must precede the `LOAD INDEX` statement. You can use this feature to create the deferred indexes and then load them.
- You identify segments by segment ID with the `SEGMENTS` clause. Segment IDs are listed in the `SYSSTHSEGMENTS` system table. If you specify an invalid segment, the data loader ignores it. The `SEGMENTS` clause applies to all types of indexes.
- The `SUBSPACE ROTATE` and `SUBSPACE number` clauses do not apply to range indexes.



## Format of LOAD INDEX statement

LOAD INDEX[ES] index\_name [ ,index\_name ]... [subspace\_clause]  
[SEGMENTS segment\_list]

where subspace\_clause is:

SUBSPACE ROTATE |  
[ VALUE | HASH ] SUBSPACE number

Argument	Format
index_name	(required) Name of the index to be loaded. You can specify multiple index names to load multiple indexes (for the same table) in one operation.
SUBSPACE ROTATE	(optional) Clause to rotate value indexes or hash indexes among subspaces. <ul style="list-style-type: none"><li>■ This clause is useful only when loading indexes for multiple segments.</li><li>■ SUBSPACE ROTATE does not apply to range indexes.</li><li>■ If you omit this clause and the SUBSPACE number clause, the FileTek FTP Data Loader uses the lowest-numbered subspace for the index type.</li></ul>
SUBSPACE number	(optional) Clause to select a specific subspace for the index type, where number is the subspace number. <ul style="list-style-type: none"><li>■ If you omit the VALUE or HASH keyword, the FileTek FTP Data Loader uses the same subspace number for both value and hash indexes.</li><li>■ If you omit this clause and the SUBSPACE ROTATE clause, the FileTek FTP Data Loader uses the lowest-numbered subspace for the index type.</li></ul>

Argument	Format
SEGMENTS segment_list	(optional) One or more segment IDs to load indexes for specific segments. If you omit this clause, the FileTek FTP Data Loader creates index entries for all segments of the table and changes the status of the index from deferred to complete.  The format of segment_list is: segment_list_item [ , segment_list_item ]...
segment_list_item	segment_range   segment
segment_range	first_segment - last_segment

---

## Example LOAD INDEX statements

This section contains example LOAD INDEX statements.

- To load index files for the ORDERS2000 index for all segments of the table:

```
LOAD INDEX ORDERS2000
```

The data loader uses the lowest-numbered subspace for the index type because the SUBSPACE number and SUBSPACE ROTATE clauses are omitted.

- To load an index file for the ORDERS2000 index for the first segment of the table, using subspace 2:

```
LOAD INDEX ORDERS2000 SEGMENTS 0 SUBSPACE 2
```

- To load index files for the ORDERS2000 index for a range of segments, rotating among subspaces that are valid for the index type:

```
LOAD INDEX ORDERS2000 SEGMENTS 0-5 SUBSPACE ROTATE
```

- To load index files for the ORDERS2000 and ORDERSDETAIL indexes for all segments of the table:

LOAD INDEX ORDERS2000, ORDERSDETAIL

## Merging segments of a table

You can merge segments in a table by using the MERGE or COALESCE statement in a control file. A *merge*, or *coalesce*, operation consolidates a group of segments into one segment (possibly more) and invalidates the input segments. You have full control over which segments are grouped, for instance, by specific segment IDs or segment tags or by size criteria. Merging segments enhances performance because it reduces the number of file and extent opens and closes for a query.

Note the following guidelines:

- LOB subsegment files are not merged. The segment ID in a LOB file name, then, will be different from the other files in the segment. LOB file names have the original segment ID, and the table data file and index files have the new segment ID.
- The FileTek FTP Data Loader invalidates input segments after merging them but does not remove the invalidated segments. See “Replacing a segment” on page 3-65 for more information about invalidated segments.
- The FileTek FTP Data Loader automatically commits the operation after creating the result segment. You should still confirm the merge operation to delete the checkpoint.
- After merging the segments, the FileTek FTP Data Loader reports the following information about the input segments: segment ID, segment size, load confirmation time, and segment tag.

## Format of MERGE statement

```
{MERGE | COALESCE} INTO TABLE table_name [subspace_clause]
[SEGMENT segment_tag] [SEGMENTS segment_list]
[EXCLUDE segment_list] [MAXINSIZE n] [MINOUTSIZE n]
```

where subspace\_clause is:

```
SUBSPACE ROTATE |
[ VALUE | HASH | TABLE ] SUBSPACE number
```

Argument	Format
table_name	(required) Name of the table with the segments to be merged.
SUBSPACE ROTATE	(optional) Clause to rotate the components (table data, value indexes, hash indexes) among subspaces. This clause is useful for merge operations that produce multiple result segments. If you omit this clause and the SUBSPACE number clause, the FileTek FTP Data Loader uses the lowest-numbered subspace for the component type.
SUBSPACE number	<p>(optional) Clause to select specific subspaces for components in a result segment, where number is the subspace number.</p> <ul style="list-style-type: none"> <li>■ If you omit the TABLE, VALUE, or HASH keyword, the FileTek FTP Data Loader uses the same subspace number for all components.</li> <li>■ If you omit this clause and the SUBSPACE ROTATE clause, the FileTek FTP Data Loader uses the lowest-numbered subspace for the component type.</li> </ul>
SEGMENT segment_tag	(optional) Name of the result segment. A segment tag cannot exceed 40 characters and must follow SQL identifier conventions (see page 3-3) or be quoted. If you omit this clause, the FileTek FTP Data Loader uses the load ID as the segment tag.

Argument	Format
SEGMENTS segment_list	<p>(optional) One or more input segments to be merged. You can identify the input segments by segment ID or segment tag. If you omit this clause, the FileTek FTP Data Loader considers all segments in the table. If you specify an invalidated or non-existent segment, the data loader issues a warning message but continues the operation.</p> <p>The format of segment_list is:</p> <p>IDS segment_list_item [ , segment_list_item ]...   TAGS segment_tag [ , segment_tag ]...</p>
segment_list_item	segment_range   segment
segment_range	first_segment - last_segment
EXCLUDE segment_list	<p>(optional) One or more segments to exclude from the merge operation. You can identify excluded segments by segment ID or segment tag. If you omit this clause, the FileTek FTP Data Loader includes all segments in the table. The format of the segment_list is the same as for the SEGMENTS clause.</p>
MAXINSIZE n	<p>(optional) Maximum size of input segments. This clause excludes segments that are larger than the specified number of bytes or a number followed by K(x1024), M(xKK), or G(xKM). If you omit this clause, the FileTek FTP Data Loader assumes no size limit on input segments.</p>
MINOUTSIZE n	<p>(optional) Minimum size of the result segment. This clause groups the input list into a result segment that is no smaller than the specified value. If you omit this clause, the FileTek FTP Data Loader assumes no minimum size limit on result segments.</p> <ul style="list-style-type: none"> <li>■ If there aren't enough qualifying segments to create a group of the minimum size, the data loader does not merge them.</li> <li>■ When grouping segments, once the size of the result segment exceeds this value, the data loader starts to group segments for another result segment.</li> <li>■ The size (n) is the number of bytes or a number followed by K(x1024), M(xKK), or G(xKM).</li> </ul>

## Example MERGE statements

This section contains example MERGE statements.

- To merge all segments of the CALLSDetail table into one segment:

```
MERGE INTO TABLE CALLSDetail
```

- To merge the first five segments of the CALLSDetail table into one segment:

```
MERGE INTO TABLE CALLSDetail SEGMENTS IDS 0-4
```

- To merge all segments of the CALLSDetail table except segment ID 0:

```
COALESCE INTO TABLE CALLSDetail EXCLUDE IDS 0
```

- To merge all small segments that are less than 100MB of the CALLSDetail table into segments of at least 1GB but ignore segments with the segment tag late\_entries:

```
MERGE INTO TABLE CALLSDetail MAXINSIZE 100M  
MINOUTSIZE 1G EXCLUDE TAGS "late_entries"
```

# The FTP commands

This chapter describes the FTP commands you use to:

- Log into the StorHouse FTP server
- Set the transfer type
- Validate the data file and load control file before loading
- Transfer the control file
- Transfer the data file
- Pipe the data
- Check the status of an operation
- Restart an operation
- Confirm an operation
- Abort an operation
- 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 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 4-9 for other guidelines about submitting FTP commands to the StorHouse FTP server.

## FTP commands for loading data

You enter the following user-level FTP commands with your client FTP tool to load 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.
open	USER PASS	Logs into the StorHouse FTP server at alternate port 1985.
put or send	PORT STOR ALLO	Transfers the control file and the data file or pipes the input from a program; confirms, restarts, aborts, and checks the status of an operation.
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.
	PASR	Runs in FTP passive mode. The user-level command varies by client FTP tool or may be the default used by the client.



If your client FTP tool supports the FTP quote command, you can also send the following server-level commands directly to the StorHouse FTP server:

- SYST – determine the operating system of the server
- STAT – print information about the status of the connection
- SITE DRAIN seconds – set the drain on error time-out

For example:

```
ftp> quote SITE DRAIN 60
```

Use the FTP remotehelp command to obtain more information about these commands. See page 4-14 for more information about the SITE DRAIN command.

## The put command

You submit the FTP put (or send) command to define the operation you want to perform: transfer a control or data file, pipe data, restart or abort an operation, check the status of an operation, confirm an operation. The format of the put command is:

```
{put | send} local-file keyword=value[,keyword=value]...
```

Argument	Description
put   send	(required) Command verb. The case of the verb must conform to the conventions of your client FTP tool. Most tools require that you type FTP command verbs in lowercase characters.
local-file	(required) Name of the control file or the data file. If you're piping the data, specify the program name and arguments here.
keyword=value	(required) Customized keywords and values necessary to load data, load indexes, and merge segments. This series of keywords and values replaces the typical remote file name argument on the FTP put command.

## Keywords for the put command

The following table describes the keywords and values for the put command.

put keyword	Description and values																				
[load_type=]	<p>(required for each put command) Identifies the type of operation. Valid values:</p> <ul style="list-style-type: none"><li>■ load – Transfers the control file or the combined control/data file. This is used for control files containing LOAD DATA, LOAD INDEX, MERGE, and SQL statements.</li><li>■ data – Transfers the data file or pipes the data.</li><li>■ confirm or end – Drops all restart information for the operation. For a data load, also updates the metadata.</li><li>■ restart – Restarts a previous incomplete operation.</li><li>■ status – Checks the status of one or more completed or in-progress operations.</li><li>■ abort – Aborts a previous operation.</li><li>■ validate – Checks for data file and control file errors. This value is not valid for an index load operation.</li></ul>																				
db_ref=	<p>(optional) Identifies which database to emulate for defaults when processing a LOAD DATA statement. Valid values:</p> <ul style="list-style-type: none"><li>■ ANSI (default)</li><li>■ Oracle</li><li>■ DB2</li></ul> <p>The db_ref value affects the following:</p> <table><tr><th></th><th>ANSI</th><th>Oracle</th><th>DB2</th></tr><tr><td><b>FLOAT length</b></td><td>8</td><td>4</td><td>8</td></tr><tr><td><b>DISCARDS 0</b></td><td>None allowed</td><td>None allowed</td><td>No limit</td></tr><tr><td><b>POSITION conflicts</b></td><td>Length of data type used</td><td>Length of data type used</td><td>Column numbers used</td></tr><tr><td><b>Short records*</b></td><td>Error generated</td><td>Null value loaded</td><td>Error generated</td></tr></table>		ANSI	Oracle	DB2	<b>FLOAT length</b>	8	4	8	<b>DISCARDS 0</b>	None allowed	None allowed	No limit	<b>POSITION conflicts</b>	Length of data type used	Length of data type used	Column numbers used	<b>Short records*</b>	Error generated	Null value loaded	Error generated
	ANSI	Oracle	DB2																		
<b>FLOAT length</b>	8	4	8																		
<b>DISCARDS 0</b>	None allowed	None allowed	No limit																		
<b>POSITION conflicts</b>	Length of data type used	Length of data type used	Column numbers used																		
<b>Short records*</b>	Error generated	Null value loaded	Error generated																		

\* See page 3-58 for details about short records.

put keyword	Description and values
sql_ccsid=	(optional) Defines the character set of 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) Defines the default character set for character input data. See “Specifying the character set of the input data” on page 3-18 for ways 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>
dbn[ame]=	(required for load_type load, restart, validate, and abort) Specifies the name of the StorHouse database. The database name is case sensitive.
fixed=	(optional) Indicates data is composed of fixed-length records with the specified record length (in bytes). If you created the data file with an editor, be sure to include one byte for the carriage return. The maximum record length is 32767 bytes. If you omit the fixed or var keyword, then the FileTek FTP Data Loader interprets the data as text records.
var	(optional) Indicates data is composed of variable-length records. Each data record is preceded by the shortword length (in nvk format). If you omit the var or fixed keyword, then the FileTek FTP Data Loader interprets the data as text records.
loadid[ent]=	(required for load_type load, restart, and abort) Uniquely identifies a load. The load ID is alphanumeric and can contain an underscore (_). It is case sensitive and cannot exceed 40 characters. You can use the same load ID in different databases. You can reuse a load ID in the same database only after you confirm or abort the load. A load ID is the default segment tag on the SEGMENT clause. If the load_type is validate, then the FileTek FTP Data Loader ignores the load ID.

put keyword	Description and values
nvk=	(optional) Supplies the host hardware type for interpreting lengths and values of binary data. See "Data type considerations" on page 2-4 for more information about native data types. Valid values: <ul style="list-style-type: none"> <li>■ SPARC (default)</li> <li>■ IBM</li> <li>■ DOS</li> <li>■ VAX</li> </ul>

---

## Guidelines for entering a put command

When specifying a put command, note the following:

- Type keywords in any order and in any case because they are not case sensitive. For instance, you can specify nvk, NVK or Nvk.
- You can omit the part of the keyword enclosed in brackets [ ]. For instance, you can omit the [load\_type=] keyword entirely (just specify the value), 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).
- The loadident and dbname values are case sensitive. For example, if you assigned a load ID of LOAD1 and need to restart that load, you'd specify LOAD1, not load1 or Load1.
- If a value contains special characters, enclose it in single quotes. For example, to assign a load ID of load\$, enclose it in single quotes like this: loadid='load\$'
- Separate each keyword and value pair with a comma and no intervening spaces, and be sure the entire put command fits on one line. For example,

put load\_type=load,var (correct), but put load\_type = load, var (incorrect, contains spaces).

- If you specify load\_type=load but the control file contains an UNLOAD statement, the load fails and you must abort it.

## When to use a keyword

You must include a load\_type value (load, data, confirm, restart, or abort) for *every* put command you issue. Other keywords are required or optional depending on the value of load\_type. For example, when you transfer a control file (load\_type=load), you must include the dbname and loadident keywords with values, but all other keywords are optional. The following table identifies the keywords to use with specific values of load\_type.

**Keywords to use with load\_type values**

put keyword	load_type values						
	load	data	confirm	status	restart	abort	validate
db_ref	optional	no	ignored	ignored	optional	ignored	optional
sql_ccsid	optional	no	ignored	ignored	optional	ignored	optional
data_ccsid	optional	no	ignored	ignored	optional	ignored	optional
dbn[ame]	<b>required</b>	no	optional	optional	<b>required</b>	<b>required</b>	<b>required</b>
fixed	optional	optional	ignored	ignored	optional	ignored	optional
var	optional	optional	ignored	ignored	optional	ignored	optional
loadid[ent]	<b>required</b>	no	optional	optional	<b>required</b>	<b>required</b>	<b>ignored</b>
nvk	optional	no	ignored	ignored	optional	ignored	optional

required – you must provide the keyword or the operation will fail.

optional – if you provide the keyword it's processed; if you don't, the default is used.

ignored – the keyword isn't processed so you don't need to use it.

no – you cannot use the keyword or the operation will fail.

## Sample session

The following session illustrates the user-level FTP commands you enter to log in, transfer files, confirm a load, and log off. It also shows some of the replies you receive during a load.

Start FTP and log in. → fltksun.1> ftp alpha1 1985

Connected to alpha1.

220 alpha1 LD/FTP server ... ready. Enter StorHouse account.

Enter your StorHouse account ID. → Name (jjw:juliette): juliette

331 Password required for StorHouse account juliette.

Enter your StorHouse password. → Password:

230 StorHouse User (account) juliette logged in.

Set the transfer type. → ftp> binary

200 Type set to I.

Transfer the control file. → ftp> put control.inp load,loadid=1,dbn=database1

200 PORT command successful.

150 Opening BINARY mode data connection.

226 \*\*\*\*\* OK, SQL stmts parsed. Now send (put) data file \*\*\*\*\*

Transfer the data file. → ftp> put data.inp data

200 PORT command successful.

150 Opening BINARY mode data connection.

226- %L-I-XLLOADST, Table loading started on server \at record 1\

226- %L-I-XLDINFO, \09-MAR-2002:13:36:35 12500 records processed...\

226- %L-I-XLDINFO, \09-MAR-2002:13:36:43 25000 records processed...\

226- %L-I-XLDINFO, \09-MAR-2002:13:36:56 50000 records processed...\

```
226- %L-I-XLDINFO, \09-MAR-2002:13:37:25 100000 records processed...\
226- %L-I-XLDINFO, \09-MAR-2002:13:38:15 200000 records processed...\
226- %L-I-XLDINFO, \09-MAR-2002:13:38:22 216312 total records
processed\
226- %WORLD-S-XWOK, A request was successfully completed.
226 ***** Load operation/request finished *****sqlcode=<0>
```

Confirm the load. →ftp> put - confirm

```
200 PORT command successful.
226- %WORLD-S-XWOK, A request was successfully completed.
226 ***** Load operation/request finished *****sqlcode=<0>
```

Log off. →ftp> quit

```
221 Goodbye.
```

## GUI and client FTP tool 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 load 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 GUI client FTP tool must provide a way for you to specify an alternate port. Be sure to open a session at the alternate port instead of the default port. Most client FTP tools let you open a session with the FTP open command. Some tools let you open a session with the UNIX ftp command, for example (where alpha1 is the name of the StorHouse system and 1985 is the alternate port):

```
fltksun.1> ftp alpha1 1985
```

## Specifying keywords for a remote file name or target directory

If your GUI client FTP tool requests a remote file name, enter put keywords and values instead. Note this:

- Don't type the command verb put; simply supply its keywords and values.
- Type a semicolon last to indicate the end of the keywords and values.
- Use all other conventions described on page 4-6.

Below is an example:

**Enter remote file name:**

If your GUI client FTP tool requests a target directory (and then appends a remote file name), do the same thing: type the keywords and values and terminate the string with a semicolon. The FileTek FTP Data Loader ignores any additional text that the tool appends to the string (such as a remote file name).

## Listing the remote directory

Some GUI tools automatically list the remote directory (using the PWD, NLST, and LST server-level FTP commands) before and after a file transfer. The FileTek FTP Data Loader lets your tool do this, but a remote directory and listing do not apply to this system so responses are meaningless. For instance, the response to a PWD command is:

257 "FTP\_Data\_Loader" is pseudo-directory



## Using automatic binary transfer type

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 Loader supports BINARY transfer type, 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 GUI 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 load succeeds or fails.

## Resetting the transfer type to ASCII

If you close a connection using the FTP close command after a BINARY transfer and then open another connection using the FTP open command, your client FTP tool may not reset the transfer type to ASCII. If this occurs, any data sent to the StorHouse FTP server could be misinterpreted (that is, no end of lines would be found) and an error would occur. To avoid this, do one of the following:

- Always quit the FTP session using the FTP bye or quit command rather than just close the connection using the FTP close command.
- Always set the transfer type after opening a new connection.

## Globbering remote file names

Some UNIX client FTP tools preprocess (glob) file names in put 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 characters that require quotes in a put keyword string

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
[ ]	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 prints whatever text was entered after doing any globbing dictated by the shell.

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

## Using CtrlC to abort a transfer

For most UNIX implementations, when you press CtrlC to stop a file transfer, the client FTP tool will close the data channel and wait for the server's normal reply. Since closing the data channel is the typical way the EOF is indicated, the StorHouse FTP server will think that the transfer ended normally.

**Caution:** If you press CtrlC or kill a client-side process involved in a load (for instance, the FTP session or a data feeder), abort the load using `load_type=abort` on the FTP put command because a restart is not possible.

The FileTek FTP Data Loader will try to detect a premature EOF by enforcing the constraint that the last line in the data stream must be followed by a newline (text data) or the data ends cleanly at the end of a data record (fixed and var data). This way, if you use CtrlC or kill a process, the EOF will likely occur during the middle of a line (or block), triggering an error.

## Handling client-side errors

Because the end of the data stream is always interpreted as a normal EOF, there is no straightforward way to abort a load if an error is detected on the client side of the load. This is especially a problem when you use a data feeder program (such as a data generator, ESQLE program, or data filter) instead of a data file. A data feeder is more likely to encounter an error between records and therefore the premature EOF will look completely normal (that is a “clean” break at the end of a record). This will likely result in the load ending successfully, masking the fact that an error has occurred.

A suggested strategy to prevent this problem is for any data feeder program (on an error) to send data that will trigger a loader error to stdout. The suggested data depends on the FTP record format used:

- For text data, send a line not terminated by a newline
- For fixed-length data, send a different-length record
- For variable-length data, send a non-zero var length with no data (or short data)

For example, you could place the following C statements in the error handling code for a data feeder program. Each of these will cause an XLEOFTERM error in the FileTek FTP Data Loader:

#### C statements in client error handling code

Data format	C statement in error handler
text (default)	<code>printf ("*Client-side ERROR detected*");</code>
fixed-length	<code>printf ("*Client-side ERROR detected*");</code> <code>/* Note: make sure fixed length is different */</code>
variable-length	<code>short varlen = 12345;</code> <code>write (fileno(stdout), &amp;varlen, sizeof(short));</code> <code>write (fileno(stdout), "*Client-side ERROR detected", 27);</code>

## Setting a timer to drain the data channel

If an error occurs during a load or restart and the StorHouse FTP server closes the data channel before receiving all of the data, your client FTP tool receives a SIGPIPE signal. Some client FTP tools read the status reply after a SIGPIPE error, and other tools discard the reply. If your client FTP tool does not read replies after SIGPIPE signals, you can:

- Issue a put command with `load_type=status` to obtain the status of the load. See page 4-26 for more information about determining the status of a load.

- Set a timer with the quote SITE DRAIN command that enables the StorHouse FTP server to continue reading the input before closing the data channel. The default is 300 seconds (5 minutes). You can increase or decrease this value depending on how long you want to wait to get the status of a failed load.

For example, to increase the SITE DRAIN value to 600 seconds (10 minutes), type:

```
ftp> quote SITE DRAIN 600
```

If your client FTP tool returns replies after a SIGPIPE error, you can set the SITE DRAIN value to 0. For example:

```
ftp> quote SITE DRAIN 0
```

## Automating command entry

Although there's no client software to install or files to configure on your client computer to run the FileTek FTP Data Loader, there are some ways that you can automate the login process and the entry of the put command.

### Using FTP auto login

For SunOS, the .netrc file on your home directory contains information for automating the FTP login process to a remote computer. You can set up this file to connect to the StorHouse FTP server just as you would for any remote computer. Refer to your FTP publication for more information about the .netrc file and using auto login, but note the following:

- The .netrc file must reside on the home directory of the computer you're using to load data.

- If you include the StorHouse account ID and password in your `.netrc` file, disallow read access to that file for everyone other than the owner (you); otherwise, the client FTP tool won't perform the auto login.
- If you omit the StorHouse password and include only the account ID in your `.netrc` file, then you can allow read access to that file and you'll be prompted to enter the StorHouse password during login.

Here's a sample auto login entry in a `.netrc` file:

```
machine alpha1 login juliette password romeo
```

In this example, the StorHouse name is `alpha1`, the StorHouse account ID is `juliette`, and the StorHouse password is `romeo`. With this setup, Juliette won't be prompted for her StorHouse account ID and password when logging into the StorHouse FTP server.

## Creating macros for the put command

Your `.netrc` file can also contain up to 16 macros that you create with the FTP `macdef` command. This is especially useful for the `put` command. For example, instead of typing both `put` keywords and values, you can simply type the values at the command line.

Refer to your client FTP documentation for more information about the `macdef` command. This section contains sample macros and examples of invoking those macros at the command line.

### Macro for loading data that's in the control file

<b>In <code>.netrc</code> file</b>	<code>macdef l</code> <code>put \$1 load,loadid=\$2,dbname=\$3</code>
------------------------------------	--

<b>At command line</b>	<code>ftp&gt;\$l file1.inp A database1</code>
------------------------	---

\$l invokes the macro named l (for load)  
file1.inp is the name of the file containing control statements and data (\$1)  
A is the load ID (\$2)  
database1 is the database name (\$3)

### **Macro for loading data in a separate data file**

**In .netrc file**      macdef ld  
                     put \$1 load,loadid=\$2,dbname=\$3  
                     put \$4 data

**At command line**      ftp>\$ld control.inp 2 database1 data.inp

\$ld invokes the macro named ld (for load data)  
control.inp is the name of the control file (\$1)  
2 is the load ID, which must be unique from other active loads (\$2)  
database1 is the database name (\$3)  
data.inp is the name of the data file (\$4)

### **Macro for confirming a load during the same FTP session**

**In .netrc file**      macdef e  
                     put - end

**At command line**      ftp>\$e

\$e invokes the macro named e (for end)

### **Macro for restarting a load**

**In .netrc file**      macdef r  
                     put \$1 restart,loadid=\$2,dbname=\$3

**At command line**     ftp>\$r file1.inp 1 database1

\$r invokes the macro named r (for restart)  
file1.inp is the name of the file transferred during the load (\$1)  
1 is the load ID originally assigned to the load (\$2)  
database1 is the database name (\$3)

### **Macro for aborting a load**

**In .netrc file**     macdef a  
                     put \$1 abort,loadid=\$2,dbname=\$3

**At command line**     ftp>\$a file1.inp 1 database1

\$a invokes the macro named a (for abort)  
file1.inp is the name of the file transferred during the load (\$1)  
1 is the load ID originally assigned to the load (\$2)  
database1 is the database name (\$3)

## **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 allow 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.

**Note:** If you're set up for auto login and depending on how you set up your .netrc file, you won't be prompted for the StorHouse account ID and possibly the password. See "Using FTP auto login" on page 4-15 for more information about setting up your computer for auto login.



**▼ 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 **J** :

```
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 **J** .
3. At Password:, type your StorHouse account password and press **J** .

**▼ To log in with the open command**

1. At the UNIX prompt, type ftp and press **J** .
2. At the ftp prompt, type the following (replace sth\_name with the name of your StorHouse system) and press **J** :

```
open sth_name 1985
```

For example (where sth\_name is alpha1):

```
open alpha1 1985
```

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

## Setting the transfer type

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

- You can transfer control statements in ASCII or BINARY mode.
- You *must* transfer fixed-length data and variable-length data in BINARY mode.
- You can transfer text data in ASCII or BINARY mode, but in BINARY mode, the text lines must be terminated by ASCII newlines (\n).
- When transferring in ASCII mode from a non-ASCII machine, your client FTP tool will convert the data. You must *not* specify the data\_ccsid keyword and value on the put command.

The default transfer type is ASCII, so if that's the one you want, you don't have to set the transfer type. 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. In this case, any data you send to the StorHouse FTP server may be misinterpreted (no end of lines), and you'd get an error. FileTek recommends that you always set the transfer type after logging in.

### ▼ To set the transfer type to ASCII

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

### ▼ To set the transfer type to BINARY

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

## Validating the data and control file before loading

You can validate the data file and the control file before starting a data load or a merge operation. The validate feature is not available for an index load operation. The loader reports any errors without creating or merging segments, discard files, and checkpoints. To perform a validate operation, you use the same options and steps as a normal load except you use the validate keyword instead of the load keyword, any load ID is ignored, and the restart, status, abort, and confirm operations are not applicable.

### ▼ To validate the data and control file

1. Transfer the control file (or combination data and control file). At ftp>, type the following and press **J** :

- put
- followed by the name of the control file
- followed by validate
- followed by the database name and desired put keywords and values separated by commas with no intervening spaces

For example, to validate a control file or combination data and control file called control.inp, for a database called database1, and to use the defaults for the other keywords, type:

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

2. For a load operation, if the data is in a separate data file, transfer the data file. At ftp>, type the following and press **J** :

- put
- followed by the name of the data file
- followed by data
- followed by the fixed keyword and value or the var keyword, if appropriate

For example, to transfer a data file called `data.inp` containing fixed-length data with a record length of 50, type:

```
ftp> put data.inp data,fixed=50
```

3. If the data and control files contain errors, fix the errors and repeat this procedure to revalidate them.

## Transferring the control file

Use this procedure when you're:

- Loading deferred indexes
- Merging segments
- Issuing SQL statements only
- Loading data in the same file as the LOAD DATA statement
- Loading data in a separate file from the LOAD DATA statement
- Loading data by piping from a program
- Loading data in a VRAM file on StorHouse

You must include `load_type=load` as well as the `dbname` and `loadident` keywords and values on the `put` command. For a data load, if you omit all other keywords and values, then the FileTek FTP Data Loader assumes you're loading text data and using the following defaults:

- `db_ref=ANSI`
- `sql_ccsid=819` (for ISO 8859-1)
- `data_ccsid=819` (for ISO 8859-1)
- `nvk=SPARC`

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

**▼ To transfer the control file**

1. At ftp>, type the following and press **J** :

- put
- followed by the name of the control file
- followed by load
- followed by the required and desired put keywords and values separated by commas with no intervening spaces

For example, to specify a control file called control.inp, a database called database1, and a load ID called 1, type:

```
ftp> put control.inp load,dbn=database1,loadid=1
```

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

- You receive an error message (one that starts with 4 or 5), then restart or abort the operation.
- The transfer completed successfully (you receive message 226- %WORLD-S-XWOK, A request was successfully completed) and you're loading data that's in the same file as the control statements, then confirm the load.
- The transfer completed successfully and you're loading data that's in a separate data file or you're piping the data, then transfer the data file after you receive message 226 OK, now send data.

## Transferring the data file

If the data and control statements are in separate files, transfer the data file after the control file. You must specify `load_type=data` and include the fixed keyword and record length value (if the data is fixed-length format) or the var keyword (if the data is variable-length format). Remember that when transferring fixed-length and variable-length data, the transfer type must be `BINARY`.

**Note:** If you are loading LOB data from separate LOB data files, simply transfer the data file. The FileTek FTP Data Loader uses a separate FTP connection to retrieve the LOB data files based on the `BLOB_FILE` or `CLOB_FILE` specification in the control file and the file names in the data file.

### ▼ To transfer the data file

1. At `ftp>`, type the following and press `J` :
  - `put`
  - followed by the name of the data file
  - followed by `data`
  - followed by the fixed keyword and value or the var keyword, if appropriate

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

```
ftp> put data.inp data,fixed=50
```

2. Your next step depends on the result of the data file transfer. If:
  - You receive an error message, then restart or abort the load.
  - The transfer completed successfully, confirm the load.

## Piping the data

If you're piping the data from a program instead of using a data file, transfer the control file, then follow this procedure.

### ▼ To pipe the data

1. At ftp>, type the following and press **J** :

- put
- followed by the name of the program and any arguments, preceded by a vertical bar (the pipe symbol) and enclosed in quotes
- followed by data
- followed by the fixed keyword and value or the var keyword, if appropriate

For example, to pipe variable-length data from a program called myprogram, type:

```
ftp> put "| myprogram myargs" data,var
```

Be sure to use quotes for put keyword strings that contain special characters. Don't put any spaces between the first quote and the pipe symbol.

2. Your next step depends on the result. If:

- You receive an error message, then restart or abort the load.
- The piping completed successfully, confirm the load.

## Checking the status of an operation

You can check the status of one or more operations in one or more databases by using the `load_type=status` keyword and value on the `put` command. The operation may be in progress or have completed successfully or unsuccessfully. Note that StorHouse removes status information after you confirm or abort an operation, so it's only available before then.

Status information includes the following:

- The load ID and start time for all selected operations
- If the operation is in progress, the current status and the number of any completed control statements
- If the operation is *not* in progress, the final status (successful or failed)
- If a data load operation finished OK, the total number of records loaded
- If the operation finished with an error, the failed control statement number along with the SQL code, the data record number, and (if applicable) the user table and data field where the error occurred

### ▼ To check the status of an operation

1. At `ftp>`, type the following and press `J` :
  - `put`
  - followed by a `-` (dash) in place of the local file name
  - followed by `status`
  - (optional) followed by a complete database name for a specific database or the wildcard character `(*)` in part of a database name for a group of databases. Omit the database name to check the status in all databases.



- (optional) followed by the exact load ID for a specific operation or an \* or % wildcard character in part of a load ID for a group of operations. Omit the load ID to display the status of all active operations.

The \* wildcard character represents 0 or more characters and the % wildcard character represents any one character. Some client FTP tools require single quotes around special characters. If your tool has this requirement, remember to use single quotes around a database name or load ID with a wildcard character.

For example:

- To check the status of all active operations in all StorHouse databases:

```
put - status  
or  
put - status dbn='*'
```

- To check the status of all active operations in a database called database1:

```
put - status,dbn=database1
```

- To check the status of all active operations in databases that start with db:

```
put - status,dbn='db*'
```

- To check the status of a specific operation called load1 in database1:

```
put - status,dbn=database1,loadid=load1
```

- To check the status of any operation that starts with the letter a in database1:

```
put - status,dbn=database1,loadid='a*'
```

The following output shows the status of two loads in two databases. The database names and load IDs are shown in angle brackets after \Load, for example

## 4

## The FTP commands

Restarting an operation

The database name and load ID were omitted, so status information for all active loads in all databases is displayed.

→ ftp> put - status

200 PORT command successful.

load1 in dbase1 completed successfully and has not been confirmed.

→ 550- %L-I-XLDINFO,\Load <dbase1:load1> started 19-MAR-2002:16:33:36\  
 550- %L-I-XLDINFO,\10 total records were successfully loaded\  
 550- %L-I-XLDINFO,\%WORLD-S-XWOK, A request was successfully completed.\  
 550- %L-I-XLDINFO, \-----\  
 load4 in dbase2 failed at the first control statement. SQL code 20234 indicates an invalid time string. The user table is EXTR4 and the field in error is the fourth one (F4).

→ 550- %L-I-XLDINFO,\Load <dbase2:load4> started 19-MAR-2002:18:36:58\  
 550- %L-I-XLDINFO,\Load failed at stmt #1 (1=first):\  
 550- %L-I-XLDINFO,\sqlcode=<-20234> data rec#1\  
 550- %L-I-XLDINFO,\error table=EXTR4 field=F4\  
 550- %L-I-XLDINFO,\%L-E-XLDSQLERR, Error returned from SQL engine.\  
 550- %L-I-XLDINFO, \-----\  
 ...

## Restarting an operation

If you received an error message when transferring a control file or the data, you can restart that operation after fixing any errors. You can also abort the operation. You must abort the operation if loading LOB data because a restart operation is not supported with LOBs. If StorHouse is shut down while an operation is in progress, you can restart or abort the operation after StorHouse is started. Note the following:

- If the LOAD DATA statement and data are in separate files, restart at the beginning (transfer the control file first and the data file second).
- If you try to restart a load but there is no checkpoint for that load ID, the FileTek FTP Data Loader treats the operation like a load instead of a restart.

- If an operation fails and you don't want to restart it, abort it. Don't restart it with a null control file.
- If there's a problem with the control file:
  - You can fix syntax errors in control statements.
  - You cannot add or delete any clauses or keywords. The control statement must contain the same clauses as the original operation.
  - You cannot add or delete control statements. For example, if the control file contains a CREATE TABLE statement, don't delete it. The control file must contain the same number of control statements as the original operation.
  - You cannot change the order of control statements. They must be in the exact order as the original operation.
  - You cannot change the type of control statement. For instance, you can't change a LOAD DATA statement to a DROP TABLE statement.
- If the control file needs to be modified (order changed, statements added, and so on), you must abort the operation and start over.
- For a load operation, the data file must contain the same data. You cannot add or delete data records.
- When restarting a load operation, the following keywords and values on the FTP put command must be the same as the original load: db\_ref, nvk, and data\_ccsid.

▼ **To restart a load when the data's in the control file, an index load operation, or a merge operation**

1. At ftp>, type the following and press **J** :
  - put
  - followed by the name of the control file
  - followed by restart
  - followed by the same put keywords and values you supplied for the original operation

For example, to restart a load with load ID A, type:

```
ftp> put file1.inp restart,dbn=database1,loadid=A
```

2. Confirm the operation when it completes successfully.

▼ **To restart a load when the data's in a separate data file**

1. Transfer the control file. At ftp>, type the following and press **J** :
  - put
  - followed by the name of the control file
  - followed by restart
  - followed by the same put keywords and values you supplied for the original load

For example, to restart a load with load ID 1, type:

```
ftp> put control.inp restart,dbn=database1,loadid=1
```

2. Transfer the data file. At ftp>, type the following and press **J** :
  - put
  - followed by the name of the data file
  - followed by data

- followed by the fixed keyword and value or the var keyword, if appropriate

For example, to specify a data file called data.inp containing fixed-length data with a record length of 50, type:

```
ftp> put data.inp data,fixed=50
```

3. Confirm the load when it completes successfully.

▼ **To restart a load when you piped the data**

1. Transfer the control file. At ftp>, type the following and press **J** :

- put
- followed by the name of the control file
- followed by restart
- followed by the same put keywords and values you supplied for the original load

For example, to restart a load with load ID 1, type:

```
ftp> put control.inp restart,dbn=database1,loadid=1
```

2. Pipe the data. At ftp>, type the following and press **J** :

- put
- followed by the name of the program and any arguments, preceded by a vertical bar (pipe symbol) and enclosed in quotes
- followed by data
- followed by the fixed keyword and value or the var keyword, if appropriate

For example, to pipe variable-length data from a program called myprogram, type:

```
ftp> put "| myprogram myargs" data,var
```

3. Confirm the load when it completes successfully.

## Confirming an operation

After an operation completes successfully, you must confirm it to remove all restart information stored in checkpoint files. Confirming an operation also allows you to re-use a load ID in a database. If you're not sure whether an operation completed successfully, check the status before confirming the operation. You can confirm an operation that completed in the same or a different FTP session.

Note the following:

- Confirming a data load operation updates the metadata. Users can access new segments *after* you confirm a load.
- Confirming an index load or merge operation simply removes restart information stored in checkpoint files.
- When you confirm an operation in the same FTP session, you may not have to provide the dbname and loadid keywords. The values default to the last operation in the same FTP session.
- When you confirm an operation from a different FTP session, provide the dbname and loadid keywords and values.

To confirm an operation, use the end or confirm keyword with the put command. You can use a dash in place of the file name.

▼ **To confirm an operation in the same FTP session**

At ftp>, type one of the following commands and press **J** :

put - end  
put - confirm

▼ **To confirm an operation from a different FTP session**

At ftp>, type the following and press **J** :

- put
- followed by a dash (-)
- followed by confirm or end
- followed by the dbname and loadid keywords and values for the operation you are confirming

For example:

ftp> put - confirm,dbn=database1,loadid=2

## Aborting an operation

Instead of restarting a failed operation, you can abort it. Aborting an operation deletes all checkpoint information. Aborting a data load operation deletes and removes any partially written segments. You must abort a load if:

- A segment exceeded the maximum size
- You are loading LOB data
- You pressed CtrlC to end a client-side process and the load appeared to end successfully

Consider aborting a load when you need to:

- Add or delete control statements in the control file
- Change the order or type of control statements
- Add or remove data records
- Change the put keywords and values for your operation

▼ **To abort an operation**

At ftp>, type the following and press **J** :

- put
- followed by a dash (-)
- followed by abort
- followed by the database name and load ID values you supplied for the original operation

For example, to abort a load with load ID 1, type:

```
ftp> put - abort,dbn=database1,loadid=1
```

## Displaying help for StorHouse FTP server commands

You can display a list of server-level FTP commands as well as a definition for a specific command by using the `remotehelp` command.

▼ **To list all StorHouse FTP server commands**

At ftp>, type `remotehelp` and press **J** .

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



▼ **To display the definition for a specific command**

At ftp>, type remotehelp followed by the server-level FTP command name, and then press **J** .

Some of the server-level commands are: ALLO, HELP, PASS, PORT, QUIT, SITE DEBUG, SITE DRAIN, STOR, STAT, SYST, TYPE A, TYPE I, USER. For example, to display a definition for the user-level FTP 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 load another user table, be sure to set the appropriate transfer type before you transfer any files.

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

At ftp>, type bye or quit and press **J** .

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

At ftp>, type close and press **J** .

# 4

## The FTP commands

---

Logging off the StorHouse FTP server

## Examples

This chapter contains examples that show how to:

- Submit an SQL statement
- Load text data, transferring one file
- Load fixed-length data, transferring two files
- Load variable-length data, transferring two files
- Replace segments
- Load a deferred index for all segments
- Merge all segments of a table

## Submitting an SQL statement

To submit an SQL statement with the FileTek FTP Data Loader, prepare a control file with the statement(s) and then transfer the control file. Even though you're not loading data, the `load_type` on the FTP put command must be `load`. You can transfer a control file in ASCII or BINARY transfer type. The FileTek FTP Data Loader commits each SQL statement after it completes.

**Control file**    The SQL CREATE TABLE statement resides in a control file called `case1.ctf`.

```
CREATE TABLE JACK.ORDERS
(ORDER_NUM SMALLINT NOT NULL,
 REP_LASTNAME CHAR(15) NOT NULL,
 REP_FIRSTNAME CHAR(15) NOT NULL)
TABLE SPACE ORDERS00 ;
```

**FTP commands**

1. Transfer the control file.

```
ftp> put case1.ctl load,dbn=database1,loadid=1
```

2. Confirm the transfer to remove the checkpoint information.

```
ftp> put - end
```

## Loading text data, transferring one file

This example shows how to load two user tables with text data in the control file. You can transfer text data in ASCII or BINARY transfer type.

**CREATE TABLE statements**

Both user tables contain three columns.

```
CREATE TABLE JACK.ORDERS  
(ORDER_NUM SMALLINT NOT NULL,  
REP_LASTNAME CHAR(15) NOT NULL,  
REP_FIRSTNAME CHAR(15) NOT NULL)  
TABLE SPACE ORDERS00
```

```
CREATE TABLE SUE.ORDERS  
(ORDER_NUM SMALLINT NOT NULL,  
REP_LASTNAME CHAR(15) NOT NULL,  
REP_FIRSTNAME CHAR(15) NOT NULL)  
TABLE SPACE ORDERS00
```

**Control file**

The LOAD DATA statement, data delimiter record, and data records reside in a control file called case2.ctl.

```
LOAD DATA  
INTO TABLE JACK.ORDERS  
FIELDS TERMINATED BY ','  
(ORDER_NUM INT EXTERNAL(4),  
REP_LASTNAME CHAR(15),  
REP_FIRSTNAME CHAR(15))
```

```
INTO TABLE SUE.ORDERS  
FIELDS TERMINATED BY ','  
(ORDER_NUM INT EXTERNAL(4),  
REP_LASTNAME CHAR(15),  
REP_FIRSTNAME CHAR(15));
```

```
BEGINDATA;  
2839,McGuire,Jack  
2388,Cornflake,Sue  
1439,McGuire,Jack  
3095,Cornflake,Sue
```

**FTP commands**

1. Set the transfer type to ASCII.

```
ftp> type ascii
```

2. Transfer the combined control/data file.

```
ftp> put case2.ctl load,dbn=database1,loadid=2
```

3. Confirm the load.

```
ftp> put - end
```

## Loading fixed-length data, transferring two files

When loading fixed-length data, you must set the transfer type to BINARY and include the `fixed` keyword and value on the FTP `put` command. You can set the transfer type before or after you transfer the control file.

## Loading fixed-length data, transferring two files

3. Transfer the data file, specifying the fixed keyword and the record length.

```
ftp> put case3.dat data,fixed=34
```

**Note:** Remember to add one byte for the carriage return if you created the data file using an editor. For instance, in this example, fixed=35.

4. Confirm the load.

```
ftp> put - end
```

## Loading variable-length data, transferring two files

When loading variable-length binary data, you must set the transfer type to BINARY, precede the data records with the length, and include the var keyword on the FTP put command. You can set the transfer type before or after you transfer the control file.

### CREATE TABLE statement

The user table called JACK.STATS contains two columns.

```
CREATE TABLE JACK.STATS
(STAT1 CHAR(5) NOT NULL,
STAT2 VARCHAR(5) NOT NULL)
TABLE SPACE Q400
```

### Data file

The data records reside in a data file called case4.dat. Each variable-length record is preceded by a SMALLINT shortword.

**Note:** Data is represented here in hex, two hexits per column. Character data is represented in ASCII coding.

0	0	0	A	6	1	6	2	6	3	6	4	6	5	0	0	0	3	6	7	6	8	6	9
0	0	0	8	6	2	6	3	6	1	6	5	6	2	0	0	0	1	6	7				

**Control file**     The LOAD DATA statement resides in a control file called case4.ctl.

```
LOAD
INTO TABLE JACK.STATS
(STAT1 POSITION(1) CHAR(5),
STAT2 POSITION(6) VARCHAR(5));
```

**FTP commands**     1. Set the transfer type to BINARY because you must transfer variable-length data in BINARY mode.

```
ftp> binary
```

2. Transfer the control file.

```
ftp> put case4.ctl load,dbn=database1,loadid=4
```

3. Transfer the data file, specifying the var keyword.

```
ftp> put case4.dat data,var
```

4. Confirm the load.

```
ftp> put - end
```

## Replacing segments without loading

To replace a segment without loading, you must create a control file with the LOAD keyword, INTO TABLE clause(s), and REPLACE clause(s). Even though you're not loading data, the load\_type on the FTP put command must be load and you must include the data delimiter record. You can transfer a control file in ASCII or BINARY transfer type.



In this example, assume you need to replace the segments created when the tollfree, tollcall, and justlocals user tables were loaded with this LOAD DATA statement:

```
load
into table tollfree
(fromnum position(1) binary external(10),
tonum binary external(10),
account char(12))
when (11:13) = '800' or (11:13) = '888'
```

```
into table tollcall
(fromnum position(1) binary external(10),
tonum binary external(10),
account char(12))
when (11:13) != '800' and (11:13) != '888'
```

```
into table justlocals
(fromnum position(1) binary external(10),
tonum binary external(10),
account char(12))
when ((1:3) = '301' or (1:3) = '410') and ((11:13) = '301' or (11:13) = '410');
```

The FileTek FTP Data Loader created one segment for each user table. The SEGMENT clause was omitted from the LOAD DATA statement, so the load ID is the default segment tag for all segments. Assume the load ID was callload (lowercase).

**Note:** StorHouse/RM normalizes segment tags explicitly provided on SEGMENT clauses to uppercase. StorHouse/RM does not normalize load IDs to uppercase. This means that if you specify a load ID in lower or mixed case on the FTP put command or if the load ID does not conform to SQL identifier conventions, you must use the same case and delimit the load ID (with double quotes) on the REPLACE clause. In this example, you must delimit callload and type it in lowercase on the REPLACE clause.

**Control file**      The LOAD DATA statement resides in a control file called case5.ctf.

```
load
into table tollfree
replace segment "callsload"

into table tollcall
replace segment "callsload"

into table justlocals
replace segment "callsload";

begindata;
```

**FTP commands**      1. Transfer the control file.

```
ftp> put case5.ctf load,dbn=database1,loadid=5
```

2. Confirm the transfer.

```
ftp> put - end
```

## Loading a deferred index for all segments

To load a deferred index, prepare a control file with the LOAD INDEX statement and then transfer the control file. The load\_type on the FTP put command must be load. You can transfer a control file in ASCII or BINARY transfer type.

**CREATE INDEX  
statement**

The deferred index is called ORDERS2000.

```
CREATE VALUE INDEX ORDERS2000
ON ORDERS (ORDER_NO, CUSTOMER_NAME)
DEFERRED
```

**Control file** The LOAD INDEX statement resides in a control file called loadindex.txt. This statement uses all of the defaults. The FileTek FTP Data Loader loads all index files for the ORDERS2000 index for all segments of the table.

```
LOAD INDEX ORDERS2000 ;
```

**FTP commands** 1. Transfer the control file containing the LOAD INDEX statement.

```
ftp> put loadindex.txt load,dbn=database1,loadid=1
```

2. Confirm the operation.

```
ftp> put - end
```

## Merging all segments of a table

To merge all segments of a table, prepare a control file with the MERGE statement and then transfer the control file. The load\_type on the FTP put command must be load. You can transfer a control file in ASCII or BINARY transfer type.

**Control file** The MERGE statement resides in a control file called merge.txt. This statement uses all of the defaults. The FileTek FTP Data Loader merges all segments of the JACK.ORDERS table.

```
MERGE INTO TABLE JACK.ORDERS ;
```

**FTP commands** 1. Transfer the control file containing the MERGE statement.

```
ftp> put merge.txt load,dbn=database1,loadid=1
```

2. Confirm the operation.

```
ftp> put - end
```

**5**

**Examples**

---

Merging all segments of a table

# **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

## Symbols

( ) in SQL syntax 3-3, 3-41  
... in SQL syntax 3-3  
.netrc file 4-15  
{ } in SQL syntax 3-3  
| in SQL syntax 3-3  
' in SQL syntax 3-3, 3-24

## Numerics

1985, alternate port 4-9  
65535 CCSID 3-106

## A

aborting a load  
    description 1-40  
    guidelines 4-33  
    macro for 4-18  
    procedure 4-34  
    with CtrlC 4-13

access privileges 1-7  
account, StorHouse 1-7  
administrator, StorHouse 1-8  
ALLO FTP command 4-2  
alternate port 4-9  
AND keyword  
    in delimiter specification 3-53  
    in WHEN clause 3-42  
application program interface (API) 3-12  
ASCII  
    text 2-2  
    transfer type 1-5  
ascii FTP command 4-2, 4-20  
ATF command privilege 1-7  
auto login 4-15  
automatic binary mode 4-11  
automating command entry 4-15

## B

BEGINDATA record 3-116

## Index

### C

- big-endian 2-5
  - BINARY data type 3-85, 3-106
  - binary data, interpreting 2-4
  - BINARY EXTERNAL data type 3-85
  - binary FTP command 4-2, 4-20
  - binary mode, automatic 4-11
  - BINARY transfer type 1-5
  - BLANK keyword 3-54
  - blanks
    - definition 2-9
    - trimming 2-9
  - BLANKS keyword
    - in CONTINUEIF clause 3-30
    - in DEFAULTIF clause 3-112
    - in NULLIF clause 3-111
  - BLOB data type 3-86
  - BLOB\_FILE data type 3-87
  - braces in SQL syntax 3-3
  - brackets in SQL syntax 3-3
  - BY keyword in FIELDS clause 3-52
  - bye FTP command 4-2
  - byte order 2-5
- ## C
- case
    - column names 3-3, 3-77
    - in control file 3-2
    - in put keywords 4-6
    - of database names 4-6
    - of load IDs 4-6
    - table names 3-3
  - CCSID 65535 3-106
  - CCSIDs 3-20
  - CHAR field\_specs 3-56
  - CHAR keyword 3-52
  - CHARACTER data type 3-88
  - character set names 3-20, 3-21
  - character strings 3-2
  - CHARACTERSET clause
    - description 3-18
    - examples 3-20
    - format 3-20
    - overriding 3-19
  - CHARSET keyword in data type specification 3-105
  - checking the status of a load 4-26
  - choosing which rows to load
    - format of WHEN clause 3-42
    - overview 3-40
    - specifying multiple test values 3-47
    - specifying starting and ending columns 3-43
    - specifying the starting column 3-43
    - using a character string as selection criteria 3-46
    - using a column name 3-44
    - using a field name 3-45
    - using a hexadecimal string as selection criteria 3-46
  - clauses
    - CHARACTERSET 3-18
    - CONCATENATE 3-20
    - CONSTANT 3-79
    - CONTINUEIF 3-22
    - DEFAULTIF 3-111
    - DIFFERENT SEGMENT 3-60
    - DISCARD DN 3-15
    - DISCARD FILE 3-15
    - DISCARD MAX 3-17



- DISCARDS 3-17
- EXCLUDE 3-145
- FIELDS 3-48
- INDDN 3-10
- INFILE 3-10
- INTO TABLE 3-39
- MAXINSIZE 3-145
- MINOUTSIZE 3-145
- NULLIF 3-111
- POSITION 3-80
- PRESERVE BLANKS 3-31
- REPLACE SEGMENT 3-65
- SAME SEGMENT 3-60
- SEGMENT 3-64, 3-144
- SEGMENTS 3-142, 3-145
- SEQUENCE 3-78
- SUBSPACE number 3-68, 3-141, 3-144
- SUBSPACE ROTATE 3-33, 3-141, 3-144
- summary of 3-7
- TRAILING NULLCOLS 3-58
- WHEN 3-40
- client FTP software 1-4
- CLOB data type 3-89
- CLOB\_FILE data type 3-90
- close FTP command 4-2
- coalesce operation 3-143
- collecting discarded records
  - limiting the number of discarded records 3-17
  - overview 3-15
- column
  - definition 2-6
  - in input data record 2-6
- column default values
  - account ID 3-112
  - current date 3-112
  - current time 3-112
  - literal 3-111
  - null value 3-112
- column definition in CREATE TABLE 3-101
- column name
  - case 3-3
  - in field specification 3-76, 3-77
  - in WHEN clause 3-42
- column numbers
  - in CONTINUEIF clause 3-23
  - in POSITION clause 3-81
- combining a varied number of records
  - current one with next one 3-24
  - format of CONTINUEIF 3-22
  - next one with previous one 3-25
  - overview 3-22
  - specifying a comparison value 3-29
  - specifying the starting and ending column numbers 3-28
  - specifying the starting column number 3-27
  - using a hexadecimal string 3-29
  - using a not equal comparison operator 3-31
  - using blanks 3-30
  - using last non-blank column 3-26
- command privileges 1-7
- commands
  - FTP server
    - ALLO 4-2
    - HELP 4-2
    - LST 4-10
    - NLST 4-10
    - PASS 4-2
    - PORT 4-2
    - PWD 4-10
    - QUIT 4-2
    - STOR 4-2
    - SYST 4-11
    - TYPE A 4-2
    - TYPE I 4-2
    - USER 4-2
  - FTP user
    - ascii 4-2
    - binary 4-2

## Index

### C

- bye 4-2
- close 4-2
- open 4-2
- put 4-2
- quit 4-2
- quote 4-2
- remotehelp 4-2
- type 4-2
- StorHouse
  - CREATE ACCOUNT 1-8
  - CREATE FILE 1-9
  - CREATE FSET 1-8
  - CREATE VSET 1-8
- commands, FTP server
  - SITE DRAIN 4-3, 4-15
  - STAT 4-3
- commands, FTP user
  - type ascii 4-2
  - type image 4-2
- comments, in control file 3-2
- comparison operators 3-23
- comparison value
  - BLANKS 3-24, 3-30
  - character 3-24, 3-29
  - converting 3-31
  - definition 2-7
  - hexadecimal 3-29
  - hexdigits 3-24
  - last non-blank column 3-26
  - padding 3-28
  - trimming 3-28
- CONCATENATE clause
  - description 3-20
  - example 3-21
  - format 3-21
- concatenating a fixed number of records
  - example 3-21
  - format of CONCATENATE 3-21
  - overview 3-20
- Concepts and Facilities Manual, StorHouse xvii
- condition, field 3-42
- confirming a load 1-10, 1-34, 4-32
- conflict of data type lengths 3-103
- CONSTANT clause 3-79
- contiguous string in SQL identifiers 3-3
- continuation field
  - definition 2-7
  - removing from physical records 3-23
  - specifying a comparison value 3-24, 3-42
  - specifying the location of 3-28
- CONTINUEIF clause
  - arguments 3-23
  - description 3-22
  - examples 3-29
  - format 3-22
  - LAST keyword 3-26
  - NEXT keyword 3-25
  - THIS keyword 3-24
- Control Center, description xiv
- control file
  - definition 3-1
  - guidelines 3-2
  - what it can contain 3-1
  - when restarting 4-29
- control statement
  - definition 3-1
  - LOAD DATA 3-5
  - LOAD INDEX 3-9
  - MERGE 3-10
- conversion
  - character set 3-46
  - data type 1-2, 3-101
  - selection criteria in WHEN clause 3-46
- CREATE FSET, StorHouse command 1-8

CREATE VSET, StorHouse command 1-8

creating

- account IDs 1-8

- discard files 1-9

- file sets 1-8

- volume sets 1-8

CRLF 1-5

CtrlC, to abort a load 4-13

## D

data channel, draining 4-14

data delimiter, in control file 3-115

data field

- column 2-6

- description 2-1

- field 2-6

- truncation 3-105

data file

- creating 2-1

- description 2-1

- including data in control file 3-115

- LOB 2-1

- transferring 4-24

- when restarting 4-29

data record 2-1

data type conversion 3-101

data type specification

- providing the length of a data type 3-103

- specifying a character set 3-105

- specifying a delimiter 3-106

- specifying a host name 3-107

- specifying a path name 3-107

- specifying a user name and password 3-108

data types

BINARY 3-85

BINARY EXTERNAL 3-85

BLOB 3-86

BLOB\_FILE 3-87

CHARACTER 3-88

CLOB 3-89

CLOB\_FILE 3-90

DATE EXTERNAL 3-91

DECIMAL 3-92

DECIMAL EXTERNAL 3-93

DOUBLE 3-94

FLOAT EXTERNAL 3-95

FLOAT or REAL 3-95

INTEGER 3-96

INTEGER EXTERNAL 3-96

SMALLINT 3-97

TIME EXTERNAL 3-97

TIMESTAMP EXTERNAL 3-98

VARBINARY 3-99

VARCHAR 2-5, 3-100

database name, specifying 4-6

DATE or DATE EXTERNAL data type 3-91

db\_ref put keyword 4-4

db\_ref=

- ANSI 3-17, 3-94, 3-104

- DB2 3-17, 3-82, 3-94, 3-104

- Oracle 3-17, 3-58, 3-94, 3-104

DBA privilege 1-8

dbname put keyword 4-5

DEC VAX native data type representation 2-5

DECIMAL EXTERNAL data type 3-93

DECIMAL or NUMERIC data type 3-92

default delimiter 3-50

default lengths of data types 3-104

default StorHouse group 3-13

**Index****D**

- default subspace 1-15
- default value, loading into a table 3-111
- DEFAULTTIF clause 3-112
- DEFAULTTIF keyword 3-54
- deferred index
  - definition 1-34
  - loading 1-34, 3-140
- definitions
  - CHARACTERSET clause 3-8
  - CHARSET clause 3-8
  - column (in input data) 2-6
  - comparison value 2-7
  - CONCATENATE clause 3-8
  - condition 3-22
  - CONSTANT keyword 3-8
  - continuation field 2-7
  - CONTINUEIF clause 3-8
  - control file 1-10
  - control statement 3-1
  - data field 2-1
  - data file 1-10
  - database reference (db\_ref) 4-4
  - default delimiter 3-50
  - default length 3-104
  - default subspace 1-15
  - DEFAULTTIF clause 3-9
  - deferred index 1-34
  - delimited data 2-8
  - delimiter 2-7
  - DIFFERENT SEGMENT clause 3-60
  - DISCARD DDN clause 3-7
  - discarded records 3-15
  - DISCARD FILE clause 3-7
  - DISCARD MAX clause 3-7
  - DISCARDS clause 3-7
  - enclosed data 2-8
  - enclosure delimiters 2-8
  - EXCLUDE clause 3-145
  - explicit length 3-103
  - field (in input data) 2-6
  - field specification 3-74
  - FIELDS clause 3-8
  - fixed position 3-80
  - HOST keyword 3-8
  - implied lengths 3-103
  - INFILE clause 3-7
  - in-line LOB 1-11
  - input data file 2-1
  - input data record 2-1
  - INTO TABLE clause 3-8
  - LOB data file 2-1
  - logical record 2-7
  - MAXINSIZE clause 3-145
  - MINOUTSIZE clause 3-145
  - native data type 2-4
  - NULLIF clause 3-9
  - nvk 4-6
  - out-of-line LOB 1-11
  - PATH keyword 3-8
  - physical record 2-7
  - POSITION clause 3-8
  - PRESERVE BLANKS clause 3-8
  - RECNUM keyword 3-8
  - relative position 3-80
  - REPLACE SEGMENT clause 3-8
  - SAME SEGMENT clause 3-60
  - SEGMENT clause 3-8
  - segment tag 3-63
  - SEGMENTS clause 3-142, 3-145
  - SEQUENCE clause 3-8
  - SQL identifier 3-3
  - SQL\_LDF\_ENGINES system parameter 1-36
  - SQL\_LDF\_MAXLOAD system parameter 1-36
  - SQL\_LDR\_MAXINTO system parameter 3-39
  - SQL\_SESSIONS system parameter 1-36
  - subspace 1-14
  - SUBSPACE clause 3-8
  - SUBSPACE number clause 3-68
  - SUBSPACE ROTATE clause 3-8, 3-33
  - SYSDATE keyword 3-8
  - target data types 3-101
  - terminated data 2-8
  - termination delimiter 2-8

- TRAILING NULLCOLS clause 3-8
  - transfer types 1-5
  - USER keyword 3-9
  - WHEN clause 3-8
  - DELETE command privilege 1-7
  - delimited SQL identifiers 3-4
  - delimiter specification
    - AND keyword 3-53
    - character delimiter 3-53
    - examples 3-54, 3-55
    - hexadecimal delimiter 3-53
    - in a data type specification 3-106
    - in a FIELDS clause 3-52
    - WHITESPACE keyword 3-53
  - describing each column to load
    - generating a sequence of values 3-78
    - loading a constant value 3-79
    - loading a default value 3-111
    - loading a null value 3-111
    - loading a record number 3-78
    - loading the system date 3-79
    - providing a column name 3-77
    - providing the data type length 3-103
    - providing the data type name 3-83
    - specifying a character set 3-105
    - specifying a delimiter 3-106
    - specifying a host name 3-107
    - specifying a path name 3-107
    - specifying a user name and password 3-108
    - specifying the position 3-80
  - DIFFERENT SEGMENT clause 3-60
  - directory path 2-11
  - discard file 1-9
  - discarded records
    - accessing 3-15
    - collecting 3-14, 3-15
    - definition 3-15
    - limiting the number of 3-17
    - loading 3-13
    - overwriting 3-16
    - when they're not saved 3-15
  - DISCARDFILE/DISCARDDN clause
    - description 3-15
    - discard file name 3-16
    - guidelines 3-16
    - StorHouse group name 3-16
  - DISCARDS/DISCARDMAX clause
    - description 3-17
    - example 3-18
    - format 3-18
    - guidelines 3-17
  - displaying help for StorHouse FTP server commands 4-34
  - displaying remote server messages 4-11
  - DOS native data type representation 2-5
  - DOUBLE data type 3-94
  - draining the data channel 4-14
- ## E
- EBCDIC character set 3-20
  - ellipsis points in SQL syntax 3-3
  - empty data field 3-49
  - EMPTY keyword 3-53
  - enclosed data 2-8
  - ENCLOSED keyword in FIELDS clause 3-53
  - enclosure delimiter 2-8
  - ending a load 4-32
  - engine 1-10, 1-34

## Index

### F

E-notation 3-93, 3-95

EOF 2-2

EOL 2-2

error reporting 1-3

errors, client-side 4-13

examples

- CHAR keyword 3-56

- CHARACTERSET clause 3-20

- CHARSET keyword 3-106

- CONCATENATE clause 3-21

- CONSTANT clause 3-80

- CONTINUEIF clause 3-25, 3-26, 3-27

- delimiter specification for a data field 3-106

- DIFFERENT SEGMENT clause 3-61

- DISCARDFILE clause 3-17

- DISCARDS/DISCARDMAX clause 3-18

- FIELDS clause 3-54, 3-55, 3-56, 3-57

- HOST keyword 3-107

- INFILE/INDDN clause 3-13, 3-14

- INTO TABLE clause 3-40

- LOAD DATA statement 3-117

- LOAD INDEX statement 3-142

- LOB data 3-135

- macros 4-17, 4-18

- MERGE statement 3-146

- multiple INTO TABLE specifications 3-112

- NULLFLAGS keyword 3-57

- NULLIF clause 3-111

- PATH keyword 3-107

- POSITION 3-81

- PRESERVE BLANKS clause 3-32

- put command 3-66, 4-21, 4-23, 4-33

- RECNUM keyword 3-78

- relative positioning 3-125

- REPLACE SEGMENT clause 3-67

- SAME SEGMENT clause 3-61

- sample load session 4-8

- SEGMENT clause 3-64

- SEQUENCE clause 3-78

- SUBSPACE number clause 3-69

- SUBSPACE ROTATE clause 3-35

- SYSDATE keyword 3-79

- TRAILING NULLCOLS clause 3-59

- USER keyword 3-108

- WHEN clause 3-43, 3-44, 3-45

exception processing 1-3

EXCLUDE clause 3-145

explicit lengths 3-103

extents 1-11

## F

features, FileTek FTP Data Loader 1-1

field

- definition 2-6

- in input data record 2-6

- specification 3-74

field condition 3-42

field name

- in field specification 3-76, 3-77

- in WHEN clause 3-42

field selection criteria

- padding 3-44

- specifying multiple test values 3-47

- specifying starting and ending columns 3-43

- specifying the starting column 3-43

- testing for blanks 3-47

- using a character string 3-46

- using a column name 3-44

- using a field name 3-45

- using a hexadecimal string 3-46

- with NULLIF clause 3-111

field specification

- format 3-75

- generating data 3-78, 3-79

- identifying the position of a data field 3-80
  - including a delimiter specification 3-106
  - providing the data type length 3-103
  - specifying a column name 3-76
  - specifying a field name 3-76
  - using the DEFAULTIF clause 3-111
  - using the NULLIF clause 3-111
- FIELDS clause
  - AND keyword 3-56
  - BY keyword 3-52
  - ENCLOSED keyword 3-53, 3-55
  - examples 3-54, 3-55
  - OPTIONALLY keyword 3-53, 3-56
  - TERMINATED keyword 3-54
  - WHITESPACE keyword 3-55
- file name, LOB 2-14
- file set, VRAM file 1-8
- File Transfer Protocol (FTP) xiii
- FileTek FTP Data Loader
  - features 1-1
  - how it works with FTP 1-3
- fixed put keyword 4-5
- fixed-length record format 2-4
- FLOAT (REAL) data type 3-95
- FLOAT EXTERNAL data type 3-95
- format conventions 3-3
- formats
  - CHARACTERSET clause 3-20
  - CHARSET keyword 3-105
  - CONCATENATE clause 3-21
  - CONSTANT clause 3-80
  - CONTINUEIF clause 3-22
  - datatype\_spec 3-85
  - DEFAULTIF clause 3-112
  - delimiter\_spec 3-52
  - DIFFERENT SEGMENT clause 3-61
  - DISCARDFILE/DISCARDDN clause 3-16
  - DISCARDS/DISCARDMAX clause 3-18
  - field\_spec 3-75
  - FIELDS clause 3-52
  - HOST keyword 3-107
  - INTO TABLE clause 3-40
  - LOAD INDEX statement 3-141
  - MERGE statement 3-144
  - NULLIF clause 3-111
  - PATH keyword 3-107
  - POSITION clause 3-81
  - PRESERVE BLANKS clause 3-32
  - put command 4-3
  - RECNUM keyword 3-78
  - REPLACE SEGMENT clause 3-67
  - SAME SEGMENT clause 3-61
  - SEGMENT clause 3-61
  - SEQUENCE clause 3-78
  - SUBSPACE number clause 3-69
  - SUBSPACE ROTATE clause 3-35
  - SYSDATE keyword 3-79
  - TRAILING NULLCOLS clause 3-59
  - USER keyword 3-108
  - WHEN clause 3-42
- FTP (file transfer protocol) 1-3
- ftp (UNIX) command 4-9, 4-19
- FTP commands
  - displaying help 4-34
  - server
    - ALLO 4-2
    - HELP 4-2
    - LST 4-10
    - NLST 4-10
    - PASS 4-2
    - PORT 4-2
    - PWD 4-10
    - SITE DRAIN 4-3, 4-15
    - STAT 4-3
    - STOR 4-2
    - SYST 4-3
    - TYPE A 4-2

## Index

---

### G

TYPE I 4-2

USER 4-2

#### user

ascii 4-20

binary 4-20

bye 4-2

close 4-2

open 4-2

put (abort) 4-34

put (check status) 4-26

put (general format) 4-3

put (pipe data) 4-25

put (restart) 4-30, 4-31, 4-33

put (transfer control file) 4-21, 4-23

put (transfer data file) 4-24

quit 4-2

remotehelp 4-2

send 4-3

type image 4-20

FTP session 1-5

## G

generating a sequence of values 3-78

GET command privilege 1-7

globbing remote file names 4-11

GRANT statement 1-9

group name, StorHouse 3-12

#### GUI considerations

displaying remote server messages 4-11

globbing remote file names 4-11

handling client-side errors 4-13

listing the remote directory 4-10

opening a session at an alternate port 4-9

resetting the transfer type to ASCII 4-11

setting a timer to drain the data channel 4-14

specifying keywords for a remote file name 4-10

using automatic binary mode 4-11

using CtrlC 4-13

## H

handling client-side errors 4-13

HASH keyword in SUBSPACE number clause 3-69

HELP FTP command 4-2

hexadecimal strings 3-2

#### hexdigits

in CONTINUEIF clause 3-24

in WHEN clause 3-42

HOST keyword 3-107

## I

IBM S/370 native data type representation 2-5

identifiers, delimited 3-4

identifying the user table to load

overview 3-39

using the fully qualified table name 3-40

IMAGE transfer type 1-5

implied lengths 3-103

in SQL syntax 3-3

#### index

deferred 1-34

file 1-11

loading 3-140

index load operation 3-140

index name 3-141



**INFILE/INDDN clause**

- description 3-10
- loading data from a host data file 3-14
- loading data from a VRAM file 3-12
- loading discarded records 3-13
- NOENVIRON keyword 3-12, 3-13
- StorHouse file name in 3-12

**in-line LOBs 1-11, 3-86, 3-87, 3-89****input data record**

- column 2-6
- delimited 2-7
- field 2-6
- logical 2-7
- physical 2-7

**INSERT database component privilege 1-7****INSERT privilege on user table 1-9****INTEGER data type 3-96****INTEGER EXTERNAL data type 3-96****INTO TABLE clause**

- description 3-39
- example 3-40
- format 3-40
- maximum number 3-39
- multiple 3-39
- owner name 3-40
- table name 3-40

**INTO TABLE specification**

- multiple 3-112
- syntax 3-40

**ISO 8859-1 character set 3-20****K****keywords, LOAD DATA statement**

- AND in FIELDS clause 3-53

**AND in WHEN clause 3-42****BLANKS 3-23****BY in FIELDS clause 3-52****CHAR in FIELDS clause 3-52****CHARACTERSET 3-20****CHARSET in data type specification 3-105****CONCATENATE 3-21****CONSTANT 3-79****CONTINUEIF 3-22****DEFAULTIF in a field specification 3-111****DEFAULTIF in a position\_spec 3-112****DEFAULTIF in FIELDS clause 3-54****DIFFERENT SEGMENT 3-61****DISCARDFILE/DISCARDDN 3-16****DISCARDS/DISCARDMAX 3-18****ENCLOSED in FIELDS clause 3-53****FIELDS 3-52****HASH in SUBSPACE number clause 3-69****HOST in data type specification 3-107****INFILE/INDDN 3-12****INTO TABLE 3-35, 3-37, 3-40****LAST in CONTINUEIF clause 3-23****LOB in SUBSPACE number clause 3-69****NEXT in CONTINUEIF clause 3-23****NOENVIRON in INFILE/INDDN clause 3-12****NULLFLAGS in FIELDS clause 3-52****NULLIF in a field specification 3-111****NULLIF in a position\_spec 3-111****NULLIF in FIELDS clause 3-53****OPTIONALLY in FIELDS clause 3-53****OR in WHEN clause 3-47****PATH in data type specification 3-107****POSITION 3-81****PRESERVE BLANKS 3-32****RECNUM 3-78****REPLACE SEGMENT 3-67, 3-69****SAME SEGMENT 3-60****SEGMENT 3-63****SEQUENCE 3-78****SUBSPACE 3-69****SUBSPACE ROTATE 3-35****SYSDATE 3-79****TABLE in SUBSPACE number clause 3-69**

## Index

### L

- TERMINATED in FIELDS clause 3-52
- THIS in CONTINUEIF clause 3-23
- TRAILING NULLCOLS 3-59
- USER in data type specification 3-108
- VALUE in SUBSPACE NUMBER clause 3-69
- WHEN 3-42
- WHITESPACE in FIELDS clause 3-53
- keywords, LOAD INDEX statement
  - SEGMENTS 3-142
  - SUBSPACE 3-141
  - SUBSPACE ROTATE 3-141
- keywords, MERGE statement
  - EXCLUDE 3-145
  - MAXINSIZE 3-145
  - MINOUTSIZE 3-145
  - SEGMENT 3-144
  - SEGMENTS 3-145
  - SUBSPACE 3-144
  - SUBSPACE ROTATE 3-144
- keywords, put command
  - data\_ccsid 4-5
  - db\_ref 4-4
  - dbname 4-5
  - fixed 4-5
  - load\_type 4-4
  - loadident 4-5
  - nvk 4-6
  - sql\_ccsid 4-5
  - var 4-5
- L**
- large objects (LOBs)
  - BLOB data type 3-86
  - BLOB\_FILE data type 3-87
  - CLOB data type 3-89
  - CLOB\_FILE data type 3-90
  - data files 2-1, 2-9
  - example load 3-135
  - file name 2-14
  - in-line 1-11
  - out-of-line 1-11
  - record 2-10, 3-137
  - specifying a host name 3-107
  - specifying a path name 3-107
  - specifying user information 3-108
  - subsegment file 1-11
- LAST keyword in CONTINUEIF clause 3-23, 3-26
- least significant byte 2-5
- limiting the number of discarded records 3-17
- listing the remote directory 4-10
- little-endian 2-5
- LOAD DATA statement
  - examples
    - all records, one user table 3-117
    - binary and variable-length data 3-123
    - combining records, null values 3-121
    - combining records, one user table 3-119
    - delimited data, multiple user tables 3-120, 5-2
    - LOB data from local LOB files 3-135
    - multiple logical records from one physical record 3-113
    - multiple selection criteria, data in control file 3-129
    - same input data file, multiple user tables 3-114
    - selecting subspaces 3-132
  - general
    - components 3-5
    - summary of clauses and keywords 3-7
  - specifications
    - data\_spec 3-76
    - datatype\_spec 3-76, 3-84
    - field\_spec 3-74
    - into\_table\_spec 3-40
    - position\_spec 3-76
  - syntax
    - CHARACTERSET clause 3-20
    - CONCATENATE clause 3-21

CONSTANT clause 3-79  
 CONTINUEIF clause 3-22  
 DEFAULTIF clause 3-111  
 DIFFERENT SEGMENT clause 3-60  
 DISCARDFILE/DISCARD DDN clause 3-16  
 DISCARDS/DISCARDMAX clause 3-18  
 FIELDS clause 3-52  
 INTO TABLE clause 3-40  
 NULLIF clause 3-111  
 POSITION clause 3-81  
 PRESERVE BLANKS clause 3-32, 3-35  
 REPLACE SEGMENT clause 3-67  
 SAME SEGMENT clause 3-61  
 SEGMENT clause 3-64  
 SEQUENCE clause 3-78  
 SUBSPACE number clause 3-69  
 SUBSPACE ROTATE clause 3-35  
 TRAILING NULLCOLS clause 3-59  
 WHEN clause 3-42

## tasks

collecting discarded records 3-15  
 combining a varied number of records 3-22  
 concatenating a fixed number of records 3-20  
 describing data fields 3-74  
 generating a sequence of values 3-78  
 identifying the user table 3-39  
 including data in the control file 3-115  
 limiting the number of discarded records 3-17  
 loading a constant value 3-79  
 loading a record number 3-78  
 loading missing data fields with null values 3-58  
 loading one or more segments 3-60  
 loading the system date 3-79  
 naming a segment 3-63  
 overriding a character set 3-105  
 preserving blanks 3-31  
 rotating among subspaces 3-33  
 selecting logical records 3-40  
 selecting subspaces 3-68  
 setting a column to a null value 3-111  
 setting a column to the default value 3-112  
 specifying a host name for LOB data 3-107  
 specifying a path name for LOB data 3-107  
 specifying the character set 3-18  
 specifying the position of a data field 3-80  
 specifying user information for LOB data 3-108  
 using data on StorHouse 3-10  
 using multiple INTO TABLE specifications 3-112

load ID 4-5  
 load index operation 1-34  
 LOAD INDEX statement  
   examples 3-142  
   format 3-141  
   guidelines 3-140  
   purpose 3-140  
 load process 1-10  
 load\_type put keyword 4-4  
 loadident put keyword 4-5  
 loading  
   a column with a null value 3-111  
   a constant value 3-79  
   a default value 3-111  
   a deferred index 3-140  
   a record number 3-78  
   all records into one user table 3-118  
   data from a host data file 3-14  
   delimited data into multiple user tables 3-120  
   different tables in multiple loads 1-37  
   different tables in one load 1-36  
   discarded records 3-13  
   LOB data on a client computer 2-10  
   LOB data on a remote host 2-13, 3-107  
   multiple segments 3-60  
   multiple segments of a table in one load 1-37  
   multiple segments of multiple tables in multiple loads 1-38  
   multiple segments of multiple tables in one load 1-37  
   multiple user tables 3-114  
   null values 3-121, 3-125  
   SMALLINT, DECIMAL, and VARCHAR data 3-123

## Index

### M

- some records into one user table 3-119
- the same table in multiple loads 1-38
- the system date 3-79

LOB keyword in SUBSPACE number clause 3-69

local-file argument 4-3

locks 1-39

logging into the StorHouse FTP server 4-18

logging off the StorHouse FTP server 4-35

logical record 2-7

lowercase in SQL syntax 3-3

LST FTP command 4-10

## M

MAXINSIZE clause 3-145

merge operation 1-14, 3-143

MERGE statement

- examples 3-146
- format 3-144
- guidelines 3-143
- purpose 3-143

merging segments 3-143

messages 1-40

Messages and Codes Manual, StorHouse xvi

metadata updates

- data load operation 1-42
- load index operation 1-43
- merge operation 1-44
- replace operation 1-43

MINOUTSIZE clause 3-145

most significant byte 2-5

multiple logical records in one physical record 3-113

## N

naming a segment 3-63

native data type 2-4

native values key 2-5, 4-6

NEXT keyword in CONTINUEIF clause 3-23,  
3-25

NLST FTP command 4-10

NOENVIRON keyword in INFILE/INDDN clause  
3-12

not equal comparison operators 3-31

NOT NULL 3-58

null value

- loading a column with (NULLIF clause) 3-111
- loading missing data fields (TRAILING NULLCOLS  
clause) 3-58

NULLFLAGS keyword 3-52, 3-57

NULLIF clause 3-111

NULLIF keyword 3-53

number, subspace 3-69

nvk put keyword 4-6

nvk, definition 2-5

## O

object identifier 3-86, 3-87, 3-89

OBJECT\_TYPE parameter 1-15

OID 3-86, 3-87, 3-89  
 open FTP command 4-2, 4-19  
 opening a session at an alternate port 4-9  
 operators, comparison 3-23  
 OPTIONALLY keyword in FIELDS clause 3-53  
 OR keyword in WHEN clause 3-47  
 Oracle clauses 3-1  
 out-of-line LOBs 1-11, 3-86, 3-87, 3-89  
 owner  
     in INTO TABLE clause 3-40  
     in REPLACE SEGMENT clause 3-67

## P

padding  
     comparison values 3-28  
     selection criteria 3-44  
 parallelism  
     loading different tables in multiple loads 1-37  
     loading different tables in one load 1-36  
     loading multiple segments of a table in one load 1-37  
     loading multiple segments of multiple tables in multiple loads 1-38  
     loading multiple segments of multiple tables in one load 1-37  
     loading the same table in multiple loads 1-38  
     querying a table while it's being loaded 1-39  
     system parameters 1-35  
 PASS FTP command 4-2  
 password, StorHouse account 1-7  
 PATH keyword 3-107  
 path, LOB data 2-11

PC character set 3-20  
 physical record 2-7  
 piping data 4-3, 4-25  
 PORT FTP command 4-2  
 port, alternate 4-9  
 POSITION clause  
     arguments 3-81  
     description 3-80  
     examples 3-81  
     format 3-81  
 preparing input 1-10, 1-34  
 PRESERVE BLANKS clause  
     example 3-32  
     format 3-32  
 privileges for loading 1-7, 1-9  
 process of a load 1-10  
 program name and arguments when piping 4-25, 4-31  
 put command  
     automating command entry 4-15  
     conventions 4-6  
     description 4-2  
     example 3-66, 4-21, 4-23, 4-33  
     keywords and values 4-4  
     macros 4-16  
     quoting the keyword string 4-11  
     when to use a keyword 4-7  
 PUT command privilege 1-7  
 PWD FTP command 4-10

## Q

qualified SQL identifiers 3-4

quick reference xvii

QUIT FTP command 4-2

quit FTP command 4-2

quoted strings 3-2

quoting a put keyword string 4-11

## R

RECNUM keyword 3-78

RECORD command privilege 1-7

record formats

fixed-length 2-4

text 2-2

variable-length 2-3

record, LOB 2-10, 3-137

relative positioning 3-125

remote directory 4-10

remote file name 4-10

remote host 2-13

remote-file argument 4-3

remotehelp FTP command 4-2

REPLACE SEGMENT clause 3-67

replacing a segment 3-65

replies 1-40

reply code 1-42

reserved words 3-3

RESOURCE privilege 1-8

restarting a load

after shutdown 4-28

description 1-39

guidelines 4-28

macro for 4-17

procedure 4-30

reusing load IDs 4-5

RFC 959 xvi

rotating among subspaces 1-27, 3-33

## S

SAME SEGMENT clause 3-60

sample loading session 4-8

SEGMENT clause 3-64, 3-144

segment ID 1-42

segment list 3-142

segment tag 3-64, 3-67

segmentation 1-11

segments

description 1-11

loading multiple 3-60

merging 3-143

naming 3-63

replacing 3-65

size of 1-13

SEGMENTS clause 3-142, 3-145

selecting subspaces 1-19, 3-68

semicolon, in control file 3-2

send FTP command 4-3

SEQUENCE clause 3-78

session, FTP 1-5

- SETGROUP command privilege 1-7
- setting a timer to drain the data channel 4-14
- setting the transfer type 4-20
- shared lock 1-39
- short record 3-58
- shortword 2-3
- shutdown, restart after 4-28
- SIGPIPE errors 4-14
- silently truncated 3-87
- SITE DRAIN FTP command 4-15
- sizes, segment 1-13
- SMALLINT data type 3-97
- SMALLINT in VAR-type data fields 2-5
- spaces
  - between put keywords and values 4-6
  - in control file 3-2
  - in data 2-9
- SPARC native data type representation 2-5
- specifying a default delimiter
  - describing data fields enclosed by different delimiters 3-56
  - describing data fields enclosed by the same delimiter 3-55
  - describing data fields terminated by blanks 3-55
  - describing data fields terminated with a character 3-54
  - describing data fields that are both terminated and enclosed 3-56
- specifying put keywords
  - for a remote file name 4-10
  - for a target directory 4-10
- specifying the character set of the input data
  - overview 3-19
  - with the data\_ccsid keyword 4-5
- SQL
  - format conventions 3-3
  - identifiers 3-3
  - reserved words 3-3
  - status codes 1-42
- SQL statements
  - CREATE INDEX 1-9, 3-132
  - CREATE TABLE 1-8, 3-132
  - CREATE TABLE SPACE 1-8, 3-132, 3-135
  - GRANT 1-9
- SQL syntax
  - braces { } 3-3
  - commas , 3-3
  - ellipsis points ... 3-3
  - lowercase terms 3-3
  - semicolon 3-2
  - single quotes ' 3-29
  - uppercase terms 3-3
  - vertical bar | 3-3
- sql\_ccsid put keyword 4-5
- SQL\_LDR\_ENGINES system parameter 1-36
- SQL\_LDR\_MAXINTO system parameter 1-36, 3-39, 3-60
- SQL\_LDR\_MAXLOAD system parameter 1-36
- SQL\_SESSIONS system parameter 1-36
- SQLCOMMAND access privilege 1-7
- SQLEXECUTE access privilege 1-7
- status code 1-42
- status information 4-26
- status of a load 4-26
- STOR FTP command 4-2

## Index

### S

- StorHouse
  - account information 1-8
  - group name 3-12, 3-16
  - messages 1-40
  - privileges 1-7
  - product description xiii
  - system parameters 1-36
  - rotating among 1-27, 3-33, 3-141, 3-144
  - selecting 1-19, 3-68
- SunOS commands 4-2
- symbolic name 1-41
- synonym for a table name 3-39
- syntax
  - CHARACTERSET clause 3-20
  - CHARSET keyword 3-105
  - CONCATENATE clause 3-21
  - CONSTANT clause 3-80
  - CONTINUEIF clause 3-22
  - datatype\_spec 3-85
  - DEFAULTIF clause 3-112
  - delimiter\_spec 3-52
  - DIFFERENT SEGMENT clause 3-61
  - DISCARDFILE/DISCARDDN clause 3-16
  - DISCARDS/DISCARDMAX clause 3-18
  - FIELDS clause 3-52
  - INTO TABLE clause 3-40
  - LOAD DATA statement 3-5
  - LOAD INDEX statement 3-141
  - MERGE statement 3-144
  - NULLIF clause 3-111
  - PATH keyword 3-107
  - POSITION clause 3-81
  - PRESERVE BLANKS clause 3-32
  - put command 4-3
  - RECNUM keyword 3-78
  - REPLACE SEGMENT clause 3-67
  - SAME SEGMENT clause 3-61
  - SEGMENT clause 3-64
  - SEQUENCE clause 3-78
  - SQL statements 3-3
  - SUBSPACE number clause 3-69
  - SUBSPACE ROTATE clause 3-35
  - SYSDATE keyword 3-79
  - TRAILING NULLCOLS clause 3-59
  - USER clause 3-108
  - WHEN clause 3-42
- SYSDATE keyword 3-79
- StorHouse Database Administration Guide xvii
- StorHouse documentation
  - Concepts and Facilities Manual xvii
  - Messages and Codes Manual xvi
- StorHouse FTP server
  - description 1-4
  - logging into 4-18
- StorHouse system administrator 1-8
- StorHouse/RM code 1-42
- StorHouse/RM Concepts xvi
- StorHouse/RM, description xiv
- StorHouse/SM code 1-42
- StorHouse/SM, description xiii
- strings 3-2
- subsegment file 1-11
- SUBSPACE number clause
  - description 3-68, 3-141, 3-144
  - examples 3-69
  - format 3-69
- SUBSPACE ROTATE clause
  - description 1-27, 3-33, 3-141, 3-144
  - examples 3-35
  - format 3-35
- subspaces
  - default 1-15
  - definition 1-14
  - number 3-69



SYSINDEXES system table 1-43  
 SYSSTHFILES system table 1-42, 1-43, 1-44  
 SYSSTHSEGMENTS system table 1-42  
 SYSTABLES system table 1-42  
 system parameters  
     SQL\_LDR\_ENGINES 1-36  
     SQL\_LDR\_MAXINTO 1-36, 3-39, 3-60  
     SQL\_LDR\_MAXLOAD 1-36  
     SQL\_SESSIONS 1-36  
 system table updates 1-42  
 SYSTHSEGMENTS system table 1-43

## T

table ID 1-42  
 TABLE keyword in SUBSPACE number clause 3-69  
 table name  
     case 3-3  
     in INTO TABLE clause 3-40  
     in MERGE statement 3-144  
     in REPLACE SEGMENT clause 3-67  
 tables  
     CHARACTERSET clause 3-20  
     CONCATENATE clause 3-21  
     CONSTANT clause 3-80  
     CONTINUEIF clause 3-23  
     data type specifications 3-84  
     DISCARDFILE/DISCARDDN clause 3-16  
     DISCARDS/DISCARDMAX clause 3-18  
     field specification 3-76, 3-78, 3-80  
     FIELDS clause 3-52  
     INFILE/INDDN clause 3-12  
     INTO TABLE clause 3-40  
     LOAD DATA format 3-5  
     LOAD INDEX arguments 3-141  
     LOAD INDEX format 3-9  
     MERGE arguments 3-144  
     MERGE format 3-10  
     POSITION clause 3-81  
     REPLACE SEGMENT clause 3-67, 3-69  
     SEGMENT clause 3-64  
     SEQUENCE clause 3-78  
     SQL format conventions 3-3  
     StorHouse tasks 1-8  
     SUBSPACE number clause 3-69  
     summary of LOAD DATA clauses 3-7  
     WHEN clause 3-42  
 target directory 4-10  
 TCP/IP 1-4  
 terminated data 2-8  
 TERMINATED keyword in FIELDS clause 3-52  
 termination delimiter 2-8  
 text data 2-2  
 THIS keyword in CONTINUEIF clause 3-23, 3-24  
 TIME EXTERNAL data type 3-97  
 TIMESTAMP EXTERNAL data type 3-98  
 TRAILING NULLCOLS clause  
     example 3-59  
     format 3-59  
 transfer modes 1-5  
 transfer type  
     ASCII 1-5  
     BINARY 1-5  
     IMAGE 1-5  
     resetting 4-11  
     setting 4-20  
 transferring  
     control file 4-22  
     data file 4-24

## Index

---

### U

transferring files 1-10, 1-34

#### trimming

- comparison values 3-28
- data fields 3-105
- leading blanks 2-9
- selection criteria 3-44
- trailing blanks 2-9

TYPE A FTP command 4-2

type FTP command 4-2

TYPE I FTP command 4-2

type image FTP command 4-20

### U

uppercase in SQL syntax 3-3

USER FTP command 4-2

USER keyword 3-108

#### using data on StorHouse

- loading data in a VRAM file 3-12
- loading discarded records 3-13
- overview 3-10

### V

validate load\_type 4-21

validating the data file or control file 4-21

VALUE keyword in SUBSPACE number clause 3-69

var put keyword 4-5

VARBINARY data type 3-99, 3-106

VARCHAR data type

- actual length 2-5

- as selection criteria 3-46

- conversion 2-7

- description 3-100

- maximum length 3-103

- starting column 3-80

variable-length record format 2-3

VAR-type data type considerations 2-5

vertical bar in SQL syntax 3-3

volume set, VRAM file 1-8

VRAM file 1-8, 2-1

VTF command privilege 1-7

### W

#### WHEN clause

- charstring 3-42
- column name 3-42
- description 3-40
- field name 3-42
- format 3-42
- hexdigits 3-42
- options 3-41

whitespace 2-9

WHITESPACE keyword in FIELDS clause 3-53

### X

XLDINFO replies 1-41

XLEOFTERM error 4-14