TSnosql

by

Charles E. Wegrzyn

Twisted Storage, Inc.

back cover

1 Introduction

Very simply TSnosql is a key-value database. *Keys* are simple strings that are used to represent the specific value. The *value* can take one of three types: strings, lists or sets. All operations submitted to the server are done atomically – the operation of one client request will not interfere with the operation of another client.

In addition to the three different data types, TSnosql also supports the notion of multiple spaces within the running server. Each space is given a client assigned name, and can contain key-value pairs that are different from other spaces.

Unlike the *memcached* system, TSnosql allows the entire key-value data to be backed up to or read from a disk file. Writing data to a file allows it to be recovered at a later time when the server is re-run. The TSnosql sever provides commands to write the backup file or read it on demand.

Also unlike the *memcached* system, TSnosql supports multiple slaves connected to a master. One of the TSnosql servers is selected as a master, which can read values and change or add new key-values. Other TSnosql can be connected to the master and become slaves. As slaves they will keep an exact copy of the state of the master. This allows a convenient way to provide distributed, overloaded services.

Finally TSnosql is open source and covered by the GNU license. While the software is copyrighted by Twisted Storage it will always be openly available.

1.1 Name spaces

The TSnosql server supports multiple name spaces. Each of the independent name spaces can hold key-value pairs which are unknown to the other spaces. Spaces can be given any simple name, which are sequences of alpha-numeric characters. Thus a name can be 0. Another name can be SpaceToHoldMyKeys. Note that the case of a character is significant: a is different from A!

There is no limit to the size of the name space name, and there is no limit to the number of spaces available.

There is one default name space in every TSnosql server: 0. If the client doesn't select a particular name space to use, the name space 0 is always used.

1.2 Keys

A key is a simple string of alpha-numeric characters used to identify a specific value. The keys in one name space are unique from the keys in another name space.

As in the case of name space names, key names are case sensitive. Hence FOO is different from Foo or foo.

There is no limit to the size of a key string nor is there any hard coded number of keys that can be stored in a name space.

1.3 Values

Values stored in the TSnosql server can be strings, lists or sets. An individual item in a list or set or a string can be upto 1MB long. The individual item can be either a string – ASCII or Unicode – or any binary data. The data stored in the TSnosql is taken from the data passed to the TSnosql server and used unprocessed or converted. This means it is possible to store image data directly in the server.

2 TSnosql Commands

2.1 Server Commands

The following TSnosql commands control the operation of the server:

Command	Function
Select	Select a name space
Save	Save the database to disk
Quit	Close the client connection to the server
Ping	See if the server connection is active
BGSave	Save the database to disk
DBSize	Number of keys in selected name space
FlushAll	Clear out all name spaces
FlushDB	Clear out selected name space
Slave	Connect slave to master server
Info	Return information about server
Shutdown	Shutdown server operation
Last	Time of last database save
Auth	Authorize client access to server

2.2 Variable Commands

The following commands affect strings, lists or sets:

Command	Function
Keys	Return number of keys that match glob pattern
Re	Return number of keys that match regexp patter

Exist	Does a key exist in the name space?
Type	Return the type of value for name
TTL	Return time to live for key-value
Expire	Set time to live timer for key-value
Rename	Rename a key to a new name
Renx	Conditional key rename
Move	Move key-value to new name space
Delete	Delete one or more key-values

2.3 String Commands

These TSnosql commands affect string key-values:

Command	Function
Get	Find Value to key and return it
Set	Set (new) value to key
Setnx	Conditional set value
GetSet	Atomic get-set key-value
Increment	Increment a key's value by 1
Increment by	Increment a key's value by some value
Decrement	Decrement a key's value by 1
Decrement by	Decrement a key's value by some value

2.4 List Commands

The following commands affect list values:

Command	Function
Lpush	Create list, add element to head
Rpush	Create list, add element to tail
Llen	Return length of list
Lrange	Return range of list
Ltrim	Trim a list
Lindex	Return indexed item from list
Lset	Set an indexed item in a list

Command	Function
Lrem	Remove an indexed item in a list
Lpop	Pop off the head of a list
Rpop	Pop off the tail of a list

2.5 Set Commands

The following commands work with set values:

Command	Function
Sadd	Create set, add member to set
Srem	Remove a member from a set
Spop	Pop a random member from a set
Smove	Move member from one set to another
Scard	Number of members in a set
Sismember	Is member in set
Smembers	Return set
Sinter	Return intersection of two or more sets
Sinterstore	Compute intersection of sets and store
Sunion	Return union of two or more sets
Sunionstore	Compute union of sets and store
Sdiff	Return difference or two or more sets
Sdiffstore	Compute difference of sets and store

3 TSnosql Protocol

3.1 Client – Server Exchange

Communicating with the TSnosql server is done through the exchange of ASCII text messages. Every exchange begins with a client sending a command to the server followed by the server sending back a response. Every command and the correspondining response terminates with an ending carriage return and line feed.

Commands to the server come in three variations. The first form, a simple command, contains only the ASCII name of the command followed by a carriage return and a line feed. An example is

```
PING\r\n
```

The second form is also a simple command containing the ASCII name of a command along with an ASCII argument followed by a carriage return and a line feed. An example of this command style is

```
EXISTS key\r\n
```

The third form of the command is used to send bulk, uninterpretted content to the server. A sample command of this kind is

```
SET key 5\r\nSNAFU\r\n
```

The number 5 is the number of bytes in the data following the initial carriage return and line feed but not including the trailing the carriage return and line feed.

Every command sent to the server will result in a reply being sent to the client. Replies can take one of number of different formats. The first type of reply is a single ASCII text reply; it has the form

```
+This is a single line of text\r\n
```

The meaning of the text in the reply is dependent on the command and there will not be any embedded carriage return or line feed characters in the text.

The second type of reply is an integer, which has the form

```
:22\r\n
```

Once again the meaning of the integer reply depends on the command.

A command might return some bulk, uninterpreted data to the client. In this case the reply has the form

```
$64\r\nThis data may or may not be ASCII but will be 64 characters long\r\n
```

Note the length does not include the terminating carriage return and line feed. Also the returned data is not interpreted, meaning that is can be anything including binary, Unicode, etc. If the integer after the $ character is -1 it means there is no data (this represents the NULL case).

Some commands might return zero or more bulk data values. If this is the case the reply will begin with

```
*NN\r\n
```

where NN are the number of bulk data items sent; this value can be -1 if none are returned.

Finally executing a command might return an error. All error replies in TSnosql have the form

```
-ERR NNNN This is an error message\r\n
```

The meaning of the error message is dependent on the command being executed. The value NNNN is a number in ASCII form that identifies the error message. If there is text in the message it will be returned in U.S. English.

3.2 TSnosql Commands

The following are the commands supported in version 1.0 of the TSnosql server.

3.2.1 SELECT: Select default database table

All commands that affect a key will use the named database table. If the database table doesn't exist it will be created empty.

To Server	SELECT <DBName>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.2 GET: Get string variable

Return the value of the specified key but only if the key is a string. If the key represents a list or set an error is returned. If the key doesn't exist the server will return the $-1 value.

To Server	GET <key>\r\n
From Server	$<len>\r\n <value of length 'len' bytes>\r\n
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.3 SET: Set string variable

This command will result in the named key being tagged as a string variable with the supplied value.

To Server	SET <key> <len>\r\n <value of length 'len' bytes>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.4 SETNX: SET key if it doesn't exist

This command will set the value of a string key variable but only if the key doesn't already exist. If the key was created, the server will return :1. Otherwise it returns :0.

To Server	SETNX <key> <len>\r\n <value of length 'len' bytes>\r\n
From Server	:1\r\n

or	:0\r\n
or	-ERR <msg>\r\n

3.2.5 MGET string variables

Return the values of the supplied keys. Any key that doesn't exist will return a $-1 for its value (which should be interpretted as NIL).

To Server	MGET <key1> ... <keyN>\r\n
From Server	*<number N>\r\n <Entry1>\r\n ...\r\n <EntryN>\r\n Where each <EntryN> is $<len>\r\n<value of length 'len' bytes> or $-1 if the variable doesn't exist.
or	-ERR <msg>\r\n

3.2.6 GETSET string variable

This command will return the original value of the named variable and setting it to a new value. This is done as an atomic operation.

To Server	GETSET <varName> <newValueLen>\r\n <new value of length 'newValueLen'>\r\n
From Server	$<len>\r\n <original value of length 'len' bytes>\r\n
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.7 DELETE string variables

Delete one or more string variables from the selected database.

To Server	DELETE <varName1>…<varNameN>\r\n
From Server	:<count>\r\n
or	-ERR <msg>\r\n

where <count> is the number of successfully deleted variables, which should equal N.

3.2.8 INCREMENT BY string variable

Increment the named variable by an integer amount. If the named variable doesn't exist it is assumed to be a value of 0. If the named variable can't be interpreted as an integer an error is returned to the client.

To Server	INCRBY <varName> <incrValue>\r\n
From Server	:<newValue>\r\n
or	-ERR <msg>\r\n

3.2.9 INCREMENT string variable

See INCREMENT BY with an incrValue of 1.

3.2.10 DECREMENT BY string variable

Decrement the named variable by an integer amount. If the named variable doesn't exist it is assumed to be a value of 0. If the named variable can't be interpreted as an integer an error is returned to the client.

To Server	DECRBY <varName> <incrValue>\r\n
From Server	:<newValue>\r\n
or	-ERR <msg>\r\n

3.2.11 DECREMENT string variable

See DECREMENT BY with an decrValue of 1.

3.2.12 SAVE database to file

This command will force the server to dump the complete database to disk. The file is named rdump.db and stored in the current working directory of the server.

To Server	SAVE\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.13 QUIT the server

Sending the QUIT command to the server will close the connection with the client.

To Server	QUIT\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.14 PING server to see if alive

The PING command will elicit a response from the server if it is running.

To Server	PING\r\n
From Server	+PONG\r\n

3.2.15 KEYS: Return matched keys

This command will search the currently selected database for keys with with certain names. The key has the "glob" format, meaning that * is replaced with zero or more characters.

To Server	KEYS <globName>\r\n
From Server	$<len>\r\n <string of length 'len' characters>\r\n
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.16 RE: Return matched keys using RegExp

Similar to KEYS except using the RegExp matcher instead of the Glob matcher.

To Server	KEYS <regExpName>\r\n

From Server	$<len>\r\n <string of length 'len' characters>\r\n
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.17 EXIST: Does a variable exist?

Look in the currently selected database for given variable and return an indication if it exists.

To Server	EXISTS <varName>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.18 RENAME: Rename a variable

Rename a variable to a new name. If the new name is the same as the old name or the variable doesn't exist an error is returned.

To Server	RENAME <origName> <newName>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.19 RENX: Rename variable if exists

Similar to RENAME but it returns a 1 if the operation succeeded or a 0 if it failed.

To Server	RENX <origName> <newName>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.20 MOVE: Move variable

This command will move a variable in the currently selected database to a new database. The variable must exist in the source database and must not exist in the target database. If the key was moved, the server will return :1. Otherwise it will return :0.

To Server	MOVE <varName> <newDB>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.21 BGSAVE: Save database tables

Save the databases in the background. In TSnosql this is always the case.

To Server	BGSAVE\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.22 DBSIZE: Return size of database table

This command will return the number of variables in the database.

To Server	DBSIZE\r\n
From Server	:<NoOfVariables>\r\n
or	-ERR <msg>\r\n

3.2.23 FLUSHALL: Clear all database tables

Sending this command to the server will result in all the database tables being cleared.

To Server	FLUSHALL\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.24 FLUSHDB: Clear current database table

To clear out the currently selected database table issue the FLUSHDB command.

To Server	FLUSHDB\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.25 TYPE: Return the type of a variable

This command will return the type of a variable stored in the currently selected database table.

To Server	TYPE <varName>\r\n
From Server	+none\r\n
or	+string\r\n
or	+list\r\n
or	+set\r\n
or	-ERR <msg>\rn

3.2.26 EXPIRE: Set a timeout value on a variable

This command will set a time to live on an existing variable. Setting the expiration value to N seconds will cause the variable to be removed from the currently selected database table. If the database is saved (see BGFUNC or SAVE) and the server is stopped before the expiration occurs, when the server is restarted the variable will be removed (if the expiration has passed).

To Server	EXPIRE <varName> <secondsToLive>\r\n
From Server	:<secondsToLive>\r\n
or	-ERR <msg>\r\n

3.2.27 INFO: Return info about the server

Return information about the server.

To Server	INFO\r\n

From Server	$<len>\r\n <string of length 'len' characters>\r\n
or	-ERR <msg>\r\n

3.2.28 SHUTDOWN: Shutdown the server

This command will terminate the server, absolutely.

To Server	SHUTDOWN\r\n
From Server	If successful, connection closed
or	-ERR <msg>\r\n

3.2.29 LAST: Return the last time of a save

This command will return the Unix time of the last database save.

To Server	LASTSAVE\r\n
From Server	:<lastSaveTime>\r\n
or	-ERR <msg>\r\n

3.2.30 AUTH: Authorize user

If the server requires that users of the system be authorized, this command will provide that information to the server.

To Server	AUTH <password>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.31 TTL: Time To Live function

See if a variable has an expiration time out on it.

To Server	TTL <varName>\r\n
From Server	:1\r\n
or	:0\r\n

or	-ERR <msg>\r\n

3.2.32 SLAVE: Assign a slave server

This command will connect another TSnosql as a slave to this one.

To Server	SLAVE <ip> <port>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.33 LPUSH: Create new list/add element

This command will create a new list, if it doesn't exist already. Add the element to the head of the list.

To Server	LPUSH <varName> <element>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.34 RPUSH: Create new list/add element

This command will create a new list, if it doesn't exist. The element will be added to the end of the list.

To Server	RPUSH <varName> <element>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.35 LLEN: Return the length of the list

This command, when sent to the server, will result in the length of the list be returned to the client.

To Server	LLEN <varName>\r\n
From Server	:<No of elements in list>\r\n
or	-ERR <msg>\r\n

3.2.36 LRANGE: Return a subset of a list

This command will return a subset of a list. The list range is defined by two integers and is considered inclusive.

To Server	LRANGE <varName> <from> <to>\r\n
From Server	*<cnt>\r\n<elem1>\r\n…<elemN>\r\n Where each <elem> is $<len>\r\n<value of length 'len' bytes>
or	-ERR <msg>\r\n

3.2.37 LTRIM: Trim a list

A list variable will be trimmed to a specific subset of list elements.

To Server	LTRIM <varName> <from> <to>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.38 LINDEX: Return an element of a list

This command will return a specific element of a list. Index values can take both positive and negative values. Positive values start at the beginning of the list and negative values start at the end of the list. The positive values begin at 0, which is the first element of the list, and continue to N-1, the last element of the list. The negative values begin at -1 and continue to -N, with -1 being the last element of the list, etc.

If the key isn't a list, an error will be returned. If the indexed element doesn't exist the server will return $-1.

To Server	LINDEX <key> <index>\r\n
From Server	$<len>\r\n <value of length 'len' bytes>\r\n
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.39 LSET: Set an element of a list

This command will set an element of list to some value. See LINDEX for information on index values and their meaning. If the index is out of range an error will be generated.

To Server	LSET <key> <index> <len>\r\n <value of length 'len' bytes>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.40 LREM: Remove elements from a list

To remove *count* occurrences of a *value* from a list issue the LREM command. If *count* is 0, all occurrences will be removed. If *count* is negative, the occurrences are removed from the tail to the head otherwise they are removed from the head to the tail.

To Server	LREM <key> <count> <len>\r\n <value of length 'len' bytes>\r\n
From Server	:<no. elements removed>\r\n
or	-ERR <msg>\r\n

3.2.41 LPOP: Pop off first element from list

Remove and return the first element from a list. If the list is empty the server will return $-1.

To Server	LPOP <key>\r\n
From Server	$<len>\r\n <value of length 'len' bytes>\r\n
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.42 RPOP: Pop off the last element from list

Remove and return the last element from a list. If the list is empty the server will return $-1.

To Server	RPOP <key>\r\n
From Server	$<len>\r\n <value of length 'len' bytes>\r\n

or	$-1\r\n
or	-ERR <msg>\r\n

3.2.43 SADD: Add a member to a set

Add the *member* to the set *key* if the member doesn't already exist in the set. If the set *key* doesn't exist, it is initially created as empty before performing the add.

If the *key* exists, and isn't a set an error is returned to the client. If the member is added to the set the server returns :1. Otherwise it returns :0.

To Server	SADD <key> <len>\r\n <value of length 'len' bytes>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.44 SREM: Remove a member from a set

Remove the specified *member* from the given set, *key*. If the *member* doesn't exist in the set, the operation is a no op.

If *key* doesn't exist or isn't a set an error is returned. If the member is in the set, the server will return :1. Otherwise it will return :0

To Server	SREM <key> <len>\r\n <value of length 'len' bytes>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.45 SPOP: Pop random member

Pop a random member from a set and return it. If *key* doesn't exist or is the empty set the server will return $-1.

If *key* isn't a set, an error is returned.

To Server	SPOP <key>\r\n
From Server	$<len>\r\n <value of length 'len' bytes>\r\n

or	$-1\r\n
or	-ERR <msg>\r\n

3.2.46 SMOVE: Move set

This command will move the *member* from the set *src* to the set *dest*. If the source set *src* doesn't exist or doesn't contain *member* the server will return :0. Otherwise it will return :1.

If *src* or *dest* aren't sets, the server will return an error.

To Server	SMOVE <src> <dest> <len>\r\n <member of length 'len' bytes>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.47 SCARD: Return the cardinality

Returns the number of elements (cardinality) in a set *key*.

If *key* isn't a set an error is returned.

To Server	SCARD <key>\r\n
From Server	$<cardinality>\r\n
or	-ERR <msg>\r\n

3.2.48 SISMEMBER: Is member of set?

Return :1 if *member* is in the set *key*. Otherwise return :0.

If set *key* isn't a set, return an error.

To Server	SISMEMBER <key> <len>\r\n <member of length 'len' bytes>\r\n
From Server	:1\r\n
or	:0\r\n
or	-ERR <msg>\r\n

3.2.49 SINTER: Intersection of sets

This command will take the intersection of any number of sets and return the results to the client. A non-existing set will be considered empty with the result of forcing the result to be empty.

If any *key* isn't a set, an error will be returned.

To Server	SINTER <key1> <key2> … <keyN>\r\n
From Server	*<cnt>\r\n<elem1>\r\n…<elemN>\r\n Where each <elem> is $<len>\r\n<value of length 'len' bytes>
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.50 SINTERSTORE: Compute intersection

This command will compute the intersection of one or more sets, and store the resulting set in another variable. It is like SINTER buts doesn't return the resulting set.

To Server	SINTERSTORE <dest> <key1> <key2> … <keyN>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.51 SUNION: Union of sets

This operation will take the union of any number of sets and return the resulting set to the client. Any argument that doesn't exist is taken to be an empty set. Any argument that isn't a set will result in the operation returning an error.

To Server	SUNION <key1> <key2> … <keyN>\r\n
From Server	*<cnt>\r\n<elem1>\r\n…<elemN>\r\n Where each <elem> is $<len>\r\n<value of length 'len' bytes>
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.52 SUNIONSTORE: Compute union

This command will compute the union of any number of sets and store the resulting set into a new key, *dest*. Any *key* that doesn't exist is taken as an empty set. Any *key* that isn't a set will result in an error being returned.

To Server	SUNIONSTORE <dest> <key1> <key2> … <keyN>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.53 SDIFF: Difference of sets

This command will generate the difference between the first set, *key1*, and the remaining sets, *keyN*. Any key that doesn't exist is assumed to be an empty set. Any key that isn't a set will result in an error being returned by the operation.

To Server	SDIFF <key1> <key2> … <keyN>\r\n
From Server	*<cnt>\r\n<elem1>\r\n…<elemN>\r\n Where each <elem> is $<len>\r\n<value of length 'len' bytes>
or	$-1\r\n
or	-ERR <msg>\r\n

3.2.54 SDIFFSTORE: Compute difference

This command will compute the difference between the first set, *key1*, and the remaining sets, *keyN* and store the resulting set in the destination key, *dest*. Any key that doesn't exist is assumed to be an empty set. Any key that isn't a set will result in an error being returned.

To Server	SDIFFSTORE <dest> <key1> <key2> … <keyN>\r\n
From Server	+OK\r\n
or	-ERR <msg>\r\n

3.2.55 SMEMBERS: Return the members of a set

This command will return all the members of a set, *key*. If *key* isn't a set, this command will return an error. A non-existant *key* will return an empty set, $-1.

To Server	SMEMBERS <key>\r\n
From Server	*<cnt>\r\n<elem1>\r\n...<elemN>\r\n Where each <elem> is $<len>\r\n<value of length 'len' bytes>
or	$-1\r\n
or	-ERR <msg>\r\n

4 Running TSnosql

Running TSnosql is as easy as:

python Tsnosql.py

The following command line options can be used:

Option	Meaning
-h, --help	Return help information
-d, --debug	Turn on debug logging
-H <dir>, --home=<dir>	Home directory for TSnosql
-n <ip>, --nosql=<ip>	IP address and port for NoSql cmds
-i <file>, --initfile=<file>	ASCII initial contents file
-m, -master`	Run server as Master
-r <file>, --rdump=<file>	Binary dump file

In the current working directory if a file called '.redis.cfg' is present, it will be checked to find specific configuration. This file has the general format of an INI with sections and variables.

If there is a section named “security” and a variable named “auth”, it indicates all clients must be authenticated before access is allowed. The value of the “auth” variable is the authentication value (see the AUTH command above).

If there is a section named “general” and a variable named “updateCount” is found it is the number of updates against the name spaces before the key-values are written to the disk file. If no updateCount value is provided, a default is set to 10.

Two options, -i and -r control the initial contents of the server. The -r option is used to specify the name of the bininary dump file that is used by the server; the default is Tsnosql-rdump.db. The -i option is used to provide the name of an ASCII initialization file; the default name is MDB.TXT.

When the server starts it will look for the binary dump file (see the -r or –rdump option). If this file exists it is taken to be the binary contents of the key-value database from a previous running of the server.

If the binary dump file doesn't exist, the server will look for the ASCII initialization file (see the -i or –initfile option). If this file exists it is an INI file and the contents are used to set the initial contents of the key-value store.

If both the binary dump file and ASCII initialization file do not exist, the key-value store of the server is set to empty.

5 Ascii Initialization File Sample

The following shows an example MDB.TXT file:

```
#  This is an example line...
#  Some more comments.
#
#  Notice critical.IP is a string with the value of localhost and critical.PORT is a string with
#  value of 10000. The values are uninterpreted and will be returned as strings to the client
#  programs.
#
#  The space between the key and value is ignored – whitespace is merged.
critical.IP          localhost
critical.PORT        10000

# General information for the data transfer support services.
#
FTP.base             /tmp
FTP.port             10004
HTTP.port            10005
FTP.login            freestore
FTP.password         freestore

# Read a file called CONFIG.TXT followed by reading a file called FOO.BAR
@include CONFIG.TXT
@include FOO.BAR
```