TDS 5.0 Functional Specification

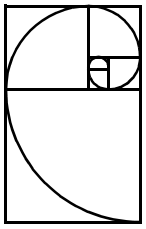
Version 3.8

Sybase Confidential

00000-01-00000-00

Last Revised January 2006

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.



Introduction

1. Overview

The Tabular Data Stream (TDS) is an application level protocol used to send requests and responses between clients and servers. A client's request may contain multiple commands. The response from the server may return one or many result sets.

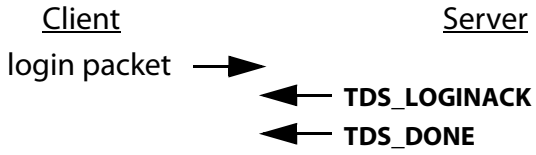
TDS relies on a connection oriented transport service. Session, presentation, and application service elements are provided by TDS. TDS does not require any specific transport provider. It can be implemented over multiple transport protocols if they provide connection oriented service.

TDS provides support for login capability negotiation, authentication services, and support for both database specific and generic client commands. Responses to client commands are returned using a self-describing, table oriented protocol. Column name and data type information is returned to the client before the actual data.

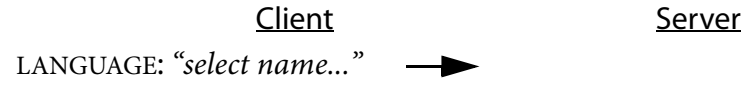
For example, here is a high-level description of the TDS tokens exchanged by a client and a server to establish a dialog and then execute a simple SQL query. The SQL statement is, "select *name from sysobjects where id* < 3". This query causes two table rows to be returned to the client.

The client first requests a transport connection to the server and then sends a login record to establish a dialog. The login record contains capability and authentication information.

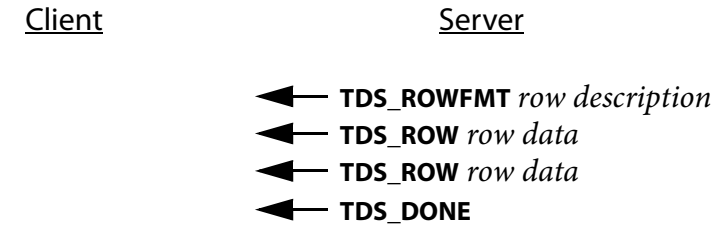
The server responds with a acknowledgment token followed by a completion token indicating that it has accepted the dialog request.



Now that a dialog has been established between the client and the server, the client sends the SQL query to the server and then waits for the server to respond.



The server executes the query and returns the results to the client. First the data columns are described by the server, followed by the actual row data. A completion token follows the row data indicating that all row data associated with the query has been returned to the client.



The TDS PDUs are described in *TDS 5.0 Reference Pages*.

The TDS protocol is mostly a token based protocol where the contents of a Protocol Data Unit (PDU) are tokenized. The token and its data stream describe a particular command or part of a result set returned to a client.

For example, there is a token called **TDS_LANGUAGE** which is used by a client to send language, typically SQL, commands to a server. There is also a token called **TDS_ROWFMT** which describes the column name, status, and data type which is used by a server to return column format information to a client.

2. Protocol Data Units

A TDS request or response may span multiple PDUs. The size of the PDU sent over the transport connection is negotiated at dialog establishment time. Each PDU contains a header, which is usually followed by data.

2.1. Protocol Data Unit Header

A PDU header contains information about the size and contents of the PDU as well as an indication if it is the last PDU in a request or response. The format of a TDS PDU is described in detail in the Message Buffer Header reference page. The TDS protocol is half-duplex. A client writes a complete request and then reads a complete response from the server. Requests and responses cannot be intermixed and multiple requests cannot be outstanding.

2.2. Protocol Data Unit Data

In addition to a header, PDUs usually include some data. Control PDUs do not contain any other data. They consist of a header only. Requests and response PDUs contain TDS tokens that describe the request or response.

3. Client Protocol Data Units

PDUs sent from a client to a server can contain the following data:

- Dialog establishment information
- Language command
- Cursor command
- Database Remote Procedure Call
- Attentions
- Dynamic SQL command
- Message command

3.1. Dialog Establishment

To establish a dialog with a server a client must:

- Create a transport connection
- Send a login record
- Send a capability data stream
- Perform any required authentication handshaking
- Read the login acknowledgment

A client application may have multiple dialogs established with the same or multiple servers, but this is transparent to the TDS protocol. All of the steps above must be completed for each active dialog supported by a client application.

3.2. Language Commands

The **TDS_LANGUAGE** token is used to send language commands to a server. When a client is communicating with a SQL Server, this language is a SQL command. A language command may span multiple PDUs, but its total length is limited by the length field in the **TDS_LANGUAGE** token. See TDS_LANGUAGE on page 233 for details.

The character set that the language command is sent in is negotiated during dialog establishment. The server will perform any required character set translations as required.

3.3. Cursor Commands

There are two ways to send cursor commands to a server:

- Language commands
- Cursor TDS tokens

Cursor commands can be sent to a server using the **TDS_LANGUAGE** token and the SQL dialect as described above. However, this requires the server to parse the language to implement the requested cursor operation.

TDS also provides native token support for all ANSI specified cursor operations. This provides a more efficient mechanism for sending cursor commands to a server since it eliminates the parsing step. It also allows servers built using the Open Server product to implement cursor emulation on top foreign data sources without implementing a parser.

A complete description of the cursor tokens is in **TDS_CUR*** reference pages.

3.4. Database Remote Procedure Calls (RPC)

To execute a remote procedure call on the server, the client sends a **TDS_RPC** data stream to the server. This is a binary stream that contains the RPC name, options, and parameters. Each RPC must be in a separate message and not intermixed with SQL commands or other RPC commands. For a detailed description of the RPC request data stream (page 293).

COMMENTS:Need to rewrite to reflect change in RPC protocol

3.5. Attentions

The client can cancel the current request by sending an attention to the server. Once the client sends an attention, the client reads until it gets an attention acknowledgment. After sending an at-

tention to a server the client will discard any data received until it receives an attention acknowledgment.

TDS 5.0 attentions are sent using the non-expedited data transfer service provided by the transport provider. Earlier versions of TDS sent attentions using the expedited data transfer service if it was provided by the transport provider.

Expedited attentions will still be supported by clients and servers that implement 5.0 TDS so that they can continue to communicate with earlier versions of TDS.

3.6. Dynamic SQL Commands

To execute dynamic sql on the server, the client sends a **TDS_DYNAMIC** or **TDS_DYNAMIC2** data stream to the server to prepare and/or execute the dynamic sql. TDS_DYNAMIC2 has a four (4) byte length while TDS_DYNAMIC has a two (2) byte length field. The binary stream indicates the type of operation (prepare, execute, dealloc, etc.), whether there are arguments to the sql, the statement identification, and the sql statement. For more information, see **TDS_DYNAMIC** (page 205)

3.7. Message Commands

4. Server Protocol Data Units

PDUs sent from a server to a client can contain the following data:

- Dialog establishment acknowledgment
- Row results
- Return status
- Return parameters
- Response completion
- Error information
- Attention acknowledgment
- Cursor status
- Message responses

4.1. Dialog Establishment Acknowledgment

The acknowledgment to a dialog establishment request is a token stream consisting of information about a server's characteristics, informational messages and a completion indication. There are optionally authentication handshake messages.

The **TDS_CAPABILITY**, **TDS_LOGINACK**, and **TDS_DONE** tokens are used to communicate information to the client regarding the dialog establishment request.

If there are any information messages in the dialog response, an **TDS_EED** data stream is returned from the server to the client.

A **TDS_DONE** token is always sent to terminate the dialog establishment response.

4.2. Row Results

If a client request results in data being returned, the data will precede any other data streams returned from the server. Row data is always preceded by a description of the column names and data types. For a detailed description of the data stream see the reference pages for **TDS_ROWFMT**, and **TDS_ROW**.

4.3. Return Status

A return status can be returned in response to a client command. A return status is returned to a client using the **TDS_RETURNSTATUS** token.

4.4. Return Parameters

Return parameters can be sent to a client in response to either a language or RPC command. Return parameters are returned to a client using the **TDS_PARAMFMT** and **TDS_PARAM** tokens.

When a RPC is invoked, some or all of it's parameters may be designated as output parameters. This allows RPC parameters to act like variables that are passed by reference. All output parameters will have values returned by the server.

Return parameters can also be returned to a client in response to a language command. This is the normal case for stored procedures on a SQL Server. If the stored procedure is executed via a language command, any parameters designated as output parameters are returned using the TDS **TDS_PARAMFMT** and **TDS_PARAM** tokens.

4.5. Response Completion

The end of a server response can be determined using the TDS PDU header length field. However, the **DONE** token is used to report command completion.

When executing a language command that contains a batch of SQL commands, there will be a **TDS_DONE** data stream for each set of results. All but the last **TDS_DONE** will have the **TDS_DONE_MORE** bit set in the Status field of the **TDS_DONE** data stream. Therefore, the client can

always tell after reading a **TDS_DONE** whether or not there are more results associated with the current command.

For stored procedures, completion of statements in the stored procedure is indicated by a **TDS_DONEINPROC** data stream for each statement and a **TDS_DONEPROC** data stream for each completed stored procedure. For example, if a stored procedure executes two other stored procedures, a **TDS_DONEPROC** data stream will signal the completion of each stored procedure.

4.6. Error Information

TDS provides support for returning error numbers, severity, and error message text to a client. This information is returned to clients using the **EED** token. In previous versions of TDS the **TDS_ERROR** and **TDS_INFO** tokens were both used. These tokens are now obsolete.

4.7. Attentions Acknowledgments

Once a client has sent an attention to a server, the client must continue to read data until the attention has been acknowledged. Attentions are acknowledged by servers using the status field of the TDS header. Please see Cancel Protocol on page 21 for details.

4.8. Cursor Status

4.9. Message Responses

5. Protocol Data Unit Definition

TDS supports two types of PDUs; token oriented and tokenless. A token oriented PDU contains TDS tokens in the user data portion of the PDU. Tokenless PDUs contain un-formatted binary data in the user data portion.

5.1. Tokenless Stream

Tokenless data streams are only used for the client login record and bulk copy operations. The PDU header is used to determine the type of data being sent in the PDU. The actual length of the data in the PDU is determined from the length field in the header.

5.2. Token Stream

Tokens are single byte identifiers that are sent in the user data portion of a PDU. They are followed by token specific data. Tokens are either fixed length or variable length. Variable length tokens are followed by a length field. Fixed length tokens do not have a length field.

The size of the length field following a token is encoded in the token value. There are five possible classes of token length fields. They are listed here along with their bit pattern encoding:

5.2.1. Zero Length - 110xxxxx

This is a token which is not followed by a length. There is no data associated with the token.

5.2.2. Fixed Length - xx11xxxx

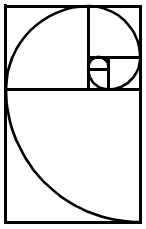
This is a token which is followed by a 1, 2, 4, or 8 bytes of data. No length field follows this token since the data length is encoded in the token value. Bits 3 and 4 are always on. Bits 5 and 6 indicate the length of the fixed length data.

- xx1100xx indicates 1 byte of data.
- xx1101xx indicates 2 bytes of data.
- xx1110xx indicates 4 bytes of data.
- xx1111xx indicates 8 bytes of data.

5.2.3. Variable Length - any other pattern

This token is followed by a length. The size of the length field, in bytes, is also encoded in the token value.

- 1010xxxx indicates 2 bytes of length. (NOTE: KEY token in this group is a “zero-length” token, there is no length field.)
- 1110xxxx indicates 2 bytes of length. (NOTE: ROW, ALTROW, PARAM tokens are in this group, but are “zero-length” tokens. The length field is absent.)
- 1000xxxx indicates 2 bytes of length.
- 001000xx or 011000xx indicates 4 bytes of length.
- 001001xx, 001010xx, 011001xx, or 011010xx indicates 1 byte of length.



Features for 5.0

This chapter describes the additions and changes made in TDS 5.0. The current subversion of this spec is 3.8 The following products will implement support for 5.0 TDS:

- 10.0 DB-Library and later product versions.
- 10.0 Client-Library and later product versions.
- 10.0 Server-Library and later product versions.
- 10.0 SQL Server and later versions.

The 5.0 TDS features fall into the following general areas:

- Cursor support
- Dynamic SQL support
- Extended Error Data
- Additional data types
- Internal changes to improve layering, support, and migration to future releases.

A general description of each of these areas is at the beginning of this chapter followed by examples on how the features are used. Details on the various new features in 5.0 TDS are in the appropriate reference pages in this document.

6. Cursors Support

5.0 TDS provides full protocol support for all ANSI specified cursor commands. This protocol support allows the System 10 Client-Library to provide a call level interface to cursor

functionality implemented in the System 10 SQL Server. It also allows System 10 Server-Library applications to provide support for foreign cursors via another RDBMS or other data source.

This section includes a general discussion and outline of the TDS data stream that supports cursor operations through both language and a non-language call-level interface. Following the outline are some examples which illustrate the relationship of the cursor data streams and a client application. Detailed reference pages for the new cursor tokens are in the **CUR*** reference pages.

6.1. SQL Server Cursor Support

The System 10 release of SQL Server supports cursors as defined in the ANSI SQL 89 specification. Client-Library provides applications access to the SQL Server's cursor functionality or a Server-Library application's via a set of APIs. Client-Library applications can access the cursor functionality in Sybase Server products either through the Client-Library cursor APIs or using SQL language commands.

6.2. Support of Foreign Cursors (Open Server)

System 10 Open Server provides support for all TDS cursor commands. A set of APIs and a new cursor event provide an Open Server application access to all cursor requests made by a Client-Library application via the call-level interface. This eliminates the requirement for an Open Server application to parse T-SQL commands to implement cursor support.

6.3. Cursors and TDS

6.3.1. Client Cursor Requests

Both language and call-level interface cursor requests are supported by System 10 Client-Library.

If a Client-Library application is using language based cursor commands, the cursor command is sent to the server using the **LANGUAGE** token. The disadvantage of this technique is that it requires the server to parse the language command to implement the cursor request. It also makes it more difficult to build an Open Server application to support foreign cursors since a parse would be required to parse the T-SQL cursor command.

If a Client-Library application uses the call-level interface the following TDS tokens are sent to the server instead of a language string:

CURDECLARE	Declare a cursor.
CUROPEN	Open a cursor.
CURFETCH	Fetch "fetch count" number of rows through a cursor.
CURUPDATE	Update the current cursor row.

CURDELETE	Delete the current cursor row.
CURCLOSE	Close, and optionally deallocate, a cursor.
CURINFO	Status info on this cursor.

NOTE: Some tokens appear in more then one version, e.g. the **CURDECLARE** token has three versions, **CURDECLARE**, **CURDECLARE2** and **CURDECLARE3**. Other tokens which have more then one version are **ROWFMT**, **DYNAMIC**, **ORDERBY**,**PARAMFMT** and **CURINFO**. Thus reading e.g. **ROWFMT** implies either **ROWFMT** or **ROWFMT2**.

Cursor tokens can be batched together in the same PDU with some restrictions.

The advantage of using the call-level interface, and cursor tokens, is that it eliminates the parsing required by the server. This improves cursor performance and also makes it easier to provide support for foreign cursors in an Open Server application.

6.3.2. Cursor results

Cursor results are returned to a client using the same **ROWFMT** and **ROW** tokens used to return non-cursor results to a client. The number of rows returned by a cursor fetch is controlled by the current cursor fetch count.

6.3.2.1. Setting "current" cursor row

One complication with cursors is that cursor rows are not passed between the server and client a single row-at-time if the cursor fetch count is greater than one. This means that when the client does an update or delete based on the "current cursor row", the client's idea of the cursor row may not be the same as the server's. This is handled by the client identifying the current row to the server by sending the key for the current row to the server before performing the update or delete.

Key column information is returned to the client in the **ROWFMT** token.

For example, consider the following cursor:

```
DECLARE CURSOR csr AS
SELECT a, b FROM mytable

FOR UPDATE
```

In this example, the unique key for "mytable" is columns "a" and "c". Although the column "c" is not part of the select list, the server will send it back with the **ROW** token as a "hidden" field. The **ROWFMT** token will identify column "a" as a "key" field and column "c" as a "key" and "hidden" field. This tells Client-Library that column "c" is not a column as far as the client application is

concerned, but it is part of the key for the row. Then if any updates or deletes are performed on this cursor, Client-Library will send the key for the current row back to the server as a **KEY** token along with the update or delete request.

The server does **not** send back a new key value if an update changes a key value. The client must remember that this row has been updated, and if the application attempts to update this row again, it should set the **TDS_CUR_CONSEC_UPDS** bit in any future update to this row.

6.3.2.2. Matching cursor results to a particular cursor.

5.0 TDS supports multiple open cursors over the same dialog. However, only one cursor can be the current cursor at any given time. The **CURINFO** token is used to indicate the current cursor on a dialog. The **CURINFO** token is also used by a server to assign a cursor ID when a cursor is first opened, and by a client to set the current cursor fetch count.

Whenever a client or server wants to change the current cursor it sends a **CURINFO** token with the cursor ID set to the new current cursor. A cursor remains current until it is explicitly changed by another **CURINFO**. See TDS_CURINFO on page 145 for complete details.

7. Dynamic SQL Support

8. Extended Error Data

9. Additional Data Types

TDS 5.0 provides support for **NUMERIC**, **DECIMAL**, **LONGVARCHAR**, and **LONGVARBINARY** data types. A new TDS token was added for each of these new data types. The new data types are supported in the **ROWFMT**, **PARAMFMT**, **ALTFMT**, **RETURNVALUE**, **ROW**, **ALTROW**, **KEY**, **PARAMS**, or **RPC** data streams.

Also introduced (version 3.1 of this specification) is the **TDS_BLOB** datatype. It is a chunked or streaming datatype useful for moving larger data. Neither the sender nor the receiver needs to know how large the total data will be when it begins sending it.

See TDS Datatypes on page 167 for details on the new data type tokens.

9.1. TDS Header File

All TDS tokens, defines, and typedefs are now defined in one header file, **tds.h**. **tds.h** is the sole definition for TDS values. It should be used by all Sybase products to ensure product consistency for all TDS values.

9.2. Options and Capabilities

TDS 5.0 adds support for tokenized option commands. The token added to support options is **OPTIONCMD**. Support is provided for setting, clearing, and inquiring about server options.

Previous versions of TDS had no support for options requiring all products that provided option support to use hard coded T-SQL option strings, "set option".

TDS 5.0 also adds support for clients and servers to exchange capabilities during dialog establishment. Clients send a list of requested capabilities to a server for both request and response. The capability list includes both commands that a client can send and a list of data types that can be supported. A server returns the complete list of capabilities that it is willing to support on this dialog. This list may by different than the original list sent by a client. If the list of capabilities is different than the original one requested by the client it can chose to continue using the server's capabilities or to terminate the dialog.

The token added in 5.0 to support this feature is **CAPABILITY**.

Previous versions of TDS required the client and server to use the TDS version to determine the capabilities that are supported on a dialog. This made it very difficult to migrate to future releases of TDS. Capabilities solves this problem by providing a finer level of control over the actual functions supported on a dialog.

9.3. TDS Protocol Data Unit Changes

TDS 5.0 eliminates the use of the packet header type to determine the command contained in the PDU. In previous versions of TDS, both language and RPC commands used the packet header. This made it impossible in previous versions of the protocol to send more than one command at a time since only one packet header exists in a PDU. Now it is possible to mix command and response types in the same PDU.

For example, an option command could be bundled with a language command in the same PDU.

Removing the use of the packet header to indicate the command type also more clearly defines the layering of the TDS protocol. The packet header provides PDU delineation only. This functionality is session level functionality in the OSI Reference Model. The command type indicated by the token is an application level function.

The new packet type added to support this is **NORMAL**. This will be used for all packets that contain completely tokenized data.

10. Wide Result support

Version 3.4 of this specification adds TDS support to remove the 255 byte limit on columns and the 250 column limit per table. **TDS_WIDETABLE** Request and **TDS_NOWIDETABLE** Response capability bits were added to indicate that clients can make requests using new bigger tokens and can handle response streams with these wider result sets.

TDS_CURDECLARE2, TDS_DYNAMIC2, TDS_ORDERBY2, TDS_ROWFMT2, and TDS_PARAMFMT2 tokens were added to address size limitations in the existing CURDECLARE, DYNAMIC, ORDERBY, PARAMFMT, and ROWFMT tokens respectively.

11. Scrollable Cursor support

Version 3.6 of this specification adds TDS support for scrollable cursors. CURDECLARE3 and CURINFO2 tokens have been added to provide more status and option bits necessary for scrollable-cursor support. There are also several examples in the Examples section of this document.

New TDS Tokens

- **CAPABILITY — 0xE2**
  Dialog capability negotiation.
- **CURDECLARE — 0xA3**
  Declare a cursor.
- **CURDECLARE2 — 0x23**
  Declare a cursor.
- **CURDECLARE3 — 0x10**
  Declare a (scrollable) cursor.
- **CUROPEN — 0x31**
  Open a cursor.
- **CURFETCH — 0x2E**
  Fetch through a cursor.
- **CURUPDATE -- 0xEA**
  Update through a cursor.
- **CURDELETE — 0x2C**
  Delete through a cursor.
- **CURCLOSE — 0x33**
  Close a cursor.

- **CURINFO — 0x83**
  Report and set cursor characteristics.
- **CURINFO2 — 0x87**
  Report and set cursor characteristics.
- **DYNAMIC — 0xE7**
  Describes a statement to be “prepared” or a prepared statement to be “executed”.
- **DYNAMIC2 — 0xA3**
  Describes a statement to be “prepared” or a prepared statement to be “executed”.
- **EED -- 0xE5**
  Describes Extended Error Data message(s).
- **KEY — 0xCA**
  Cursor key data.
- **MSG — 0xE5**
  Peer-to-peer message.
- **ORDERBY2 — 0x22**
  Describes the sorting order of the result set to follow based on ORDER BY clauses of the select statement.
- **ROWFMT — 0xEE**
  Describes format of row or key columns.
- **ROWFMT2 — 0x61**
  Describes format of row or key columns.
- **LANGUAGE — 0x21**
  Client language command.
- **LOGOUT — 0x71**
  Dialog termination.
- **OPTIONCMD — 0xA6**
  Setting, clearing, and checking options.
- **PARAMFMT — 0xEC**
  Parameter format.
- **PARAMFMT2 — 0x20**
  Parameter format.
- **PARAMS — 0xD7**
  Parameter data.
- **RPC — 0xE0**
  Database Remote Procedure Call command.

New TDS Packet Types

- **normal** packet type — **20**
  Tokenized request/response packet type.
- **urgent** packet type — **21**
  Tokenized packet type containing attention or event notification.

New TDS Datatypes

- **DECN — 0x6A**
  The decimal data type.
- **NUMN — 0x6C**
  The numeric data type.
- **LONGBINARY — 0xE1**
  The long binary data type.
- **LONGCHAR — 0xAF**
  The long character data type.
- **SENSITIVITY —**
  The sensitivity data type for secure user authentication
- **BOUNDARY —**
  The boundary data type for secure user authentication

New TDS Datatypes with revision 3.6

- **TDS_DATE**
- **TDS_DATEN**
- **TDS_TIME**
- **TDS_TIMEN**
- **TDS_INTERVAL**

Changed TDS Datastreams

- **Language Requests**
  Now tokenized — see **LANGUAGE**.

- **LOGINACK — 0xAD**
  Dropped *interface* argument and added *status* to facilitate handshake login sequence. The interface information is now handled by capabilities.
- **Remote Procedure Call Requests**
  Now tokenized — see **RPC**.

TDS Datastreams No Longer Supported

- **ALTCONTROL — 0xAF**
  Was never implemented.
- **COLNAME — 0xA0**
  Replaced by **ROWFMT**.
- **COLFMT — 0xA1**
  Replaced by **ROWFMT**.
- **PROCID — 0x7C**
  Dropped. Never used.

TDS 5.0 changes with revision 3.7

- Description of TDS attentions was modified to reflect implementation.
- Extended password encryption to allow passing of cipher suite information.
- Added new and renumbered/renamed some old datatypes (see below for list.)
- Login redirection added.
- Cluster failover capability.
- TDS_EED was modified to carve out TDS message space.

The following datatypes were added:

- **TDS_UNITEXT — 0xae**
  Unicode UTF-16 Text

The following datatypes were renumbered:

- **TDS_INT8 — 0xbf**
  8-byte Integer — was 0x39
- **TDS_SINT1 — 0xb0**
  Signed Integer — was 0x40

TDS 5.0 changes with revision 3.8

• **TDS_CURINFO3** was added to restore the usage of **TDS_CUR_ISTAT_ROWCOUNT** so that old cursor applications would continue to work correctly.

• Added capabilities to support large identifiers. Large identifiers have a maximum length of 255.

• Added **TDS_XML** data type (value 0xA3) and associated capabilities.

• Added request and response capabilities for server set packet size.

• Connection Migration messages and capabilities.

• Reserved **TDS_OPTIONCMD2** token value and corrected the TDS Token List to reflect actual token values (a couple of cursor tokens were either missing or incorrect.)

• New section provided to give some detail on cluster related features. This includes some additional information on High Availability (HA) features.

• Misc. spelling errors were fixed.

• Added **TDS_DBRPC2** token to allow longer procedure names. Also added a request capability to indicate that it can be used.

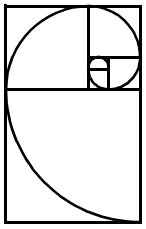
• Performance enhancements were added dealing with the usage of **TDS_ROWFMT** and **TDS_ROWFMT2**. A capability was added to indicate that additional meta-data **TDS_ROWFMT2** provides is not needed. Additionally, a capability was added to indicate that **TDS_ROWFMT** should be sent in preference to **TDS_ROWFMT2**. Also, a capability bit was added that indicates that **TDS_DYNAMIC** and **TDS_DYNAMIC2** support the setting of **TDS_DYNAMIC_SUPPRESS_FMT** in the status field.

• New message buffer header type was added to support migration.

• New TDS Options were added.

• Capability added to suppress **TDS_DONEINPROC** usage.

• **TDS_NONINT_RETURN_VALUE** capability added.



Canceling Operations

Clients require the ability to cancel an outstanding request. For example, the client may submit a query to a server which returns several hundred rows. While the rows are being returned to the client, the client decides that it is no longer interested and wishes to tell the server. This is done by cancelling the request. The operation is typically used to stop the processing of a client request to the server and is known as a cancel.

This chapter describes the 5.0 TDS behavior for handling cancels in terms of the TDS protocol. It also describes how cancels work with new 5.0 TDS features, such as cursors.

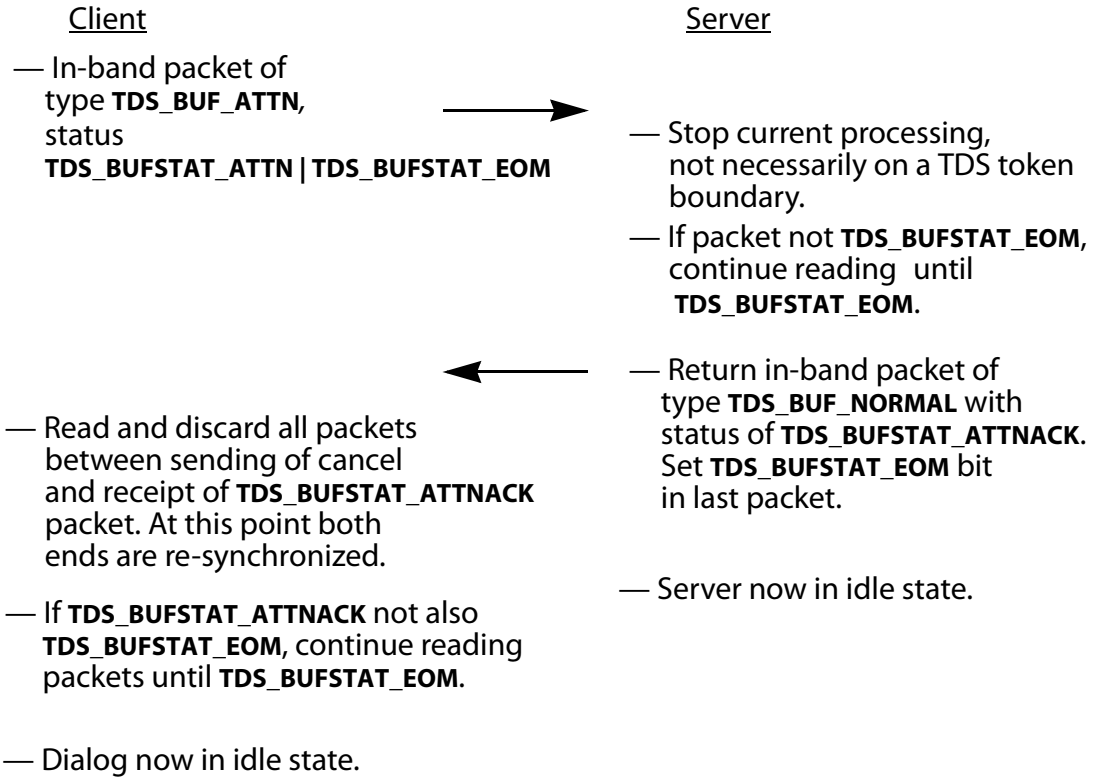
A major change to cancels in 5.0 is that cancels are sent as "normal" data instead of "expedited" data. The elimination of expedited data solves a lot of race conditions caused by using expedited data. Also, not all transport protocols support expedited data. However, the switch to using normal data delivery for cancels is not without cost. Because the cancel is delivered in the normal data stream, cancels can come to the attention of the recipient more slowly than expedited data. This is because any data in front of the cancel must be read first.

12. Cancel Protocol

A cancel request is sent using a non-expedited TDS packet with the header type set to **TDS_BUF_ATTN** and the packet header status bit set to **TDS_BUFSTAT_ATTN**. The client will then read packets from the server until the cancel is acknowledged with a packet of header type **TDS_BUF_NORMAL** and the packet header status bit set to **TDS_BUFSTAT_ATTNACK**. The data, if any, in the packet with the **TDS_BUFSTAT_ATTNACK** bit set is discard. Once the client receives a packet with the header status bit **TDS_BUFSTAT_ATTNACK** set, the dialog state is returned to an idle state. The client may now issue another request.

When a **TDS_BUFSTAT_ATTN** is sent by a client the **TDS_BUFSTAT_EOM** bit must also be set in the header status field. The **TDS_BUFSTAT_ATTNACK** returned by a server in response to a **TDS_BUFSTAT_ATTN** must have the **TDS_BUFSTAT_EOM** bit set at the end of the response. However, the **TDS_BUFSTAT_ATTNACK** can have a data length of 0 or greater. All data in the **TDS_BUFSTAT_ATTNACK** response can be safely discard by the client.

Any dialog state information required by the sender of a **TDS_BUFSTAT_ATTN** is explicitly requested by the sender after the **TDS_BUFSTAT_ATTNACK** has been received. The only state information currently required by a client is the state of all open cursors on the dialog. This state information is requested by the client by sending a **TDS_CURINFO** token with a cmd argument of **TDS_CUR_CMD_INQUIRE** and a cursor id of 0.



Older servers may respond differently than the protocol described above. The original attention acknowledgement protocol used **TDS_DONE** with a done status of **TDS_DONE_ATTN**. This means that a client application needs to examine both the TDS message header and, if the message header does not contain **TDS_BUFSTAT_ATTNACK**, the data stream. Fortunately, only the last 8 bytes of the data stream needs to be examined and can be done by finding a message header with **TDS_BUFSTAT_EOM** set.

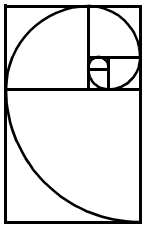
13. Cancels and Cursors

Because cursors, unlike other commands, may have a life that spans multiple requests, the relationship of cancels and cursors needs to be discussed separately. Unlike a regular request, a cursor may, and usually does, have a life beyond a single request. Therefore a cancel does not necessarily cause a cursor to disappear. There is also the problem of row context with cursor. Unless the cursor row count is 1, the server's and client's notion of the current row is usually different. If a cancel is received during a cursor fetch, there is really no way of re-synchronizing the server's and client's row context.

Canceling a batch that includes cursor commands really means that the condition of the cursor or cursors in the request is unknown. The cancel may cause a cursor to be closed or it may have no effect at all if the server has already completed the cursor-related commands in a request. The problem is further compounded by the fact that the server may have finished a cursor operation, e.g., fetch, before it received the cancel and the client doesn't see the data because it comes between the time the cancel was sent and the **TDS_BUFSTAT_ATTNACK** packet was received from the server. There is also the problem that cursor may be either language or function based. A language-based cursor is one that was opened and operated using T-SQL commands. These commands are sent to the server using a **TDS_LANGUAGE** TDS token. A function-based cursor is one that was opened and operated using Client-Library cursor APIs. These commands are sent to the server using TDS **TDS_CUR*** tokens.

These problems are solved in the following way.

- When the client sends a cancel, the client will request the cursor state (open or closed) for all cursors on the dialog. This information is requested using the **TDS_CURINFO** token.
- The client will update its notion of the cursor state, if needed, for every function-based cursor referenced in the request that was canceled.
- The server will enforce the rule that cursors opened via language may be manipulated only with language commands and cursors opened via TDS cursor functions may be manipulated only with TDS cursor tokens. In other words, a cursor may not be manipulated using both language and cursor tokens.



Event Notifications

In pre-5.0 TDS, event notifications were sent using the **TDS_EVENTNOTICE** data stream in a **TDS_BUF_RESPONSE** message. The only token in this response message was **TDS_EVENTNOTICE**. Event notifications are always sent at the end of a complete TDS token stream.

The old way of sending event notifications causes problems when attentions are sent as non-expedited or normal data. A client would miss an event notification that is sent by a server after a client has sent an attention. If event notifications were sent to 5.0 clients using the pre-5.0 protocol, a client could not discard received message data after sending an attention because it would have to parse the token stream looking for event notifications. This defeats the purpose of attentions.

To solve this problem, event notifications in 5.0 are sent in a **TDS_BUF_URGENT** message with the Status field set to **TDS_BUFSTAT_EVENT**. This allows 5.0 clients to discard received data following an attention based on the message header only. The event notification parameters will also be sent using the **TDS_PARAMFMT/PARAMS** data stream, instead of **TDS_RETURNVALUE**.

14. Event Notification Capabilities

The type of event notification protocol to use will be controlled using a new request capability value called **TDS_REQ_URGEVT**. If this capability is requested by a client, the new event notification protocol will be used. If this capability is not requested, the old event notification protocol will be used. This will allow DB-Library to only support the old event notification protocol.

15. Pre-5.0 Event Notification Protocol

This is a summary of the pre-5.0 event notification protocol.

Event Notification Protocol

Message Type: **TDS_BUF_RESPONSE**
Message Status: Undefined

Token Stream
TDS_EVENTNOTICE
TDS_RETURNVALUE
TDS_DONE(TDS_DONE_EVENT|TDS_DONE_FINAL)

NOTE: These are the only tokens in this response message.

Dropped Procedure Protocol

Message Type: **TDS_BUF_RESPONSE**
Message Status: Undefined

Token Stream
TDS_ERROR (MsgNo = 16500)
"Procedure %s no longer exists in the server"
TDS_DONE(TDS_DONE_EVENT|TDS_DONE_FINAL)

NOTE: These are the only tokens in this response message.

16. 5.0 Event Notification Protocol

This is a summary of the 5.0 event notification protocol.

5.0 Event Notification Protocol

Message Type: **TDS_BUF_URGENT**
Message Status: **TDS_BUFSTAT_EVENT|TDS_BUFSTAT_EOM**

Token Stream

TDS_EVENTNOTICE
TDS_PARAMFMT
TDS_PARAMS
TDS_DONE(TDS_DONE_EVENT)

NOTE: These are the only tokens in this response message.

5.0 Dropped Procedure Protocol

Message Type: **TDS_BUF_URGENT**
Message Status: **TDS_BUFSTAT_EVENT|TDS_BUFSTAT_EOM**

Token Stream

TDS_EED (MsgNo = 16500)
"Procedure %s no longer exists in the server"
TDS_DONE(TDS_DONE_EVENT|TDS_DONE_FINAL)

NOTE: These are the only tokens in this response message.

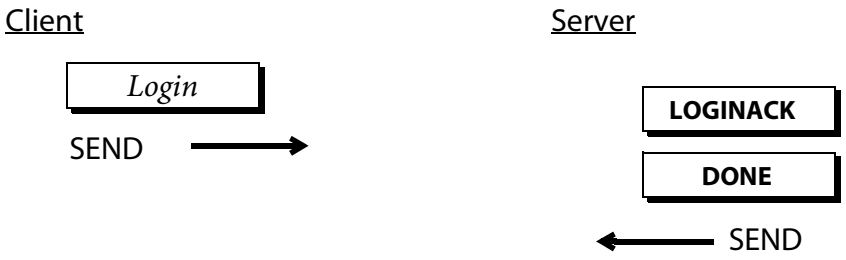
Examples

Command-Based Cursor Operations

Here is an example of a simple cursor client application. This program opens a connection to a server, and obtains a command handle for that connection. The application then declares and opens a cursor, setting cursor rows to 10. The rows of the cursor result set are then fetched one at a time, and an update of a particular row is made. Note that this example is for a forward-only non-scrollable cursor; scrollable cursor examples can be found later in this chapter.

The TDS tokens that are sent and received are identified in the diagrams below.

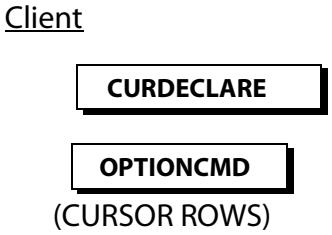
```
/*
** Open a connection to a server.
*/
login = dblogin();
dbproc = dbopen(login, SERVER_NAME);
```



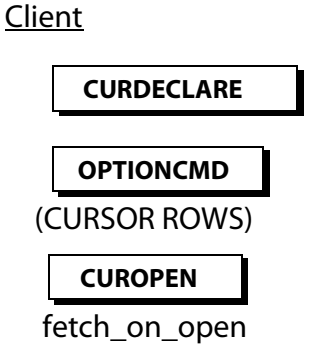

```
/*
** Now get a command handle.
*/
cmd = dbinitcmd(dbproc);

/*
** Let's declare the cursor.
*/
strcpy(charbuf "select * from A_Table");
dbinitop(cmd, DB_CURSOR_DECLARE, "my_cursor", charbuf, DBFORUPDATE);
```


```
/*
```

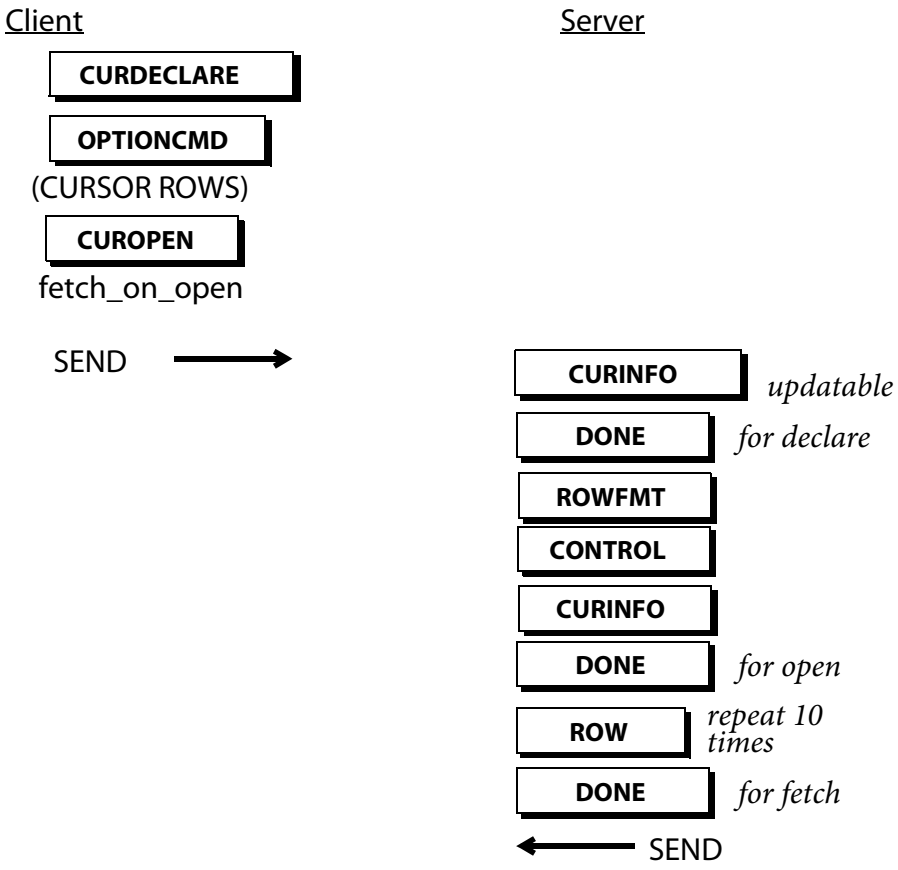
```
** Set the cursor rows to 10.
*/
dbcmdoptions(cmd, DBCURROWS, 10);
```




```
/*
** Let's open the cursor in the same operation.
*/
dbinitop(cmd, DB_CURSOR_OPEN, NULL, NULL, FETCH_ON_OPEN);
```

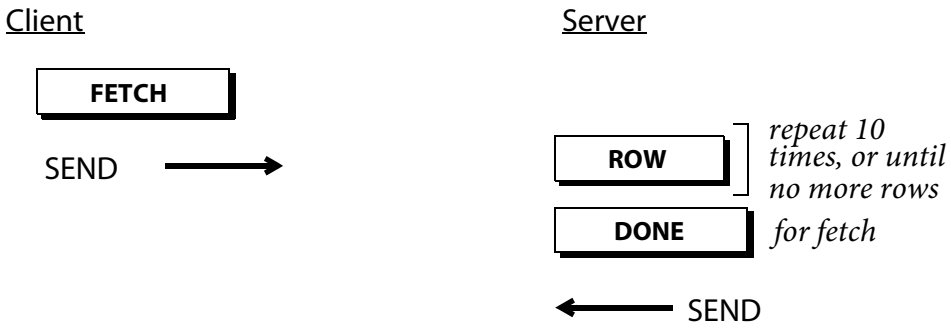


```
/*
** Now send the open to the server.
*/
dbcmdsend(cmd);
```



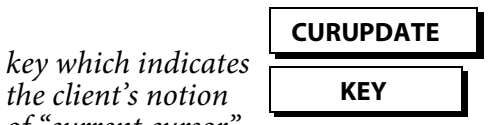

```
/*
 ** Process the results of the cursor.
 */
while ((ret = dbcmdresults(cmd)) != DBNOMORERESULTS)
{
      switch (ret)
      {
       case DBREGRESULT:
            /*
            ** Bind the columns here.
            */
            dbcmdbind(cmd, DBREGROW, SYBINTBIND, 1,
                  4, 1, NULL, &intbuf);
```

```
dbcmdbind(cmd, DBREGROW, SYBNTSBIND, 1,
      255, 1, NULL, charbuf);

/*
** Now fetch the rows.
*/
while (dbfetch(cmd, 1, 0, 0)
                  != DBNOMOREROWS)
```



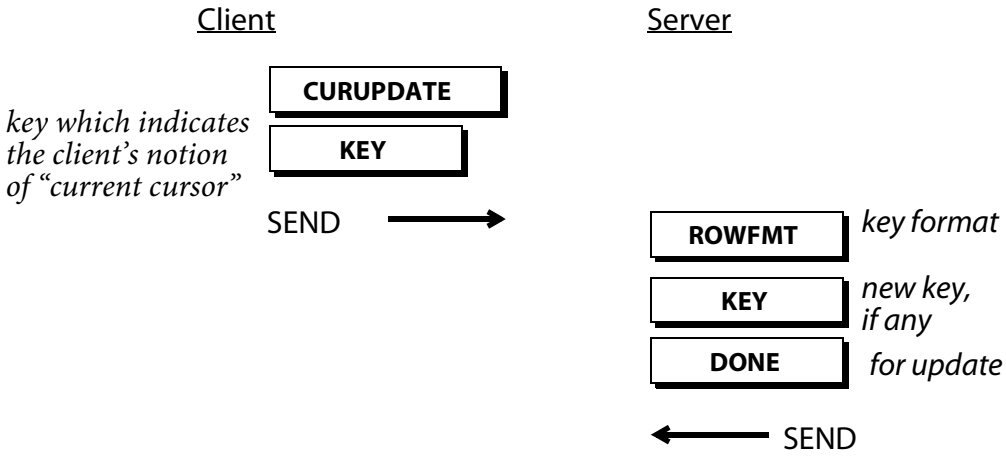
The fetch will be automatically sent when 10 rows are consumed by the client application.

```
{
      /*
      ** Update a particular row.
      */
      if (intbuf == 25)
      {
            /*
            ** Define the update clause.
            */
            dbinitop(cmd, DB_CURSOR_UPDATE, NULL,
                  "set col1 = 55",0);
```



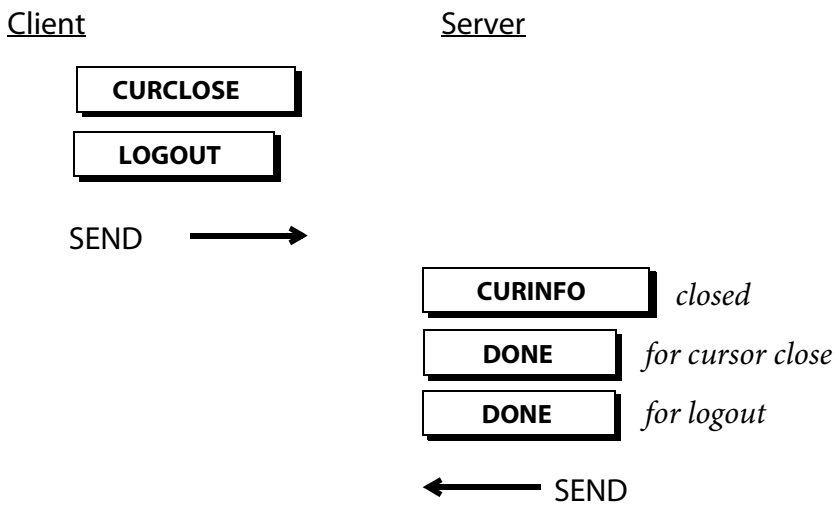

```
            /*
            ** Send the update command
            ** to the server.
            */
```

```
dbcmdsend(cmd);
```



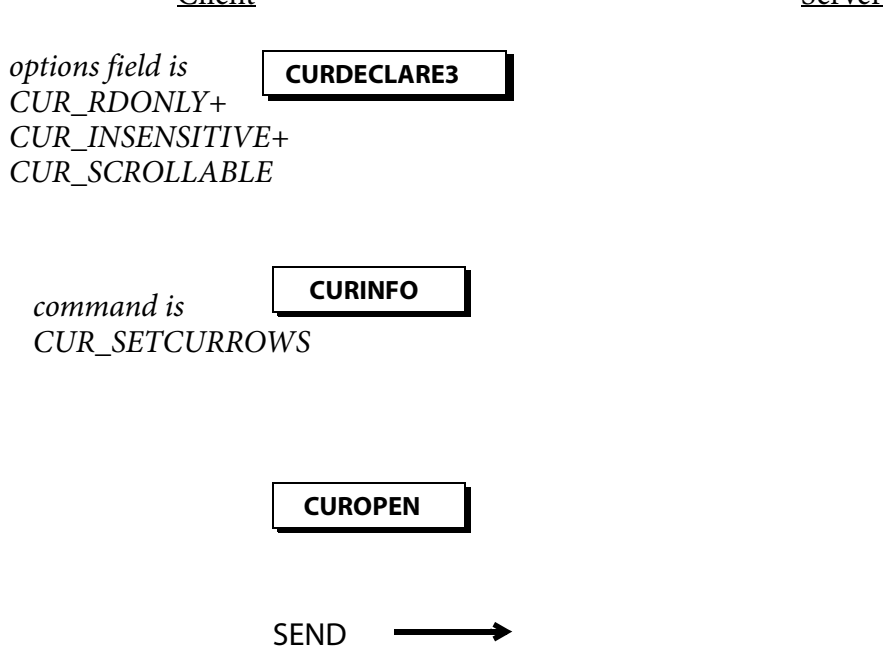

```
                        /*
                        ** Verify that the update
                        ** succeeded.
                        */
                        if (dbcmdresults(cmd) == FAIL)
                        {
                              exit(1);
                        }
                  }

                  /*
                  ** Go on to the next row.
                  */
            }

            break;

      case FAIL:
      default:

            /*
            ** This is an error - Open Client has
            ** already called the application's
            ** error handler, so just exit.
            */
            exit(1);
      }

      /*
      ** Go on to the next result set.
      */
```

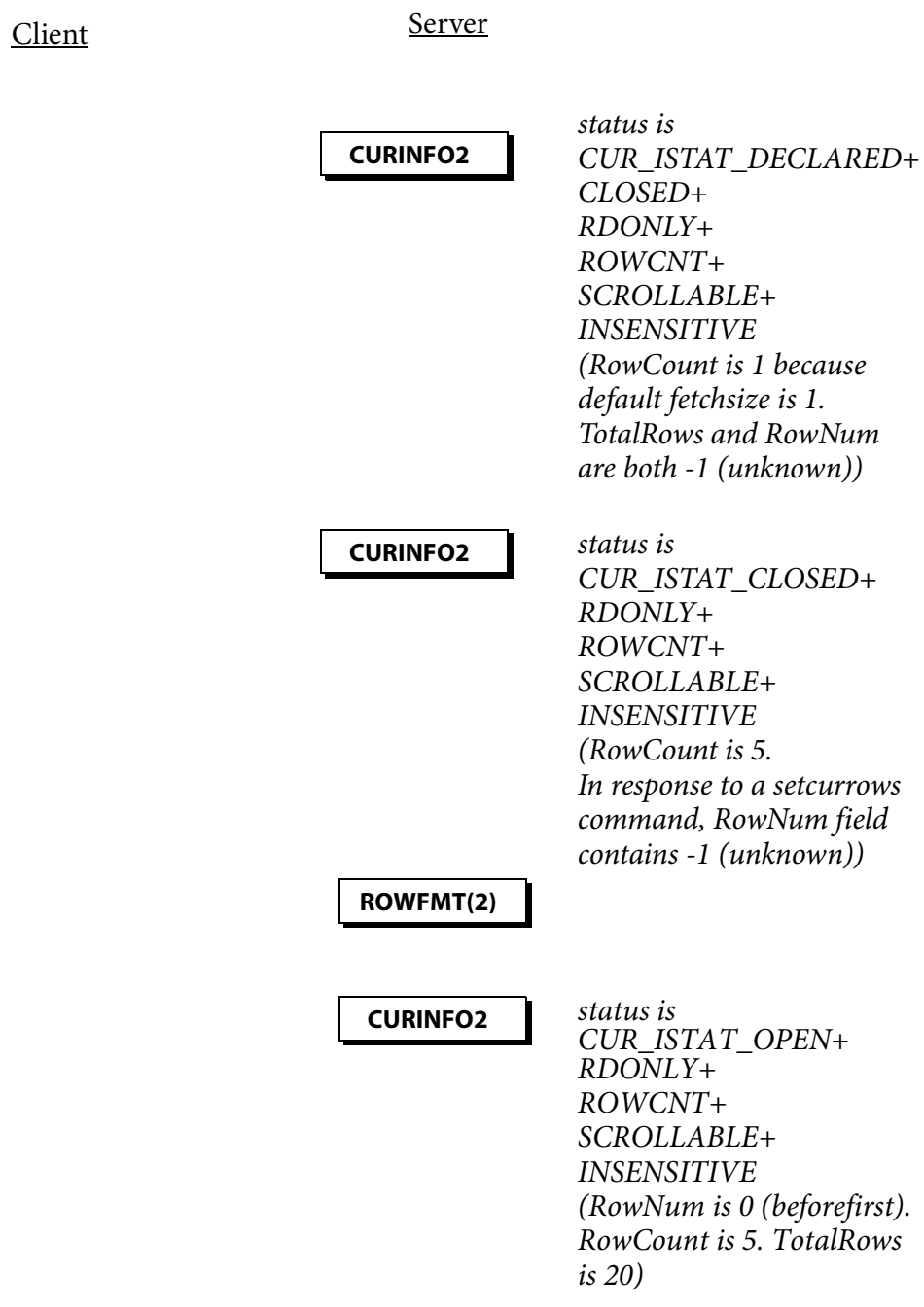
```
}

/*
** All done.
 */
 dbclose(dbproc);
```

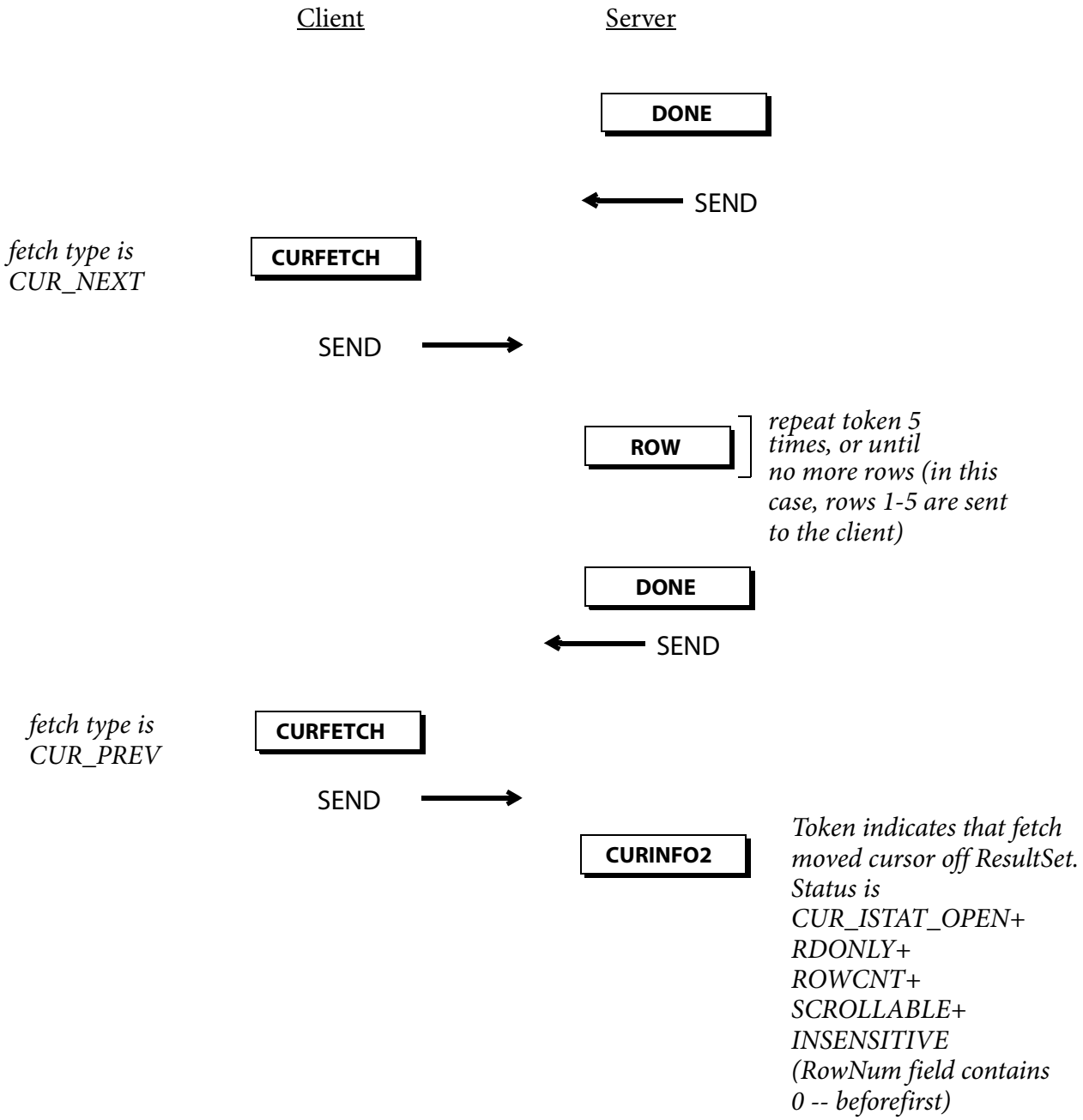


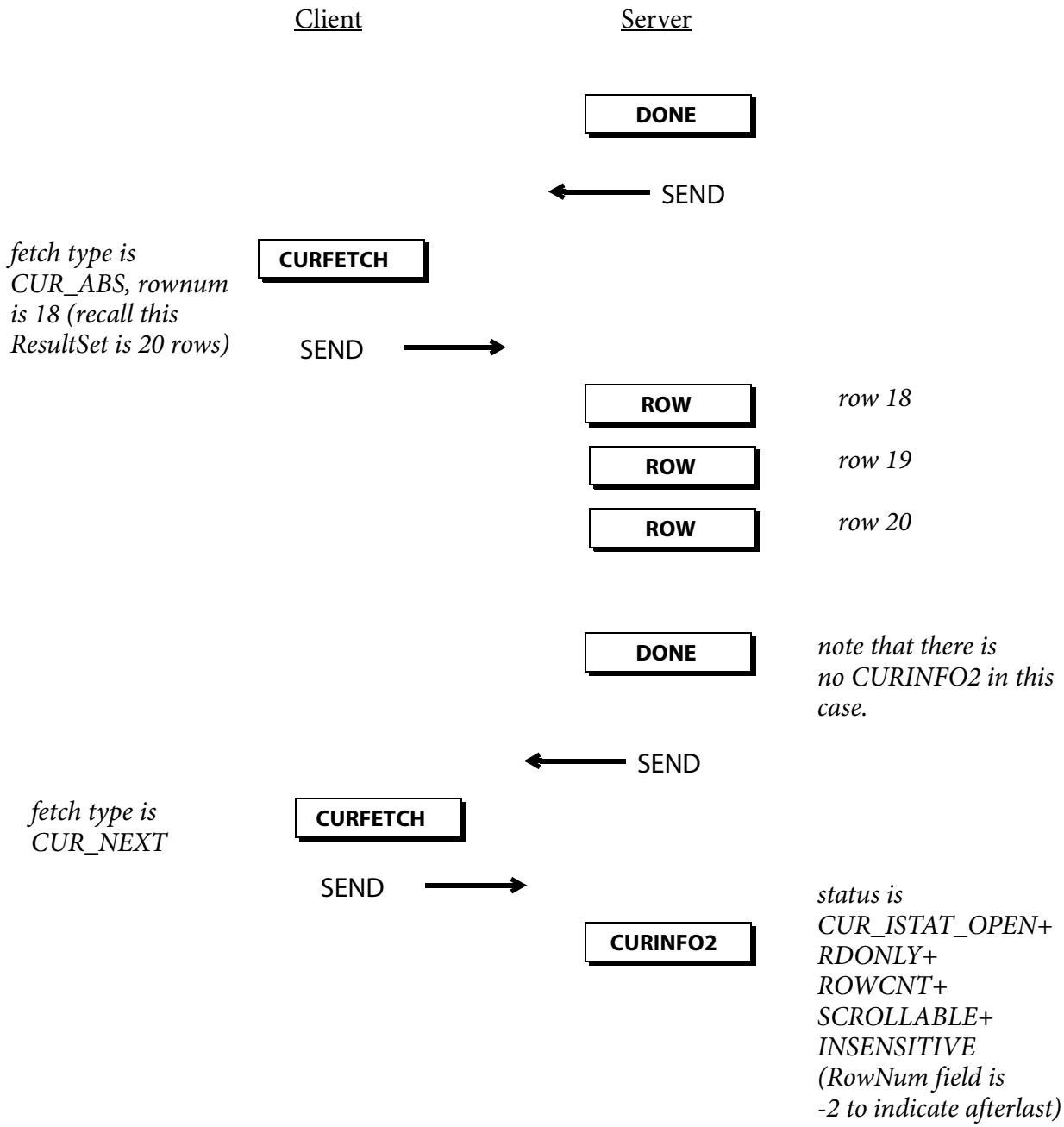
Example - TDS token Scrollable Cursor Operations (scrollable example 1)

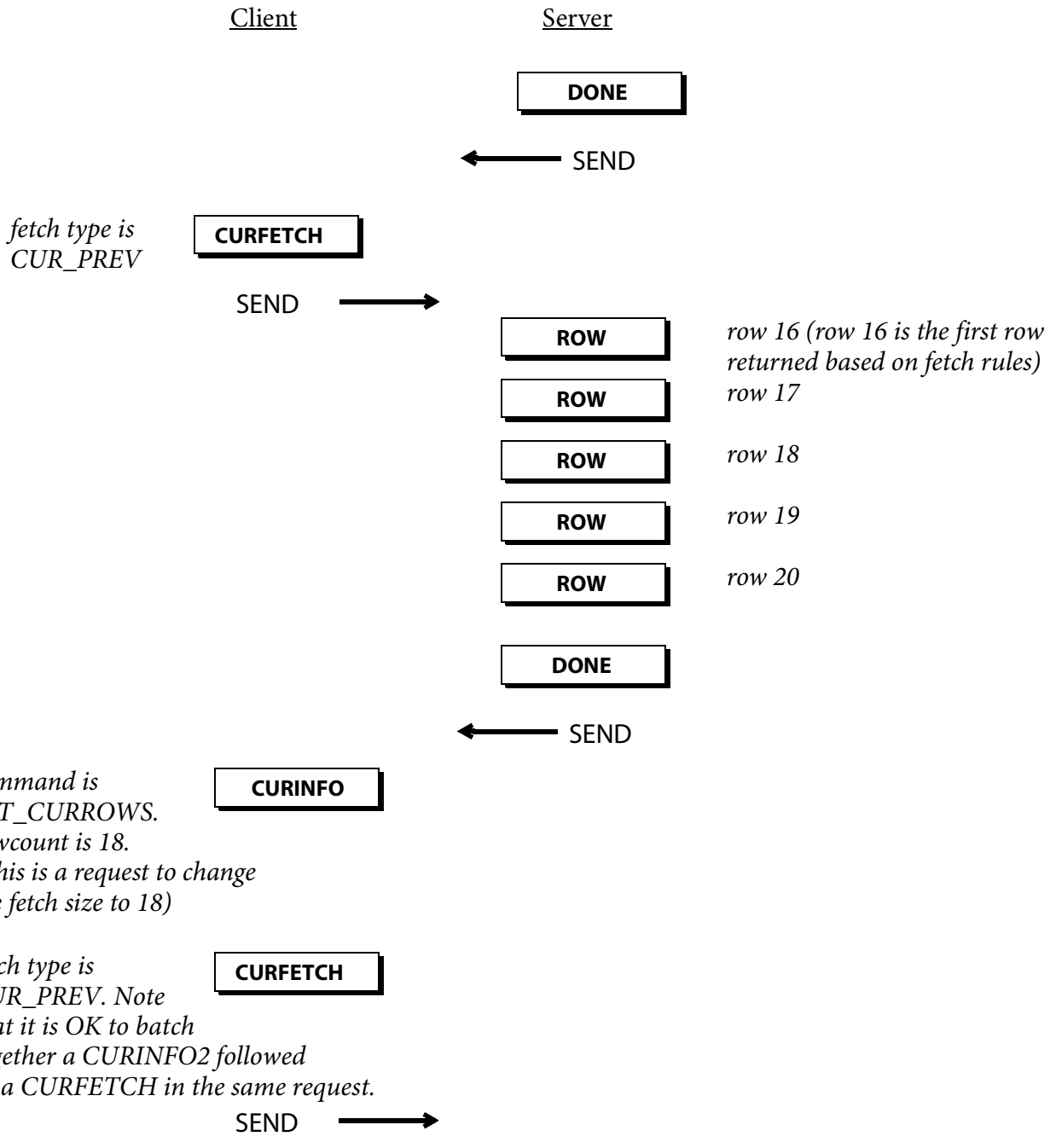
The following example shows the TDS token exchange that can take place when a client opens a read-only insensitive scrollable cursor on a query that returns 20 rows. The fetch size is initially set to 5.

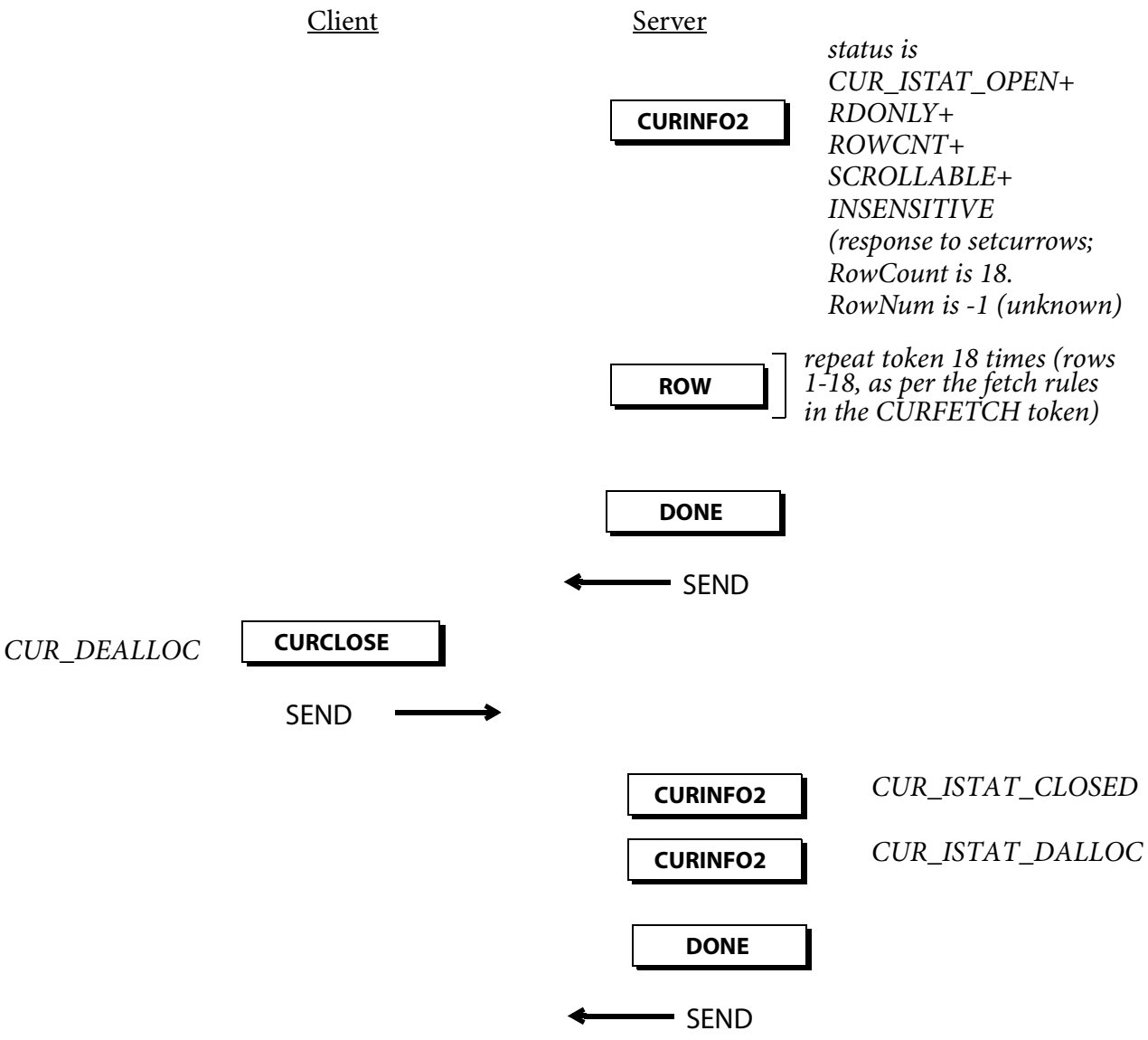



Client
Server
CURINFO2
status is
CUR_ISTAT_DECLARED+
CLOSED+
RDONLY+
ROWCNT+
SCROLLABLE+
INSENSITIVE
(RowCount is 1 because
default fetchsize is 1.
TotalRows and RowNum
are both -1 (unknown))
CURINFO2
status is
CUR_ISTAT_CLOSED+
RDONLY+
ROWCNT+
SCROLLABLE+
INSENSITIVE
(RowCount is 5.
In response to a setcurrows
command, RowNum field
contains -1 (unknown))
ROWFMT(2)
CURINFO2
status is
CUR_ISTAT_OPEN+
RDONLY+
ROWCNT+
SCROLLABLE+
INSENSITIVE
(RowNum is 0 (beforefirst).
RowCount is 5. TotalRows
is 20)


Client
Server
DONE
SEND
fetch type is CUR_NEXT
CURFETCH
SEND
ROW
repeat token 5 times, or until no more rows (in this case, rows 1-5 are sent to the client)
DONE
SEND
fetch type is CUR_PREV
CURFETCH
SEND
CURINFO2
Token indicates that fetch moved cursor off ResultSet. Status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ SCROLLABLE+ INSENSITIVE (RowNum field contains 0 -- beforefirst)

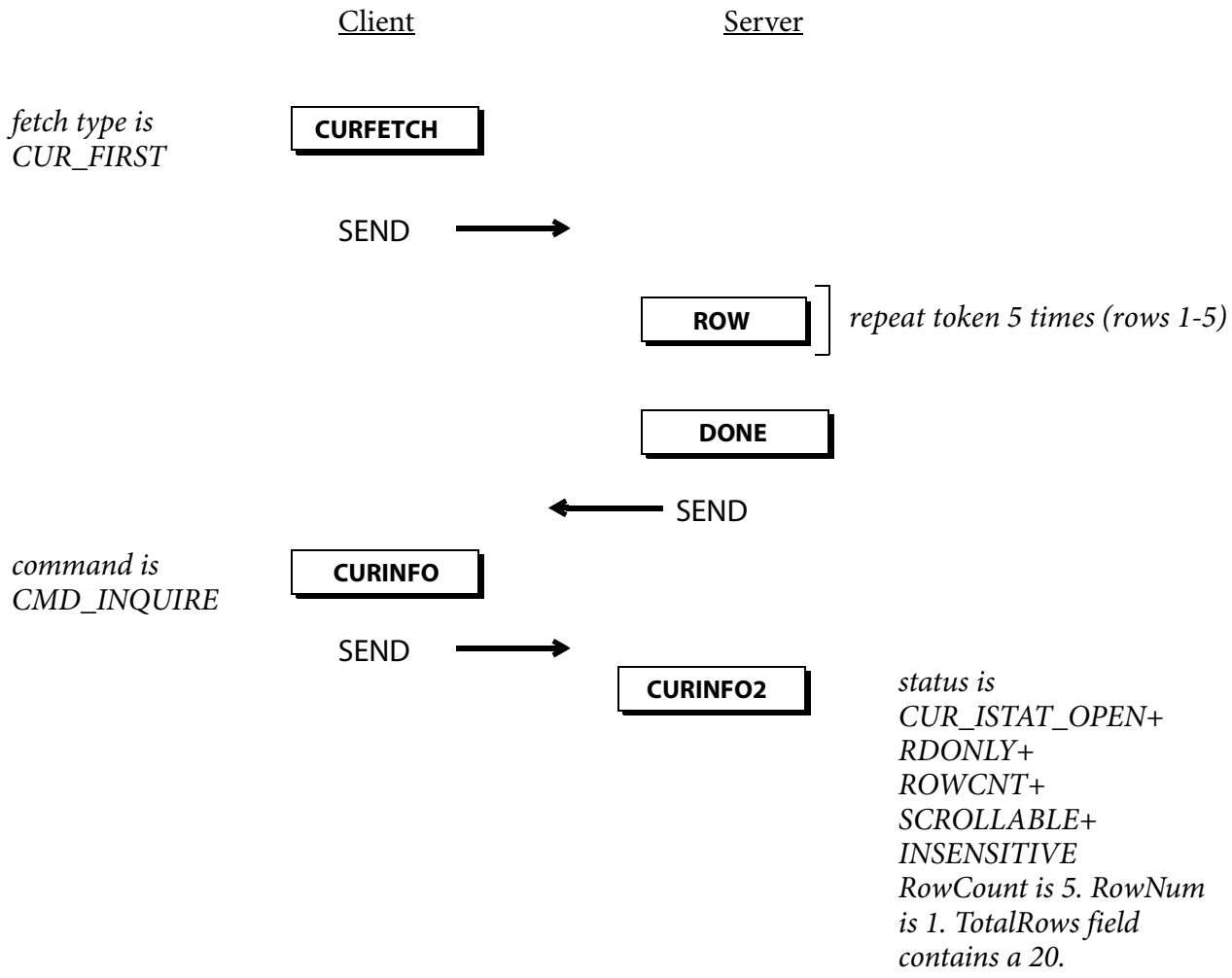

Client
Server
DONE
SEND
fetch type is CUR_ABS, rownum is 18 (recall this ResultSet is 20 rows)
CURFETCH
SEND
ROW
row 18
ROW
row 19
ROW
row 20
DONE
note that there is no CURINFO2 in this case.
SEND
fetch type is CUR_NEXT
CURFETCH
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ SCROLLABLE+ INSENSITIVE (RowNum field is -2 to indicate afterlast)

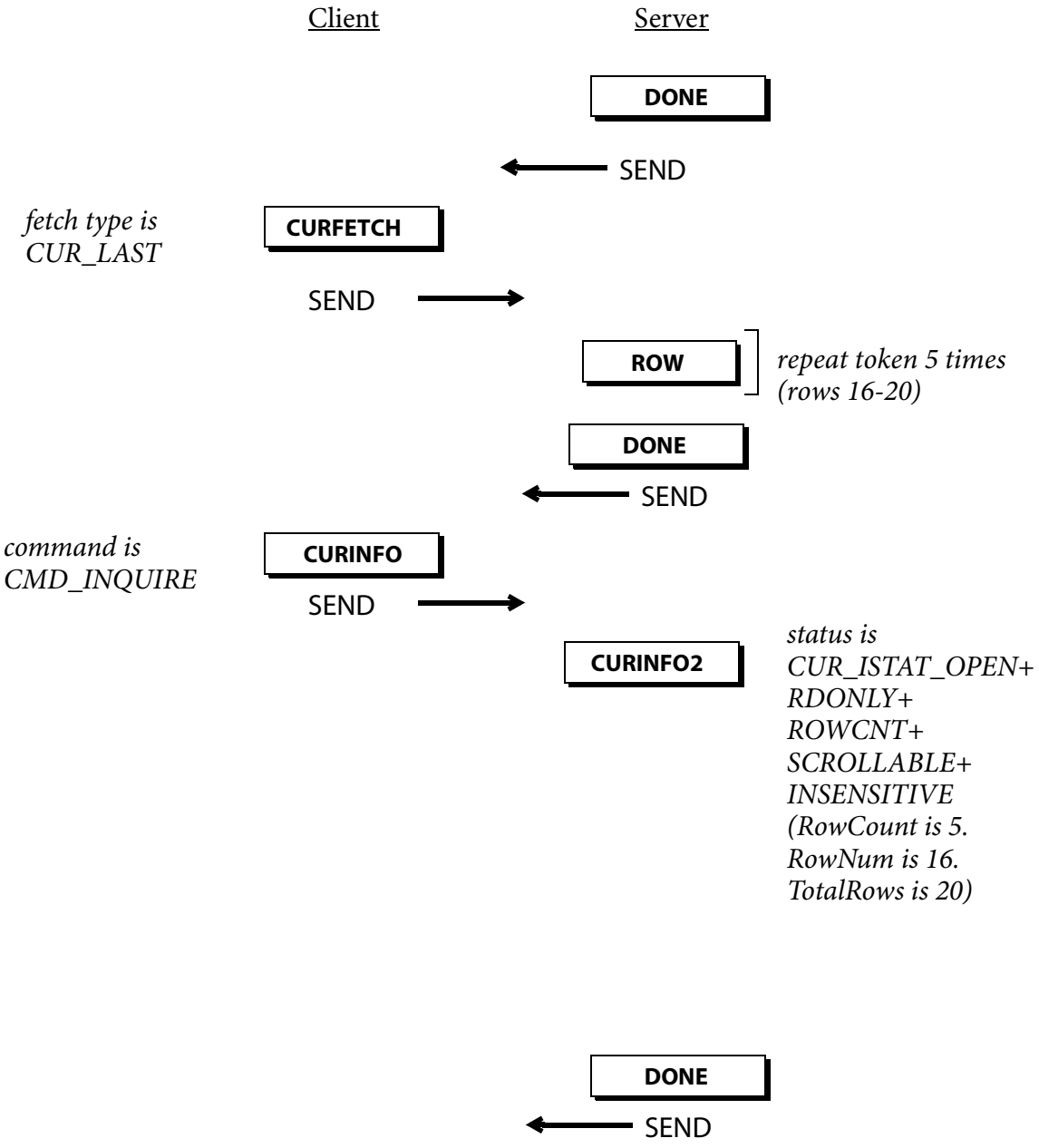

Client
Server
DONE
SEND
fetch type is CUR_PREV
CURFETCH
SEND
ROW
row 16 (row 16 is the first row returned based on fetch rules)
ROW
row 17
ROW
row 18
ROW
row 19
ROW
row 20
DONE
SEND
command is SET_CURROWS. rowcount is 18. (This is a request to change the fetch size to 18)
CURINFO
fetch type is CUR_PREV. Note that it is OK to batch together a CURINFO2 followed by a CURFETCH in the same request.
CURFETCH
SEND

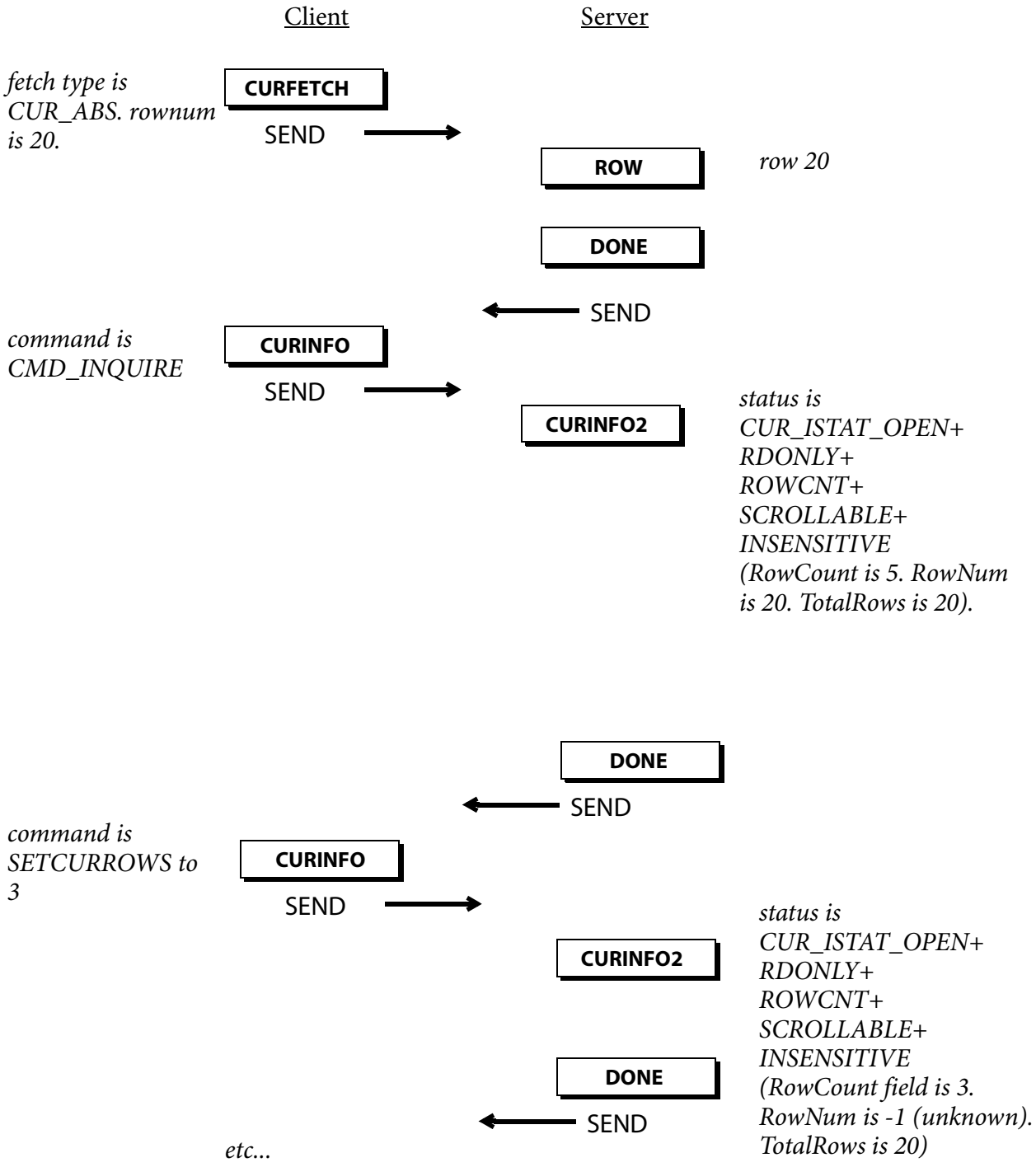

Client
Server
status is
CUR_ISTAT_OPEN+
RDONLY+
ROWCNT+
SCROLLABLE+
INSENSITIVE
(response to setcurrows;
RowCount is 18.
RowNum is -1 (unknown)
CURINFO2
repeat token 18 times (rows
1-18, as per the fetch rules
in the CURFETCH token)
ROW
DONE
SEND
CUR_DEALLOC
CURCLOSE
SEND
CURINFO2
CUR_ISTAT_CLOSED
CURINFO2
CUR_ISTAT_DALLOC
DONE
SEND

Example - TDS token Scrollable Cursor Operations (scrollable example 2)

The following example shows the TDS token exchange that can take place when a client opens a read-only insensitive scrollable cursor on a query that returns 20 rows. The fetch size is initially set to 5. Note that unlike scrollable example 1, we will not include the cursor declaration or open token stream here. Assume that the cursor has already been opened.

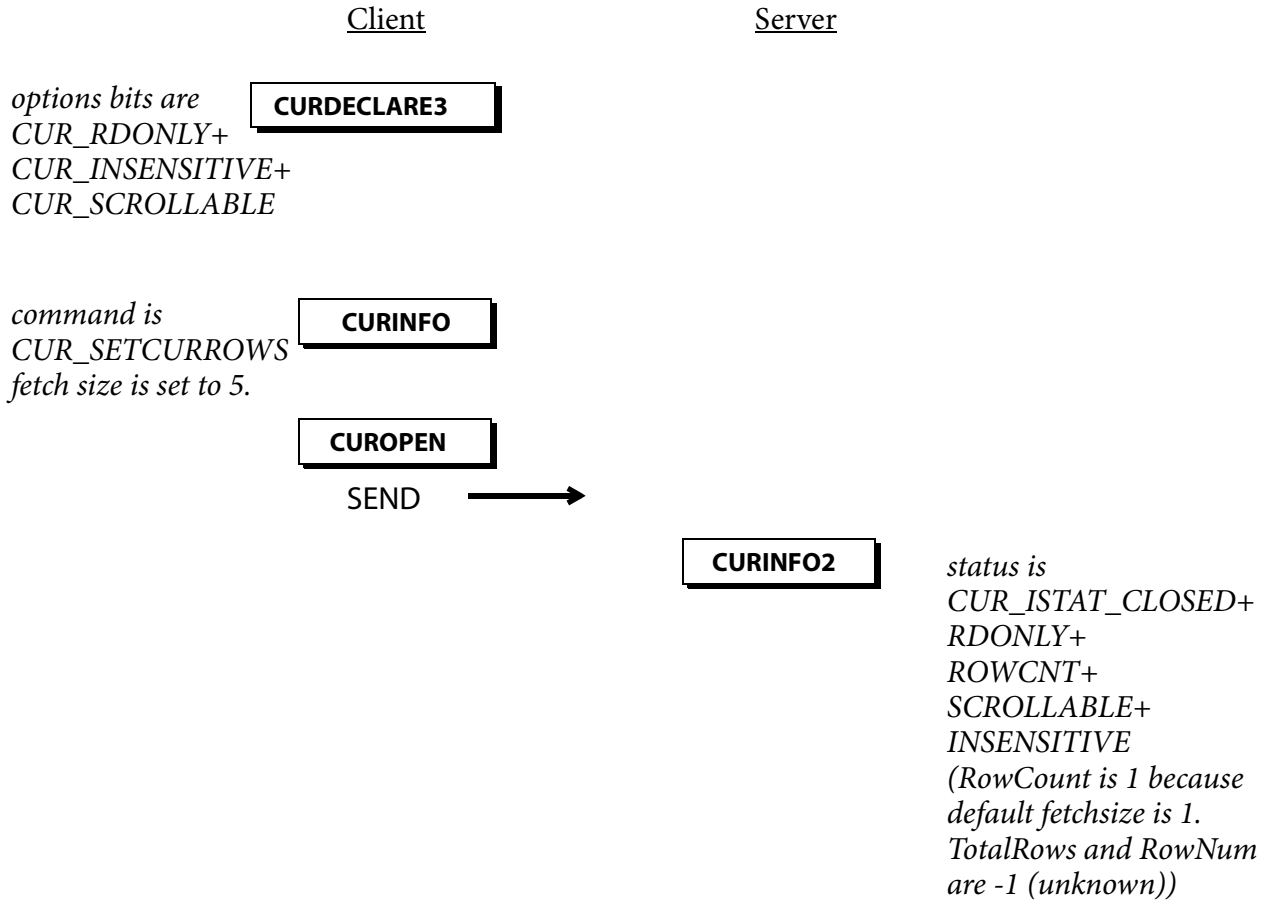


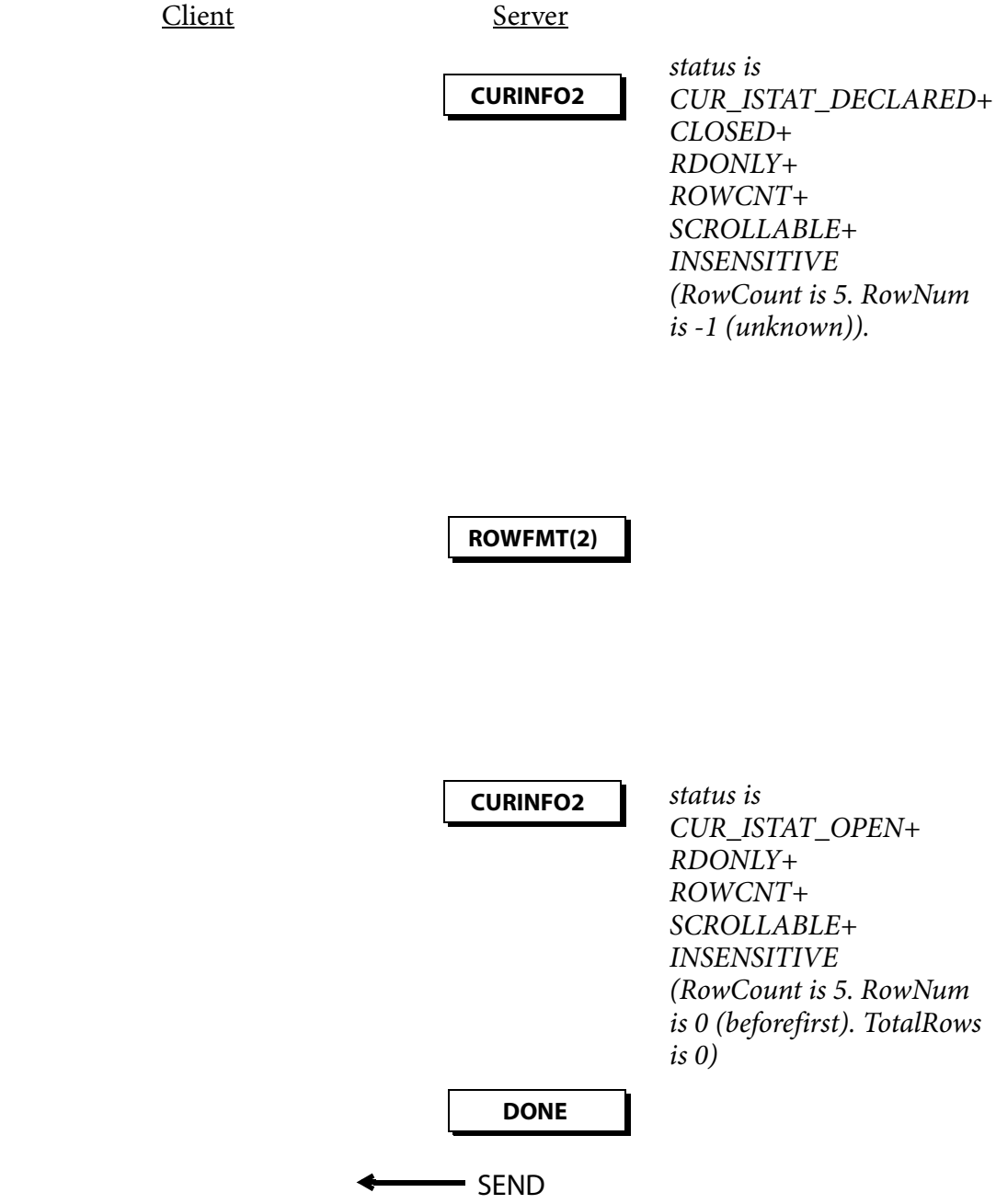

Client
Server
DONE
SEND
fetch type is CUR_LAST
CURFETCH
SEND
ROW
repeat token 5 times (rows 16-20)
DONE
SEND
command is CMD_INQUIRE
CURINFO
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ SCROLLABLE+ INSENSITIVE (RowCount is 5. RowNum is 16. TotalRows is 20)
DONE
SEND

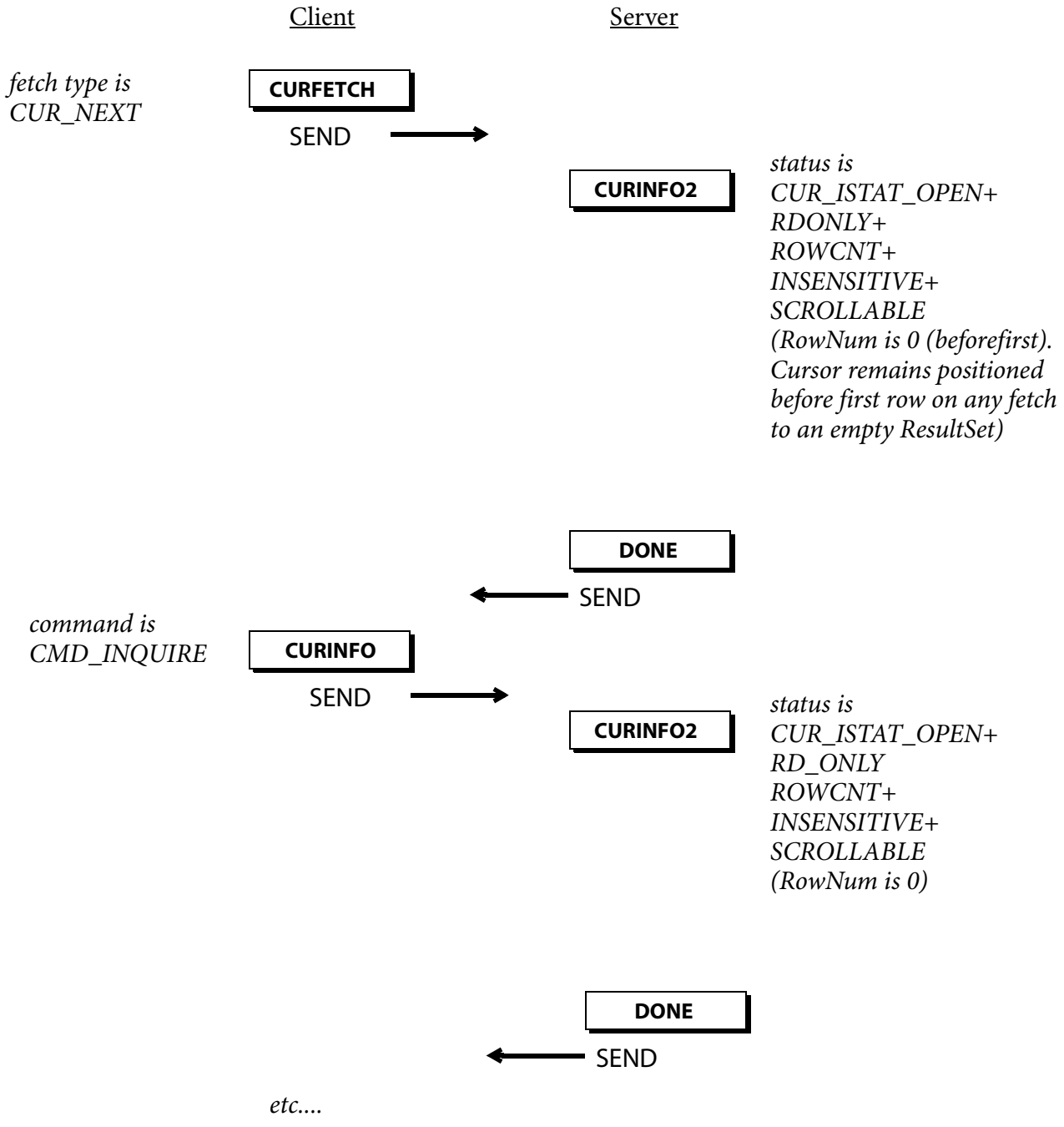

Client
Server
fetch type is CUR_ABS. rownum is 20.
CURFETCH
SEND
ROW
row 20
DONE
SEND
command is CMD_INQUIRE
CURINFO
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ SCROLLABLE+ INSENSITIVE (RowCount is 5. RowNum is 20. TotalRows is 20).
DONE
SEND
command is SETCURROWS to 3
CURINFO
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ SCROLLABLE+ INSENSITIVE (RowCount field is 3. RowNum is -1 (unknown). TotalRows is 20)
DONE
SEND
etc...

Example - TDS token Scrollable Cursor Operations (scrollable example 3)

The following example shows the TDS token exchange that can take place when a client opens a read-only insensitive scrollable cursor on a query that returns no rows (i.e. an empty ResultSet). The fetch size is initially set to 5.

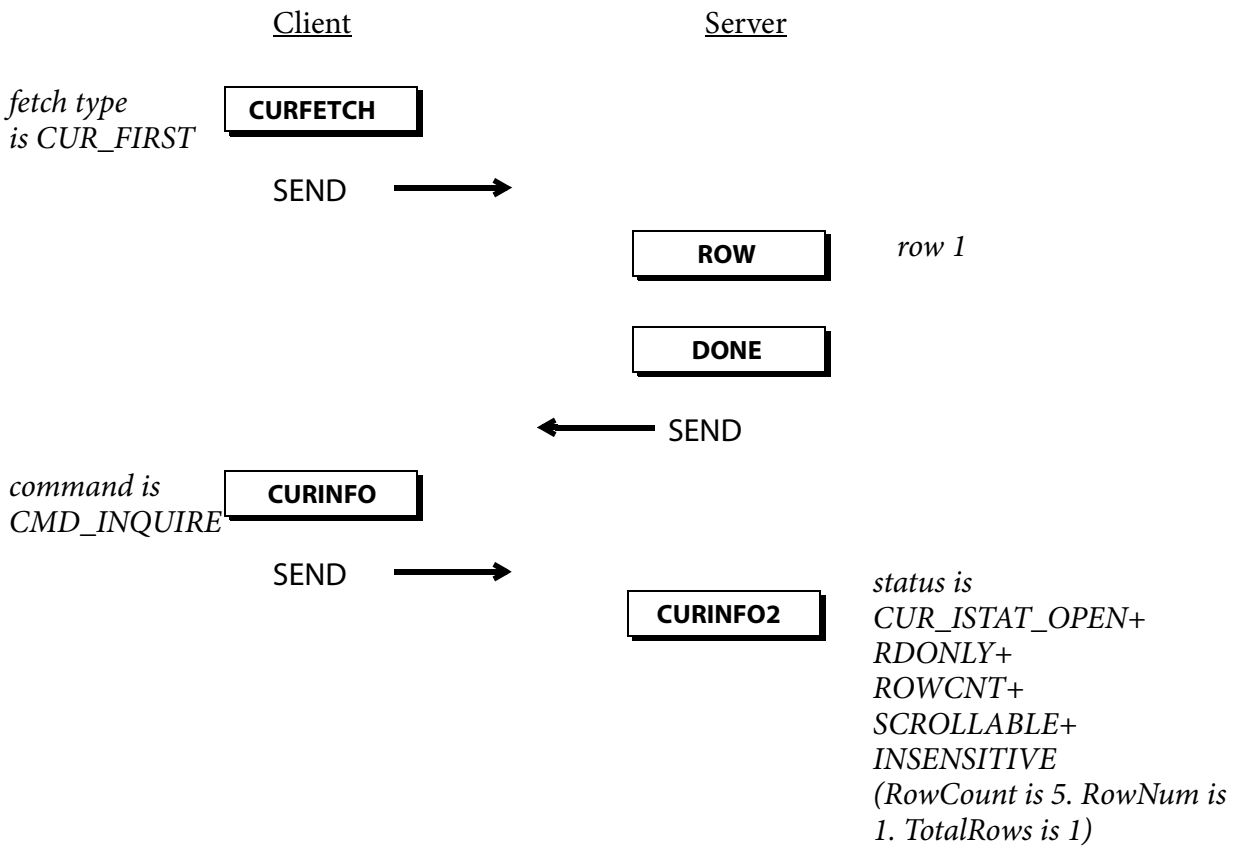


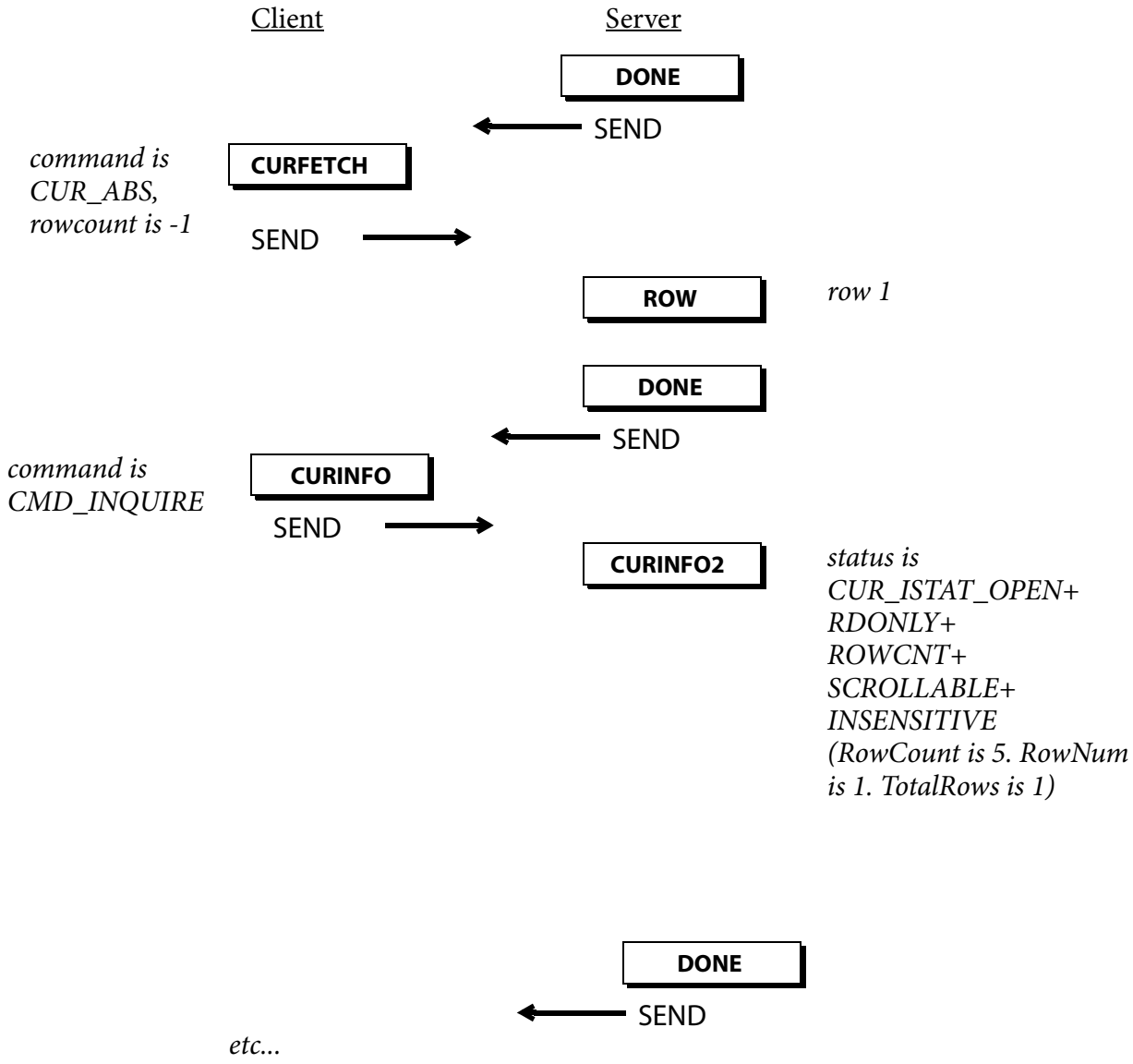

Client
Server
CURINFO2
status is
CUR_ISTAT_DECLARED+
CLOSED+
RDONLY+
ROWCNT+
SCROLLABLE+
INSENSITIVE
(RowCount is 5. RowNum
is -1 (unknown)).
ROWFMT(2)
CURINFO2
status is
CUR_ISTAT_OPEN+
RDONLY+
ROWCNT+
SCROLLABLE+
INSENSITIVE
(RowCount is 5. RowNum
is 0 (beforefirst). TotalRows
is 0)
DONE
SEND


Client
Server
fetch type is CUR_NEXT
CURFETCH
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ INSENSITIVE+ SCROLLABLE (RowNum is 0 (beforefirst). Cursor remains positioned before first row on any fetch to an empty ResultSet)
DONE
SEND
command is CMD_INQUIRE
CURINFO
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RD_ONLY ROWCNT+ INSENSITIVE+ SCROLLABLE (RowNum is 0)
DONE
SEND
etc....

Example - TDS token Scrollable Cursor Operations (scrollable example 4)

The following example shows the TDS token exchange that can take place when a client opens a read-only insensitive scrollable cursor on a query that returns one row. The fetch size is initially set to 5. Like example 2, we assume the cursor has already been opened, and that we have not fetched any rows yet.




Client
Server
DONE
SEND
command is CUR_ABS, rowcount is -1
CURFETCH
SEND
ROW
row 1
DONE
SEND
command is CMD_INQUIRE
CURINFO
SEND
CURINFO2
status is CUR_ISTAT_OPEN+ RDONLY+ ROWCNT+ SCROLLABLE+ INSENSITIVE (RowCount is 5. RowNum is 1. TotalRows is 1)
DONE
SEND
etc...

Note:

The 4 scrollable cursor examples depicted above all use **TDS_CURINFO2**. If a client and server are used which support **TDS_CURINFO3**, the following differences apply:

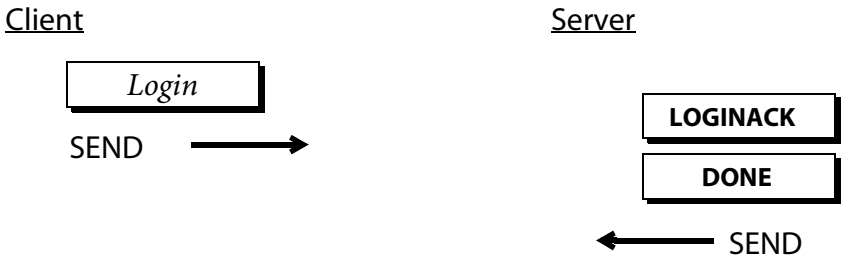
- If a client sends a **TDS_CURINFO** token to the server with **command** set to **TDS_CUR_CMD_INQUIRE**, **TDS_CUR_CMD_SETCURROWS**, **TDS_CUR_CMD_INFORM** or **TDS_CUR_CMD_LISTALL** the server will respond using **TDS_CURINFO3** with the **TDS_CUR_ISTAT_ROWCNT** status bit set and the *RowCount* field must be present and contain the correct (pre-)fetch size.
- In all other cases, if the server responds with **TDS_CURINFO3** because the client issued a **TDS_CURDECLARE3**, **TDS_CUROPEN**, **TDS_CURFETCH** or **TDS_CURCLOSE**, **TDS_CUR_ISTAT_ROWCNT** must not be set and the *RowCount* field must not be present in the stream.

See the description for the **TDS_CURINFO3** token and differences with respect to **TDS_CURINFO2**.

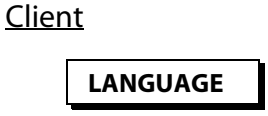
Example — Language-based Cursor Operations

The previous examples accessed cursor functionality in a server via the TDS cursor tokens. Clients may also use language commands for cursor operations. In order to illustrate the ability to access cursor functionality in SQL Server via Transact-SQL queries, we will rewrite the previous example, sending a language command to the server containing cursor operations.

```
/*
** Open a connection to the server.
*/
login = dblogin();
dbproc = dbopen(login, SERVER_NAME);
```

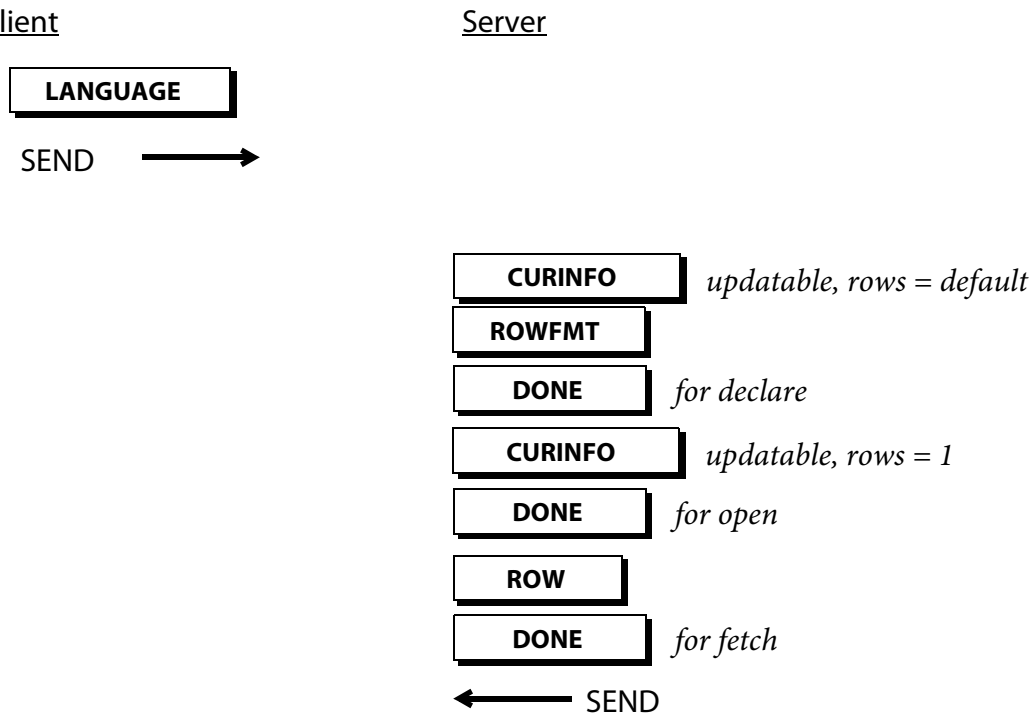



```
/*
** Now get a command handler.
*/
cmd = dbinitcmd(dbproc);

/*
** Let's build our command string. This command batch
** will declare and open the cursor. It will also set
** cursor rows to 1.
*/
strcpy(charbuf, "declare cursor my_cursor for ");
strcat(charbuf, "select * from A_Table for update ");
strcat(charbuf, "set cursor rows 10 for my_cursor ");
strcat(charbuf, "open my_cursor ");
strcat(charbuf, "fetch my_cursor");

dbinitop(cmd, DB_LANG_CMD, NULL, charbuf, 0);
```



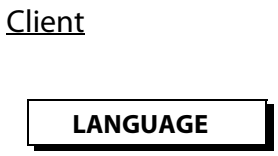

```
/*
** Send the query to the Server.
*/
```

```
dbcmdsend(cmd);
```

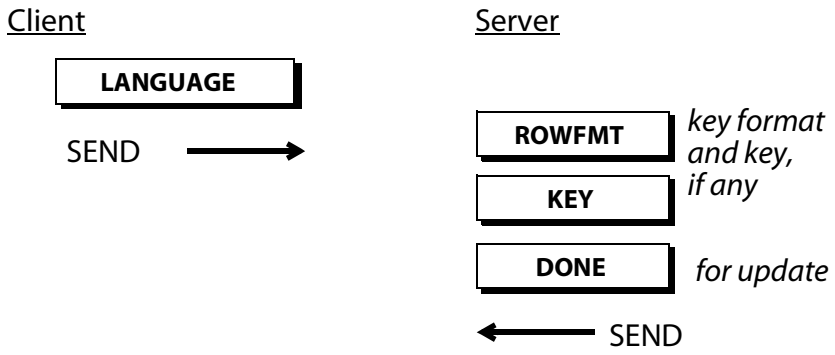



```
/*
** Process the results of the cursor.
*/
while ((ret = dbcmdresults(cmd)) != DBNOMORERESULTS)
{
      switch (ret)
      {
       case DBREGRESULT:

            /*
            ** Bind the columns here.
            */
            dbcmdbind(cmd, DBREGROW, SYBINTBIND, 1,
                  4, 1, NULL, &intbuf);
            dbcmdbind(cmd, DBREGROW, SYBNTSBIND, 1,
                  255, 1, NULL, charbuf);

            /*
            ** Now fetch the rows.
            */
            while (dbfetch(cmd, 1, 0, 0) !=
```

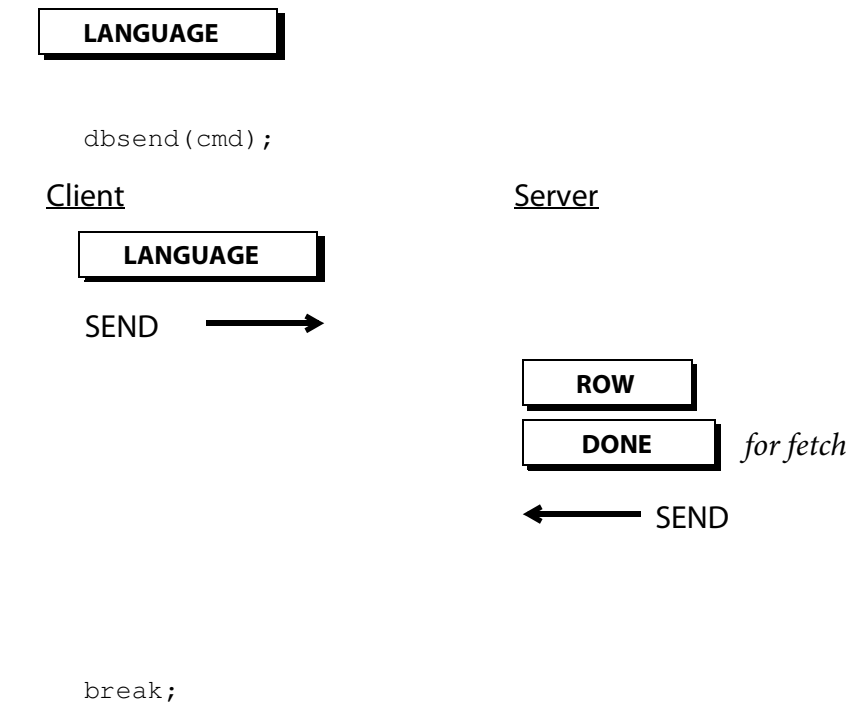
```
                                DBNOMOREROWS)
{

        /*
        ** Update a particular row.
        */
        if (intbuf == 25)
        {
              /*
              ** Define the update clause.
              ** Change the first column
              ** the value 1.
              */
              strcpy(charbuf,
              "update A_Table set col1 = 1 ");
              strcat(charbuf,
              "where current of my_cursor");

              dbinitop(cmd, DB_LANG_CMD,
                    NULL, charbuf, 0);
```




```
              /*
              ** Send the update command
              ** to the server.
              */
              dbcmdsend(cmd);
```

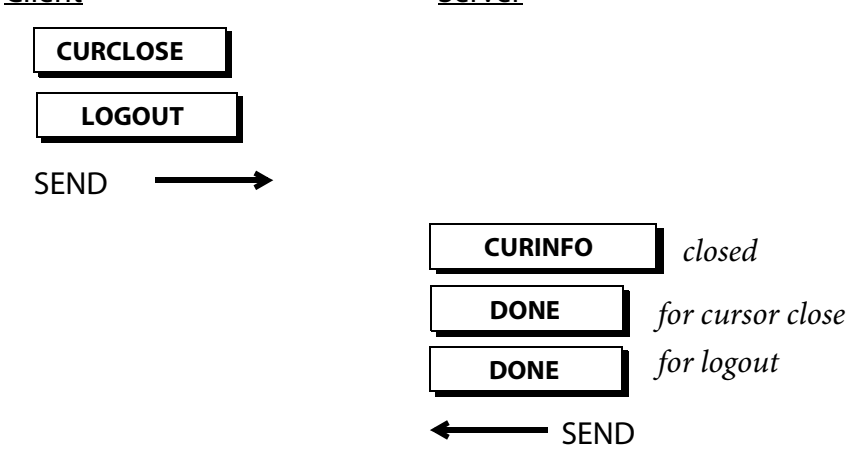


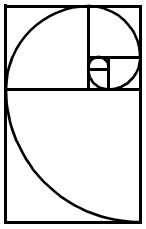
```
                /*
                ** Verify that the update
                ** succeeded.
                */
                if (dbcmdresults(cmd) == FAIL)
                {
                      fprintf(stderr, "ERROR - update
                            failed!\n");
                      exit(1);
                }
          }

          /*
          ** Go on to the next row.
          */
    }

    /*
    ** Send another fetch to see if there are more rows.
    */
    strcpy(charbuf, "fetch my_cursor");
    dbinitop(cmd, DB_LANG_CMD,NULL, charbuf, 0);
```




```
    break;
```

```
        case FAIL:
        default:

             /*
             ** This is an error. Open Client has
             ** already printed an error message,
             ** so just exit here.
             */
             exit(1);
        }
 }

/*
** All done.
*/
dbclose(dbproc);
```





Identity Columns

Identity columns are used to uniquely identify a row in a table. They are a column of type numeric. They must have a scale value of 0. The status field of the **TDS_ROWFMT** token is used to determine if a column is an identity column. Identity columns will have a status of **TDS_ROW_IDENTITY**.

17. Identity Column Options

There are two ways an identity column in a table is updated:

- Implicitly: The server generates a unique value for the identity column
- Explicitly: Client provides a value for the identity column.

These two methods of updating an identity column are controlled using options. The T-SQL option command is:

```
set identity_insert <tablename> <on/off>
```

If identity is turned on, the client is expected to provide a value for the identity column. This is the explicit case. If identity is turned off, the server will generate a value for the identity column. The client does not provide a value for the identity column. This is the implicit case.

Identity can only be turned on for one table at a time on a given dialog.

The option definitions for the **TDS_OPTIONCMD** token to support identity columns are **TDS_OPT_IDENTITYON** and **TDS_OPT_IDENTITYOFF**. See TDS_OPTIONCMD on page 273 for details on this options.

18. Bulk Copy Support

When Bulk Copy loads or retrieves table information it must account for the identity column. In the default case (implicit) the identity column is not returned to the user of the bcp stand-alone or the bulk copy library API. The bulk copy library must provide/strip the identity column based on the table description information received from the server during initialization. In the explicit case the identity column will be provided by and returned to the user of the bcp stand-alone or bulk copy API.

Table 1: Sample Table Description

Table Description	Data Type	Identity?
Column 1	Character	No
Column 2	Numeric	Yes
Column 3	Integer	No

For implicit inbound, the data provided in the bcp input file or via the bulk copy API by a BCP user would not include any information for column 2. When the BCP library is building the formatted row, it would insert a 0 placeholder in the row for the identity column before sending the row to the server.

For explicit inbound, the data provided in the bcp input file or via the bulk copy API by a BCP user would include information for column 2. If this information is not provided the bulk copy library would report an error. The formatted row is built entirely from data provided by the user.

For implicit outbound, the bulk copy library and bcp would not return description information or data for column 2 to the user. If a user asked for a description of column 2, they would receive the description for column 3.

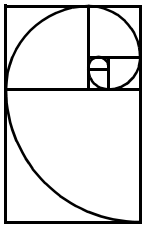
For explicit outbound, descriptions for all columns would be available to the user. Column 2 should be identified as an identity column.

To support identity columns an external configuration option must be made available for both bcp and the bulk copy library. This configuration option is used to indicate whether implicit or explicit identity column behavior is wanted. This configuration option should be made available via a command line option for the bcp stand-alone, and either a new property API to the bulk library, or a new argument to blk_init.

If explicit identity column support is requested, the bulk copy library must request the current setting of the **TDS_OPT_IDENTITYON** option. It then must send a **TDS_OPT_IDENTITYON** option for the table that will be loaded. When the load is complete, the bulk library must generate a **TDS_OPT_IDENTITYOFF** option for the table that was just loaded, and reset the current state of the **TDS_OPT_IDENTITYON** option using the initial setting requested before the bulk copy was started.

Bulk copies on tables that contain identity columns will not be supported in TDS versions < 5.0. If a bulk copy is attempted on a table with an identity column using TDS < 5.0, the server will generate an error and the bulk copy will be aborted.

—



Security Support

TDS 5.0 added support for negotiated login and security specific data types.

19. Data Types

Two new data types were added to support the secure server. Both of these data types are 1 byte variable length data types. Their names are:

- **TDS_SENSITIVITY**
- **TDS_BOUNDARY**

Servers will perform character set translation on these data types. There are no conversions defined for these data types. These data types are used during security handshake, during login, and as column values in a row.

If a client uses capabilities bits to indicate that these data types are not supported, a server automatically sends these data types as **TDS_VARCHAR**s instead. The capability bits are:

- **TDS_DATA_SENSITIVITY** (**TDS_CAP_REQUEST**)
- **TDS_DATA_BOUNDARY** (**TDS_CAP_REQUEST**)
- **TDS_DATA_NOSENSITIVITY** (**TDS_CAP_RESPONSE**)
- **TDS_DATA_NOBOUNDARY** (**TDS_CAP_RESPONSE**)

20. Login Record Support

The lseclogin field in the login record is used to indicate that a client is willing to perform the indicated security handshaking. The server has the final say over whether this handshaking will occur.

The lseclogin field can have any combination of the following bits set:

Table 2: Negotiated Login Bits

Name	Description
TDS_SEC_LOG_ENCRYPT	Perform password encryption. No plain text passwords are sent in either lpw/lpwnlen or lrempw/lrempwlen fields (lpwnlen and lrempwlen should be set to 0). Any information in these fields is ignored by the server.
TDS_SEC_LOG_CHALLENGE	perform challenge/response login sequence.
TDS_SEC_LOG_LABELS	Perform security label exchange.
TDS_SEC_LOG_APPDEFINED	Perform application specific security hand-shake.
TDS_SEC_LOG_SECSESS	Use external security mechanisms.
TDS_SEC_LOG_ENCRYPT2	Use extended password negotiation if available.

21. Security Messages

The message numbers in the table below are reserved for secure login negotiation.

Table 3: Negotiated Login Messages

Name	Client/ Server	Description
TDS_MSG_SEC_ENCRYPT	Server	Start encrypted login protocol. This message has one **TDS_VARBINARY** parameter containing the encryption key.
TDS_MSG_SEC_LOGPWD	Client	Send encrypted user password to a server. This message has one **TDS_VARBINARY** parameter containing the encrypted user password.

Table 3: Negotiated Login Messages

Name	Client/ Server	Description
TDS_MSG_SEC_REMPWD	Client	Send a list of remote servers and encrypted passwords to a server. The message parameters consist of pairs of **TDS_VARCHAR/ TDS_VARBINARY** parameters that contain the remote server name and the encrypted password for that remote server.
TDS_MSG_SEC_CHALLENGE	Server	Start challenge/response protocol. This message has one **TDS_VARBINARY** parameter which contains an un-encrypted challenge byte string. This message is only used for the probe account and the backup server.
TDS_MSG_SEC_RESPONSE	Client	Return the encrypted challenge byte string to a server. This message is only used for the probe account and the backup server.
TDS_MSG_SEC_GETLABELS	Server	Start trusted user login protocol. There are no parameters to this message.
TDS_MSG_SEC_LABELS	Client	Return security labels to a server. This message has an undefined number of parameters of type **TDS_SENSITIVITY**. These parameters contain the security labels. The number of security labels returned to the server is undefined by the TDS protocol.
TDS_MSG_SEC_OPAQUE	Both	External security mechanism meta-data such as data signatures and authentication tokens.

Table 3: Negotiated Login Messages

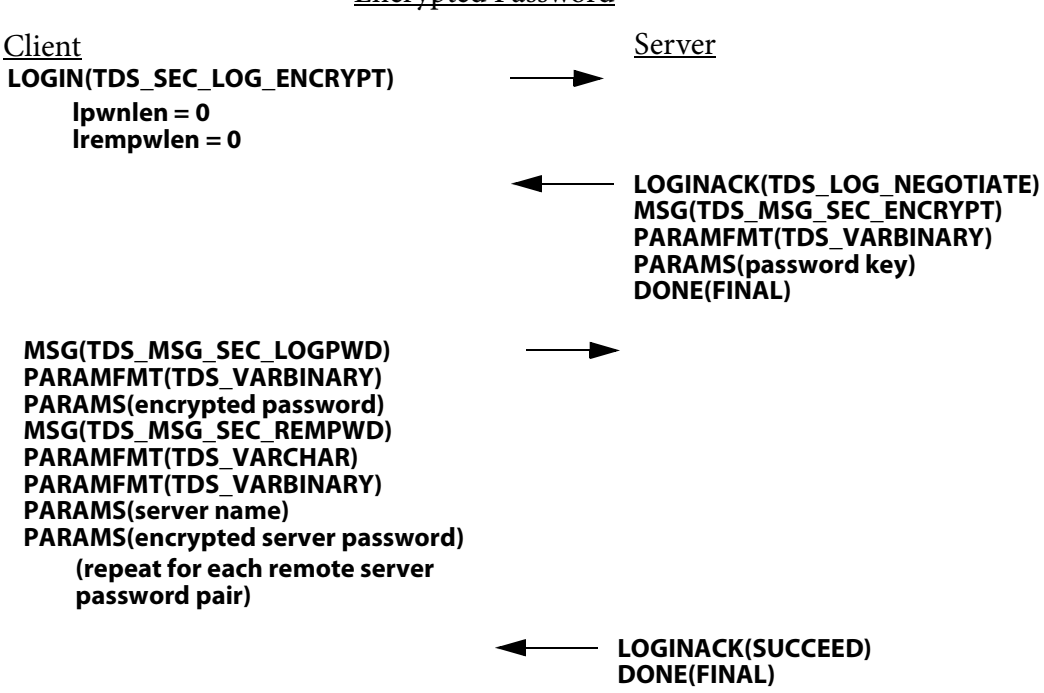
Name	Client/ Server	Description
TDS_MSG_SEC_ENCRYPT2	Server	Start extended encrypted password login protocol. This message has two parameters, a **TDS_INT4** containing the cipher suite to use and a **TDS_LONGBINARY** parameter containing the encryption key corresponding to the cipher suite selected.
TDS_MSG_SEC_LOGPWD2	Client	Send encrypted passwords to a server. This message has one **TDS_LONGBINARY** parameter containing the encrypted user password followed by **TDS_LONGCHAR**/ **TDS_LONGBINARY** parameter pairs representing remote server names and their encrypted passwords.
TDS_MSG_SEC_SUP_CIPHER	Client	Send list of supported ciphers. At this time, this list is sent if the server sends a **TDS_MSG_SEC_ENCRYPT2** message containing a cipher suite the client does not support. The parameters are a list of **TDS_INT4** values containing supported cipher suites.

Some comments on Negotiated Login Messages on page 60. The **TDS_MSG_SEC_ENCRYPT**/ **TDS_MSG_SEC_LOGPWD**/**TDS_MSG_SEC_REMPWD** messages all assume that the client and server have only one agreed encryption algorithm. **TDS_MSG_SEC_OPAQUE** is used to send external security mechanism information between the client and server during both login time and to implement data packet signing. **TDS_MSG_SEC_ENCRYPT2**/**TDS_MSG_SEC_LOGPWD2**/**TDS_MSG_SEC_SUP_CIPHER** cipher suite values are from RFC 2246, section A.5. Note that the cipher suite values are composed of an asymmetric cipher, symmetric cipher, and a hash function. Currently, only the server will send **TDS_MSG_SEC_ENCRYPT2** with a public key for the asymmetric algorithm of the cipher suite.

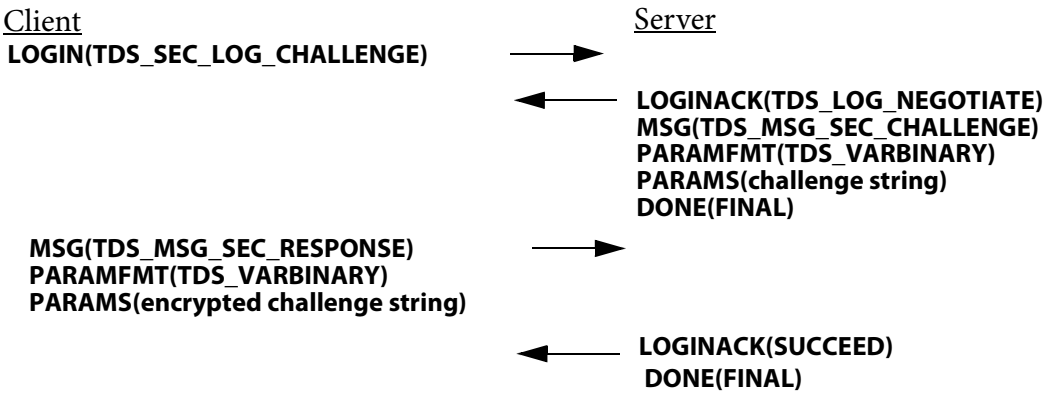
22. Security Protocols

The client program is responsible for requesting that a security hand-shake should occur using one or more of the negotiated login bits in the login record. A client can abort a security handshake at any time by closing the connection.

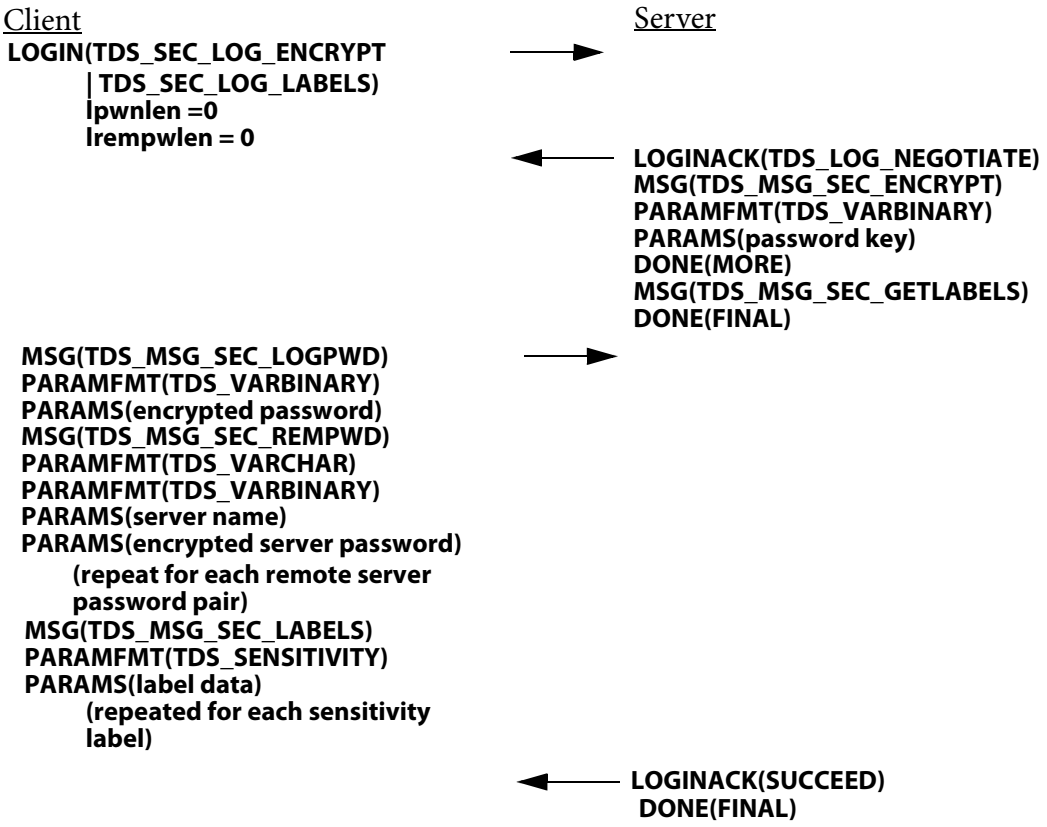
Security hand-shaking is done using the messages defined above. The protocol for the encrypted password, challenge/response, and trusted user are below.



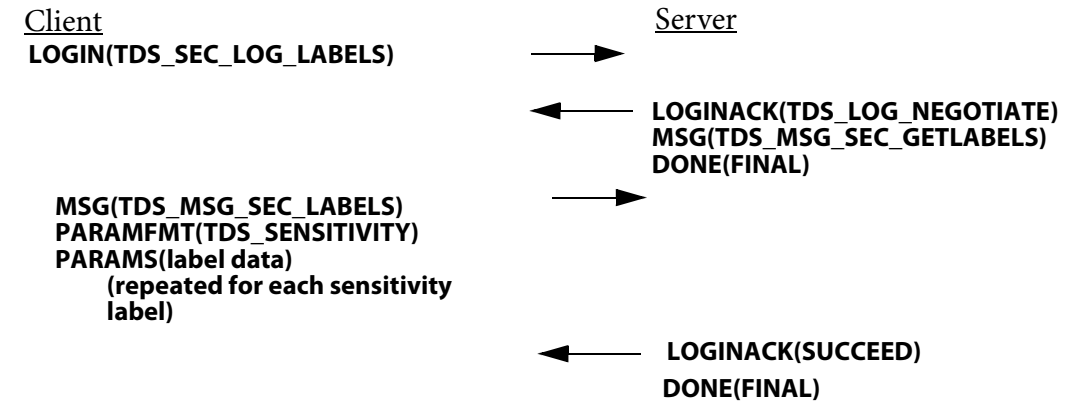
Challenge/Response



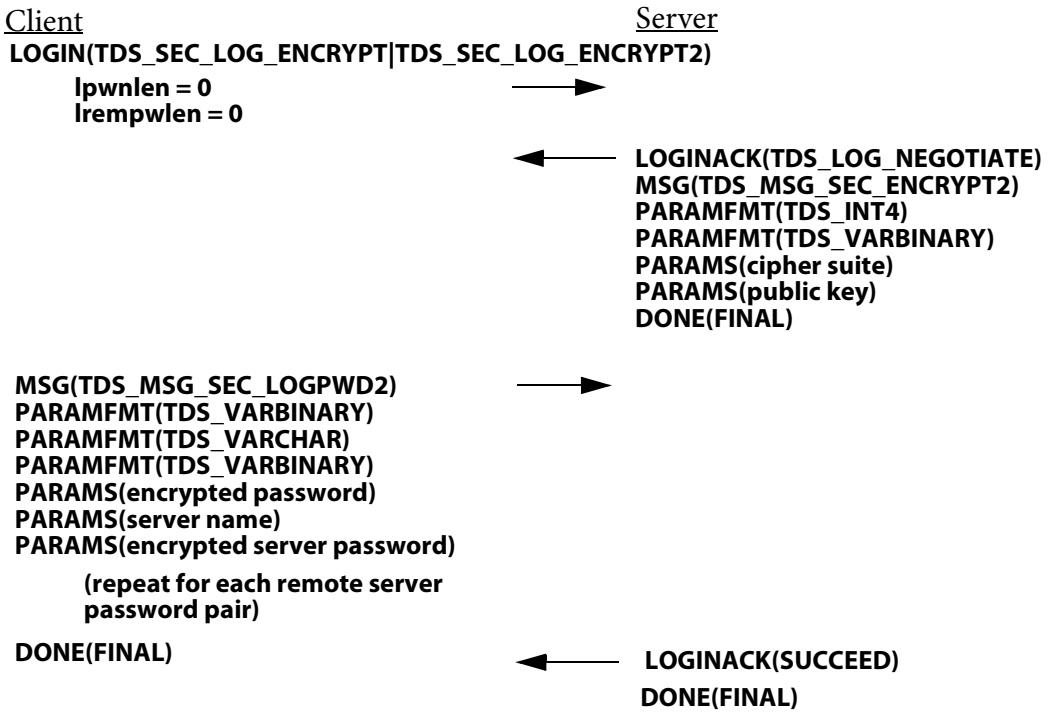
Encrypted Password and Trusted User

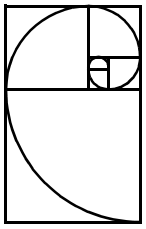


Trusted User



Extended Encrypted Password





Cluster Support

Cluster support refers to three areas of the TDS specification that were added to support environments that require a high degree of availability. These areas are connection re-establishment in the event of a server failure (refered to here are the HA feature,) login redirection (which allows a group of servers to load balance client connections among other things,) and connection migration (which allows a server to re-connect a client from itself to another server.)

23. Login Record Support

The lhalogin field in the login record is used to indicate that a client is willing to perform HA related functionality as well as login redirection. If a client is capable of any of these services, the server is allowed to send the appropriate message to the client.

The lhasessionid field was added to support client session resumption. This field is 6-bytes in length and will be valid only during a failover or migration. The server sends a 6-byte value to the client during negotiated login for the client to use during session resumption in the event of a connection failure.

The lhalogin field can have any combination of the following bits set:

Table 4: HA Login Bits

Name	Description
TDS_HA_LOG_SESSION	Client application wants automatic failover to occur in the event of a connection failure.
TDS_HA_LOG_RESUME	Client application is attempting to failover to server and the lhasessionid field is valid.

Table 4: HA Login Bits

Name	Description
TDS_HA_LOG_FAILOVERSRV	If set, this server is considered the secondary server.
TDS_HA_LOG_REDIRECT	Client library supports login redirection to alternate server.
TDS_HA_LOG_MIGRATE	Client application is being migrated to server.

24. Migration Messages

The message numbers in the table below are used for HA and connection migration.

Table 5: Redirection and Migration Related Messages

Name	Client/ Server	Description
TDS_MSG_HAFAILOVER	Both	This message has one parameter, the session id that the client should use during a failover or migration operation.
TDS_MSG_MIG_REQ	Server	Initiate a connection migration. The parameters to this message are interpreted as the results of a directory service lookup of a server. The format of the parameters is documented in the **TDS_EED** comments section (page 220)
TDS_MSG_MIG_SYNC	Client	Client acknowledgement of the server's migration request. Once the client sends this, it cannot send any further information to the server except to signal an attention, send a **TDS_LOGOUT**, or notify the server of a migration failure via **TDS_MSG_MIG_FAIL**. There are no parameters.

Table 5: Redirection and Migration Related Messages

Name	Client/ Server	Description
TDS_MSG_MIG_CONT	Server	Server has received client data up to **TDS_MSG_MIG_SYNC** and wants the client to commence migration to previously sent query entries. There are no parameters.
TDS_MSG_MIG_IGN	Server	Server wants the client to ignore the previous **TDS_MSG_MIG_REQ** token. There are no parameters.
TDS_MSG_MIG_FAIL	Client	Client's attempt to migrate to another server failed. This message may have a parameter which should indicate why the migration failed. This parameter value has not been defined.

The **TDS_MSG_HAFAILOVER** message is used during login. It is sent via the negotiated login mechanisms. The server sends the session id and the client echos this back to the server during login.

TDS_MSG_MIG_xxx messages are only used in **TDS_BUF_MIGRATE** packets. This allows migration messages to be interpreted outside the normal data stream.

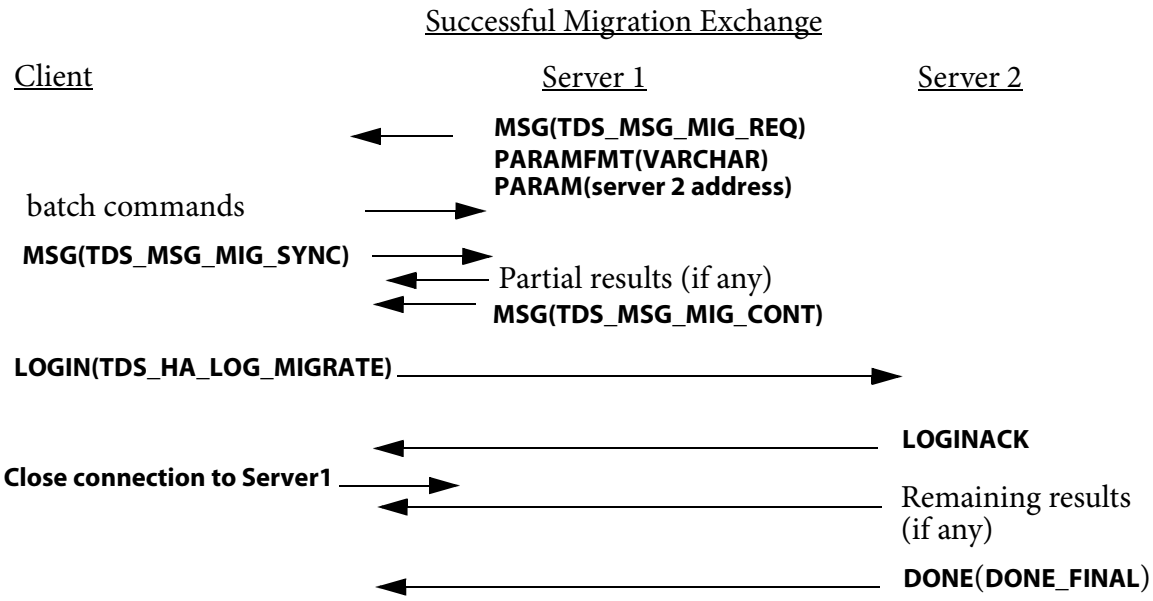
25. Login Redirection

In the lhalogin field, a client can set the **TDS_HA_LOG_REDIRECT** bit to indicate that the client library understands the login redirection protocol. If the server sees this bit set, then it may send a **TDS_EED** with **TDS_EED_INFO** set in the status argument and a message number of **TDS_REDIRECT** to the client. This message requires at least a single argument which will be used by the client to connect to and restart the login process. Unless a client has set the **TDS_CAP_CLUSTERFAILOVER** capability, the state value should have **TDS_EED_IMMEDIATE_REDIRECT** set. The server can suggest to the client that it continue to set **TDS_HA_LOG_REDIRECT** by also setting **TDS_EED_SET_REDIRECT** in the state value field. When the client sees this message, it should close the current connection and attempt to connect to the first server it can as specified via the message parameters. If the client is HA enabled and has set the **TDS_CAP_CLUSTERFAILOVER** bit, then the server can send this message with the **TDS_EED_IMMEDIATE_REDIRECT** clear. The client will then use the message parameters as failover targets instead of the **HAFAILOVER** entry from the directory service.

26. Migration Protocol

If a client indicates at login that it is capable of migration and the server might need to migrate the client, the server should send TDS_MSG_HAFAILOVER with a session id to be used during migration. Note that this same session id is used for HA-Failover.The server is responsible for initiating a connection migration. The following diagrams will attempt to show possible migration interaction scenarios. Note that all **TDS_MSG**s related to migration must be sent in a packet with a buffer type of **TDS_BUF_MIGRATE.** Furthermore, the migration protocol should be viewed as an interruption of the normal token exchange. In particular, once the client sends the **TDS_MSG_MIG_SYNC**, it should not send any token other than an attention, **TDS_LOGOUT**, or **TDS_MSG_MIG_FAIL**.

The first diagram shows a successful migration token exchange.

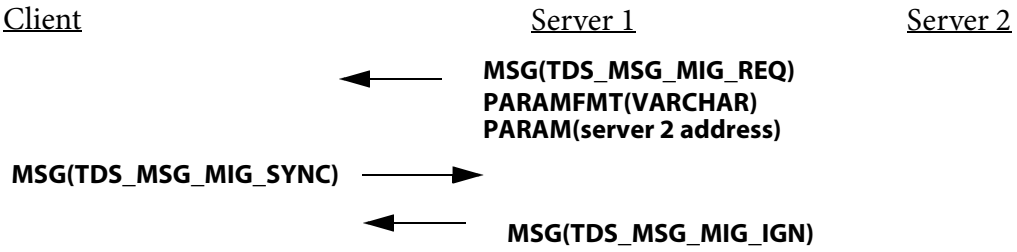


Note that when the client closes the connection to Server 1, it does not send a **TDS_LOGOUT** token. The migration protocol is best thought of as an interruption of the current token stream. Because of the structure of the TDS protocol, many clients will not see the migration message until they are processing batch results. Because of this, a client will need to suspend the TDS processing state once it receives **TDS_MSG_MIG_CONT** and move to a login state to connect to Server 2. Server 1 is responsible for forwarding any relevant batch commands to Server 2 as part of the migration. Server 1 is allowed to satisfy the current batch from the client before sending **TDS_MSG_MIG_CONT**. Once Server 1 sends **TDS_MSG_MIG_CONT**, it should not send any further data to the Client unless the client sends an attention or **TDS_MSG_MIG_FAIL**. Also note that regardless of whether Server 1

completely satisfies the Client's active batch results or not, the final **TDS_DONE** should be sent by Server 2 in this case.

The next diagram shows a aborted migration request. In this case, the server had sent a migration

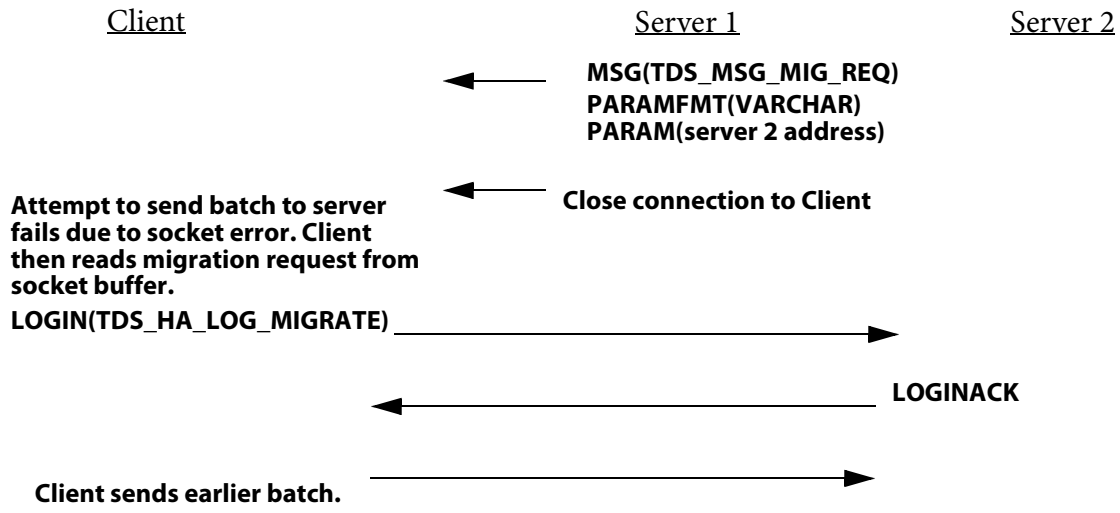
Aborted Migration Exchange



request. By the time the server received **TDS_MSG_MIG_SYNC**, the server decided that the migration was no longer needed. In this case the server sends **TDS_MSG_MIG_IGN**.

Under certain circumstances, the server may not be able to wait for the **TDS_MSG_MIG_SYNC** to be sent by the client. In these cases, after the **TDS_MSG_MIG_REQ** has been sent, it can close the connection. The client will be expected to check that the socket does not have any pending data before reporting network connection failure. If the client has received a migration request from the server and detects a socket failure, then it is expected to attempt a migration to new system. Should the migration fail, then a network connection error should be reported back to the application. It is the servers responsibility to ensure that all relevant client context is moved between servers. Note

Closed Migration Exchange

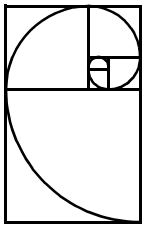


that the server does NOT send a **TDS_DONE(FINAL)** in this situation. Server 2 will send a **TDS_DONE(FINAL)** in any situation where the migration interrupts returning results during a batch.

Within the context of a SQL server, the following migration types have been identified (in order of decreasing complexity):

- In the middle of a multi-batch transaction
- In the middle of a batch (i.e., after sending some results, but before completing all requests within the client's batch.)
- At the end of a batch before sending **TDS_DONE**(**TDS_DONE_FINAL**) (post-batch)
- At the start of a batch after receiving client data but before sending results (pre-batch)
- Idle connection

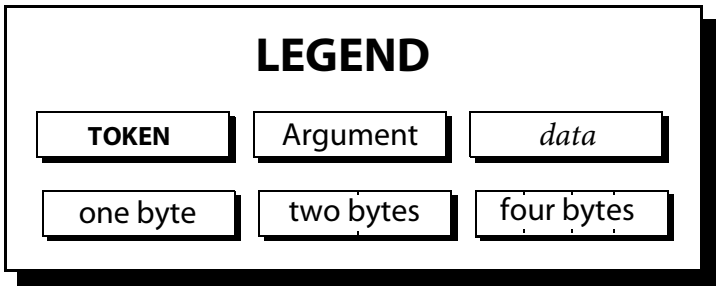
Since a server makes the determination to migrate a client, it is free to restrict which clients can be migrated. Also note that for many connections, the Idle connection type will devolve into the pre-batch case. This means that a server must be prepared to save all outstanding client transmissions up to the point it receives the **TDS_MSG_MIG_SYNC**. For the migration to succeed, all data the client has transmitted is considered to be part of the client context within the server. Should the server decide to, it can process the client's batch and start sending results before sending the **TDS_MSG_MIG_CONT**. The server may also decide to cancel the migration and then restart it to a different target system based on the contents of the client's batch. It is extremely important that the migration be as transparent as possible to the client application.



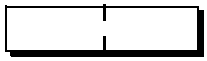
TDS 5.0 Reference Pages

Each TDS token has a reference page which provides a detailed description of the format of the token's data stream and of its usage. Each reference page contains a graphic description of the data stream's syntax, comments on various aspects of its usage, and a detailed description of each argument.

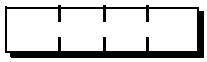
In most cases, the graphic syntax gives enough information to be used as a quick reference to the datastream. This is the legend for the graphics.



A box without any marks indicates a one byte argument.



A box with one pair of marks indicates a two byte argument.



A box with three pairs of marks indicates a four byte argument.

TOKEN — A bold-faced font indicates the TDS token for the data stream.

Argument — A Helvetica font indicates that the argument is part of the data stream description but not part of the actual data of the data stream.

data — An italic font indicates that this argument is replaced by actual data in the data stream.

All multi-byte length fields in the data streams are sent in the client's byte order. The server receiving the token converts the length field as required.

TDS Token List

This is a complete list of all assigned TDS tokens, not including the data type tokens. See the data type man page for a complete list of the data type tokens.

TDS_ALTCONTROL . . . . . .0xAF (obsolete)
TDS_ALTFMT 0xA8
TDS_ALTFMT2 . . . . . .0x?? (token not implemented yet)
TDS_ALTNAME 0xA7
TDS_ALTROW. 0xD3
TDS_ALTROW2 . . . . . .0x?? (token not implemented yet)
TDS_CAPABILITY. 0xE2
TDS_COLFMT . . . . . .0xA1 (obsolete)
TDS_COLFMTOLD . . . . . .0x2A (obsolete)
TDS_COLINFO 0xA5
TDS_COLINFO2 . . . . . .0x20
TDS_COLNAME . . . . . .0xA0 (obsolete)
TDS_CONTROL. 0xAE
TDS_CURCLOSE. 0x80
TDS_CURDECLARE 0x86
TDS_CURDECLARE2 . . . . . .0x23
TDS_CURDECLARE3 . . . . . .0x10
TDS_CURDELETE. 0x81
TDS_CURFETCH. 0x82
TDS_CURINFO 0x83
TDS_CURINFO2 . . . . . .0x87
TDS_CURINFO3 . . . . . .0x88
TDS_CUROPEN. 0x84
TDS_CURUPDATE 0x85
TDS_DBRPC 0xE6
TDS_DBRPC2 . . . . . .0xE8
TDS_DEBUGCMD. . . . . .0x60 (obsolete)
TDS_DONE 0xFD
TDS_DONEINPROC 0xFF
TDS_DONEPROC 0xFE

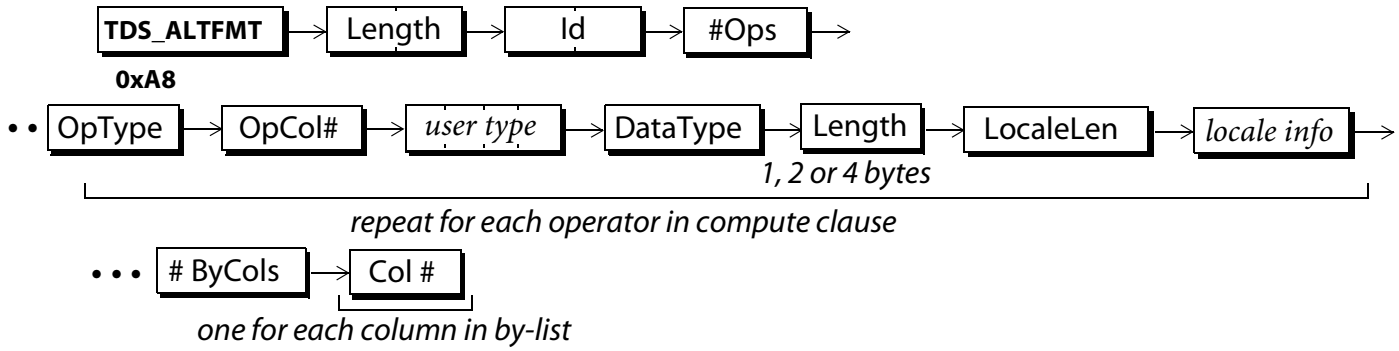
TDS_DYNAMIC.....0xE7
TDS_DYNAMIC2 .....0x62
TDS_EED 0xE5
TDS_ENVCHANGE.....0xE3
TDS_ERROR.....0xAA (obsolete)
TDS_EVENTNOTICE 0xA2
TDS_INFO .....0xAB (obsolete)
TDS_KEY 0xCA
TDS_LANGUAGE 0x21
TDS_LOGINACK.....0xAD
TDS_LOGOUT.....0x71
TDS_MSG.....0x65
TDS_OFFSET.....0x78
TDS_OPTIONCMD.....0xA6
TDS_OPTIONCMD2.....0x63 (reserved)
TDS_ORDERBY.....0xA9
TDS_ORDERBY2 .....0x22
TDS_PARAMFMT.....0xEC
TDS_PARAMFMT2 .....0x20
TDS_PARAMS.....0xD7
TDS_PROCID.....0x7C (obsolete)
TDS_RETURNSTATUS.....0x79
TDS_RETURNVALUE.....0xAC (obsolete)
TDS_RPC .....0xE0 (obsolete)
TDS_ROW 0xD1
TDS_ROWFMT 0xEE
TDS_ROWFMT2 .....0x61
TDS_TABNAME 0xA4

TDS_ALTFMT

Function

The data stream for describing the data type, length, and status of **COMPUTE** data.

Syntax



Arguments

TDS_ALTFMT This is the data stream token that indicates that this is a data stream containing a description of compute data. This token is one byte and has the value **0xA8**.

Length This length specifies the number of bytes remaining in the data stream. It is an unsigned, two-byte integer.

Id This is the id which identifies the compute statement to which the compute column formats apply. Because a Transact-SQL statement may have more than one compute clause, the id is necessary. The id is used later in order to correctly interpret the compute row data which comes in the **TDS_ALTROW** data stream. Id is a two-byte, unsigned integer.

Ops This is the number of aggregate operators in the compute clause. For example, the clause *"compute count(x), min(x), max(x)"* has three aggregate operators. This field is a one-byte, unsigned integer.

OpType This is the type of aggregate operator. The operands for the aggregate are described by the # ByCols and Col # fields. The possible operators are:

Table 6: Aggregate Operator Types

Operator Name	Operator Value	Description
TDS_ALT_AVG	0x4F	The average value.
TDS_ALT_COUNT	0x4B	The summary count value.
TDS_ALT_MAX	0x52	The maximum value.
TDS_ALT_MIN	0x51	The minimum value.
TDS_ALT_SUM	0x4D	The sum value.

OpCol# This is the column number associated with OpType. The first column in the select list is 1. This argument is a one-byte, unsigned integer.

user type This is the user-defined datatype of the data. It is a signed, four-byte integer.

DataType This is the data type of the data and is a one-byte unsigned integer. Fixed length datatypes are represented by a single datatype byte and have no following Length argument. Variable length datatypes are followed by Length which gives the maximum datatype length, in bytes.

Length This is the maximum length, in bytes, of DataType. The size of Length depends on the datatype. This argument only exists for variable length datatypes.

LocaleLen This is the length of the localization information. It is a one-byte, unsigned integer which may have a value of 0. If LocaleLen is 0, no localization information follows.

locale info This is the localization information for the column. It is a character string of LocaleLen bytes. This argument only exists if the LocaleLen argument is not equal to 0.

ByCols This is the number of columns in the by-list of the compute clause. For example, the compute clause "**compute count(sales) by year, month, division"** has three by-columns. It is legal to have no by-columns. In that case, # ByCols is 0. The argument is a one-byte, unsigned integer.

Col # When there are by-columns in a compute (#ByCols not equal to 0), there is one Col# argument for each select column listed in the by-columns clause. For example, "**select a, b, c order by b, a compute sum(a) by b, a"** will return # ByCols as 2 followed by Col# 2 and Col# 1. The first column number is 1. This argument is a one-byte, unsigned integer.

Comments

This is the data stream used to describe the format of a compute clause.

- A compute clause may have multiple operators.
- A compute clause may have only one by-list.
- A Transact-SQL statement may have multiple compute clauses.
- Each compute clause is described by a separate **TDS_ALTFMT** data stream.
- The information in **TDS_ALTFMT** describes the data in the **TDS_ALTROW** data stream.

Examples

See Also

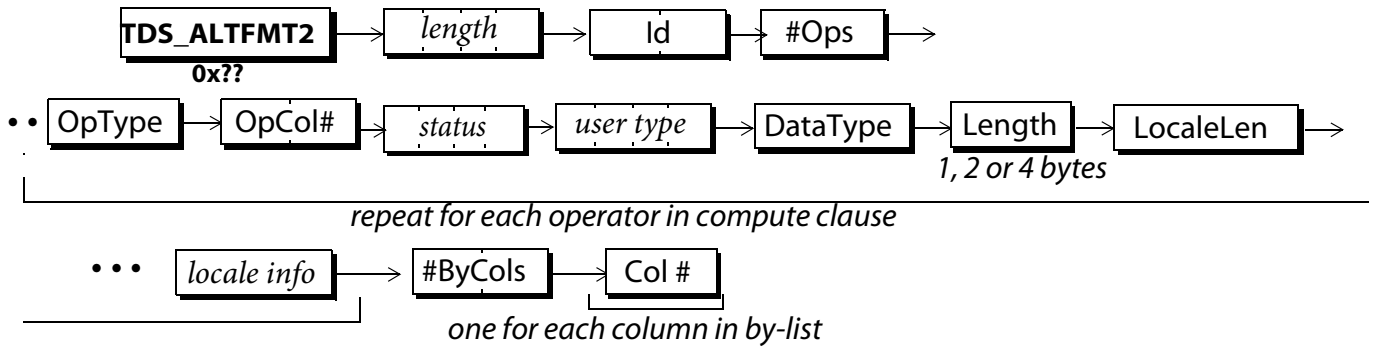
TDS_ALTNAME, **TDS_ALTROW**

TDS_ALTFMT2

Function

The data stream for describing the data type, length, and status of **COMPUTE** data.

Syntax



Arguments

TDS_ALTFMT2 This is the data stream token that indicates that this is a data stream containing a description of compute data. This token is one byte and has the value **0x??**.

length This length specifies the number of bytes remaining in the data stream. It is an unsigned, four-byte integer.

Id This is the id which identifies the compute statement to which the compute column formats apply. Because a Transact-SQL statement may have more than one compute clause, the id is necessary. The id is used later in order to correctly interpret the compute row data which comes in the **TDS_ALTROW** data stream. Id is a two-byte, unsigned integer.

Ops This is the number of aggregate operators in the compute clause. For example, the clause *"compute count(x), min(x), max(x)"* has three aggregate operators. This field is a one-byte, unsigned integer.

OpType This is the type of aggregate operator. The operands for the aggregate are described by the # ByCols and Col # fields. The possible operators are:

Table 7: Aggregate Operator Types

Operator Name	Operator Value	Description
TDS_ALT_AVG	0x4F	The average value.
TDS_ALT_COUNT	0x4B	The summary count value.
TDS_ALT_MAX	0x52	The maximum value.
TDS_ALT_MIN	0x51	The minimum value.
TDS_ALT_SUM	0x4D	The sum value.

OpCol# This is the column number associated with OpType. The first column in the select list is 1. This argument is a one-byte, unsigned integer.

status This field is used to provide additional information about the datastream, such as if the columnstatus byte is present within the corresponding TDS_ALTROW datastream.

Table 8: Valid Status Values

Name	Value	Description
TDS_ALTFMT_COLUMNSTATUS	0x08	Indicates whether column-status bit is included in corresponding TDS_ALTROW datastream.

user type This is the user-defined datatype of the data. It is a signed, four-byte integer.

DataType This is the data type of the data and is a one-byte unsigned integer. Fixed length datatypes are represented by a single datatype byte and have no following Length argument. Variable length datatypes are followed by Length which gives the maximum datatype length, in bytes.

Length This is the maximum length, in bytes, of DataType. The size of Length depends on the datatype. This argument only exists for variable length datatypes.

LocaleLen This is the length of the localization information. It is a one-byte, unsigned integer which may have a value of 0. If LocaleLen is 0, no localization information follows.

locale info This is the localization information for the column. It is a character string of LocaleLen bytes. This argument only exists if the LocaleLen argument is not equal to 0.

#ByCols This is the number of columns in the by-list of the compute clause. For example, the compute clause "**compute count(sales) by year, month, division"** has three by-columns. It is legal to have no by-columns. In that case, # ByCols is 0. The argument is a two-byte, unsigned integer.

Col # When there are by-columns in a compute (#ByCols not equal to 0), there is one Col# argument for each select column listed in the by-columns clause. For example, "**select a, b, c order by b, a compute sum(a) by b, a"** will return # ByCols as 2 followed by Col# 2 and Col# 1. The first column number is 1. This argument is a one-byte, unsigned integer.

Comments

This is the data stream used to describe the format of a compute clause.

- As of TDS revision 3.5, this token was added to the spec only as a 'place-holder', meaning there are no plans to implement it currently. The token does provide a way for the widetable and columnstatus byte features to be properly implemented, so if a business case arises for either feature, this token can be implemented in the future.

- A compute clause may have multiple operators.
- A compute clause may have only one by-list.
- A Transact-SQL statement may have multiple compute clauses.
- Each compute clause is described by a separate **TDS_ALTFMT2** data stream.
- The information in **TDS_ALTFMT2** describes the data in the **TDS_ALTROW2** data stream.

Examples

See Also

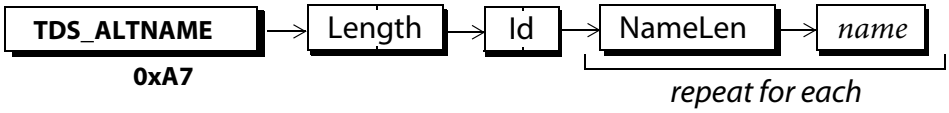
TDS_ALTNAME, **TDS_ALTROW, TDS_ALTROW2**

TDS_ALTNAME

Function

Describes the number and name of a compute clause.

Syntax



Arguments

TDS_ALTNAME This token indicates that this datastream describes a compute clause. The token's length is one byte and it's value is **0xA7**.

Length This is the total length, in bytes, of the remaining data stream. It is a two-byte, unsigned integer.

Id This is the id of the compute clause being described. It is legal for a Transact-SQL statement to have multiple compute clauses. The id is used to associate **TDS_ALTNAME**, **TDS_ALTFMT**, and **TDS_ALTROW** data streams. The field is a two-byte unsigned integer.

NameLen This the length, in bytes, of the name or heading for each of the aggregate operators in the compute clause. Aggregate operators are not required to have headings and usually don't. In the null heading case, NameLen will be 0 and no name field will follow. There is a NameLen for each operator in a compute clause.

name This is the compute clause heading. This argument is NameLen bytes long. If NameLen is 0, this argument does not exist.

Comments

- This token is used to describe the number of aggregate operators in a compute clause. It optionally associates names with each of the aggregate operators.

- There may be more than one compute statement in a Transact-SQL compute clause. Each compute clause is assigned an Id. Id is used to associate the **TDS_ALTFMT** and **TDS_ALTROW** data streams.
- All **TDS_ALTNAME** data streams are grouped together and precede any **TDS_ALTFMT** data streams. If there is more than one compute statement, all the **TDS_ALTNAME** data streams for the compute come first, followed by the **TDS_ALTFMT** data streams.

Examples

See Also

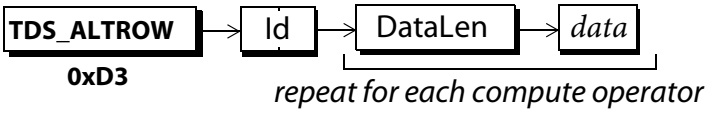
TDS_ALTFMT, TDS_ALTROW

TDS_ALTROW

Function

A row of data for a compute clause.

Syntax



Arguments

TDS_ALTROW This token indicates that this is a data stream containing data for a compute clause. This is a one byte with a value of **0xD3**.

Id This is the id of the compute clause data. It is legal for a Transact-SQL statement to have multiple compute clauses. The id is used to associate **TDS_ALTNAME**, **TDS_ALTFMT**, and **TDS_ALTROW** data streams. The field is a two-byte unsigned integer.

DataLen This is the length, in bytes, of the data. This field is optional, depending on the datatype of the following data.The details for representing TDS datatypes in a data stream are covered in the Datatypes reference page.

data This is the actual data of the compute clause. It's format is identical to a **TDS_ROW** data stream. Each aggregate operator in the compute clause is represented in the data stream as a column.

The data received is always in the native format of the client machine. For example, if integers are represented differently on the server than on the client, the server will perform any conversion before sending data.

Comments

- An **TDS_ALTROW** includes a complete row of compute data. It is in the format described by the **TDS_ALTFMT** data stream for a particular compute clause.

- An **TDS_ALTROW** data stream consists of DataLen and data pairs, one for each aggregate operator in the compute clause. The DataLen argument is only included for variable length and nullable datatypes.
- An **TDS_ALTROW** data stream is identical to a **TDS_ROW** data stream except that it has an Id field following the **TDS_ALTROW** token. Because there may be more than one compute clause in a Transact-SQL statement, each compute clause is given a unique Id. This Id is used to associate all TDS **TDS_ALT*** data streams.

Examples

See Also

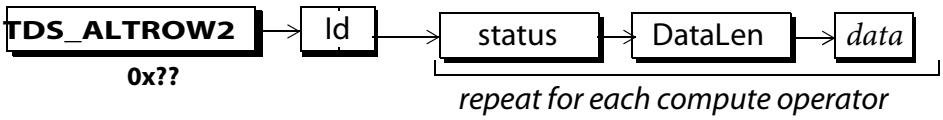
TDS_ALTFMT, TDS_ALTNAME, TDS_ROW

TDS_ALTROW2

Function

A row of data for a compute clause.

Syntax



Arguments

TDS_ALTROW2 This token indicates that this is a data stream containing data for a compute clause. This is a one byte with a value of **0x??**.

Id This is the id of the compute clause data. It is legal for a Transact-SQL statement to have multiple compute clauses. The id is used to associate **TDS_ALTNAME**, **TDS_ALTFMT2**, and **TDS_ALTROW2** data streams. The field is a two-byte unsigned integer.

status This is the columnstatus byte. For information on the usage and meaning of this byte, see the TDS_Datatypes chapter.

DataLen This is the length, in bytes, of the data. This field is optional, depending on the datatype of the following data.The details for representing TDS datatypes in a data stream are covered in the Datatypes reference page.

data This is the actual data of the compute clause. Its format is identical to a **TDS_ROW** data stream. Each aggregate operator in the compute clause is represented in the data stream as a column.

The data received is always in the native format of the client machine. For example, if integers are represented differently on the server than on the client, the server will perform any conversion before sending data.

Comments

- The **TDS_ALTROW2** token is, as of the 3.5 edition of this spec, only a place-holder. It is meant to be used in conjunction with the **TDS_ALTFMT2** token. It is exactly the same as the **TDS_ALTROW** token, except that it contains the columnstatus byte. Note that this token will not contain the columnstatus byte if the corresponding **TDS_ALTFMT2** token's status bytes do not have the TDS_COLUMNSTATUS bit on.
- A **TDS_ALTROW2** includes a complete row of compute data. It is in the format described by the **TDS_ALTFMT2** data stream for a particular compute clause.
- A **TDS_ALTROW2** data stream consists of DataLen and data pairs, one for each aggregate operator in the compute clause. The DataLen argument is only included for variable length and nullable datatypes.
- A **TDS_ALTROW2** data stream is identical to a **TDS_ROW** data stream except that it has an Id field following the **TDS_ALTROW2** token. Because there may be more than one compute clause in a Transact-SQL statement, each compute clause is given a unique Id. This Id is used to associate all TDS **TDS_ALT*** data streams.

Examples

See Also

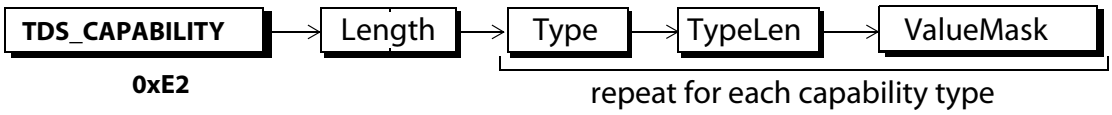
TDS_ALTFMT2, TDS_ALTFMT, TDS_ALTNAME, TDS_ROW

TDS_CAPABILITY

Function

Exchange client and server capabilities during dialog establishment.

Syntax



Arguments

TDS_CAPABILITY This token indicates that this data stream contains a list of capabilities.

Length This is the length, in bytes, of the remaining data stream for this token. This argument is a two-byte, unsigned integer.

Type This is the Type to which the following value mask refers. Capabilities are grouped by Types. This argument is a one-byte, unsigned integer. The supported capability types are:

Table 9: Capability Types

Type	Value	Description
TDS_CAP_REQUEST	1	Requests and data types that can be sent on this dialog.
TDS_CAP_RESPONSE	2	Responses and data types that should not be sent on this dialog.

TypeLen This is the length of ValueMask.

ValueMask ValueMask contains the bit-field encoded capabilities being reported in the data stream. The first byte in the ValueMask contains the high order capability bits. The last byte in the ValueMask contains the low order capability bits.

Comments

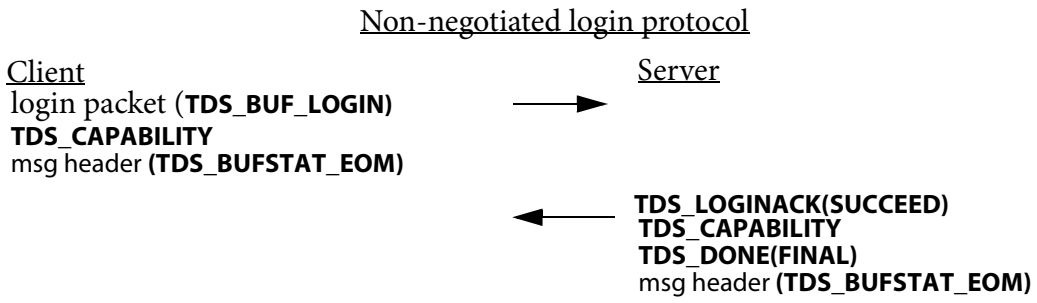
- When a client sends a login request to a server it sends a list of capabilities that it requires on the dialog. A client does not have to send all known capabilities to a server, only those it considers important.
- A server must respond to all capability requests from a client.
- The **TDS_CAPABILITY** data stream from a client is optional. It does not have to be sent. If no capability data is sent by a client, the behavior of the server with respect to TDS capabilities is undefined. The **TDS_CAPABILITY** data stream is determined to be in a login packet using the length field in the packet header.
- The **TDS_CAPABILITY** data stream is sent by a client following the login record. The server responds to the capability data stream following the **TDS_LOGINACK(SUCCEED)** token.

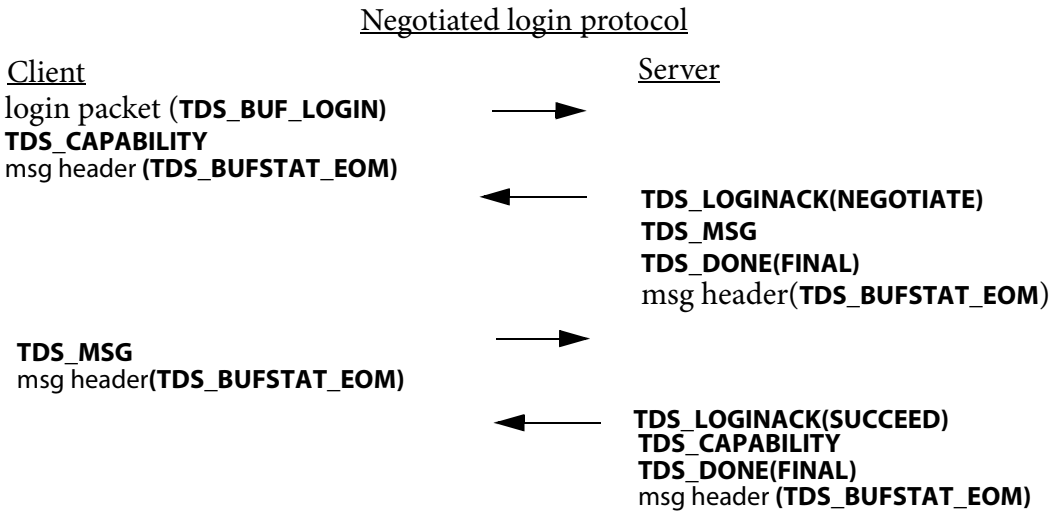
Question: There is a problem with withholding the CAPABILITY response from the server until after LOGINACK(SUCCEED) The client logically needs to know some of the datatype/parameter capabilities of the server in order to send the TDS_MSG, TDS_PARAMFMT, TDS_PARAM sequences which may required during login negotiation. (i.e. what if a client wants to send a JAVA_OBJECT parameter...? The server has not denied that datatype yet.) At a minimum we need to document the range of parameter types and token types which MUST be supported by any server before it can participate in a negotiated login sequence.

- If a negotiated login is being done on the dialog, the capability data stream only follows the **TDS_LOGINACK(SUCCEED)** token, not the **TDS_LOGINACK(NEGOTIATE)**. A capability response never follows a **TDS_LOGINACK(NEGOTIATE)** token.
- Most capabilities are not in affect until completion of the login sequence. However, there are some capabilities, such as TDS_RES_NOEED, which must be considered during the login sequence (this capability bit affects whether the server will send TDS_EED or TDS_INFO to the client -- and since these tokens can go to the client during the login sequence, the capability bit must be taken into account).

- There are some login actions which require that TDS_EED be used; for example, login redirection. In this case, if the client was to set TDS_RES_NOEED, then login redirection cannot occur. If the client application sets TDS_RES_NOEED, then it cannot request features, such as login redirection, that depend on the TDS_EED token and the server should disable those features that would require a TDS_EED be sent to the client.
- Capabilities are used on all client dialogs, both client to server and server to server.
- Capabilities are only exchanged during the login sequence. Client libraries must save a list of capabilities supported on a dialog in case the client application requests the current capabilities following the login sequence. It is illegal to send a **TDS_CAPABILITY** token following a successful login sequence

Protocol Description





- A client uses the ValueMask in the **TDS_CAPABILITY** data stream a follows:
  - setting a ValueMask bit to 1 for a **TDS_CAP_REQUEST** capability indicates that the client is requesting the server to support this capability.
  - setting a ValueMask bit to 0 for a **TDS_CAP_REQUEST** capability indicates that the client does not require support for this request type on this dialog.
  - setting a ValueMask bit to 1 for a **TDS_CAP_RESPONSE** capability indicates that the client is requesting the server to withhold this response type on this dialog.
  - setting a ValueMask bit to 0 for a **TDS_CAP_RESPONSE** capability indicates that the client is willing to receive this response type on this dialog.
- Servers use the ValueMask in the **TDS_CAPABILITY** data stream as follows:
  - converting a client's 1 bit in a **TDS_CAP_REQUEST** ValueMask to a 0 indicates that the server cannot support this request capability.
  - converting a client's 1 bit in a **TDS_CAP_RESPONSE** ValueMask to a 0 indicates that the server is not willing to withhold this response/data type from a client.

- If a server does not understand a capability Type it should set all bits to 0 in the ValueMask. This indicates to the client that the server cannot support or withhold any of these capabilities.
- If a server does not understand a bit in a ValueMask it should set this bit to 0 to indicate that it cannot support or withhold this capability.

Capabilities

The tables below summarize all of the supported request and response capabilities supported in TDS 5.0.

Table 10: TDS_CAP_REQUEST Capabilities

Name	Value	Description
TDS_REQ_LANG	1	Language requests
TDS_REQ_RPC	2	RPC requests
TDS_REQ_EVT	3	Registered procedure event notification
TDS_REQ_MSTMT	4	Support multiple commands per request
TDS_REQ_BCP	5	Bulk copy requests
TDS_REQ_CURSOR	6	Cursor command requests
TDS_REQ_DYNF	7	Dynamic SQL requests
TDS_REQ_MSG	8	**TDS_MSG** requests
TDS_REQ_PARAM	9	RPC requests will use the **TDS_DBRPC** token and **TDS_PARAMFMT**/**TDS_PARAM** to send parameters.
TDS_DATA_INT1	10	Support 1 byte unsigned integers
TDS_DATA_INT2	11	Support 2 byte integers
TDS_DATA_INT4	12	Support 4 byte integers
TDS_DATA_BIT	13	Support bit data types
TDS_DATA_CHAR	14	Support fixed length character data types

Table 10: TDS_CAP_REQUEST Capabilities

Name	Value	Description
TDS_DATA_VCHAR	15	Support variable length character data types
TDS_DATA_BIN	16	Support fixed length character data types
TDS_DATA_VBIN	17	Support variable length binary data types
TDS_DATA_MNY8	18	Support 8 byte money data types
TDS_DATA_MNY4	19	Support 4 byte money data types
TDS_DATA_DATE8	20	Support 8 byte date/time data types
TDS_DATA_DATE4	21	Support 4 byte date/time data types
TDS_DATA_FLT4	22	Support 4 byte floating point data types
TDS_DATA_FLT8	23	Support 8 byte floating point data types
TDS_DATA_NUM	24	Support numeric data types
TDS_DATA_TEXT	25	Support text data types
TDS_DATA_IMAGE	26	Support image data types
TDS_DATA_DEC	27	Support decimal data types
TDS_DATA_LCHAR	28	Support long variable length character data types
TDS_DATA_LBIN	29	Support long variable length binary data types.
TDS_DATA_INTN	30	Support NULL integers
TDS_DATA_DATETIMEN	31	Support NULL date/time
TDS_DATA_MONEYN	32	Support NULL money
TDS_CSR_PREV	33	Obsolete, will not be used.
TDS_CSR_FIRST	34	Obsolete, will not be used.

Table 10: TDS_CAP_REQUEST Capabilities

Name	Value	Description
TDS_CSR_LAST	35	Obsolete, will not be used.
TDS_CSR_ABS	36	Obsolete, will not be used.
TDS_CSR_REL	37	Obsolete, will not be used.
TDS_CSR_MULTI	38	This is possibly obsolete.
TDS_CON_OOB	39	Support expedited attentions
TDS_CON_INBAND	40	Support non-expedited attentions
TDS_CON_LOGICAL	41	Support logical logout (not supported in this release)
TDS_PROTO_TEXT	42	Support tokenized text and image (not supported in this release)
TDS_PROTO_BULK	43	Support tokenized bulk copy (not supported this release)
TDS_REQ_URGEVT	44	Use new event notification protocol
TDS_DATA_SENSITIVITY	45	Support sensitivity security data types
TDS_DATA_BOUNDARY	46	Support boundary security data types
TDS_PROTO_DYNAMIC	47	Use DESCIN/DESCOUT dynamic protocol
TDS_PROTO_DYNPROC	48	Pre-pend “create proc” to dynamic prepare statements
TDS_DATA_FLTN	49	Support NULL floats
TDS_DATA_BITN	50	Support NULL bits
TDS_DATA_INT8	51	Support 8 byte integers
TDS_DATA_VOID	52	?
TDS_DOL_BULK	53	?

Table 10: TDS_CAP_REQUEST Capabilities

Name	Value	Description
TDS_OBJECT_JAVA1	54	Support Serialized Java Objects
TDS_OBJECT_CHAR	55	Support Streaming character data
RESERVED	56	Reserved for future use
TDS_OBJECT_BINARY	57	Streaming Binary data
TDS_DATA_COLUMNSTATUS	58	Indicates that a one-byte status field can follow any length or data (etc.) for every column within a row using TDS_ROW or TDS_PARAMS. Note that when this capability is on, the ROWFMT* and PARAMFMT* tokens indicate in their status byte fields whether a particular column will contain the columnstatus byte.
TDS_WIDETABLE	59	The client may send requests using the CURDECLARE2, DYNAMIC2, PARAMFMT2 tokens.
RESERVED	60	Reserved
TDS_DATA_UINT2	61	Support for unsigned 2-byte integers
TDS_DATA_UINT4	62	Support for unsigned 4-byte integers
TDS_DATA_UINT8	63	Support for unsigned 8-byte integers
TDS_DATA_UINTN	64	Support for NULL unsigned integers
TDS_CUR_IMPLICIT	65	Support for TDS_CUR_DOPT_IMPLICIT cursor declare option.
TDS_DATA_NLBIN	66	Support for LONGBINARY data containing UTF-16 encoded data (usertypes 34 and 35)

Table 10: TDS_CAP_REQUEST Capabilities

Name	Value	Description
TDS_IMAGE_NCHAR	67	Support for IMAGE data containing UTF-16 encoded data (usertype 36).
TDS_BLOB_NCHAR_16	68	Support for BLOB subtype 0x05 (unichar) with serialization type 0.
TDS_BLOB_NCHAR_8	69	Support for BLOB subtype 0x05 (unichar) with serialization type 1.
TDS_BLOB_NCHAR_SCSU	70	Support for BLOB subtype 0x05 (unichar) with serialization type2.
TDS_DATA_DATE	71	Support for Date
TDS_DATA_TIME	72	Support for Time.
TDS_DATA_INTERVAL	73	Support for Interval
TDS_CSR_SCROLL	74	Support for Scrollable Cursor. This bit must be on for the following four capability bits to have meaning.
TDS_CSR_SENSITIVE	75	Support for Scrollable Sensitive Cursor
TDS_CSR_INSENSITIVE	76	Support for Scrollable Insensitive Cursor
TDS_CSR_SEMISENSITIVE	77	Support for Scrollable Semi-sensitive Cursor
TDS_CSR_KEYSETDRIVEN	78	Support for Scrollable Keyset-driven Cursor
TDS_REQ_SRVPKTSIZE	79	Support for server specified packet size
TDS_DATA_UNITEXT	80	Support for Unicode UTF-16 Text.
TDS_CAP_CLUSTERFAILOVER	81	Support Cluster Failover Extensions.
TDS_DATA_SINT1	82	Support for 1 byte signed integer
TDS_REQ_LARGEIDENT	83	Support for large identifiers

Table 10: TDS_CAP_REQUEST Capabilities

Name	Value	Description
TDS_REQ_BLOB_NCHAR_16	84	Support for BLOB subtype 0x05 (uni-char) with serialization type 0. Replaces **TDS_BLOB_NCHAR_16**. Added to deal with ASE coding issue in old servers.
TDS_DATA_XML	85	Support for XML datatype.
TDS_REQ_CURINFO3	86	Support for **TDS_CURINFO3** token.
TDS_REQ_DBRPC2	87	Support for **TDS_DBRPC2** token.
TDS_REQ_MIGRATE	89	Client can be migrated to another server

Table 11: TDS_CAP_RESPONSE capabilities

Name	Value	Description
TDS_RES_NOMSG	1	No support for **TDS_MSG** results
TDS_RES_NOEED	2	No support for **TDS_EED** token
TDS_RES_NOPARAM	3	No support for **TDS_PARAM**/**TDS_PARAMFMT** for return parameter. use **TDS_RETURNVALUE** to return parameters to this client.
TDS_DATA_NOINT1	4	No support for 1 byte unsigned integers
TDS_DATA_NOINT2	5	No support for 2 byte integers
TDS_DATA_NOINT4	6	No support for 4 byte integers
TDS_DATA_NOBIT	7	No support for bit data types
TDS_DATA_NOCHAR	8	No support for fixed length character data types
TDS_DATA_NOVCHAR	9	No support for variable length character data types

Table 11: TDS_CAP_RESPONSE capabilities

Name	Value	Description
TDS_DATA_NOBIN	10	No support for fixed length binary data types
TDS_DATA_NOVBIN	11	No support for variable length binary data types
TDS_DATA_NOMNY8	12	No support for 8 byte money data types
TDS_DATA_NOMNY4	13	No support for 4 byte money data types
TDS_DATA_NODATE8	14	No support for 8 byte date/time data types
TDS_DATA_NODATE4	15	No support for 4 byte date/time data types
TDS_DATA_NOFLT4	16	No support for 4 byte float data types
TDS_DATA_NOFLT8	17	No support for 8 byte float data types
TDS_DATA_NONUM	18	No support for numeric data types
TDS_DATA_NOTEXT	19	No support for text data types
TDS_DATA_NOIMAGE	20	No support for image data types
TDS_DATA_NODEC	21	No support for decimal data types
TDS_DATA_NOLCHAR	22	No support for long variable length character data types
TDS_DATA_NOLBIN	23	No support for long variable length binary data types
TDS_DATA_INTN	24	No support for nullable integers
TDS_DATA_NODATETIMEN	25	No support for nullable date/time data types
TDS_DATA_NOMONEYN	26	No support for nullable money data types
TDS_CON_NOOOB	27	No support for expedited attentions
TDS_CON_NOINBAND	28	No support for non-expedited attentions

Table 11: TDS_CAP_RESPONSE capabilities

Name	Value	Description
TDS_PROTO_NOTEXT	29	No support for tokenized text and image.
TDS_PROTO_NOBULK	30	No support for tokenized bulk copy
TDS_DATA_NOSENSITIVITY	31	No support for the security sensitivity data type
TDS_DATA_NOBOUNDARY	32	No support for the security boundary data type
TDS_RES_NOTDSDEBUG	33	No support for **TDS_DEBUG** token. Use image data instead.
TDS_RES_NOSTRIPBLANKS	34	Do not strip blank from fixed length character data
TDS_DATA_NOINT8	35	No support for 8 byte integers
TDS_OBJECT_NOJAVA1	36	No Support Serialized Java Objects
TDS_OBJECT_NOCHAR	37	No Support Streaming character data
TDS_DATA_NOCOLUMNSTATUS	38	No Support for the columnstatus byte
TDS_OBJECT_NOBINARY	39	No Streaming Binary data
	40	Reserved for future use
TDS_DATA_NOUINT2	41	No Support for unsigned 2-byte integers
TDS_DATA_NOUINT4	42	No Support for unsigned 4-byte integers
TDS_DATA_NOUINT8	43	No Support for unsigned 8-byte integers
TDS_DATA_NOUINTN	44	No Support for NULL unsigned integers
TDS_NO_WIDETABLES	45	Client cannot process the **TDS_ORDERBY2**, **TDS_PARAMFMT2**, and **TDS_ROWFMT2** tokens required to support tables with a LARGE number of columns. The server should not send them.

Table 11: TDS_CAP_RESPONSE capabilities

Name	Value	Description
TDS_DATA_NONLBIN	46	No Support for LONGBINARY data containing UTF-16 encoded data (usertypes 34 and 35)
TDS_IMAGE_NONCHAR	47	No Support for IMAGE data containing UTF-16 encoded data (usertype 36).
TDS_BLOB_NONCHAR_16	48	No Support for BLOB subtype 0x05/0.
TDS_BLOB_NONCHAR_8	49	No Support for BLOB subtype 0x05/1.
TDS_BLOB_NONCHAR_SCSU	50	No Support for BLOB subtype 0x05/2.
TDS_DATA_NODATE	51	No Support for Date
TDS_DATA_NOTIME	52	No Support for Time.
TDS_DATA_NOINTERVAL	53	No Support for Interval.
TDS_DATA_NOUNITEXT	54	No Support for Unicode UTF-16 Text.
TDS_DATA_NOSINT1	55	No Support for 1 byte signed integers.
TDS_NO_LARGEIDENT	56	No Support for Large Identifiers
TDS_NO_BLOB_NCHAR_16	57	No Support for BLOB subtype 0x05/0. Replaces **TDS_BLOB_NONCHAR_16**. Added to work around ASE coding issue.
TDS_NO_SRVPKTSIZE	58	No Support for Server specified packet size larger than the client requested. If the TDS_REQ_PKTSIZE capability is set, the server may still specify the packet size as long as it is not larger than suggested by the client.
TDS_DATA_NOXML	59	No Support for XML datatype
TDS_NONINT_RETURN_VALUE	60	Client support for non-integer return values from **TDS_RETURNVALUE**

Table 11: TDS_CAP_RESPONSE capabilities

Name	Value	Description
TDS_RES_NOXNLDATA	61	Client does not need additional metadata in **TDS_ROWFMT2**. Also, server should use **TDS_ROWFMT** instead of **TDS_ROWFMT2**.
TDS_RES_SUPPRESS_FMT	62	Server will suppress **TDS_ROWFMT** (or **TDS_ROWFMT2**) tokens if the appropriate status bit is set in **TDS_DYNAMIC**.
TDS_RES_SUPPRESS_DONEINPROC	63	Server can suppress **TDS_DONEINPROC** tokens. This allows **TDS_ROW** to be followed by **TDS_ROWFMT(2)** instead of **TDS_DONEINPROC**/**TDS_ROWFMT(2)**. Final done count will be provided by **TDS_DONEPROC**.
TDS_RES_FORCE_ROWFMT2	64	Force usage of **TDS_ROWFMT2** even if **TDS_ROWFMT** could be used.

See Also

TDS_OPTIONCMD

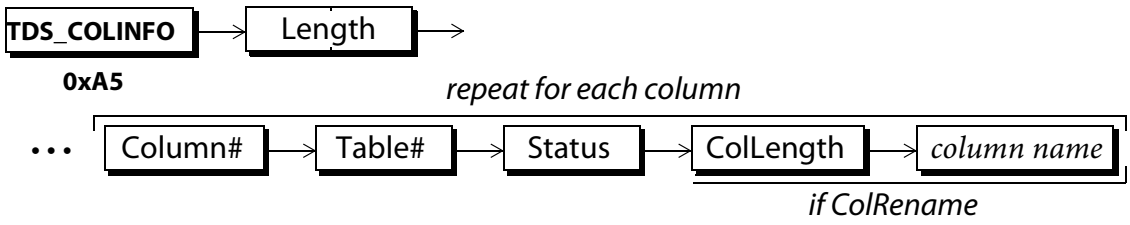
TDS_COLINFO

Function

The data stream used to provide column information for browse mode.

NOTE the select list for a SELECT WITH BROWSE must not contain > 255 columns.

Syntax



Arguments

TDS_COLINFO This token indicates that this is a data stream containing information about columns involved in a **select with** *browse mode*.

Length This is the total length of the remaining **TDS_COLINFO** data stream. It is a two-byte, unsigned integer.

Column# This is the number of the column in the **select** target list to which the column information applies. The number of the first column is 1. Column# is a one-byte, unsigned integer.

Table# This is the number of the table from which the column comes. The tables names are listed in the **TDS_TABNAME** data stream which precedes the **TDS_COLINFO** data stream. The first table in the **TDS_TABNAME** data stream is 1. Table# is a one-byte, unsigned integer.

Status This one-byte, unsigned integer is the status of the current column being described. Every column in the **select** target list is described in the **TDS_COLINFO** data stream.

Table 12: TDS_COLINFO status values

Status Name	Status Value	Description
TDS_STAT_EXPR	0x04	This column is the result of an SQL expression and not an actual column in the underlying table
TDS_STAT_KEY	0x08	The column is part of the row key. It does not have to be part of the **select** target list.
TDS_STAT_HIDDEN	0x10	This column was not in the **select** target list. It is usually not made visible to the client application by the client library. However, it was passed to the client because it is part of the row key. Hidden columns are always key columns.
TDS_STAT_RENAME	0x20	The column name returned for this column in the **select** target list (described in the **TDS_ROWFMT** data stream) is not the column's name in the table. For example, in the statement "**select orderdate = date from order",** the real column name is "date" but the name returned in the **TDS_ROWFMT** data stream was "orderdate". If the column status is **TDS_STAT_RENAME**, the real column name is in the next two arguments of the **TDS_COLINFO** data stream.

ColLength This is the length of the column's real name. Note that this field and the following column name field will appear only if the preceding Status field has **TDS_STAT_RENAME** set. This argument is a one-byte, unsigned integer.

column name This is the column's real name. It's length, in bytes, is given by the ColLength argument. The *columnname* only exists if ColLength is greater than 0.

Comments

- When browse mode is used on a select statement, the server sends back information about the tables and columns involved. With this information, the client library can build a qualification clause for any subsequent update or delete statements.
- All columns needed to make a unique key for a row are returned to the client library. Some of the returned columns may not exist in the select statement's target list. Columns not in the target list are hidden columns. They are usually not returned to the client application by the client library.
- Information for every column in the select list as well as hidden key columns is included in the **TDS_COLINFO** data stream.
- The column name and column name length fields are included only if Status is **TDS_STAT_RENAME**.
- This data stream is always preceded by a **TDS_TABNAME** data stream.
- This data stream is used only for browse mode.
- Browse mode functionality has been replaced by System 10 cursor support. New applications are encourage to use cursors instead of browse mode queries.
- Because the Column# field is only 1 byte wide, this token cannot correctly describe "for browse" results with > 255 columns.

Examples

See Also

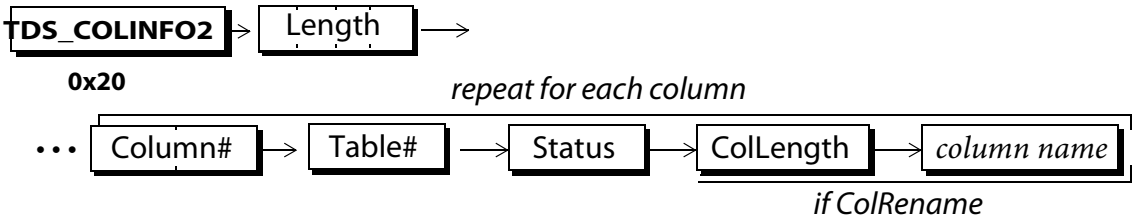
TDS_TABNAME, TDS_ROWFMT

TDS_COLINFO2

Function

The data stream used to provide column information for browse mode. Note that it is identical in purpose and most syntax to the TDS_COLINFO token, except that the Length field is 4 bytes long, and Column# is a 2 byte value. This token was introduced to allow for result sets with > 255 columns.

Syntax



Arguments

TDS_COLINFO This token indicates that this is a data stream containing information about columns involved in a **select with** *browse mode*.

Length This is the total length of the remaining **TDS_COLINFO2** data stream. It is a four-byte, unsigned integer.

Column# This is the number of the column in the **select** target list to which the column information applies. The number of the first column is 1. Column# is a two-byte, unsigned integer.

Table# This is the number of the table from which the column comes. The tables names are listed in the **TDS_TABNAME** data stream which precedes the **TDS_COLINFO2** data stream. The first table in the **TDS_TABNAME** data stream is 1. Table# is a one-byte, unsigned integer.

Status This one-byte, unsigned integer is the status of the current column being described. Every column in the **select** target list is described in the **TDS_COLINFO2** data stream. The meanings are the same as those for TDS_COLINFO.

ColLength This is the length of the column's real name. Note that this field and the following column name field will appear only if the preceding Status field has **TDS_STAT_RENAME** set. This argument is a one-byte, unsigned integer.

column name This is the column's real name. It's length, in bytes, is given by the ColLength argument. The *columnname* only exists if ColLength is greater than 0.

Comments

- Servers should not send this token to clients unless the **TDS_ORDERBY2** Response capability bit is true. This token was added in version 3.4 of this specification to support wide (64K columns) row results. Earlier clients do not support it.
- When browse mode is used on a select statement, the server sends back information about the tables and columns involved. With this information, the client library can build a qualification clause for any subsequent update or delete statements.
- All columns needed to make a unique key for a row are returned to the client library. Some of the returned columns may not exist in the select statement's target list. Columns not in the target list are hidden columns. They are usually not returned to the client application by the client library.
- Information for every column in the select list as well as hidden key columns is included in the **TDS_COLINFO2** data stream.
- The column name and column name length fields are included only if Status is **TDS_STAT_RENAME**.
- This data stream is always preceded by a **TDS_TABNAME** data stream.
- This data stream is used only for browse mode.
- Browse mode functionality has been replaced by System 10 cursor support. New applications are encourage to use cursors instead of browse mode queries.

Examples

See Also

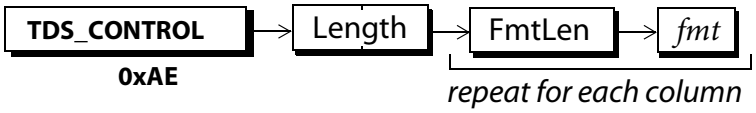
TDS_TABNAME, TDS_ROWFMT

TDS_CONTROL

Function

Describes the user control or format information for columns.

Syntax



Arguments

TDS_CONTROL This token indicates that this is a data stream containing control information.

Length This is the total length, in bytes, of the remaining data stream. It is a two-byte, unsigned integer.

FmtLen This the length, in bytes, of the control information that follows. This is an unsigned one-byte argument.

fmt This is the actual control information for a column. Its length is FmtLen. If FmtLen is 0, this argument doesn't exist in the data stream. The *fmt* field is treated as a binary byte string. There is no character set conversion performed on this argument.

Comments

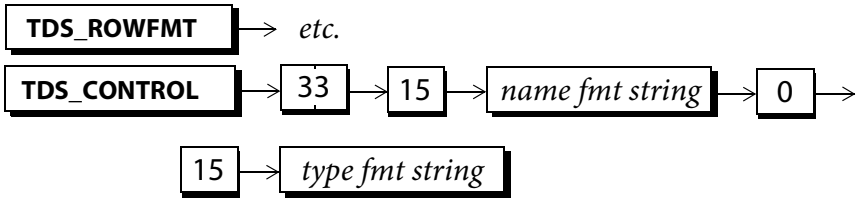
- This data stream is used to tell the client about any user-defined format information for columns. It is used to support a facility in Transact-SQL that allows arbitrary, user-defined information to be associated with **select** target-list columns and then returned to the client.
- The SQL Server option control must be on for a server to return **TDS_CONTROL** data streams.
- This feature is used internally by some Sybase front-end applications. However, it is fairly obscure and normally unused by most customer applications.

Examples

The client sends the following query:

```
select name, id, type from sysobjects
controlrow 0 "name fmt string", "", "type fmt string"
```

The data stream from the server is:



See Also

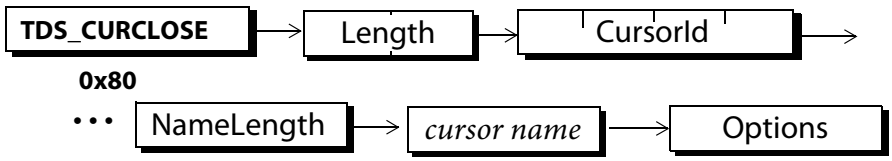
TDS_ROWFMT

TDS_CURCLOSE

Function

Describes the cursor close data stream.

Syntax



Arguments

TDS_CURCLOSE This is the token for a client request to close a cursor.

Length This is the total length of the remaining **TDS_CURCLOSE** data stream. It is a two-byte unsigned integer.

CursorId Cursor id of cursor that is being closed. If CursorId is 0 the cursor is being closed by name. This is a four-byte unsigned integer.

NameLength This is the length of the *cursor name*. It is a one-byte unsigned integer and must be > 0 and <= **TDS_MAX_NAME**. NameLength and cursor name are only included if CursorId is equal to 0.

cursor name This is the name of the cursor. The length of this field is in the NameLength argument.

Options These are the options associated with this cursor close. The value values for this argument are:

Table 13: Cursor close options

Name	Value	Description
TDS_CUR_COPT_UNUSED	0x00	No close options.

Table 13: Cursor close options

Name	Value	Description
TDS_CUR_COPT_DEALLOC	0x01	Close and de-allocate the cursor. The cursor must be re-declared before it can be reopened.

Comments

- This is the data stream generated by a client when a close cursor request is sent to a server.
- A **TDS_CURCLOSE** token can only be sent for a cursor following **TDS_CURDECLARE** and **TDS_CUROPEN** tokens.
- The cursor to close is identified in the **TDS_CURCLOSE** token.
- Multiple **TDS_CURCLOSE** data streams may be sent in the same request.
- The **TDS_CURCLOSE** token is acknowledged with a **TDS_CURINFO** token.
- Two **TDS_CURCLOSE** tokens can only be sent for the same cursor if the first one sent does not have the Option argument set to **TDS_CUR_COPT_DEALLOC**.

Examples

See Also

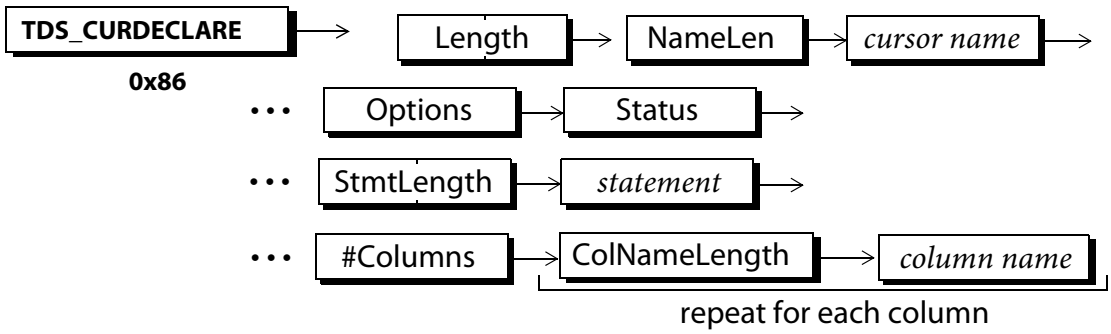
TDS_CUROPEN, TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3

TDS_CURDECLARE

Function

Describes the data stream for declaring a cursor.

Syntax



Arguments

TDS_CURDECLARE This token indicates that this is a data stream containing a client cursor request to declare a cursor.

Length This is the total length of the remaining **TDS_CURDECLARE** data stream. It is a two-byte unsigned integer.

NameLength This is the length of the *cursor name*. It may be up to 255 bytes long; however, a maximum length of **TDS_MAXNAME (30)** is widely supported.

cursor name This is the name of the cursor.

Options — These are the cursor declare options. This is a one-byte unsigned integer.

Table 14: Cursor option values

Option Name	Value	Description
TDS_CUR_DOPT_UNUSED	0x00	No options associated with this cursor.
TDS_CUR_DOPT_RDONLY	0x01	This cursor is read only.
TDS_CUR_DOPT_UPDATABLE	0x02	Updates can be performed with this cursor.
TDS_CUR_DOPT_SENSITIVE	0x04	Reserved, valid with TDS_CURDECLARE3 token only.
TDS_CUR_DOPT_DYNAMIC	0x08	This cursor is being declared against a dynamically prepared statement.
TDS_CUR_DOPT_IMPLICIT	0x10	This cursor is implicitly read-only, automatically fetches first set of rows on the CURSOPEN, and automatically closes after the last row is fetched. This option should not be specified unless the TDS_CUR_IMPLICIT request capability is set.
DO NOT ADD NEW OPTIONS		See **CURDECLARE3**

Status — This is the cursor declare status argument. It is a one-byte unsigned integer.

Table 15: Cursor Declare Status

Name	Value	Description
TDS_CUR_DSTAT_UNUSED	0x00	No status associated with this cursor declare.

Table 15: Cursor Declare Status

Name	Value	Description
TDS_CUR_DSTAT_HASARGS	0x01	The cursor declare statement is followed by parameters.

StmtLen — This is the total length of the following **SELECT** statement associated with this cursor. It is a 2-byte unsigned integer. Please note that since the total **TDS_CURDECLARE** data stream Length may be no greater than 64k-1, StmtLength can never be a full 64k-1. The maximum size of StmtLength depends on the length of the cursor name and the number and length of any update columns.

statement — This is the actual text of the cursor, without the **DECLARE CURSOR** clause. For example, in the following full ANSI cursor declaration, only the words in italics would be the statement argument.

DECLARE CURSOR csr1 FOR
SELECT a, b FROM tab1
WHERE a < 12 AND b > 15
FOR UPDATE OF a

#Columns — When a cursor is declared **FOR UPDATE**, the update columns may be specified. This argument identifies the number of columns specified for update. If this number is > 0, the column (or columns) name length and name follow. This argument is a one-byte unsigned integer. This argument is optional. If its value is 0 then the following arguments are omitted. If this number is > 0, then the statement must NOT contain a **FOR UPDATE OF** or **FOR READ_ONLY** clause. Generally, it is expected that clients will set this to 0 unless the application is using cursors within Open Server.

NameLength When a cursor is declared **FOR UPDATE**, the columns that may be updated can be specified. This, and the following, argument are repeated for each column specified for update. If the previous argument, #Columns, is 0, this argument and the following argument will not be included. Columns are represented by their column name length and column name in the **FOR UPDATE** list. This parameter is a one-byte unsigned integer.

column name This is the name of the column optionally described in the **FOR UPDATE** clause. Its length is described by the NameLength argument.

Comments

- This is the data stream generated by a client to declare a cursor.
- If the **TDS_CURDECLARE** is successful, the client's and server's notion of the current cursor context is changed to be the new cursor. The cursor id assigned by the server for the new cursor will be returned to the client in the **TDS_CURINFO** data stream that acknowledges the cursor declare.
- #Columns refers to the number of columns that are to be updated. Generally, this will be 0 and the column update information will be contained in *statement*. Should this number be > 0 and a **FOR UPDATE** clause be contained in statement or if the server decides that the cursor can only be opened READONLY, the server should generate an error and disallow the cursor declaration.
- *statement* should not contain the **DECLARE** <cursor name> **CURSOR FOR**" clause of a cursor declaration but under the following conditions the Server will report back to the client the READONLY or UPDATABILITY through CURINFO tokens.
  - If #Columns is > 0 then the statement may contain the **UPDATE** [**OF** <column name list>]" clause.
  - If #Columns is 0 then the statement may or may not contain the **UPDATE OF** <column name list>" clause. As explained above, it is the server's responsibility to parse the SQL Statement and determine the number of update columns (if any), even when the #Columns field is set to 0.

  - If Option is **not TDS_CUR_DOPT_RDONLY**, then the statement may contain the "**FOR READ ONLY"** clause.

- Information about the cursor is returned to the client in the **TDS_CURINFO** data stream once the server has received a **declare cursor** token, via cursor command.
- If the declare is successful, the **TDS_ ROWFMT** data stream describing the results will be returned to the client at cursor open time. The **TDS_ROWFMT** data stream for the results will not be returned at declare cursor time.
- A language cursor declare statement may be parameterized. If so, the description of the parameters using a **TDS_PARAMFMT** data stream must follow the **TDS_CURDECLARE** data stream. When the cursor is opened, the parameter values must be passed to the server with a **TDS_PARAMFMT/TDS_PARAMS** data stream following the **TDS_CUROPEN** data stream. Parameterized declare statements are indicated by a Status of **TDS_CUR_DSTAT_HASARGS**.
- The **TDS_CURDECLARE** token can be sent with a **TDS_CUROPEN** and **TDS_CURFETCH** token in the same request. The server will acknowledge each token with a **TDS_CURINFO**, **TDS_DONE(MORE)**, except for the final token (**TDS_CURFETCH**) which is acknowledged with a **TDS_CURINFO**, **TDS_DONE(FINAL)**.

Examples

Valid TDS token streams for TDS_CURDECLARE are as follows:

— A cursor is declared on a dynamic SQL statement, or a cursor is declared on a statement that we know to be executing a stored procedure:

TDS_CURDECLARE

TDS_CUROPEN

TDS_PARAMFMT

TDS_PARAMS

— A cursor is declared on a language statement:

TDS_CURDECLARE

TDS_PARAMFMT

TDS_CUROPEN

TDS_PARAMFMT

TDS_PARAMS

See Also

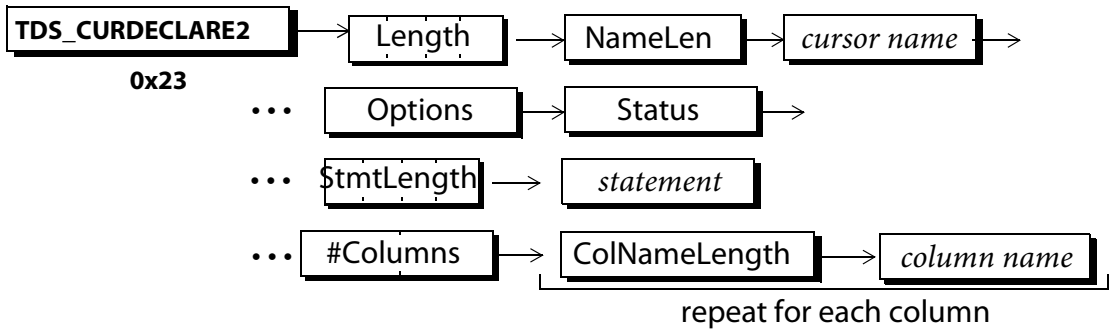
TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_CUROPEN, TDS_CURINFO, TDS_CURINFO2, TDS_CURINFO3, TDS_CURFETCH

TDS_CURDECLARE2

Function

Describes the data stream for declaring a cursor. It serves an identical purpose to TDS_CURDECLARE, but has been widened to support more columns.

Syntax



Arguments

TDS_CURDECLARE2 This token indicates that this is a data stream containing a client cursor request to declare a cursor.

Length This is the total length of the remaining **TDS_CURDECLARE2** data stream. It is a four-byte unsigned integer.

NameLength This is the length of the *cursor name*.It may be up to 255 bytes long; however, a maximum length of **TDS_MAXNAME (30)** is widely supported.

cursor name This is the name of the cursor.

Options These are the cursor declare options. This is a one-byte unsigned integer. Option values are described in the TDS_CURDECLARE section.

Status This is the cursor declare status argument. It is a one-byte unsigned integer.

StmtLength This is the total length of the following **SELECT** statement associated with this cursor. It is a 4-byte unsigned integer.

statement This is the actual text of the cursor, without the **DECLARE CURSOR** clause. For example, in the following full ANSI cursor declaration, only the words in italics would be the statement argument.

DECLARE CURSOR csr1 FOR
SELECT a, b FROM tab1
WHERE a < 12 AND b > 15
FOR UPDATE OF a

#Columns When a cursor is used **FOR UPDATE**, the update columns may be specified. This argument identifies the number of columns specified for update. If this number is > 0, the column (or columns) name length and name follow. This argument is a two-byte unsigned integer. This argument is optional. If its value is 0 then the following arguments are omitted. See **TDS_CURDECLARE** for more information on this usage.

NameLength When a cursor is declared **FOR UPDATE**, the columns that may be updated can be specified. This, and the following, argument are repeated for each column specified for update. If the previous argument, #Columns, is 0, this argument and the following argument will not be included. Columns are represented by their column name length and column name in the **FOR UPDATE** list. This parameter is a one-byte unsigned integer.

column name This is the name of the column optionally described in the **FOR UPDATE** clause. Its length is described by the NameLength argument.

Comments

- Read comments in the TDS_CURDECLARE section.
- With appearance of TDS_CURDECLARE3 including all properties of this token, TDS_CURDECLARE2 may become obsolete in the future.
- Associated TDS_ROWFMT token is TDS_ROWFMT2.

Examples

N/A

See Also

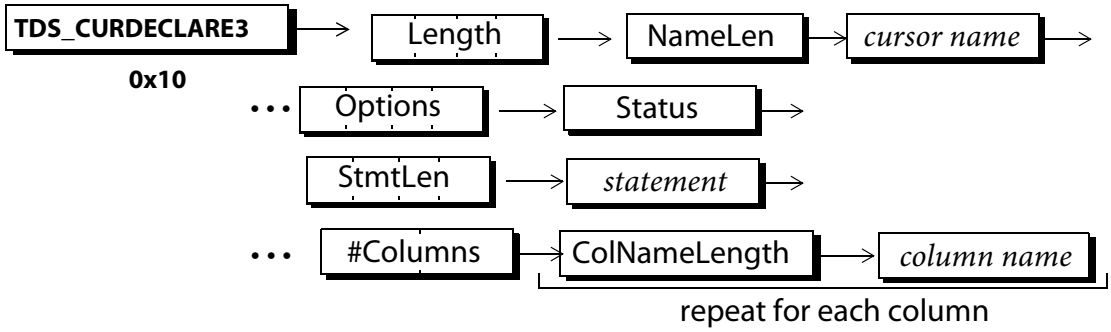
TDS_CURDECLARE, TDS_CURDECLARE3, TDS_CUROPEN, TDS_CURINFO, TDS_CURINFO2, TDS_CURINFO3, TDS_CURFETCH

TDS_CURDECLARE3

Function

Describes the data stream for declaring a cursor. It serves an identical purpose to **TDS_CURDECLARE** and **TDS_CURDECLARE2**, but has been added to support scrollable cursors and other new cursor options. It encompasses all properties of **TDS_CURDECLARE(2)**.

Syntax



Arguments

TDS_CURDECLARE3 This token indicates that this is a data stream containing a client cursor request to declare a cursor.

Length This is the total length of the remaining **TDS_CURDECLARE** data stream. It is a four-byte unsigned integer.

NameLength This is the length of the *cursor name*. It may be up to 255 bytes long; however, a maximum length of **TDS_MAXNAME (30)** is widely supported.

cursor name This is the name of the cursor.

Options These are the cursor declare options. This is a four-byte unsigned integer.

Table 16: Cursor option values

Option Name	Value	Description
TDS_CUR_DOPT_UNUSED	0x00000000	No options associated with this cursor.
TDS_CUR_DOPT_RDONLY	0x00000001	This cursor is read only.
TDS_CUR_DOPT_UPDATABLE	0x00000002	Updates can be performed with this cursor.
TDS_CUR_DOPT_SENSITIVE	0x00000004	Scrollable cursor, sensitive. Changes to the base table will be visible.
TDS_CUR_DOPT_DYNAMIC	0x00000008	This cursor is being declared against a dynamically prepared statement.
TDS_CUR_DOPT_IMPLICIT	0x00000010	This cursor is implicitly read-only, automatically fetches first set of rows on the CUROPEN, and automatically closes after the last row is fetched. This option should not be specified unless the TDS_CUR_IMPLICIT request capability is set.
TDS_CUR_DOPT_INSENSITIVE	0x00000020	Insensitive cursor, may or may not be scrollable. Result set will not be affected by changes to the base table.
TDS_CUR_DOPT_SEMISENSITIVE	0x00000040	Semi-sensitive cursor, may or may not be scrollable. Changes to base table rows not yet fetched by the cursor will be visible.
TDS_CUR_DOPT_KEYSETDRIVEN	0x00000080	Scrollable cursor, keyset driven. Non-key parts over which the cursor was declared are subject to change, Keyparts are fixed with respect to the cursor.
TDS_CUR_DOPT_SCROLLABLE	0x00000100	Cursor is scrollable. Can be used in conjunction with (IN|SEMI)SENSITIVE, KEYSETDRIVEN etc.

Status This is the cursor declare status argument. It is a one-byte unsigned integer.

Table 17: Cursor Declare Status

Name	Value	Description
TDS_CUR_DSTAT_UNUSED	0x00	No status associated with this cursor declare.
TDS_CUR_DSTAT_HASARGS	0x01	The cursor declare statement is followed by parameters.

StmtLength This is the total length of the following **SELECT** statement associated with this cursor. It is a 4-byte unsigned integer. Please note that since the total **TDS_CURDECLARE3** data stream Length may be no greater than 2^32-1, StmtLength can never be a full 2^32-1. The maximum size of StmtLength depends on the length of the cursor name and the number and length of any update columns.

statement This is the actual text of the cursor, without the **DECLARE CURSOR** clause. For example, in the following full ANSI cursor declaration, only the words in italics would be the statement argument.

DECLARE CURSOR csr1 FOR
SELECT a, b FROM tab1
WHERE a < 12 AND b > 15
FOR UPDATE OF a

#Columns When a cursor is declared **FOR UPDATE**, the update columns may be specified. This argument identifies the number of columns specified in the update list. See **TDS_CURDECLARE** for more information on the usage of this parameter.

NameLength When a cursor is declared **FOR UPDATE**, the columns that may be updated can be specified. This, and the following, argument are repeated for each column specified for update. If the previous argument, #Columns, is 0, this argument and the following argument will not be included. Columns are represented by their column name length and column name in the **FOR UPDATE** list. This parameter is a one-byte unsigned integer.

column name This is the name of the column optionally described in the **FOR UPDATE** clause. Its length is described by the NameLength argument.

Comments

- An implicit cursor (using the **TDS_CUR_DOPT_IMPLICIT** status bit) is a unique type and is not to be used with the scrollable, sensitive, insensitive, keyset-driven or semi-sensitive types.
- It is legal for a client to request a non-scrollable cursor of types insensitive, semi-sensitive, keyset driven and sensitive.
- See **TDS_CURDECLARE** and **TDS_CURDECLARE2** for detailed comments.

Examples

N/A

See Also

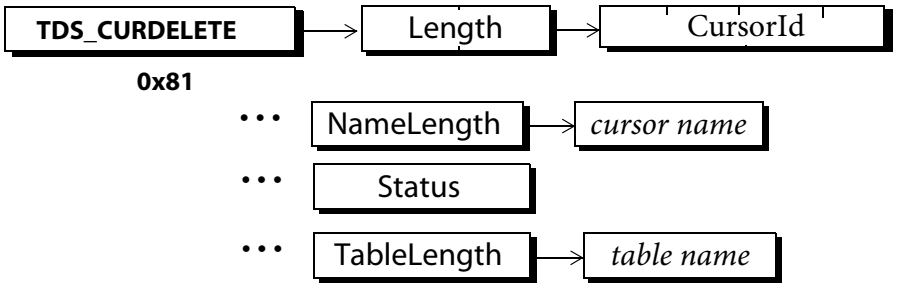
TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CUROPEN, TDS_CURINFO, TDS_CURINFO2, TDS_CURINFO3, TDS_CURFETCH

TDS_CURDELETE

Function

Describes the data stream for deleting a row through a cursor.

Syntax



Arguments

TDS_CURDELETE This is the data stream command token for a client request to **delete** through a cursor.

Length This is the total length of the remaining **TDS_CURDELETE** data stream. It is a two byte unsigned integer.

CursorId This is the internal identifier for the cursor. If CursorId is 0 it means that the cursor on which to perform the delete is identified by name using the NameLength and *cursor name* arguments. This argument is a four byte, unsigned integer.

Name Length This is the length of the *cursor name*. It is a one-byte unsigned integer and must be > 0 and <= **TDS_MAX_NAME(30)**. This part of the data stream is only included if CursorId is equal to 0.

cursor name This is the name of the cursor.

Status This is status information associated with the cursor delete. This argument is a one-byte unsigned integer. It has the following values:

Table 18: Cursor Delete Status Values

Name	Value	Description
TDS_CUR_DELSTAT_UNUSED	0x00	No status associated with the cursor delete.

TableLength This is the length of the table name which follows. It is a one byte unsigned integer.

table name This is the name of the table to which the **delete** applies. It may be a compound name such as "site.db.owner.table". It should be the same table reference as used in the **declare cursor**.

Comments

- This is the data stream generated by the client when a **delete cursor** command is sent to the server.
- The cursor to which the **TDS_CURDELETE** refers is identified in the **TDS_CURDELETE** data stream.
- When a **TDS_CURDELETE** data stream is sent to the server, it is always followed by a **TDS_KEY** data stream. The **TDS_KEY** data stream defines to the server what the client's current row is.
- A **TDS_CURINFO**, **TDS_DONE** is returned on a successful delete.
- A **TDS_CURINFO**, **TDS_EED**, **TDS_DONE** is returned on a version mismatch.
- A **TDS_EED**, **TDS_DONE** is returned for a key mismatch.

Examples

See Also

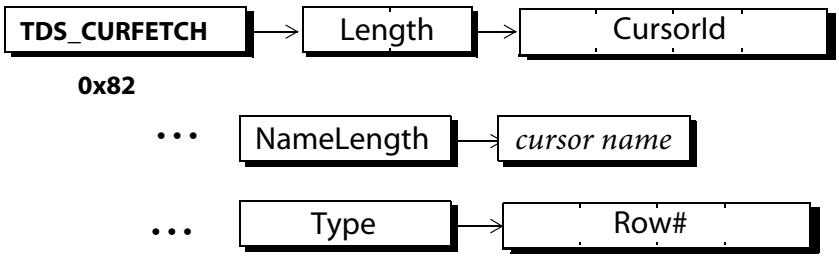
TDS_CURUPDATE, TDS_CURDECLARE, TDS_CURINFO, TDS_CURINFO2, TDS_CURINFO3, TDS_CUROPEN, TDS_KEY

TDS_CURFETCH

Function

Describes the data stream for sending a fetch command to a server.

Syntax



Arguments

TDS_CURFETCH This is the data stream command token that indicates that this is a data stream containing a client cursor request for a **cursor fetch**.

Length This is the total length of the remaining **TDS_CURFETCH** data stream. It is a two byte unsigned integer.

CursorId This is the internal cursor identifier for this cursor. It is a four-byte unsigned integer.

NameLength This is the length of the *cursor name*. It is a one byte unsigned integer and must be > 0 and <= **TDS_MAXNAME (30)**. NameLength is only included if CursorId is equal to 0.

cursor name This is the name of the cursor. Its length is NameLength

Type This argument defines the fetch orientation to get the row(s) that should be returned for this fetch. If cursor scrolling is not supported by a server (determined using capabilities), then Type must always be **TDS_CUR_NEXT**. Type is an unsigned, one byte integer, and its possible values are:

Table 19: Cursor fetch types

Type Names	Value	Row# Sent?	Description
TDS_CUR_NEXT	1	No	Return next rowset from the result set.
TDS_CUR_PREV	2	No	Return previous rowset from the result set.
TDS_CUR_FIRST	3	No	Return first rowset from the result set.
TDS_CUR_LAST	4	No	Return last rowset(s) from the result set.
TDS_CUR_ABS	5	Yes	Return rowset at position specified in Row#. The first row in a result set is 1. ABS(-1) returns the rowset starting with the last row, ABS(-2) returns the rowset starting on the second to last row, etc.
TDS_CUR_REL	6	Yes	Return rowset at current position plus Row#. Row# can be positive or negative. **Please see cursor positioning rules below**.

row # This is a signed, four-byte integer which indicates the row position. This argument is optional and is only included in the data stream as indicated in the table above.

cursor name This is the name of the cursor.

Comments

- This is the data stream generated by a client when a **fetch cursor** request is sent to a server.
- The number of rows that are returned on a **fetch cursor** are determined by the cursor fetch count set using the **TDS_CURINFO** token.

- The cursor to which the **TDS_CURFETCH** refers is identified in the **TDS_CURFETCH** token.
- If scrolling is not supported by the server, the Type argument must be **TDS_CUR_NEXT**. Version 3.6 of this spec adds scrollability. See Rules section.
- The *row #* argument is only in the data stream when Type is **TDS_CUR_ABS** or **TDS_CUR_REL**.
- *Row#* passed in can be less then 0 in case of cursor scrolling. E.g. passing **TDS_CUR_ABS** (-5) is a valid command. See Cursor Positioning Rules below for detailed information.
- If the cursor is updatable, the cursor key is imbedded in the **TDS_ROW** that is returned. Please see **TDS_ROWFMT** for a description of the cursor key.
- **TDS_CURFETCH** tokens are typically responded to with row results and a **TDS_DONE**. A **TDS_CURFETCH** token may also result in the server sending back a **TDS_CURINFO2** token followed by a **TDS_DONE** token. In this latter case, the **TDS_CURINFO2** token will indicate that the fetch moved the cursor position off of the cursor ResultSet -- either before the first row or after the last row.
- The **TDS_CURFETCH** token can be sent by a client with a **TDS_CURDECLARE** and **TDS_CUROPEN** in the same request. A server will acknowledge the **TDS_CURDECLARE** and **TDS_CUROPEN** tokens with a **TDS_CURINFO** and a **TDS_DONE(MORE)**. The **TDS_CURFETCH** will be acknowledged with the row results and a **TDS_DONE(FINAL)**.
- The **TDS_CURFETCH** command may be sent in the same request with a **TDS_CURINFO*** token immediately preceding it. The **TDS_CURINFO*** token can be used to set or alter the fetch size just before the fetch is executed.
- Note that when a cursor Resultset is empty, any attempt to fetch rows using a scrollable cursor will result in the server returning a **TDS_CURINFO2** token with the **TDS_CUR_ISTAT_EMPTYRESULTSET** and **TDS_CUR_ISTAT_ISBEFOREFIRSTROW** status bits set. This means that for an empty ResultSet, the cursor always remains positioned before the first row.

- When servers implement keyset-driven, sensitive and updatable scrollable cursors, this spec will need to be amended. Certainly, keyset-driven and updatable cursors will require the use of a hidden column or columns to contain the keys. Additionally, we may make use of the column status byte to communicate the update status/timestamp of a particular row value.

Cursor Positioning Rules (spec version 3.6 and later)

Definitions of fetch orientation.

- before_start. The cursor is positioned before the first row of the result set. As described in the comments section above, when a fetch positions the cursor at this location, the server sends a CURINFO2 token with the **TDS_CUR_ISTAT_BEFOREFIRSTROW** status bit set. A **TDS_DONE** token must follow.
- after_end. The cursor is positioned after the last row of the result set. As described in the comments section above, when a fetch positions the cursor at this location, the server sends a CURINFO2 token with the **TDS_CUR_ISTAT_AFTERLASTROW** status bit set. A **TDS_DONE** token must follow.
- currow_start. Defines the number of the first row in the current rowset.
- last_resrow. Defines the number of the last row in the result set.
- rowset_size. This is the number of rows to be returned by a cursor fetch.
- row_set. A collection of rows delivered to the client. It is a subset of the entire cursor result set. rowset_size indicates number of rows in row_set.
- fetch_offset. The value of the fetch offset. This value is an unsigned integer.

Rules

TDS_CUR_NEXT rule:

Table 20: TDS_CUR_NEXT

Condition	First row of new rowset
before_start	1
currow_start + rowset_size (a) <= last_resrow	currow_start + rowset_size (a)
currow_start + rowset_size (a) > last_resrow	after_end
after_end	after_end

(a) If the rowset_size has been changed since the previous call, this is the rowset_size that was used with the previous fetch.

TDS_CUR_PREV rule:

Table 21: TDS_CUR_PREV

Condition	First row of new rowset
before_start	before_start
currow_start = 1	before_start
1 < currow_start <= rowset_size (b)	1 (a)
currow_start > rowset_size (b)	currow_start - rowset_size (b)
after_end && last_resrow < rowset_size (b)	1 (a)
after_end && last_resrow >= rowset_size (b)	lastres_row - rowset_size (b) + 1

(a) ODBC drivers are responsible for generating a warning message indicating that the client attempted to fetch before the result set, and that the server is returning the first rowset.

(b) If the rowset_size has been changed since the previous call to fetch rows, this is the new rowset_size.

TDS_CUR_ABS rule:

Table 22: TDS_CUR_ABS

Condition	First row of new row(set)
fetch_offset <0 && |fetch_offset| <= lastres_row	lastres_row + fetch_offset + 1
fetch_offset < 0 && |fetch_offset| > lastres_row && |fetch_offset| > rowset_size (b)	before_start
fetch_offset < 0 && |fetch_offset| > lastres_row && |fetch_offset| <= rowset_size (b)	1 (a)
fetch_offset = 0	before_start
1 <= fetch_offset <= lastres_row	fetch_offset
fetch_offset > lastres_row	after_end

(a) The client is responsible for generating a warning message indicating that the client attempted to fetch before the result set, and that the server is returning the first rowset.

(b) If the rowset_size has been changed since the previous call to fetch rows, this is the new rowset_size.

TDS_CUR_REL rule:

Table 23: TDS_CUR_REL

Condition	First row of new rowset
(before_start && fetch_offset > 0) || (after_end && fetch_offset < 0)	same rowset returned as if it was called with **TDS_CUR_ABS**. (a)
before_start && fetch_offset < 0	before_start
currow_start = 1 && fetch_offset < 0	before_start
currow_start > 1 && (currow_start + fetch_offset) < 1 && |fetch_offset| > rowset_size (c)	before_start
currow_start > 1 && (currow_start + fetch_offset) < 1 && |fetch_offset| <= rowset_size (c)	1 (b)
1 < currow_start + fetch_offset <= lastres_row	currow_start + fetch_offset
currow_start + fetch_offset > lastres_row	after_end
after_end && fetch_offset >= 0	after_end

(a) Same as TDS_CUR_ABS. See next section.

(b) ODBC drivers are responsible for generating a warning message indicating that the client attempted to fetch before the result set, and that the server is returning the first rowset.

(c) If the rowset_size has been changed since the previous call to fetch rows, this is the new rowset_size.

TDS_CUR_FIRST rule:

Table 24: TDS_CUR_FIRST

Condition	First row of new rowset
Any	1

TDS_CUR_LAST rule:

Table 25: TDS_CUR_LAST

Condition	First row of new rowset
rowset_size (a) <= lastres_row	lastres_row - rowset_size (a) + 1
rowset_size (a) > lastres_row	1

(a) If the rowset_size has been changed since the previous call to fetch rows, this is the new rowset_size.

Examples, Scrollable cursor

Table 26:

1	a
2	b
3	...
4	
5	
6	
7	
8	
9	
10	
11	
12	

- fetch c (ABS -1) returns last row, row # 12.
- fetch c (ABS -3) returns row # 10.
- fetch c (ABS 0) returns no data, cursor is positioned before first row.
- See new cursor status flags in **TDS_CURDECLARE3**.
- See examples of TDS-based scrollable cursor operations in the Examples section.

See Also

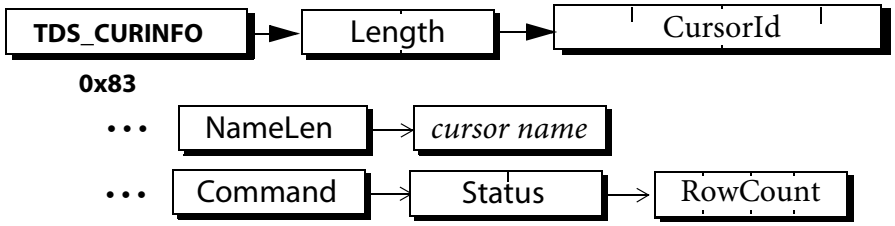
TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_CUROPEN, TDS_CURINFO, TDS_CURINFO2, TDS_CURINFO3

TDS_CURINFO

Function

The data stream for describing cursor characteristics and state.

Syntax



Arguments

TDS_CURINFO This is the data stream token that indicates that this is a data stream containing a description of a cursor.

Length This is the total length of the remaining **TDS_CURINFO** data stream. It is a two-byte unsigned integer.

CursorId This is the internal identifier for a cursor. The CursorId is always set by the server that is managing the cursor. It is never assigned by a client. This argument is a four-byte, signed integer.

NameLength This is the length of the following *cursor name*. It is a one-byte, unsigned integer. This argument only appears if CursorId is 0.

cursor name This is the cursor name for this cursor. The argument only appears if there is a NameLength parameter in the data stream. Its length is in NameLength.

Command This is the command associated with the **TDS_CURINFO** token.

Table 27: TDS_CURINFO Commands

Name	Value	Description
TDS_CUR_CMD_SETCURROWS	1	Set the fetch count.
TDS_CUR_CMD_INQUIRE	2	Ask status of a cursor.
TDS_CUR_CMD_INFORM	3	Report status of a cursor.
TDS_CUR_CMD_LISTALL	4	Report status of all open cursors.

Status This argument describes the status of the cursor. This argument is a two-byte, unsigned integer. The possible values are:

Table 28: Valid Cursor Status

Option Names	Values	Description
TDS_CUR_ISTAT_UNUSED	0x0000	The option argument is unused.
TDS_CUR_ISTAT_DECLARED	0x0001	The specified cursor has been declared.
TDS_CUR_ISTAT_OPEN	0x0002	The specified cursor is open.
TDS_CUR_ISTAT_CLOSED	0x0004	The specified cursor is closed.
TDS_CUR_ISTAT_RDONLY	0x0008	The specified cursor is read-only. Any update or delete statements against this cursor are illegal.
TDS_CUR_ISTAT_UPDATABLE	0x0010	The specified cursor is updatable. Update and delete statements may be issued against this cursor.
TDS_CUR_ISTAT_ROWCNT	0x0020	The rowcount argument is valid. This **TDS_CURINFO** command is setting the current row fetch count.
TDS_CUR_ISTAT_DEALLOC	0x0040	The specified cursor has been deallocated. It cannot be opened unless it is declared again.
DO_NOT_ADD_STATUS_BITS		See **CURINFO2**

row count This describes how many rows will be returned for a cursor **fetch**. It is a four-byte signed integer.

Comments

- This data stream is used for two purposes. It is used to communicate changes in the state of a cursor. It is also used to set the current cursor context.

- This data stream is used to set the "current cursor". This is required because there can be multiple open cursors on a single client server dialog. The **TDS_CURINFO** is used to coordinate commands and responses with a particular cursor.
- The **TDS_CURINFO** token is used by servers to return the assigned cursor id after a cursor has been declared.
- This data stream is first returned to a client when the cursor is **declared**. It is also returned to the client if the number of rows per **fetch** is changed.
- If Command is **TDS_CUR_CMD_LISTALL** the CursorId must be 0.
- NameLength and CursorName are optional. They are only in the data stream if CursorId is 0.
- RowCount is optional. It is only present if the Length argument after subtracting out the lengths of the other arguments is 4. (It was initially specified that this field would be present if and only if the **TDS_CUR_ISTAT_ROWCNT** bit was set. Open Client was not coded to this requirement, thus we are left with this silly subtraction technique.)
- Returning a RowCount equal to 0 is illegal.
- It is illegal to set the **TDS_CUR_ISTAT_ROWCNT** Status with the **TDS_CUR_CMD_INQUIRE** and **TDS_CUR_CMD_LISTALL** commands.
- Language based cursors do not cause **TDS_CURINFO** tokens to be sent.
- A client requests the status of a specified cursor using the **TDS_CUR_CMD_INQUIRE** command with CursorId set to the identifier of the cursor the client wants information on. The server responds with the **TDS_CUR_CMD_INFORM** Command and the status bits set appropriately for the cursor identifier identified in the CursorId argument. NOTE: This command is not supported by all servers.
- A client can request the status of all active cursor using the **TDS_CUR_CMD_LISTALL** Command. When a server receives this Command it returns a **TDS_CUR_CMD_INFORM** Command for all active cursors on the dialog.
- A server will acknowledge the **TDS_CUR_CMD_SETCURROWS** Command with a **TDS_CURINFO**, **TDS_DONE**. If the requested row count is invalid the server will respond with a **TDS_CURINFO**, **TDS_EED**, **TDS_DONE**.

Examples

See Also

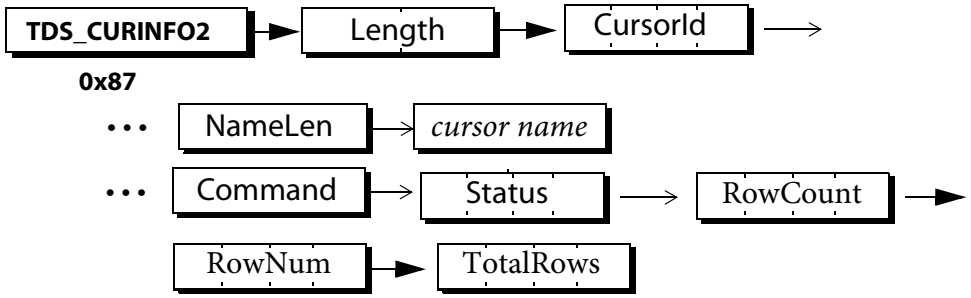
TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_CUROPEN

TDS_CURINFO2

Function

The data stream for describing cursor characteristics and state. This token can be used only when the **TDS_CSR_SCROLL** capability is on.

Syntax



Arguments

TDS_CURINFO2 This is the data stream token that indicates that this is a data stream containing a description of a cursor.

Length This is the total length of the remaining **TDS_CURINFO2** data stream. It is a two-byte unsigned integer.

CursorId This is the internal identifier for a cursor. The CursorId is originally generated by the server that is managing the cursor. It is never created by a client. Once the server has assigned this ID, both client and server can specify it in this field to exchange information about a particular cursor. This argument is a four-byte, signed integer.

NameLength This is the length of the following *cursor name*. It is a one-byte, unsigned integer. This argument only appears if CursorId is 0.

cursor name This is the cursor name for this cursor. The argument only appears if there is a NameLength parameter in the data stream. Its length is in NameLength.

Command This is the command associated with the **TDS_CURINFO2** token.

Table 29: TDS_CURINFO Commands

Name	Value	Description
TDS_CUR_CMD_SETCURROWS	1	Set the fetch count.
TDS_CUR_CMD_INQUIRE	2	Ask status of a cursor.
TDS_CUR_CMD_INFORM	3	Report status of a cursor.
TDS_CUR_CMD_LISTALL	4	Report status of all open cursors.

Status This argument describes the status of the cursor. This argument is a four-byte, unsigned integer. The possible values are:

Table 30: Valid Cursor Status

Option Names	Values	Description
TDS_CUR_ISTAT_UNUSED	0x00000000	The option argument is unused.
TDS_CUR_ISTAT_DECLARED	0x00000001	The specified cursor has been declared.
TDS_CUR_ISTAT_OPEN	0x00000002	The specified cursor is open.
TDS_CUR_ISTAT_CLOSED	0x00000004	The specified cursor is closed.
TDS_CUR_ISTAT_RDONLY	0x00000008	The specified cursor is read-only. Any update or delete statements against this cursor are illegal.
TDS_CUR_ISTAT_UPDATABLE	0x00000010	The specified cursor is updatable. Update and delete statements may be issued against this cursor.
TDS_CUR_ISTAT_ROWCNT	0x00000020	The RowCount argument is valid and is being used to report the current row fetch size to the client. A server must always set this bit when sending a **CURINFO2** to a client.
TDS_CUR_ISTAT_DEALLOC	0x00000040	The specified cursor has been deallocated. It cannot be opened unless it is declared again.
TDS_CUR_ISTAT_SCROLLABLE	0x00000080	Cursor is scrollable.
TDS_CUR_ISTAT_IMPLICIT	0x00000100	Cursor is an implicit cursor.
TDS_CUR_ISTAT_SENSITIVE	0x00000200	Cursor is sensitive.
TDS_CUR_ISTAT_INSENSITIVE	0x00000400	Cursor is insensitive.
TDS_CUR_ISTAT_SEMISENSITIVE	0x00000800	Cursor is semi-sensitive. Note: ASENSITIVE as per ANSI.

Table 30: Valid Cursor Status

Option Names	Values	Description
TDS_CUR_ISTAT_KEYSETDRIVEN	0x00001000	Cursor opened is keyset driven. Note: ASENSITIVE as per ANSI.

RowCount This field holds the current fetch size for the cursor. This field is always present in the stream since the **TDS_CUR_ISTAT_ROWCNT** status bit is always set for **TDS_CURINFO2**. This field is a four-byte unsigned integer.

RowNum This field holds the 1-based row number for the cursor's current position. RowNum is a four-byte signed integer; it may contain the values 0 (beforefirst), -1 (unknown) and -2 (afterlast).

TotalRows This field holds the total number of rows in a cursor ResultSet. TotalRows is a four-byte signed integer; it may contain the value -1 (unknown) and 0 (if the cursor ResultSet is empty).

Comments

- This data stream is used for two purposes. It is used to communicate changes in the state of a cursor. It is also used to set the current cursor context.
- This data stream is used to set the "current cursor". This is required because there can be multiple open cursors on a single client server dialog. The **TDS_CURINFO2** token is used to coordinate commands and responses with a particular cursor.
- The **TDS_CURINFO2** token is used by servers to return the assigned cursor id after a cursor has been declared.
- This data stream is first returned to a client when the cursor is **declared**. It is also returned to the client if the number of rows per **fetch** is changed. It is also returned to the client when the client performs a **CURFETCH** that moves the cursor off the ResultSet (before the first row, or after the last row). Finally, this token is returned when a client issues a **TDS_CUR_CMD_INQUIRE** or **TDS_CUR_CMD_LISTALL** command to request state information on a cursor.

- If Command is **TDS_CUR_CMD_LISTALL** the CursorId must be 0.
- NameLength and CursorName are optional. They are only in the data stream if CursorId is 0.
- Returning a RowCount equal to 0 is illegal.
- The default fetch size for any cursor is 1, so when the server responds to a **CURDECLARE*** with a **CURINFO2** token, the RowCount value will be 1.
- When a client issues a **TDS_CMD_SETCURROWS** command, the database will respond with a **CURINFO2** token to report the new fetch size in the RowCount field. In the **CURINFO2** token that is returned, the server will set the RowNum field to a value of unknown (-1). This is to avoid possible confusion when a client batches together a fetch size change with a cursor fetch in one request.
- The following constants will be used in the TotalRows field: (0) will indicate that the ResultSet is empty; (-1) will indicate that the total number of rows is unknown.
- The following constants will be used in the RowNum field: (0) will indicate that the cursor is before the first row (beforefirst); (-1) will indicate that the current row number is unknown (or not applicable, as in a sensitive cursor); and (-2) will indicate that the cursor is positioned after the last row (afterlast).
- The RowNum field value refers to the first row number of the present batch of fetched rows. For example, a client creates a cursor with a fetch size of 5 and performs a fetch-absolute (16) on a 50-row ResultSet. The server returns rows 16-20, and the client then issues a **TDS_CMD_CUR_INQUIRE**. The server would then report a value of 16 in the RowNum field in the **CURINFO2** token it returns.
- It is legal for a client to batch together a **CURINFO*** token to set the fetch size with a **CURFETCH** token immediately following it in the same request.
- In an empty resultset, the cursor will always remain positioned before the first row (i.e. the RowNum field will always contain a value of 0).

- When a cursor is scrollable and semi-sensitive, a server will not know the total number of rows until the last row of the cursor ResultSet has been fetched. Therefore, when a client issues a **TDS_CUR_CMD_INQUIRE**, the server will return a value of -1 (unknown) in the RowNum field until the cursor has passed the last row at least once. After that point, when a client issues a **TDS_CUR_CMD_INQUIRE,** the server should report the known total number of rows in the TotalRows field of any **CURINFO2** token it returns.
- It is illegal for a client to set the **TDS_CUR_ISTAT_ROWCNT** Status with the **TDS_CUR_CMD_INQUIRE** and **TDS_CUR_CMD_LISTALL** commands.
- If a client performs a cursor fetch that moves the cursor off the ResultSet, the server will return a **TDS_CURINFO2** token with the proper value set in the RowNum field to indicate the cursor position. For example, if a client tries a fetch absolute (16) in a 10-row ResultSet, the server will return a **TDS_CURINFO2** token which has a value of -2 (afterlast) in the RowNum field.
- Language-based cursors do not cause **TDS_CURINFO2** tokens to be sent.
- A client requests the status of a specified cursor using the **TDS_CUR_CMD_INQUIRE** command with CursorId set to the identifier of the cursor the client wants information on. The server responds with the **TDS_CUR_CMD_INFORM** command and the status bits set appropriately for the cursor identifier identified in the CursorId argument.
- A client can request the status of all active cursor using the **TDS_CUR_CMD_LISTALL** Command. When a server receives this Command it returns a **TDS_CUR_CMD_INFORM** Command for all active cursors on the dialog.
- A server will acknowledge the **TDS_CUR_CMD_SETCURROWS** command with a **TDS_CURINFO(2), TDS_DONE**. If the requested row count is invalid, the server will respond with a **TDS_CURINFO(2)**, **TDS_EED**, **TDS_DONE**.

Examples

See scrollable-cursor examples earlier in this document.

See Also

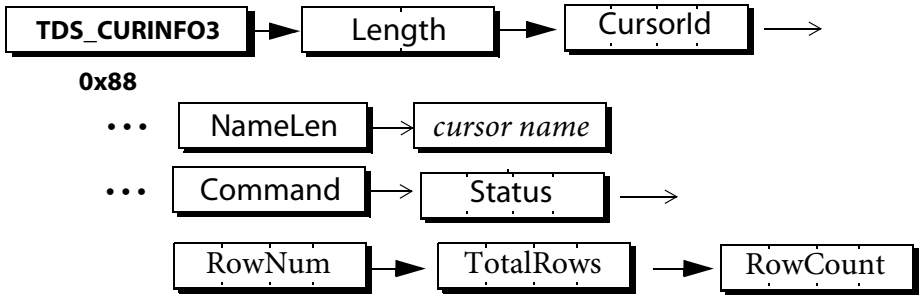
TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_CUROPEN, TDS_CURINFO, TDS_CURINFO3

TDS_CURINFO3

Function

The data stream for describing cursor characteristics and state. This token can be used only when the **TDS_REQ_CURINFO3** capability is on. This token will be used instead of a **TDS_CURINFO2** token in this case.

Syntax



Arguments

TDS_CURINFO3 This is the data stream token that indicates that this is a data stream containing a description of a cursor. See **TDS_CURINFO2** for more information on field interpretation.

RowCount This field holds the current fetch size for the cursor. This field is present in the stream only if the **TDS_CUR_ISTAT_ROWCNT** status bit is set. This field is a four-byte unsigned integer. Note that unlike **TDS_CURINFO2**, this field is not always present. Also note that *RowCount* follows all the other fields.

Comments

- This token is almost identical to **TDS_CURINFO2**. The only difference is that the **TDS_CUR_ISTAT_ROWCNT** status bit is not always set. This difference restores **TDS_CURINFO** behavior and corrects existing client library usage of this field.

Examples

See scrollable-cursor examples earlier in this document.

See Also

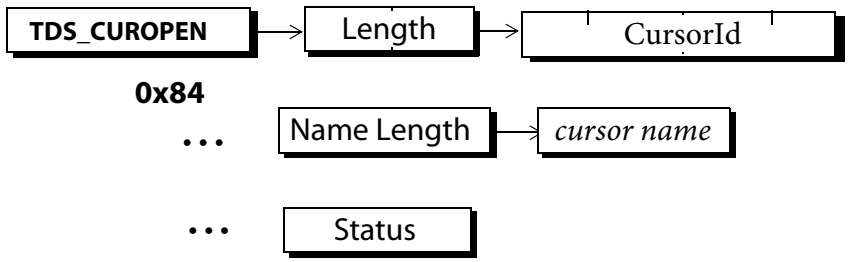
TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_CUROPEN, TDS_CURINFO, TDS_CURINFO2

TDS_CUROPEN

Function

Describes the cursor open data stream.

Syntax



Arguments

TDS_CUROPEN This is the data stream token for a client request to open a cursor.

Length This is the total length of the remaining **TDS_CUROPEN** data stream. It is a two byte unsigned integer.

CursorId This is the internal identifier for the cursor. If CursorId is 0 it means that the cursor being opened is identified by name using the NameLength and *cursor name* arguments. This argument is a four byte, unsigned integer.

NameLength This is the length of the *cursor name*. It is a one byte unsigned integer and must be > 0 and <= **TDS_MAXNAME**. This part of the data stream is only included if CursorId is equal to 0.

cursor name This is the name of the cursor. It is NameLength bytes long.

Status This argument contains status associated with the cursor open command. This argument is a one-byte, unsigned integer.

Table 31: Cursor Open Status Values

Name	Value	Description
TDS_CUR_OSTAT_UNUSED	**0x00**	This open command has no status.
TDS_CUR_OSTAT_HASARGS	**0x01**	Data for arguments associated with the cursor declare statement following the cursor open command in a **TDS_PARAM** data stream.

Comments

- This is the data stream generated by a client when a **open cursor** command is sent to a server.
- The cursor to open is identified in the **TDS_CUROPEN** data stream.
- A cursor must have been declared using **TDS_CURDECLARE** before it can be opened.
- The description of the cursor results, if any, are returned to the client using a **TDS_ROWFMT** data stream at cursor open time.
- A **TDS_CURDECLARE, TDS_CUROPEN** and **TDS_CURFETCH** can be sent in the same request if they all refer to the same cursor.
- A cursor declare statement may be parameterized. If it is, the description of the parameters is passed to the server using a **TDS_PARAMFMT** data stream following the **TDS_CURDECLARE** data stream. When the cursor is opened, the parameter values must be passed to the server with a **TDS_PARAMS** data stream following the **TDS_CUROPEN** data stream. The **TDS_CUR_OSTAT_HASARGS** status must be set in this case.
- Both a **TDS_PARAMFMT** and **TDS_PARAMS** data streams can follow a **TDS_CUROPEN**. This allows conversion to occur between the parameters specified at declare time and the actual parameters provided at open time.
- A server responds with a **TDS_CURINFO** and **TDS_ROWFMT** on success. The **TDS_CURINFO** must come before the **TDS_ROWFMT**.

- A client must be able to accept a **TDS_EED** token at any time during the server response to the **TDS_CUROPEN**.
- The **TDS_CUROPEN** token can be sent by a client with a **TDS_CURDECLARE** and **TDS_CURFETCH** in the same request. A server will acknowledge the **TDS_CURDECLARE** and **TDS_CUROPEN** tokens with a **TDS_CURINFO**, **TDS_DONE(MORE)**, and the **TDS_CURFETCH** with the rows and a **TDS_DONE(FINAL)**.

Examples

See Also

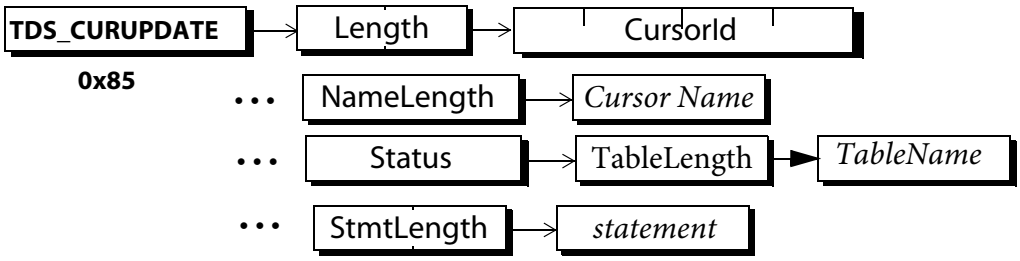
TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_ROWFMT, TDS_ROWFMT2, TDS_PARAMFMT, TDS_PARAMS, TDS_CURCLOSE, TDS_CURFETCH

TDS_CURUPDATE

Function

Describes the data stream for updating a row through a cursor.

Syntax



Arguments

TDS_CURUPDATE This is the data stream token for an update through a cursor.

Length This is the total length of the remaining **TDS_CURUPDATE** data stream. It is a two byte unsigned integer.

CursorId This is the internal identifier for the cursor. If CursorId is 0 it means that the cursor being fetched is identified by name using the NameLength and *cursor name* arguments. This argument is a four byte, unsigned integer

NameLength This is the length of the cursor name which follows. It is a one byte unsigned integer.

cursor name This is the name of the cursor to which the update applies.

Status This is the status information associated with this cursor update. This argument is a one-byte unsigned integer. It can have the following values:

Table 32: Cursor Update Status Values

Name	Value	Description
TDS_CUR_OSTAT_UNUSD	0x00	Status field is unused
TDS_CUR_OSTAT_HASARGS	0x01	Parameters follow the cursor update token
TDS_CUR_CONSEC_UPDS	0x02	Consecutive cursor updates are occurring on this cursor.

TableLength Length of table name that follows. If this argument is 0, no table name follows. This argument is a one-byte unsigned integer.

table name This is the name of the table to which the update applies. It may be a compound name such as "site.db.owner.table". It should be the same table reference as used in the declare cursor statement.

StmtLength Used in the language option case, this is the total length of the following set clause statement. It is a two-byte unsigned integer. This argument is optional.

statement Used in the language option case, this is the actual text of the **SET** clause in the update cursor statement, without the **UPDATE** *table* or **WHERE CURRENT OF** clauses. Unlike the binary option, the values in the **SET** clause need not be constants.

Comments

- This is the data stream generated by the client when an update cursor command is sent to a server.
- The cursor to which the **TDS_CURUPDATE** refers is the one that is current, according to the last **TDS_CURINFO** or **TDS_CURDECLARE** data stream received by the server.

- An update cursor data stream is optionally followed by a **TDS_KEY** data stream which defines the cursor key for the client's "current" row. No **TDS_ROWFMT** data stream is sent to the server with the **TDS_KEY** data stream.
- If a new key is generated by an update, the new key will be returned to the client by sending a **TDS_ROWFMT** and **TDS_KEY** data stream, describing the new key, before the **TDS_DONE** data stream acknowledging the update.
- The server always returns a **TDS_CURINFO**, **TDS_DONE** on a successful update.
- The server will return a **TDS_CURINFO**, **TDS_EED**, **TDS_DONE** on a version mismatch.
- The server will return **TDS_EED**, **TDS_DONE** on a key mismatch.

Examples

See Also

TDS_CURDELETE, TDS_CURDECLARE, TDS_CURDECLARE2, TDS_CURDECLARE3, TDS_KEY, TDS_ROWFMT, TDS_ROWFMT2

TDS Datatypes

Description

This is a complete description of how all data types are represented using TDS. The data type is defined using the **TDS_ROWFMT, TDS_ROWFMT2, TDS_ALTFMT**, or **TDS_PARAMFMT** data streams for rows, compute rows, and parameter data respectively. The actual data is sent using a **TDS_ROW**, **TDS_ALTROW**, or **TDS_PARAMS** data stream.

Length information is sent with variable length and nullable datatypes. Fixed length datatypes do not contain a length argument.

The length information sent in a format data stream indicates the maximum length of this datatype. The length information sent with the data is the actual length of the specific datatype being sent.

If the **TDS_DATA_COLUMNSTATUS** request capability is enabled, then all datatype representations begin with a **status** byte. Status field meanings are defined in *Table 33: Status bit meanings* .

Table 33: Status bit meanings

Bit Mask	Meaning
0x00	The data that follows can be 0-length non-NULL data. It will be 0-length, non-NULL data if the length of the data is 0. Otherwise, a columstatus of 0 just means to take the data that follows as is.
0x01	No Data follows, the value is NULL
0x02	0-length non-NULL text or image data follows.
0xFC	Reserved for future use.
Combined	Interpretation for combinations of these 3 bits
000b	Standard data, getXXX returns value that follows
001b	NULL, isNull returns true
Any other	Not valid, communication exception

Some notes on the columnstatus byte:

- If the **TDS_DATA_COLUMNSTATUS** capability bit is off, this status byte is never present. And even if the **TDS_DATA_COLUMNSTATUS** capability bit is on, the status byte will not be present unless the appropriate bit is set in the status byte of the ROWFMT* or PARAMFMT* token that describes the column or parameter.
- It will be an error for status bits 0x01 and 0x02 to be simultaneously.
- If status bit 0x01 (following data is null) is set, no length or data information follows.
- The 0x01 (data is null) status bit can be set *regardless of the datatype.* For example, even if the datatype is TDS_INT4, which is on its face a non-nullable type, if the columnstatus byte says the data is null, then clients and servers should interpret the data as null. This will allow us to deprecate the INTN and UINTN datatypes when the columnstatus byte is enabled.
- The concept of "zero-length non-nullness" will be applicable to varchar, char, univarchar, unichar, binary, varbinary, text, unitext and image data. For other datatypes, we will continue down the old codepaths, where a 0 length can be interpreted in various ways, including to mean NULL.
- Restating the above bulleted point, when the TDS_DATA_COLUMNSTATUS capability bit is off, nullness can still be indicated, just as it always was before we added the columnstatus byte. *Table 34: Datatype Summary* indicates which datatypes have a Length field. Those datatypes which have a length field can convey SQL NULL by having a length of 0.
- In a previous version of this spec, status bits were included to convey overflow/underflow conditions, as well as truncation/rounding conditions. It was decided to remove those bits from the spec until such time as a greater understanding is reached about exactly how these situations would be reported to the client. Additionally, it was decided that we should wait also until a specific business case presented itself.

TDS presentation conversion is server makes right. This means that the server is always responsible for performing any required conversions. Presentation conversion is performed for the following cases:

- Character set conversions for character and text datatypes.

- Numeric conversions for float, decimal, and numeric datatypes between the client's local representation and the server's.
- Date and time conversions between a client's local representation and the server's.
- Byte ordering conversions for length fields and integer datatypes.

Each of the datatypes has a request and a response capability associated with it. If the request capability bit is set after login then it is OK for the client to send parameters of that type to the server. If the response capability NOXXX is clear then it is OK for the server to send this datatype to the client (in rows, parameters, etc.). If the NOXXX response capability is set, then the server may not send this datatype. The server may convert the datatype to another which the client does accept, or may raise an error indicating that a response could not be returned due to client datatype restrictions. For example, if the server is returning rows from an unsigned short column and the client doesn't support UINT2 or UINTN datatypes, then the server may choose to convert each row to an INT4 or INTN(4) to preserve the value, or may raise an error.

A brief description of all datatypes supported by TDS is in the table below. The syntax of their data streams is in the Syntax section below.

Table 34: Datatype Summary

Datatype Name	Value	Fixed Length?	Nullable?	Converted ?	Description
TDS_BINARY	**0x2D**	**Yes**	**No**	**No**	Binary
TDS_BIT	**0x32**	**Yes**	**No**	**No**	Bit
TDS_BLOB	**0x24**	**No**	**Yes**	**No**	Serialized Object
TDS_BOUNDARY	**0x68**	**No**	**Yes**	**Yes**	Boundary
TDS_CHAR	**0x2F**	**Yes**	**No**	**Yes**	Character
TDS_DATE	**0x31**	**Yes**	**No**	**No**	Date
TDS_DATEN	**0x7B**	**No**	**Yes**	**Yes**	Date
TDS_DATETIME	**0x3D**	**Yes**	**No**	**Yes**	Date/time

Table 34: Datatype Summary

Datatype Name	Value	Fixed Length?	Nullable?	Converted ?	Description
TDS_DATETIMEN	**0x6F**	**No**	**Yes**	**Yes**	Date/time
TDS_DECN	**0x6A**	**No**	**Yes**	**Yes**	Decimal
TDS_FLT4	**0x3B**	**Yes**	**No**	**Yes**	Float
TDS_FLT8	**0x3E**	**Yes**	**No**	**Yes**	Float
TDS_FLTN	**0x6D**	**No**	**Yes**	**Yes**	Float
TDS_IMAGE	**0x22**	**No**	**Yes**	**No**	Image
TDS_INT1	**0x30**	**Yes**	**No**	**No**	Unsigned Integer
TDS_INT2	**0x34**	**Yes**	**No**	**Yes**	Integer
TDS_INT4	**0x38**	**Yes**	**No**	**Yes**	Integer
TDS_INT8	**0xbf**	**Yes**	**No**	**Yes**	Integer
TDS_INTERVAL	**0x2E**	**Yes**	**No**	**Yes**	Time Interval
TDS_INTN	**0x26**	**No**	**Yes**	**Yes**	Integer
TDS_LONGBINARY	**0xE1**	**No**	**Yes**	**No**	Binary
TDS_LONGCHAR	**0xAF**	**No**	**Yes**	**Yes**	Character
TDS_MONEY	**0x3C**	**Yes**	**No**	**Yes**	Money
TDS_MONEYN	**0x6E**	**No**	**Yes**	**Yes**	Money
TDS_NUMN	**0x6C**	**No**	**Yes**	**Yes**	Numeric
TDS_SENSITIVITY	**0x67**	**No**	**Yes**	**Yes**	Sensitivity
TDS_SHORTDATE	**0x3A**	**Yes**	**No**	**Yes**	Date/time
TDS_SHORTMONEY	**0x7A**	**Yes**	**No**	**Yes**	Money
TDS_SINT1	**0xb0**	**Yes**	**No**	**No**	Signed Integer

Table 34: Datatype Summary

Datatype Name	Value	Fixed Length?	Nullable?	Converted ?	Description
TDS_TEXT	**0x23**	**No**	**Yes**	**Yes**	Text
TDS_TIME	**0x33**	**Yes**	**No**	**No**	Time
TDS_TIMEN	**0x93**	**No**	**Yes**	**Yes**	Time
TDS_UINT2	**0x41**	**Yes**	**No**	**Yes**	Unsigned Integer
TDS_UINT4	**0x42**	**Yes**	**No**	**Yes**	Unsigned Integer
TDS_UINT8	**0x43**	**Yes**	**No**	**Yes**	Unsigned Integer
TDS_UINTN	**0x44**	**No**	**Yes**	**Yes**	Unsigned Integer
TDS_UNITEXT	**0xae**	**No**	**Yes**	**Yes**	Unicode UTF-16 Text
TDS_VARBINARY	**0x25**	**No**	**Yes**	**No**	Binary
TDS_VARCHAR	**0x27**	**No**	**Yes**	**Yes**	Character
TDS_VOID	**0x1f**	**N/A**	**N/A**	**N/A**	Void (unknown)
TDS_XML	**0xA3**	**No**	**Yes**	**Yes**	XML

Usertypes

Some TDS datatypes are used to carry more than one SQL datatype. For example, in SQL the BINARY(30) datatype is different from the VARBINARY(30) in that a BINARY is always logically 30-bytes long - it is NULL-padded to 30 bytes if < 30 bytes are contained in the corresponding **TDS_BINARY** value. A VARBINARY is a varying-length datatype, it has no implied trailing NULLs. Though the data representation for both is the same, clients and servers at either end of TDS may need to determine what sort of SQL type a given TDS_BINARY value corresponds to for proper semantic processing. We use the **usertype** field of the format (ROWFMT, PARAMFMT, ALTFMT) to distinguish among the SQL datatypes. Table *Table 35: USERTYPE mappings* lists the mappings used.

Table 35: USERTYPE mappings

TDS Datatype	SQL Datatype	Usertype	Comment
TDS_VARCHAR	char	1	blank pad to the length in the format
TDS_VARCHAR	varchar	2	
TDS_VARBINARY	binary	3	null pad to the length in the format
TDS_VARBINARY	varbinary	4	
TDS_INTN	tinyint	5	
TDS_INTN	smallint	6	
TDS_INTN	int	7	
TDS_FLTN	float	8	
TDS_NUMERIC	numeric	10	
TDS_MONEYN	money	11	
TDS_DATETIMEN	datetime	12	
TDS_INTN	intn	13	

Table 35: USERTYPE mappings

TDS Datatype	SQL Datatype	Usertype	Comment
TDS_FLTN	floatn	14	
TDS_DATETIMN	datetimn	15	
TDS_BIT	bit	16	
TDS_MONEYN	moneyn	17	
TDS_VARCHAR	sysname	18	Internal ASE datatype
TDS_TEXT	text	19	
TDS_IMAGE	image	20	
TDS_MONEYN	smallmoney	21	
TDS_DATETIMN	smalldatetime	22	
TDS_FLTN	real	23	
TDS_VARCHAR	nchar	24	
TDS_VARCHAR	nvarchar	25	
TDS_NUMERIC	decimal	26	decimal and numeric datatypes are identical on ASE, but we maintain the distinction on how they were declared so that clients can report column types in a way that is consistent with how they were declared.
TDS_NUMERIC	decimaln	27	
TDS_NUMERIC	numericn	28	
TDS_LONGBINARY	unichar	34	fixed length UTF-16 encoded data
TDS_LONGBINARY	univarchar	35	variable length UTF-16 encoded data
TDS_IMAGE	unitext	36	UTF-16 encoded data

Table 35: USERTYPE mappings

TDS Datatype	SQL Datatype	Usertype	Comment
TDS_DATE	date	37	
TDS_TIME	time	38	
TDS_INTERVAL	interval	39	
TDS_DATEN	daten	40	
TDS_TIMEN	timen	41	
TDS_INTN	bigint	42	
TDS_UINTN	usmallint	43	
TDS_UINTN	uint	44	
TDS_UINTN	ubigint	45	
TDS_XML	xml	47	Treated like text and image.
TDS_DATETIMN	date	50	The hh:mm:ss.nnnn information should be ignored
TDS_DATETIMN	time	51	The mm/dd/yyyy information should be ignored
TDS_INTN	unsigned short	52	Deprecated
TDS_INTN	unsigned int	53	Deprecated
TDS_INTN	unsigned long	54	Deprecated
TDS_LONGBINARY	serialization	55	serialized java object or instance (i.e. java object)
TDS_LONGBINARY	serialized java class	56	serialized java class (i.e. byte code)
TDS_LONGCHAR	string	57	internally generated varchar strings (e.g. select @@version), not table columns

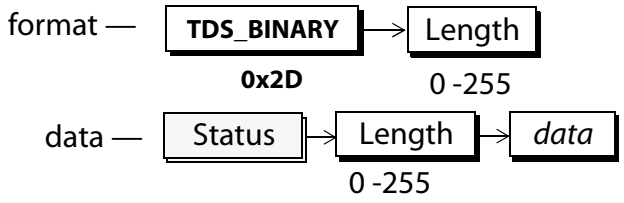
Table 35: USERTYPE mappings

TDS Datatype	SQL Datatype	Usertype	Comment
TDS_INTN	unknown	58	a describe input will return TDS_INT4 (as a simple place-holder) for all columns where it does not know the datatype. This usertype indicates that the actual type is unknown.
TDS_LONGBINARY	smallbinary	59	64K max length binary data (ASA)
TDS_LONGCHAR	smallchar	60	64K maximum length char data (ASA)
TDS_BINARY	timestamp	80	This has nothing to do with date or time, it is an ASE unique value for use with optimistic concur-rency.

Syntax

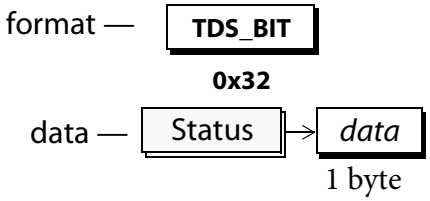
TDS_BINARY — 0x2D

The **TDS_BINARY** datatype is considered a fixed length data type. However, its network representation can vary from 0 to 255 bytes to eliminate sending non-significant trailing **0x00**s. The length is specified by a one-byte unsigned integer which precedes the datatype token and the data.



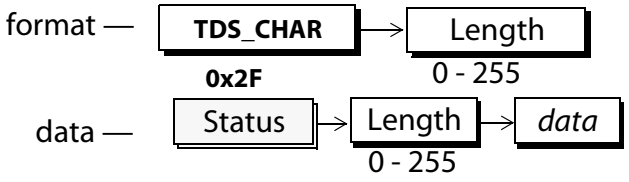
TDS_BIT — 0x32

TDS_BIT is a fixed length datatype of one byte. The only valid values for this datatype are **0x00** or **0x01**.



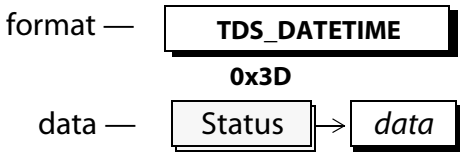
TDS_CHAR — 0x2F

The **TDS_CHAR** datatype is considered a fixed length data type. However, its network representation can vary from 0 to 255 bytes to eliminate sending non-significant trailing spaces. The length is specified by a one-byte unsigned integer which precedes the datatype token and the data.



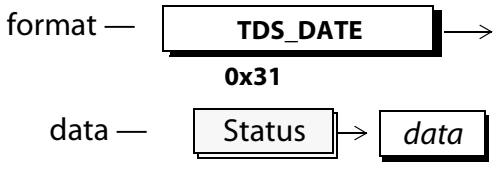
TDS_DATETIME — 0X3D

TDS_DATETIME is a fixed length datatype of 8 bytes.



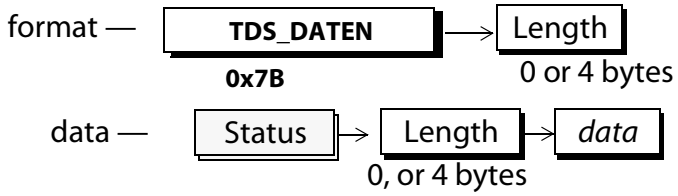
TDS_DATE — 0X31

TDS_DATE is a fixed length datatype of 4 bytes.



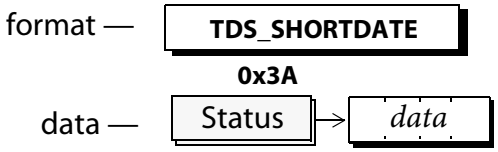
TDS_DATEN — 0X7B

TDS_DATEN is a nullable version of the TDS_DATE datatype.



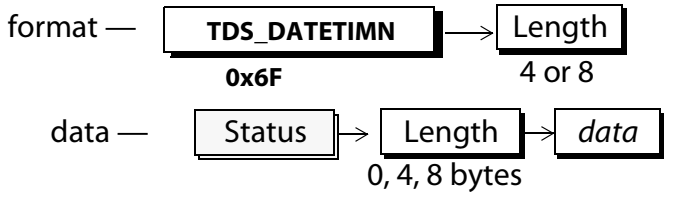
TDS_SHORTDATE — 0X3A

TDS_SHORTDATE is a fixed length datatype of4 bytes.



TDS_DATETIMN — 0X6F

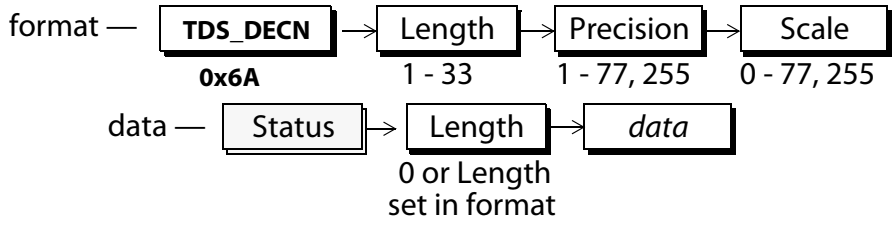
TDS_DATETIMN is a nullable version of the **TDS_DATETIME** and **TDS_DATETIME4** datatypes. The token and its data are preceded by an unsigned one-byte integer which has the value 0, 4, or 8. A NULL is indicated by a length value of 0.



The data length must either be 0 or the length specified in the format length argument.

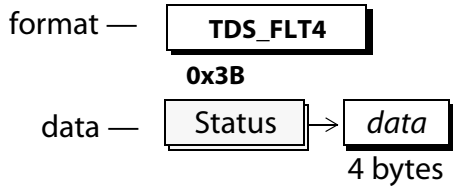
TDS_DECN — 0x6A

The **TDS_DECN** is a variable length nullable datatype. The token is followed by one byte arguments for data length, precision, and scale. The length byte is the length of the data only. It does not include the bytes for precision and scale. A Length of 0 in the data stream indicates a NULL datatype. The **TDS_DECN** has exactly the same format as the **TDS_NUMN** datatype.



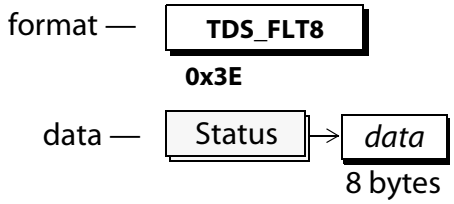
TDS_FLT4 — 0x3B

This is a fixed length four-byte floating point datatype. The precision of the floating point number is platform specific.



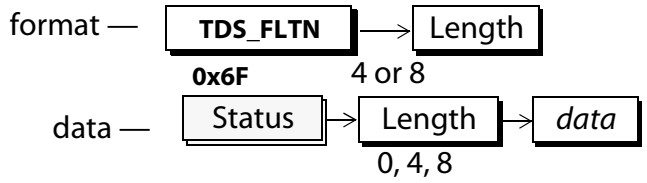
TDS_FLT8 — 0x3E

This is fixed length eight-byte floating point datatype. The precision of the floating point is platform specific.



TDS_FLTN — 0x6D

This is the same as the **TDS_FLT4** and **TDS_FLT8** datatypes except that NULLS are allowed. The token and its data are preceded by an unsigned one-byte integer which has the value 0, 4, or 8. A NULL is indicated by a Length value of 0.



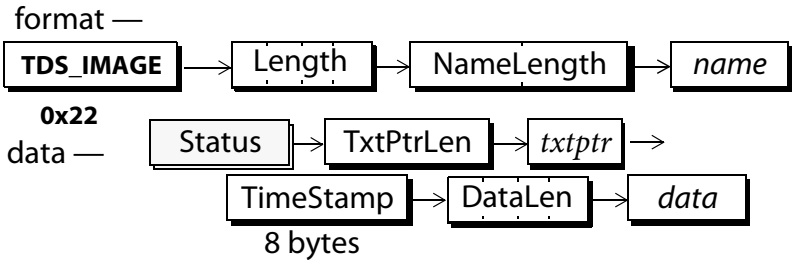
The data length must either be 0 or the length specified in the format length argument. For example, if the format length is specified as 4, the data length can not be 8.

TDS_IMAGE — 0x22

This is a large binary datatype. TxtPtrLen gives the length in bytes of the following txtptr argument. If TxtPtrLen is 0 then the value of the IMAGE data item is SQL NULL and none of the other fields follow. The txtptr is a varbinary value (of length TxtPtrLen) which the database can use to re-locate the source of this data if the client wants to modify it.

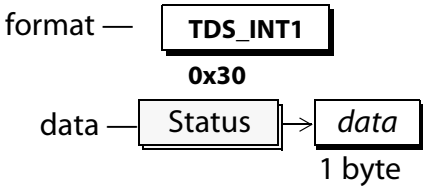
TimeStamp is an 8-byte binary value which is automatically changed on the database whenever an IMAGE value is changed. If the client uses a BULK_WRITE stream to update this value it must pass this timestamp value into the WRITETEXT clause. If the timestamp doesn't match the current value on the server the update will fail because there has been an intervening modification to that IMAGE value.

DataLen is a 4 byte, unsigned value which indicates the length in bytes of the following data . Data is the value of the IMAGE column.



TDS_INT1 — 0x30

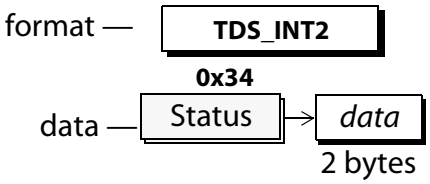
This is an unsigned, one-byte integer. It may have the value of 0 through



255.

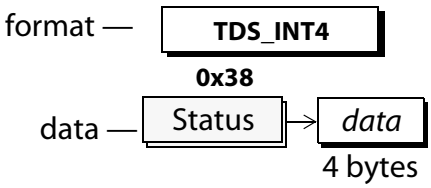
TDS_INT2 — 0x34

This is a signed, two-byte integer. It may have the value of -32,768 through 32,767.



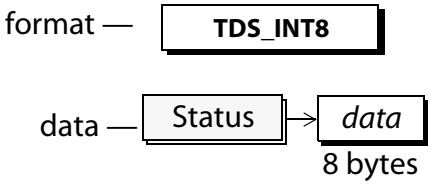
TDS_INT4 — 0x38

This is a signed, four-byte integer. It may have the value of -2,147,483,648 through 2,147,483,647.



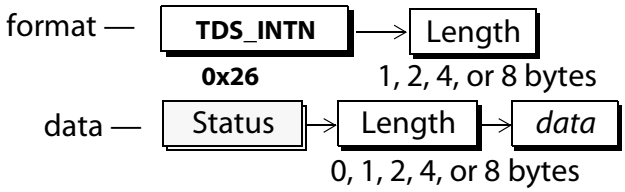
TDS_INT8 —0xBF

This is a signed, eight-byte integer. It may have the value of -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807.



TDS_INTN — 0x26

This is either an **TDS_INT1**, **TDS_INT2**, or **TDS_INT4**, or **TDS_INT8** which allows NULLS. The token and its data are preceded by an unsigned one-byte integer which specifies its length. If used to represent an **TDS_INT1**, the length must be either 0 or 1. If used to represent an **TDS_INT2**, the length must be either 0 or 2. If used to represent an **TDS_INT4**, the length must be either 0 or 4.If used to represent an **TDS_INT8**, the length must be either 0 or



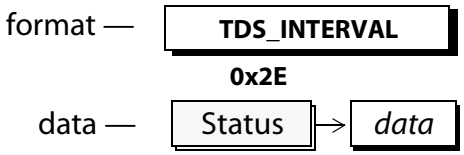
8

The data length must either be 0 or the length specified in the format length argument. For example, if the format length is specified as 4, the data length can not be 1 or 2. If the TDS_DATA_INT8 request capability is clear or the TDS_DATA_NOINT8 response capability is set, then the Length field may not indicate 8 bytes.

Note that for historical reasons, TDS_INT1 is unsigned, and a TDS_INTN with a length of 1 must be interpretted as an unsigned integer while the rest are signed. We have added a TDS_SINT1 to specifically indicate a signed 1-byte integer value.

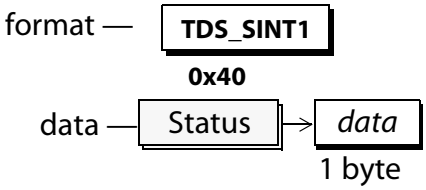
TDS_INTERVAL — 0X2E

TDS_INTERVAL is a fixed length datatype of 8 bytes.



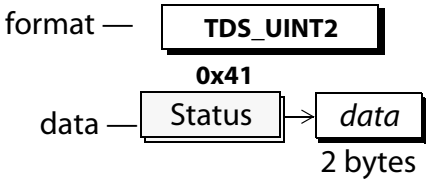
TDS_SINT1 — 0xB0

This is a signed, one-byte integer. It may have the value of -128 through 127.



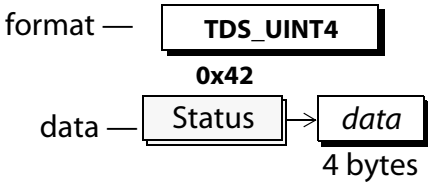
TDS_UINT2 — 0x41

This is an unsigned, two-byte integer. It may have the value of 0 through 65535.



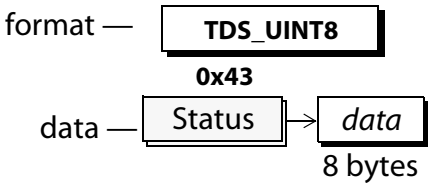
TDS_UINT4 — 0x42

This is an unsigned, four-byte integer. It may have the value of -0 through 4,294,967,295.



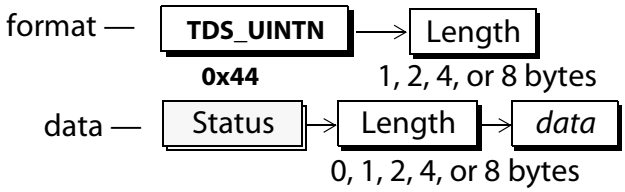
TDS_UINT8 — 0x43

This is an unsigned, eight-byte integer. It may have the value of -0 through 18,446,744,073,709,551,613.



TDS_UINTN — 0x44

This is either an **TDS_INT1**, **TDS_UINT2**, or **TDS_UINT4**, or **TDS_UINT8** which allows NULLS. The token and its data are preceded by an unsigned one-byte integer which specifies its length. If used to represent an **TDS_UINT1**, the length must be either 0 or 1. If used to represent an **TDS_UINT2**, the length must be either 0 or 2. If used to represent an **TDS_UINT4**, the length must be either 0 or 4.If used to represent an **TDS_UINT8**, the length must be either 0

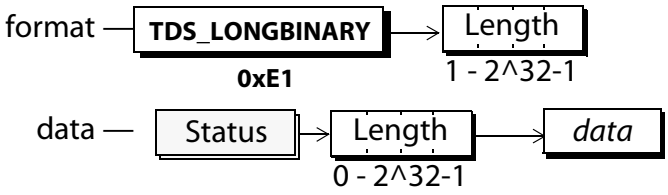


or 8

The data length must either be 0 or the length specified in the format length argument. For example, if the format length is specified as 4, the data length can not be 1 or 2. If the **TDS_DATA_UINT8** request capability is clear or the **TDS_DATA_NOUINT8** response capability is set, then the Length field may not indicate 8 bytes.

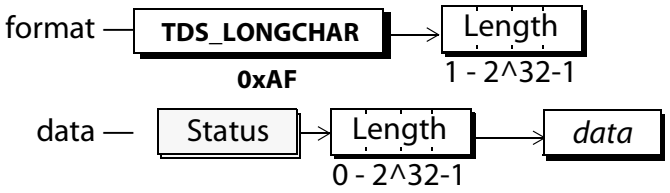
TDS_LONGBINARY — 0xE1

This is a large variable length binary datatype. This datatype can support the same length of a **TDS_IMAGE** datatype without the additional complexity. This data type has a four-byte unsigned integer length field.



TDS_LONGCHAR — 0xAF

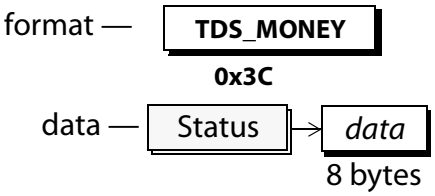
This is a large variable length character datatype. This datatype can support the same length of a **TDS_TEXT** datatype without the additional complexity.The maximum number of characters may be different than the number of bytes if the character set being used requires one, two, or four bytes to represent a character. This data type has a four byte unsigned



integer length field.

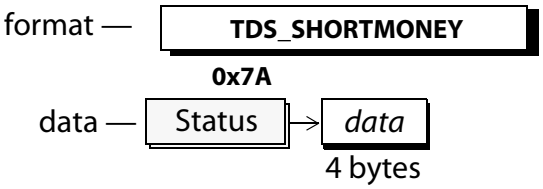
TDS_MONEY — 0x3C

This is a fixed length datatype of 8 bytes.



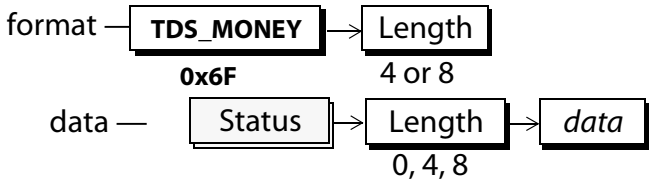
TDS_SHORTMONEY — 0x7A

This is a fixed length data type of 4 bytes.



TDS_MONEYN — 0x6E

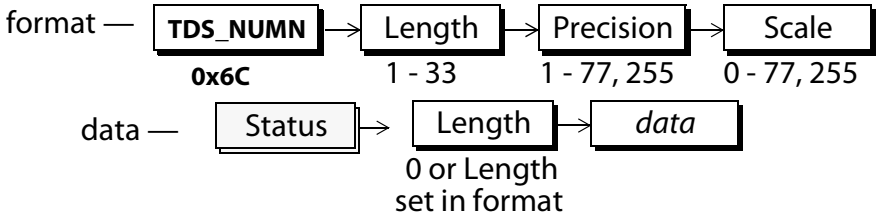
This is the same as the TDS_**MONEY** and **TDS_MONEY4** datatypes except that NULLS are allowed. The token and its data are preceded by an unsigned one-byte integer which has the value 0, 4 or 8. A NULL is indicated by a length value of 0.



The data length must either be 0 or the length specified in the format length argument. For example, if the format length is specified as 4, the data length can not be 8.

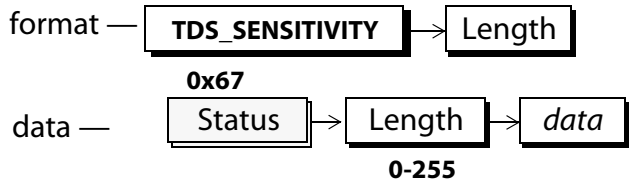
TDS_NUMN — 0x6C

This is the numeric datatype. The token is followed by bytes for data length, precision, and scale. The length byte describes the length of the data only and does not include the precision and scale bytes. Numeric has exactly the same format as the decimal datatype.



TDS_SENSITIVITY — 0x67

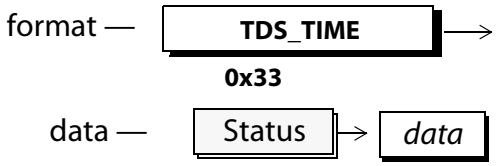
This datatype is used by secure versions of the SQL Server. It is exactly like the **TDS_VARCHAR** datatype. A NULL value has a length of 0. This datatype may be from 0 to 255 bytes. The token and its data are preceded by an unsigned one-byte integer which specifies its length. This data type is used



for security handshake during login processing. They may also exist as columns in a row. If a client uses capability bits to indicate that this data type is not supported, a server automatically converts this data type to a **TDS_VARCHAR**.

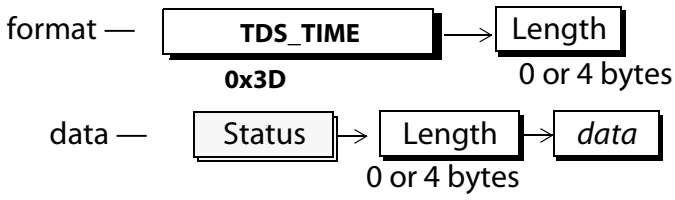
TDS_TIME — 0x33

TDS_TIME is a fixed length datatype of 4 bytes.



TDS_TIMEN — 0x93

TDS_TIMEN is a nullable version of the TDS_TIME datatype.

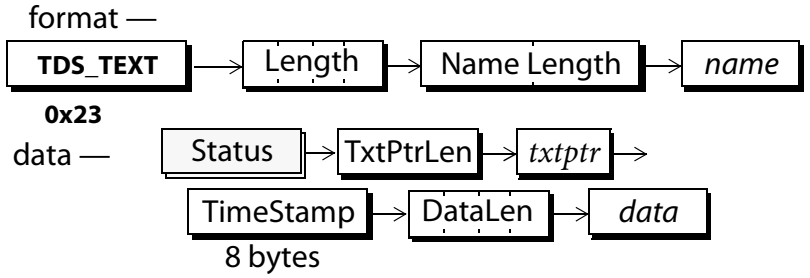


TDS_TEXT — 0x23

This is a character datatype.This is a large binary datatype. TxtPtrLen gives the length in bytes of the following txtptr argument. If TxtPtrLen is 0 then the value of the TEXT data item is SQL NULL and none of the other fields follow. The txtptr is a varbinary value (of length TxtPtrLen) which the database can use to re-locate the source of this data if the client wants to modify it.

TimeStamp is an 8-byte binary value which is automatically changed on the database whenever a TEXT value is changed. If the client uses a BULK_WRITE stream to update this value it must pass this timestamp value into the WRITETEXT clause. If the timestamp doesn't match the current value on the server the update will fail because there has been an intervening modification to that TEXT value.

DataLen is a 4 byte, unsigned value which indicates the length in bytes of the following data . Data is the value of the TEXT column.

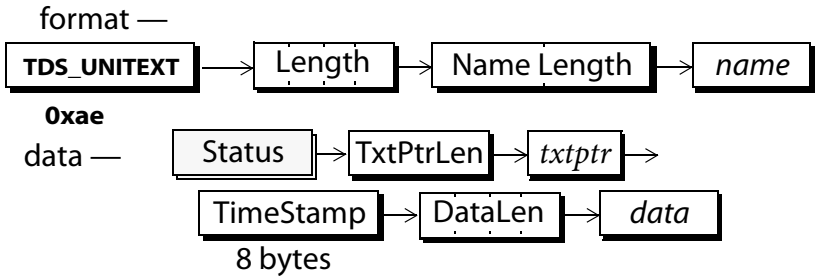


TDS_UNITEXT — 0xAE

This is a large Unicode character datatype. TxtPtrLen gives the length in bytes of the following txtptr argument. If TxtPtrLen is 0 then the value of the UNITEXT data item is SQL NULL and none of the other fields follow. The txtptr is a varbinary value (of length TxtPtrLen) which the database can use to re-locate the source of this data if the client wants to modify it.

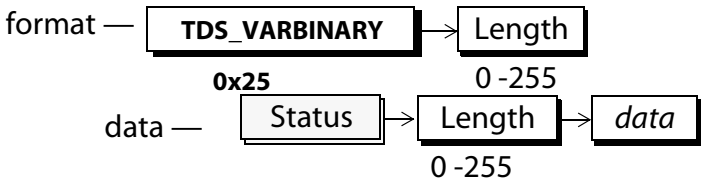
TimeStamp is an 8-byte binary value which is automatically changed on the database whenever a UNITEXT value is changed. If the client uses a BULK_WRITE stream to update this value it must pass this timestamp value into the WRITETEXT clause. If the timestamp doesn't match the current value on the server the update will fail because there has been an intervening modification to that UNITEXT value.

DataLen is a 4 byte, unsigned value which indicates the length in bytes of the following data . Data is the value of the UNITEXT column.



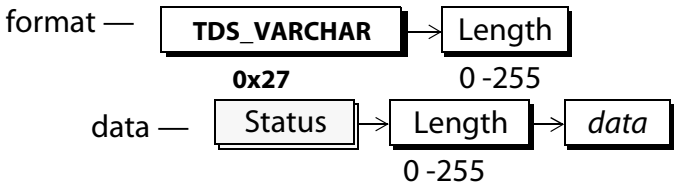
TDS_VARBINARY — 0x25

This is variable length nullable binary datatype. Its length may vary from 1 to 255 bytes. The length is specified by a one-byte unsigned integer which precedes the datatype token and the data. A NULL value has a length of 0. There is no way to represent a non-NULL empty string of length 0.



TDS_VARCHAR — 0x27

This is a variable length nullable character datatype. A NULL value has a length of 0. There is no way to represent a non-NULL empty string of length 0. This datatype may be from 0 to 255 bytes. The token and its data are preceded by an unsigned one-byte integer which specifies its length.

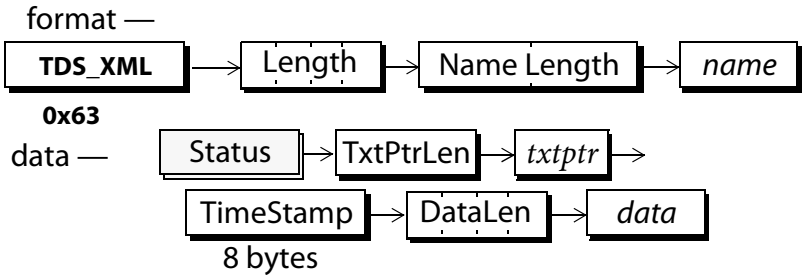


TDS_XML — 0xA3

This is an XML document datatype. TxtPtrLen gives the length in bytes of the following txtptr argument. If TxtPtrLen is 0 then the value of the TEXT data item is SQL NULL and none of the other fields follow. The txtptr is a varbinary value (of length TxtPtrLen) which the database can use to re-locate the source of this data if the client wants to modify it.

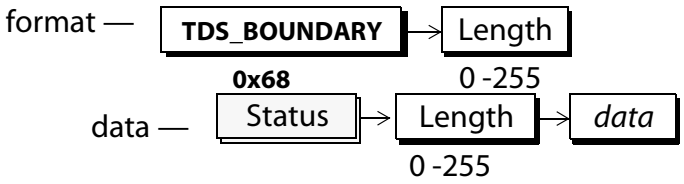
TimeStamp is an 8-byte binary value which is automatically changed on the database whenever a XML value is changed. If the client uses a BULK_WRITE stream to update this value it must pass this timestamp value into the WRITETEXT clause. If the timestamp doesn't match the current value on the server the update will fail because there has been an intervening modification to that XML value.

DataLen is a 4 byte, unsigned value which indicates the length in bytes of the following data . Data is the value of the XML column.



TDS_BOUNDARY — 0x68

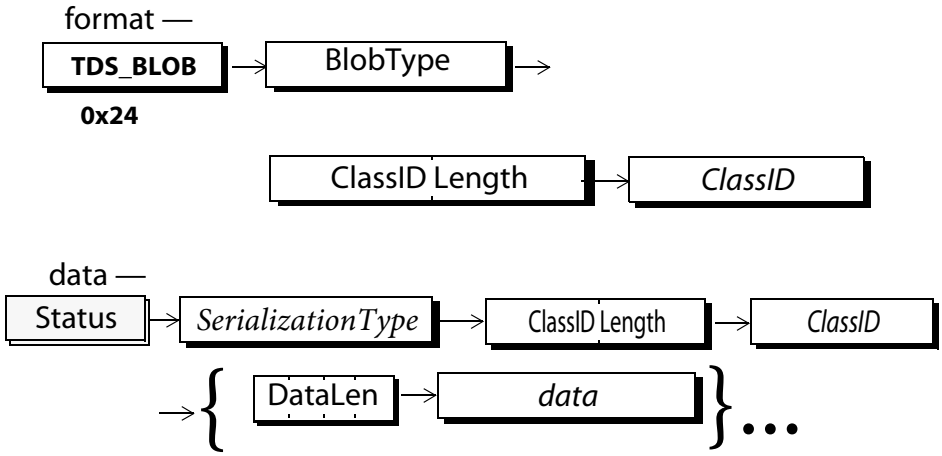
This is a variable length nullable character datatype. A NULL value has a length of 0. There is no way to represent a non-NULL empty string of length 0. This datatype may be from 0 to 255 bytes. The token and its data are preceded by an unsigned one-byte integer which specifies its length.This



data type is used for security handshake during login processing. They may also exist as columns in a row. If a client uses capability bits to indicate that this data type is not supported, a server automatically converts this data type to a **TDS_VARCHAR**.

TDS_BLOB — 0x24

This is a streaming/chunked datatype. It may represent a serialized object,



or a long binary or character datatype.

The **BlobType** - indicates what type of serialized data this is. Valid **BlobType** values are listed in *Table 36:* .

The **ClassIDLength** field indicates how long the next **ClassID** byte array is. If this value is 0, then the **ClassID** field will be absent.

The **ClassID** byte array identifies the type of Object which the column was declared to contain. All rows in that column are subclasses of this Class. How this **ClassID** should be interpreted depends on the **BlobType** value. In the case of Java Objects using Native serialization, ClassID may be missing since the serialization internally contains the name of the Class which each object is an instance of.

The **SerializationType** - indicates how the members of the object are actually

Table 36:

BlobType	ClassID meaning
0x01	The fully qualified name of the class (“com.foo.Bar”). This is a Character String in the negotiated TDS character set currently in use on this connection.
0x02	4-byte integer (database ID) 4-byte integer(sysextypes number of this class definition in this database). Both integers are in the byte-ordering negotiated for this connection.
0x03	This is long character data and has no ClassID associated with it.
0x04	This is long binary data and has no ClassID associated with it.
0x05	This is unichar data with no ClassID associated with it. It is

represented in the following **data** field **SerializationType** meanings depend on the **BlobType** and are summarized in *Table 37:* .

Table 37:

BlobType	Serialization	Meaning
0x01, 0x02	0x01	Native Java Serialization
0x03	0x00	Characters are in their native format, the character set of the data is the same as that of all other character data as negotiated on the connection during login.
0x04	0x00	Binary data in its normal form
0x05	0x00	This is unichar data with normal UTF-16 encoding with byte-order identical to that of the client.
0x05	0x01	This is unichar data in its UTF-8 encoding.
0x05	0x02	This is unichar data in SCSU (compressed) encoding.

ClassID Length gives the length of the following **ClassID** character string. If **ClassID** Length is 0, then this object is exactly an instance of the column type class (from the FORMAT) stream, and the following **ClassID** token will be missing.

ClassID Has the same meaning as ClassID from the Format token, but indicates the specific sub-class that this Object is of the declared class for the column.

DataLen is a 4-byte field The high-order bit indicates whether this is the last (0) **DataLen**/**Data** pairs, or if there is another **DataLen** value after the **Data** array (1). The low-order 31 bytes is an unsigned length of the following **Data** array.

Data is a byte array which contains the serialized value of the object.

- The DataLen/Data pairs continue until a DataLen with a clear high-bit is seen. If that final DataLen has a value of 0 then no additional Data array follows it (This is sort of a NULL terminated data stream). This allows us to pass Objects of arbitrary size with out having to first know how large these objects are).
- A value of 0x80000000 is legal, and means simply that the length of the following **Data** stream is 0, and thus the next item will be another 4-byte **DataLen**.
- There is no requirement that the lengths of the stream of Data chunks be the same.

See Also

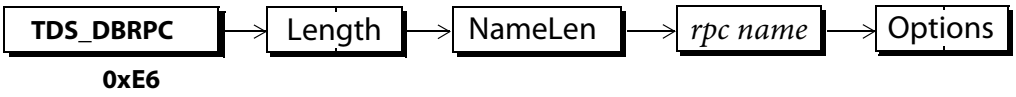
TDS_ROWFMT,TDS_ PARAMFMT, TDS_ROW, TDS_PARAMS, TDS_ALTFMT, TDS_RPC

TDS_DBRPC

Function

Describes the data stream which contains a data base remote procedure call request.

Syntax



Arguments

TDS_DBRPC This is the command token to send an data base RPC request.

Length This is the length, in bytes, of the remaining **TDS_DBRPC** data stream. It is a two-byte, unsigned integer.

NameLen This is length, in bytes, of the RPC name. It is a one-byte, unsigned integer.

rpc name This is the name of the RPC. Its length, in bytes, is given by the preceding argument.

Options This is a bit mask which contains options related to the RPC. The mask is a two-byte, unsigned integer. The defined options are:

Table 38: RPC Option Values

Name	Value	Description
TDS_RPC_UNUSED	0x000	Options field is unused.
TDS_RPC_RECOMPILE	0x0001	Recompile the RPC before execution.
TDS_RPC_PARAMS	0x0002	There are parameters associated with this RPC.

Comments

- This token is used by a client to make an RPC request to a server.
- Only one **TDS_DBRPC** token per request is allowed.
- Parameter data is sent using the **TDS_PARAMFMT/PARAMS** data stream tokens.
- There are two protocols supported for RPCs and return parameters in TDS 5.0. This is because the original 10.0 release was shipped using the **TDS_RPC** and **TDS_RETURNVALUE** tokens to send RPCs and return parameters. However, the **TDS_RPC** token had a 64K-1 byte limit that was unacceptable. This was resolved by using the **TDS_DBRPC** and **TDS_PARAMFMT/PARAMS** tokens for RPCs and return parameters.
- The **TDS_DBRPC** token will be used by clients if the **TDS_REQ_PARAM** capability bit is true.
- Return parameters will be returned to a client using the **TDS_PARAMFMT/PARAMS** tokens if the **TDS_RES_NOPARAM** capability bit is false.

Examples

See Also

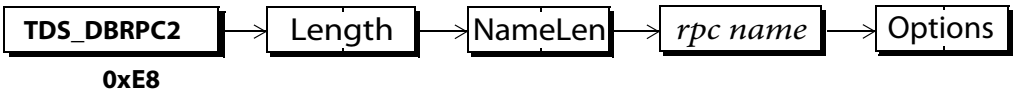
TDS_PARAMFMT, TDS_PARAMS, TDS_RPC, TDS_RETURNVALUE

TDS_DBRPC2

Function

Describes the data stream which contains a data base remote procedure call request.

Syntax



Arguments

TDS_DBRPC2 This is the command token to send an data base RPC request.

Length This is the length, in bytes, of the remaining **TDS_DBRPC2** data stream. It is a two-byte, unsigned integer.

NameLen This is length, in bytes, of the RPC name. It is a two-byte, unsigned integer.

rpc name This is the name of the RPC. Its length, in bytes, is given by the preceding argument.

Options This is a bit mask which contains options related to the RPC. The mask is a two-byte, unsigned integer. The defined options are:

Table 39: RPC Option Values

Name	Value	Description
TDS_RPC_UNUSED	0x000	Options field is unused.
TDS_RPC_RECOMPILE	0x0001	Recompile the RPC before execution.
TDS_RPC_PARAMS	0x0002	There are parameters associated with this RPC.

Comments

- This token is used by a client to make an RPC request to a server.
- The only difference between **TDS_DBRPC** and **TDS_DBRPC2** is the size of the rpc name field.
- Only one **TDS_DBRPC** or **TDS_DBRPC2** token per request is allowed.
- Parameter data is sent using the **TDS_PARAMFMT/PARAMS** data stream tokens. The use of **TDS_PARAMFMT2** is also allowed.
- There are two protocols supported for RPCs and return parameters in TDS 5.0. This is because the original 10.0 release was shipped using the **TDS_RPC** and **TDS_RETURNVALUE** tokens to send RPCs and return parameters. However, the **TDS_RPC** token had a 64K-1 byte limit that was unacceptable. This was resolved by using the **TDS_DBRPC** and **TDS_PARAMFMT/PARAMS** tokens for RPCs and return parameters.
- The **TDS_DBRPC2** token will be used by clients only if the **TDS_REQ_DBRPC2** capability bit is true. A client can use **TDS_DBRPC** in lieu of **TDS_DBRPC2** if it does not need the extra byte for NameLen.
- Note that gateway servers do not always mask capability bits to disable features not supported by the gateway. This is a long standing issue that we recently discovered. Attempts to send **TDS_DBRPC2** through these servers may fail if the server is not in full passthrough mode. It is suggested that clients only send **TDS_DBRPC2** if the extra length is needed.
- Return parameters will be returned to a client using the **TDS_PARAMFMT/PARAMS** tokens if the **TDS_RES_NOPARAM** capability bit is false.

Examples

See Also

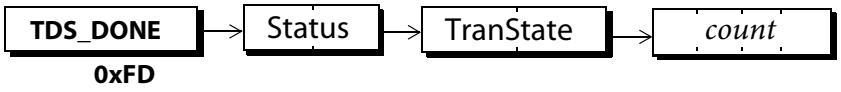
TDS_DBRPC, TDS_PARAMFMT, TDS_PARAMS, TDS_RPC, TDS_RETURNVALUE

TDS_DONE

Function

Indicates completion status of a command.

Syntax



Arguments

TDS_DONE This token is used to indicate command completion status.

Status This field is a two-byte, unsigned integer and is a bit field indicating the completion status. The possible bits are:

TDS_DONE_FINAL - 0x0000

This is the final result for the last command. It indicates that the command has completed successfully.

TDS_DONE_MORE - 0x0001

This Status indicates that there are more results to follow for the current command.

TDS_DONE_ERROR - 0x0002

This indicates that an error occurred on the current command.

TDS_DONE_INXACT - 0x0004

There is a transaction in progress for the current request.

TDS_DONE_PROC - 0x0008

This **TDS_DONE** is from the results of a stored procedure.

TDS_DONE_COUNT - 0x0010

This Status indicates that the *count* argument is valid. This bit is used to distinguish between an empty *count* field and a *count* field with a value of 0.

TDS_DONE_ATTN - 0x0020

This **TDS_DONE** is acknowledging an attention command.

TDS_DONE_EVENT - 0x0040

This **TDS_DONE** was generated as part of an event notification.

TranState This is a two-byte, unsigned integer field. It indicates the current state of the transaction on this connection.

Table 40: Transaction State Values

Name	Value	Description
TDS_NOT_IN_TRAN	0	Not currently in a transaction
TDS_TRAN_SUCCEED	1	Request caused transaction to complete successfully.
TDS_TRAN_PROGRESS	2	A transaction is still in progress on this dialog.
TDS_STMT_ABORT	3	Request caused a statement abort to occur.
TDS_TRAN_ABORT	4	Request caused transaction to abort.

count This is a four-byte integer. If **TDS_DONE_COUNT** is set in the Status argument, count contains the number of rows affected by the current command.

Comments

- **TDS_DONE** is used to indicate the completion status of a command. Multiple commands may be sent in one request. The result sets for each command are terminated by a **TDS_DONE**. When multiple result sets are returned, all but the final **TDS_DONE** will have the **TDS_DONE_MORE** bit set in the Status field.
- The server returns the current transaction state to the client in the TranState.
- The TranState field was redefined from an Info field in TDS 5.0.

- Stored procedures return **TDS_DONEINPROC** and **TDS_DONEPROC** tokens instead of **TDS_DONE**s.

Examples

See Also

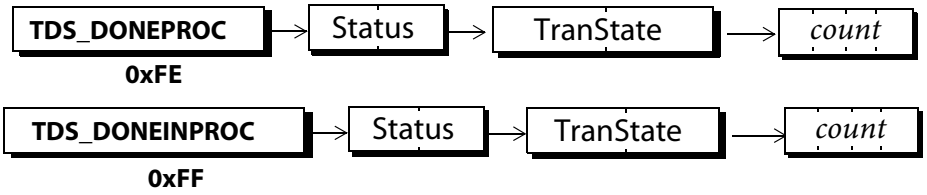
TDS_DONEPROC, TDS_DONEINPROC

TDS_DONEPROC, TDS_DONEINPROC

Function

Indicates completion status of stored procedure commands.

Syntax



Arguments

TDS_DONEPROC **TDS_DONEINPROC** These tokens are used to indicate completion status from stored procedure commands.

Status This field is a two-byte, unsigned integer and is a bit field indicating the completion status. The possible bits are:

TDS_DONE_FINAL - 0x0000

This is the final result for the last command. It indicates that the command has completed successfully.

TDS_DONE_MORE - 0x0001

This Status indicates that there are more results to follow for the current command.

TDS_DONE_ERROR - 0x0002

This indicates that an error occurred on the current command.

TDS_DONE_INXACT - 0x0004

There is a transaction in progress for the current request.

TDS_DONE_COUNT - 0x0010

This Status indicates that the *count* argument is valid. This bit is used to distinguish between an empty *count* field and a *count* field with a value of 0.

TDS_DONE_ATTN - 0x0020

This **TDS_DONE** is acknowledging an attention command.

TranState This is a two-byte, unsigned integer field. It indicates the current state of the transaction on this connection.

Table 41: Transaction State Values

Name	Value	Description
TDS_NOT_IN_TRAN	0	Not currently in a transaction
TDS_TRAN_SUCCEED	1	Request caused transaction to complete successfully.
TDS_TRAN_PROGRESS	2	A transaction is still in progress on this dialog.
TDS_STMT_ABORT	3	Request caused a statement abort to occur.
TDS_TRAN_ABORT	4	Request caused transaction to abort.

count This is a four-byte integer. If the **TDS_DONE_COUNT** bit in the Status field is set, then the count is meaningful and it gives the number of rows that were affected by the current command.

Comments

- If all the statements in a stored procedure have been executed a **TDS_DONEPROC** is returned. However, a **TDS_DONEPROC** may have the **TDS_DONE_MORE** bit set in the Status field if there are more statements to be executed. This can happen if a stored procedure has called another stored procedure. There will a separate **TDS_DONEPROC** for each stored procedure that gets called.

- Each statement in a stored procedure that executes will return a **TDS_DONEINPROC**. All statements in Transact-SQL are considered statements except variable declarations. For example, assignment of a variable is considered a separate statement and a **TDS_DONEINPROC** will be generated. The stored procedure itself is considered a statement so a stored procedure consisting of a single *select* will generate a **TDS_DONEINPROC** for the *select* followed by a **TDS_DONEPROC** for the completion of the stored procedure.
- A **TDS_DONEINPROC** is guaranteed to be followed by another **TDS_DONEINPROC** or **TDS_DONEPROC**. A **TDS_DONEPROC** will be followed by another **TDS_DONEINPROC** or **TDS_DONEPROC** only if the **TDS_DONE_MORE** bit is set in the Status field.
- For execution of stored procedures **TDS_DONEINPROC** and **TDS_DONEPROC** tokens are returned instead of **TDS_DONE**s.
- The server returns the current transaction state to the client in the TranState.
- The TranState field was redefined from an Info field in TDS 5.0.

Examples

In this example we'll execute a stored procedure which does a *select*, calls another stored procedure and then does another *select*. The procedure *proc1* looks like:

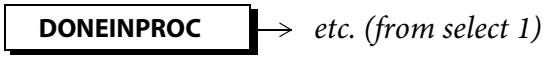
```
select 1
execute procedure proc2
select 3
```

Proc2 looks like:

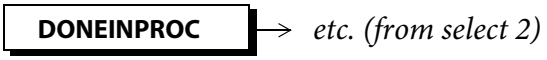
```
select 2
```

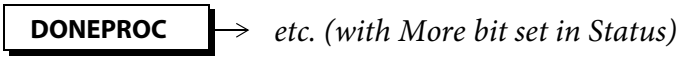
When we execute *proc1* the datastream from the server looks like:

row info and data for first select

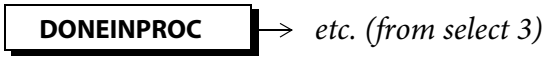


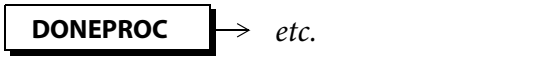
row info and data for second select





row info and data for third select





See Also

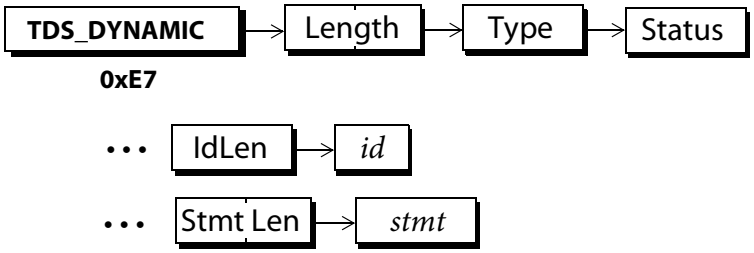
TDS_DONE

TDS_DYNAMIC

Function

A request to prepare or execute a dynamic SQL statement.

Syntax



Arguments

TDS_DYNAMIC This is token indicates that this is a dynamic SQL command.

Length This is the total length, in bytes, of the remaining datastream. It is a two-byte, unsigned integer.

Type This indicates the type of dynamic operation. Type is a one-byte integer. Its values are:

Table 42: Dynamic Operation Types

Name	Value	Description
TDS_DYN_PREPARE	0x01	This is a request to prepare *stmt*.
TDS_DYN_EXEC	0x02	This is a request to execute a prepared statement.
TDS_DYN_DEALLOC	0x04	Request to deallocate a prepared statement.
TDS_DYN_EXEC_IMMED	0x08	This a request to prepare and execute *stmt* immediately.

Table 42: Dynamic Operation Types

Name	Value	Description
TDS_DYN_PROCNAME	0x10	Is this used? If so what for?
TDS_DYN_ACK	0x20	Acknowledge a dynamic command.
TDS_DYN_DESCIN	0x40	Send input format description.
TDS_DYN_DESCOUT	0x80	Send output format description.

Status This is the status associated with this dynamic command. Status is a one-byte unsigned integer argument. It has the following valid values:

Table 43: Dynamic Status Values

Name	Value	Description
TDS_DYNAMIC_UNUSED	0x00	No status associated with this dynamic command.
TDS_DYNAMIC_HASARGS	0x01	Parameter data stream follows the dynamic command.
TDS_DYNAMIC_SUPPRESS_FMT	0x02	If this statement, as identified by id, has previously sent **TDS_ROWFMT** information and this information has not changed, do not resend **TDS_ROWFMT**.

IdLen This the length, in bytes, of the statement id which follows. The statement id may be up to 255 bytes long. It must be at least one byte long. IdLen is a one-byte, unsigned integer.

id This is the statement id. It may be up to 255 bytes long. In practice, a maximum length of 30 is widely supported. The id is a character string and must be at least one byte long.

Stmt Len This is the length of the statement. See the comments section below for information on how this argument is used.

stmt This is the statement that is to be either prepared or executed. It is a character string whose length is given, in bytes, by the previous argument. The maximum length of the statement is 32767 - 2 - the length of the statement id. This argument is only in the data stream if StmtLen is non-0.

Comments

- In SQL pre-compilers that support dynamic SQL, the prepared statement is common. It allows the client to send a SQL statement to the server to be "prepared" and then later executed, perhaps repeatedly. It is similar to a Sybase stored procedure except that it's life is limited to the client session.
- When a statement is prepared, the server will return a description of the output, if any, using the **TDS_ROWFMT** data stream. If there are any input parameters, they will be described at the same time using the **TDS_PARAMFMT** data stream.
- Each **TDS_DYNAMIC** data stream is acknowledged with a **TDS_DONE** data stream.
- The following **TDS_CAP_REQUEST** capability bits are defined for the dynamic protocol:

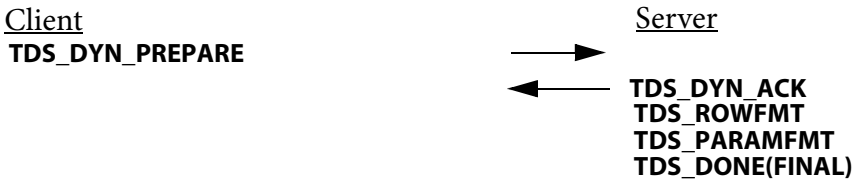
Table 44: Dynamic Protocol Capabilities

Capability	Description
TDS_PROTO_DYNAMIC	If this capability is enabled (1) the **TDS_DYN_DESCOUT/DESCIN** protocol is used to send input and output formats to a client. If this capability is disabled (0), the format information is sent back automatically by the server at **TDS_DYN_PREPARE** time.
TDS_PROTO_DYNPROC	If this capability is enabled (1) a client library will prepend "create proc" in the Stmt field of the **TDS_DYN_PREPARE** data stream. If this capability is disabled (0) a client library will just send the Stmt information un-modified.

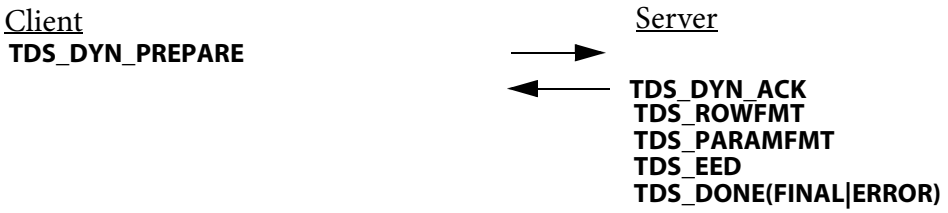
- The **TDS_CURDECLARE** token is used to declare a cursor on a prepared statement. it is the client library's responsibility to associate the prepared statement name with the **TDS_CURDECLARE** token. The prepared statement name must be in the Statement argument of the **TDS_CURDECLARE** data stream and the **TDS_CUR_DOPT_DYNAMIC** bit must be set in the Option argument.
- Only one **TDS_DYNAMIC** token can be sent in a request.
- The Stmt argument is only used in the **TDS_DYN_PREPARE** and **TDS_DYN_EXEC_IMMED** data streams. StmtLen must be set to 0 in all other dynamic data streams.
- Parameters are not supported in the **TDS_DYN_EXEC_IMMED** data stream.
- The IdLen argument must be 0 for a **TDS_DYN_EXEC_IMMED** data stream.
- No results can be returned by a server in response to a **TDS_DYN_EXEC_IMMED** command. The only valid response is a **TDS_DONE**.
- Only one **TDS_PARAMFMT/TDS_ROWFMT** is legal when responding to a **TDS_DYN_PREPARE/TDS_DYN_DESCIN/TDS_DYN_DESCOUT** command.
- Compute rows are illegal in the dynamic protocol.
- Parameter names are not supported in the **TDS_PARAMFMT** associated with the **TDS_DYN_EXEC**.
- The **TDS_DYNAMIC_SUPPRESS_FMT** status bit is valid only if **TDS_RES_SUPPRESS_ROWFMT** is set. In this case, subsequent executions of the prepared statement does not need to generate **TDS_ROWFMT** information. Note that the server may still do so. It is suggested that client libraries make the decision to suppress format information configurable since the use of stored procedures can result in different format information being returned on each execution.

Protocol Examples

Prepare - Success (**TDS_PROTO_DYNAMIC** == 0)



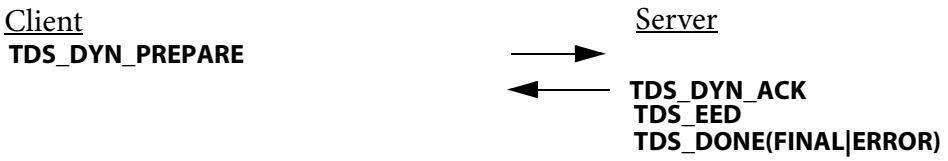
Prepare - Failure (**TDS_PROTO_DYNAMIC** == 0)



NOTE: The **TDS_EED** token could occur any where in the server response data stream. Also, the **TDS_ROWFMT/PARAMFMT** may not be returned from the server. It is possible to receive just a **TDS_ROWFMT**, a **TDS_ROWFMT/PARAMFMT**, or no format information.

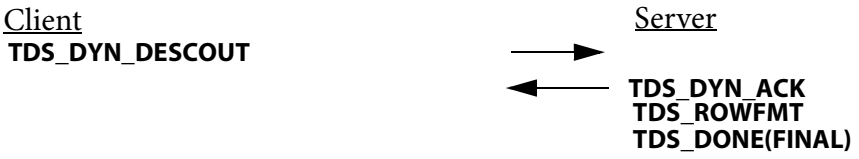
Prepare - Success (**TDS_PROTO_DYNAMIC** == 1)

Prepare - Failure (**TDS_PROTO_DYNAMIC** == 1)

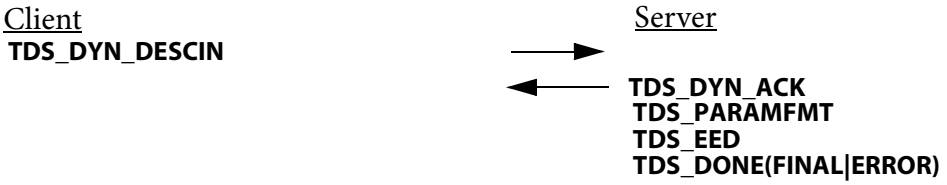


Describe Input Paramters(**TDS_PROTO_DYNAMIC** == 1)

Describe Output Rows (**TDS_PROTO_DYNAMIC** == 1)

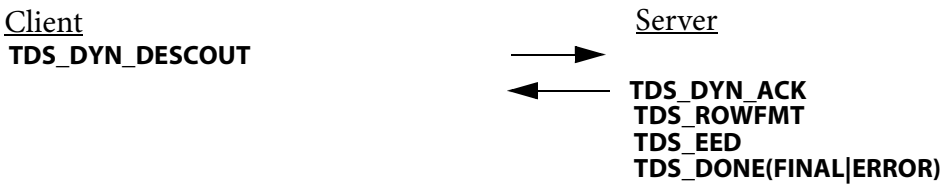


Describe Input Parameters - failure(**TDS_PROTO_DYNAMIC** == 1)



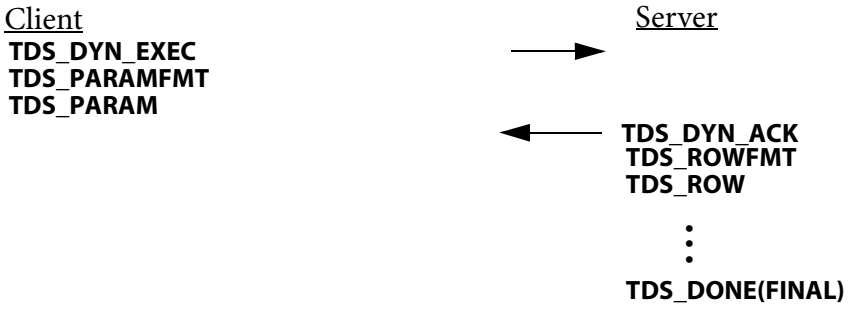
NOTE: The **TDS_PARAMFMT** may not be returned if an error is detected.

Describe Output Rows - failure(**TDS_PROTO_DYNAMIC** == 1)

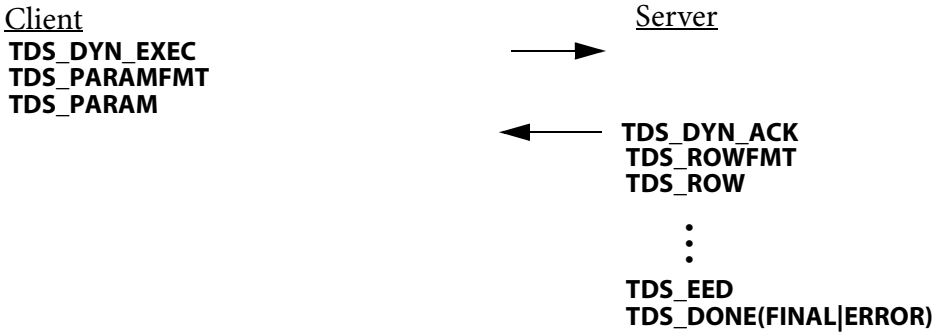


NOTE: The **TDS_ROWFMT** may not be returned if an error is detected.

Execute



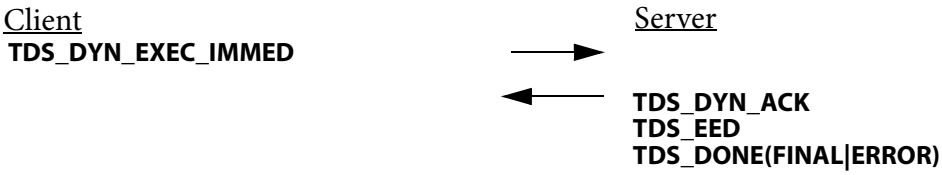
Execute - failure



NOTE: The **TDS_ROWFMT** and **TDS_ROW**(s) may not be received if an error is detected.

Execute Immediate

Execute Immediate - Failure



Deallocation

See Also

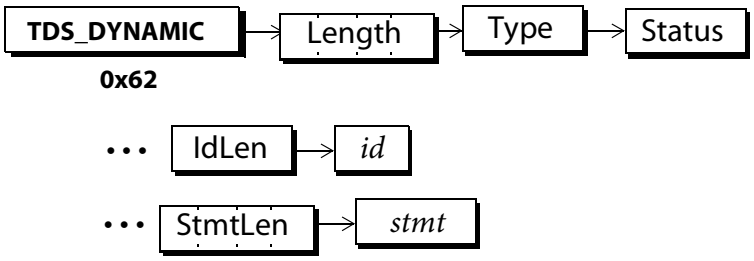
TDS_EED TDS_ROW, TDS_PARAMFMT, TDS_ROWFMT

TDS_DYNAMIC2

Function

A request to prepare or execute a dynamic SQL statement. This token is identical to the TDS_DYNAMIC token, except it has a 4-byte length and StmtLen is expanded to 4 bytes to accommodate longer statements.

Syntax



Arguments

TDS_DYNAMIC This is token indicates that this is a dynamic SQL command.

Length This is the total length, in bytes, of the remaining datastream. It is a four-byte, unsigned integer.

Type This indicates the type of dynamic operation. Type is a one-byte integer. See TDS_DYNAMIC for a description of its values.

Status This is the status associated with this dynamic command. Status is a one-byte unsigned integer argument. See TDS_DYNAMIC for its valid values:

IdLen This the length, in bytes, of the statement id which follows. The statement id may be up to 255 bytes long. It must be at least one byte long. IdLen is a one-byte, unsigned integer.

id This is the statement id. It may be up to 255 bytes long. In practice, a maximum length of 30 is widely supported. The id is a character string and must be at least one byte long.

StmtLen This is the length of the statement. See the comments section below for information on how this argument is used.

stmt This is the statement that is to be either prepared or executed. It is a character string whose length is given, in bytes, by the previous argument.

Comments

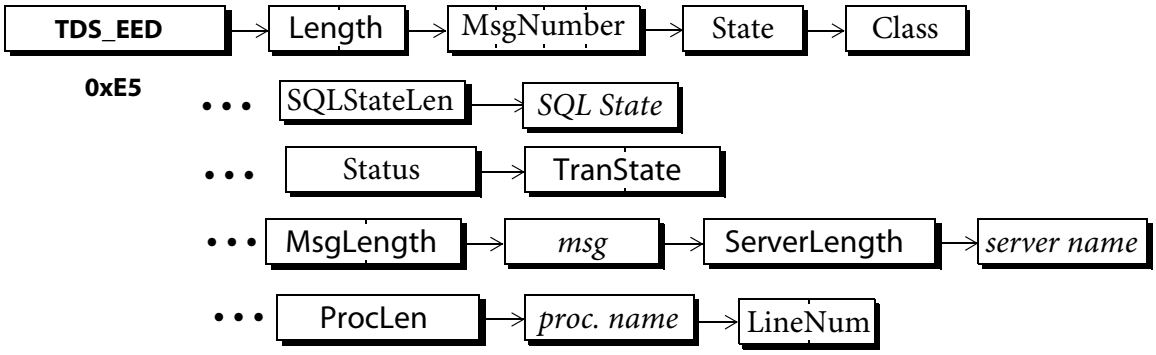
- See TDS_DYNAMIC for comments.

TDS_EED

Function

Return a text message to a client.

Syntax



Arguments

TDS_EED This is the data stream command token that indicates that this is a data stream containing a text message.

Length This is the length, in bytes, of the remaining data stream. It is a two-byte unsigned integer.

MsgNumber This is the message number for the message. MsgNumber is a four-byte, unsigned integer.

State This is the message state. It is used as a modifier to the MsgNumber. It is a one-byte, unsigned integer.

Class This is the class or severity of the message. It is a one-byte unsigned integer.

SqlStateLen This the length of the SQL state argument that follows.

SQL State This is the SQL state value associated with this message. Its length is in SQLStateLen. This argument is treated as binary data. No character set conversion will occur.

Status This is the status associated with this extended message. this argument is a one-byte unsigned integer. It has the following valid values:

Table 45: Valid Status Values

Name	Value	Description
TDS_NO_EED	0x00	No extended error data follows.
TDS_EED_FOLLOWS	0x01	Extended error data follows this token. Extended error data is sent using **TDS_PARAMFMT/PARAM**
TDS_EED_INFO	0x02	Extended, library private data follows. The error is not to be delivered to the application.

TranState This is the current state of any transactions that are active on this dialog. See the **TDS_DONE** man page for valid values for this argument. This argument is a two-byte unsigned integer.

MsgLength This is the length of the *msg* text that follows. It a two-byte, unsigned integer. Note that the total length of the **TDS_EED** data stream must be no longer than 64k-1. Since the data stream includes other information in addition to the *msg*, the actual length that *msg* can be is less than 64k-1. How much less depends on the length of the other fields in the **TDS_EED** data stream.

msg This is the actual text of the message. Its length, in bytes, is in MsgLength.

ServerLength This is the length of the server name argument which follows. It may be 0. It is a one-byte, unsigned integer.

server name This is the name of the server that is sending the message. It will be omitted if ServerLength is 0.

ProcLength This is the length of the *proc. name* argument which follows. It may be 0. It is a one-byte, unsigned integer.

proc. name This is the name of the stored procedure or RPC in which the message occurred. It will be omitted if ProcLength is 0.

LineNum This is the line number in the command batch or stored procedure that has the error, if applicable. Line numbers start at 1 so if LineNum isn't applicable to the message, it will be 0. It is a two-byte, unsigned integer.

Comments

- This is the data stream that is sent from the server to return a text message to a client. These messages are usually sent because an error was detected.
- A server may send multiple **TDS_EED** tokens in one response.
- The **TDS_EED** token is sent in place of the **TDS_ERROR/INFO** tokens when the **TDS_RES_NOEED** capability is not enabled (0).
- The Status field must be set to **TDS_EED_FOLLOWS** if extended error data follows. Any number of parameters may be sent following a **TDS_EED** token.
- A **TDS_EED** token cannot come between regular results. It either has to come before any results, or after all of the results.
- Multiple **TDS_EED** tokens can follow regular results. The multiple **TDS_EED**s are differentiated using a **TDS_DONE(MORE)**.
- Any results values that follow a **TDS_EED** for another command batch must be preceded by a **TDS_DONE(MORE)**.
- Errors generating the **TDS_EED** data stream are reported by a server by setting the **ERROR** bit in the **TDS_DONE(MORE)** token associated with the **TDS_EED**. The **ERROR** bit is the only valid status bit in the **TDS_DONE** data stream other than **MORE** for **TDS_EED** data streams.
- The **TDS_EED** token replaces both the **TDS_ERROR** and **TDS_INFO** tokens in earlier versions of TDS.
- If **TDS_EED_INFO** is set in the status field, the information that follows is intended to be interpreted only by the client library. The message number is independent of the server's message space and is defined in *Table 46: TDS_EED_INFO Message Numbers* .

Protocol Examples

Sending an Extended Error Data Stream

```
TDS_ROWFMT+
TDS_ROW+
      •
      •
      •
TDS_EED(TDS_EED_FOLLOWS)
TDS_PARAMFMT*
TDS_PARAM*
      •
      •
      •
 TDS_DONE(FINAL)**

  + Regular Results
  * Extended error data
  ** This TDS_DONE delineates both result set and the
  TDS_EED data stream.
```

Sending an Extended Error Data Stream with multiple result sets

```
TDS_ROWFMT+
TDS_ROW+
      •
      •
      •
TDS_EED(TDS_EED_FOLLOWS)
TDS_PARAMFMT*
TDS_PARAM*
      •
      •
      •
 TDS_DONE(MORE)**
 TDS_DONE(MORE)***
 TDS_ROWFMT++
 TDS_ROW++
       •
       •
       •
 TDS_DONE(FINAL)

+First Result Set
* Extended error data
** This TDS_DONE(MORE) delineates the
TDS_EED data stream.
*** This TDS_DONE(MORE) delineates the first result set
++ Second result set
```

Reporting an Error while generating a **TDS_EED** stream

TDS_ROWFMT+
TDS_ROW+
•
•
•
TDS_EED(TDS_EED_FOLLOWS)
TDS_DONE(MORE|ERROR)*
TDS_DONE(MORE)**
TDS_ROWFMT++
TDS_ROW++
•
•
•
TDS_DONE(FINAL)

+First Result Set
*** This TDS_DONE(MORE|ERROR) indicates that an error occurred while generating the TDS_EED token. NOTE: No parameters were sent in this example. It is undefined whether parameters are sent when an error occurs.**
**** This TDS_DONE(MORE) delineates the first result set.**
++ Second result set.

Table 46: TDS_EED_INFO Message Numbers

Message	State Value	Description
TDS_REDIRECT 1	**TDS_EED_IMMEDIATE_REDIRECT** 0x01 **TDS_EED_SET_REDIRECT** 0x02	This message is only sent to a client that has set **TDS_HA_LOG_REDIRECT** or **TDS_CAP_CLUSTERFAILOVER**. See notes following table.

q The **TDS_REDIRECT** message will only be generated if the client application sets the **TDS_HA_LOG_REDIRECT** bit in the TDS Login packet or the **TDS_CAP_CLUSTERFAILOVER** request bit in the capability mask. **TDS_HA_LOG_REDIRECT** modifies the login sequence. **TDS_CAP_CLUSTERFAILOVER** is used to either force a failover of an existing connection to a new server or to update a client's list of failover targets.

- When **TDS_HA_LOG_REDIRECT** is set, the server can send a **TDS_REDIRECT** message prior to sending the **TDS_LOGINACK** message. Clients that do not set **TDS_HA_LOG_SESSION** will ignore

this message if **TDS_EED_IMMEDIATE_REDIRECT** is not set. When **TDS_EED_IMMEDIATE_REDIRECT** is set, the server will follow with **TDS_LOGINACK(TDS_LOG_FAIL)** and the client is expected to attempt a connection to the systems in the parameter list that follows. If **TDS_EED_SET_REDIRECT** is set, the client application will continue to leave the **TDS_HA_LOG_REDIRECT** bit set in the subsequent connection attempts.

- When **TDS_CAP_CLUSTERFAILOVER** is set, the server can send a **TDS_REDIRECT** to cause a client to failover to a different server to execute subsequent commands. Should **TDS_EED_IMMEDIATE_REDIRECT** not be set, the connection information passed via the parameters will replace previously saved HAFAILOVER information.
- Each of the EED parameters for **TDS_REDIRECT** will describe connection information. This information will be character data (ie. **TDS_VARCHAR** or **TDS_LONGCHAR**.) The structure of each line will be "protocol connection information." The syntax is based off the information from *http://www-jdbc/xDocumentation/OIDMapping.html* site internal to Sybase. Valid protocols are tcp, decnet, spx, msnmp. If a client does not understand or support the communications implied by a particular parameter, it should ignore it and proceed to the next entry. If this means there are no valid parameters for the client, or if all connection attempts fail, the client will return a connection failure. The connection information will look like the following (note that the tcp syntax drops the placeholder between the 'tcp' keyword and the hostname):
    - tcp (hostname|address) port [filters]
    - decnet (hostname|address) (object name|object number)[filters]
    - spx netnumber nodenumber socketnumber [filters]
    - spx sapname [filters]
    - msnmp pipename [filters]
- The [filters] information is the same as the filter information stored in an interface file or LDAP directory server and may not be present. Currently, the only available filter is *ssl*. The syntax is as follows:
    - filters = filter ['' filters]

    - filter = filterName[=””DN””]
- The following are examples of connection information:
    - tcp dreamsystem 4500
    - tcp 10.10.10.10 4500 ssl

See Also

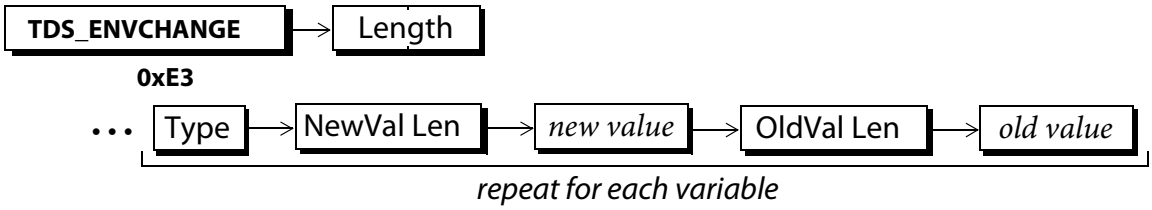
TDS_DONE, TDS_INFO, TDS_ERROR

TDS_ENVCHANGE

Function

Notify receiver of a change in the supported environmental variables.

Syntax



Arguments

TDS_ENVCHANGE This token indicates that this is a datastream containing environment change information.

Length This is the total length of the remaining environmental change data stream.

Type This one-byte, unsigned argument defines the variable affected by this command. The defined types are:

TDS_ENV_DB - 1
The current database.

TDS_ENV_LANG - 2
The current national language.

TDS_ENV_CHARSET - 3
The current character set.

TDS_ENV_PACKSIZE - 4
The current packet size, in bytes.

NewValLen This gives the length, in bytes, of the *new value* for the variable. The length may be 0.

new value This is the new value of the environment variable. Its length is given by the preceding argument. If length is 0, it will be omitted from the datastream.

OldValLen This gives the length, in bytes, of the *old value* for the variable. The length may be 0.

old value This is the old value of the environment variable. Its length is given by the preceding argument. If length is 0, it will be omitted from the datastream.

Comments

- This datastream is used to inform the receiver of any changes in any of the environment variables.
- More than one variable change can be described in a single datastream.

Examples

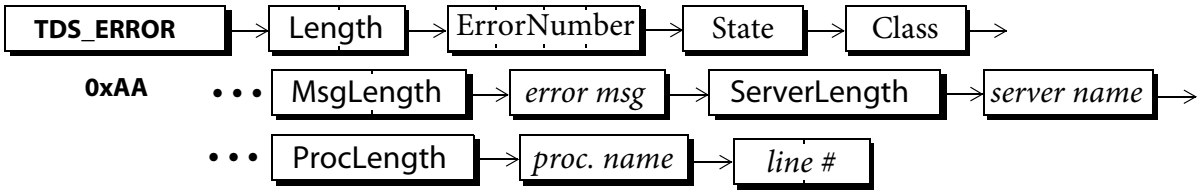
See Also

TDS_ERROR

Function

Describes the datastream which contains an error message.

Syntax



Arguments

TDS_ERROR This is the datastream command token that indicates that this is a datastream containing an error message.

Length This is the length, in bytes, of the remaining error message. It is a two-byte unsigned integer.

error number This the server-generated error number for the message. Error numbers below 20001 are reserved by the SQL Server. The number is a four-byte, signed integer.

state This is the error state. It is used as a modifier to the *error number.* It is a one-byte, unsigned integer.

class This is the class or severity of the error. In the SQL Server, a *class* of 10 or less indicates an information message. It is a one-byte unsigned integer.

MsgLength This is the length of the *msg* text that follows. It a two-byte, unsigned integer. Note that the total length of the **TDS_ERROR** datastream must be no longer than 64k-1. Since the datastream includes other information in addition to the *error msg*, the actual length that *error msg* can be is less than 64k-1. How much less depends on the length of the other fields in the **TDS_ERROR** datastream.

error msg This is the actual text of the error message. Its length, in bytes, is described in the preceding parameter.

Server Length This is the length of the server name parameter which follows. It may be 0. It is a one-byte, unsigned integer.

server name This is the name of the server that is sending the message. It will be omitted if ServerLength is 0.

ProcLength This is the length of the *proc. name* parameter which follows. It may be 0. It is a one-byte, unsigned integer.

proc. name This is the name of the stored procedure or rpc in which the error occurred It will be omitted if ProcLength is 0.

line # This is the line number in the command batch of stored procedure that has the error, if applicable. Line numbers start at 1 so if *line#* isn't applicable to the message, it will be 0. It is a two-byte, unsigned integer.

Comments

- This is the datastream that is sent from the server when an error occurs.
- A server may send multiple **TDS_ERROR** statements.
- The **TDS_ERROR** datastream is exactly the same as the**TDS_ INFO** datastream except for the token value.
- This token is obsolete and has been replaced by the **TDS_EED** token.

Examples

See Also

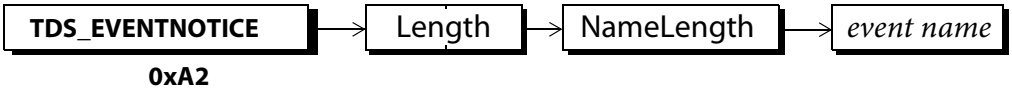
TDS_INFO, TDS_EED

TDS_EVENTNOTICE

Function

The data stream for sending a notice that an event has been raised.

Syntax



Arguments

TDS_EVENTNOTICE This is the data stream command token that indicates that this is a data stream containing an event notification.

Length This is the total length of the remaining data stream. It is a two-byte, unsigned integer.

NameLength This is the length, in bytes, of the name of the event which has been raised.

event name This is the event name of the event that has been raised. It's length is given by the preceding argument.

Comments

- This is the data stream sent by the server to the client when an event is raised. The client must have previously asked the server to send notification for a particular event.
- See the Event Notification chapter in this document for a complete description of the event notification protocol.

Examples

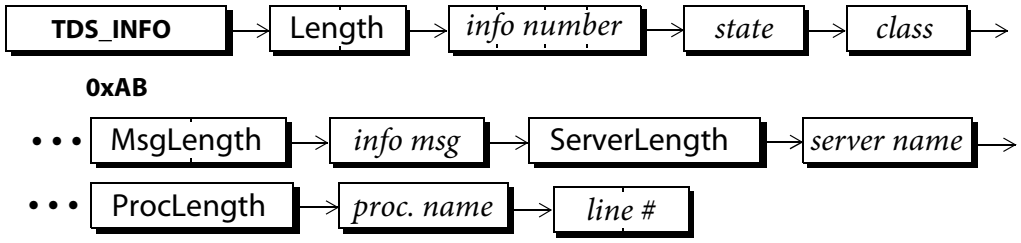
See Also

TDS_INFO

Function

Describes the datastream which contains an information message.

Syntax



Arguments

TDS_INFO This is the datastream command token that indicates that this is a datastream containing an information message.

Length This is the length, in bytes, of the remaining information datastream. It is a two-byte unsigned integer.

info number This the server-generated information number for the message. Information numbers below 20001 are reserved by the SQL Server. The number is a four-byte, signed integer.

state This is the information state. It is used as a modifier to the *info number*. It is a one-byte, unsigned integer.

class This is the class of the information message. In the SQL Server, a *class* of 10 or less indicates an information message. It is a one-byte unsigned integer.

MsgLength This is the length of the *msg* text that follows. It a two-byte, unsigned integer. Note that the total length of the **TDS_INFO** datastream must be no longer than 64k-1. Since the datastream includes other information in addition to the *info msg*, the actual length that *info msg* can be is less than 64k-1. How much less depends on the length of the other fields in the **TDS_INFO** datastream.

info msg This is the actual text of the information message. Its length, in bytes, is described in the preceding parameter.

ServerLength This is the length of the *server name* parameter which follows. It may be 0. It is a one-byte, unsigned integer.

server name This is the name of the server that is sending the message. It will be omitted if ServerLength is 0.

ProcLength This is the length of the *proc. name* parameter which follows. It may be 0. It is a one-byte, unsigned integer.

proc. name This is the name of the stored procedure or rpc in which the message occurred It will be omitted if ProcLength is 0.

line # This is the line number in the command batch of stored procedure that has the message, if applicable. Line numbers start at 1 so if *line#* isn't applicable to the message, it will be 0. It is a two-byte, unsigned integer.

Comments

- This is the datastream that is sent from the server when an informational message occurs.
- A server may send multiple **TDS_INFO** statements.
- The **TDS_INFO** datastream is exactly the same as the **TDS_ERROR** datastream except for the token value.
- This token is obsolete and has been replaced with the **TDS_EED** token.

Examples

See Also

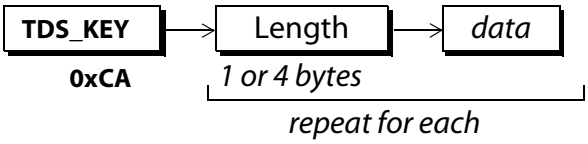
TDS_ERROR, TDS_EED

TDS_KEY

Function

The datastream for key data.

Syntax



Arguments

TDS_KEY This is the token that indicates that this is a data stream containing key data.

Length This is the actual, as opposed to maximum, data length, in bytes, of the following data. It is one-byte unsigned integer. If the following data is a fixed length datatype of standard length, e.g., ints, floats, datetimes, then there is no Length argument.

data This is the actual data for the key column. Its length, if variable, is indicated by the preceding Length argument. It is in the format requested in the login request from the client.

Comments

- This is the data stream that contains all the key for a particular row. The key data is returned to the server along with a cursor update command to tell the server the client's current row. The server will also return the new key to the client when the key is changed on a cursor update or cursor delete.
- The key data is described in the **TDS_ROWFMT** for the row with the key. The "key" column status tells the client that a particular column in a row is part of the key for that row. The key is "embedded" in the regular row. If the key column was not specifically requested by the client request, the key column is also a "hidden" column.

- No format information is passed back to the server with the **TDS_KEY** since the server already has that information. The **TDS_KEY** data stream which identifies the client's current row follows the **TDS_CURUPDATE** or **TDS_CURDELETE** data streams.
- If the key changes as a result of the **TDS_CURUPDATE**, the server will return the new key data in a **TDS_KEY** data stream, preceded by a **TDS_ROWFMT** data stream.
- When a client sends a **TDS_KEY** to the server, no **TDS_ROWFMT** data stream is sent. However, when a server sends a **TDS_KEY** data stream to a client, a **TDS_ROWFMT** data stream describing the key data precedes the **TDS_KEY** data stream.
- A **TDS_KEY** data stream consists of Length and parameter pairs, one for each parameter described by the associated **TDS_ROWFMT** data stream. The Length component doesn't appear if the data is a fixed datatype of standard length, *e.g.*, **TDS_INT2**, **TDS_MONEY**, **TDS_DATETIME**, etc. If the datatype allows nulls then the data will always be preceded by a Length argument. Fixed length datatypes that are not of a standard length, e.g., **TDS_CHAR** and **TDS_BINARY** are also preceded by a Length.
- The **TDS_PARAMS** data stream has exactly the same format as the **TDS_ROW** and **TDS_KEY** data streams. Three tokens are used for the same data stream in order to provide data stream state information. The formats will remain the same so that client and server code used to encode and decode the data streams can be the same.
- Note that if the cursor update request is made via a language request and not a **TDS_CURUPDATE** data stream, a **TDS_KEY** will not be passed to the server with the request.

Question: Verify when KEY data streams are returned to client.

Examples

See Also

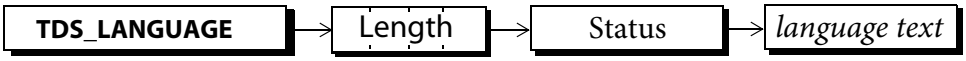
TDS_ROWFMT, TDS_ROW

TDS_LANGUAGE

Function

The token used to send a language command.

Syntax



Arguments

TDS_LANGUAGE This is the token that indicates that this is a language command.

Length This is the length, in bytes, of the rest of the token. It includes the status byte and the length of the language command. It is a four-byte, unsigned integer.

Status This status byte is a bit-mask. The only currently defined bit is 0x01 which indicates that the command is parameterized and that **PARAMFMT/PARAM** tokens follow

language text This the text of the language command. Presentation conversion is performed by the server if required.

Comments

- This is the token that is used by a client to send a language command to a server.
- Language commands may be parameterized. In that case, the Status 0x01 bit is set and the character and content of the parameters are described following the **TDS_LANGUAGE** data stream using the **TDS_PARAMFMT** and **TDS_PARAMS** data streams.
- Currently, only one **TDS_LANGUAGE** command is supported per client request.

Examples

See Also

TDS_RPC, TDS_CURDECLARE, TDS_PARAMFMT, TDS_PARAMS

Login Record

Description

This is the record that is sent to request that a dialog be established between a client and a server.

Syntax

```
typedef struct loginrec
{
    TDS_BYTE        lhostname[TDS_MAXNAME];
    TDS_BYTE        lhostnlen;
    TDS_BYTE        lusername[TDS_MAXNAME];
    TDS_BYTE        lusernlen;
    TDS_BYTE        lpw[TDS_MAXNAME];
    TDS_BYTE        lpwnlen;
    TDS_BYTE        lhostproc[TDS_MAXNAME];
    TDS_BYTE        lhplen;
    TDS_BYTE        lint2;
    TDS_BYTE        lint4;
    TDS_BYTE        lchar;
    TDS_BYTE        lflt;
    TDS_BYTE        ldate;
    TDS_BYTE        lusedb;
    TDS_BYTE        ldmpld;
    TDS_BYTE        linterfacespare;
    TDS_BYTE        ltype;
    TDS_BYTE        lbufsize[TDS_NETBUF];
    TDS_BYTE        lspare[3];
    TDS_BYTE        lappname[TDS_MAXNAME];
    TDS_BYTE        lappnlen;
    TDS_BYTE        lservname[TDS_MAXNAME];
    TDS_BYTE        lservnlen;
    TDS_BYTE        lrempw[TDS_RPLEN];
    TDS_BYTE        lrempwlen;
    TDS_BYTE        ltds[TDS_VERSIZE];
    TDS_BYTE        lprogname[TDS_PROGNLEN];
```

```
    TDS_BYTE        lprognlen;
    TDS_BYTE        lprogvers[TDS_VERSIZE];
    TDS_BYTE        lnoshort;
    TDS_BYTE        lflt4;
    TDS_BYTE        ldate4;
    TDS_BYTE        llanguage[TDS_MAXNAME];
    TDS_BYTE        llanglen;
    TDS_BYTE        lsetlang;
/*
** The following 13 bytes were used by 1.0 secure servers. Actually 2 bytes in
** the middle are unused. Since we do not support logins to 1.0 secure servers,
** we can re-use these 13 bytes.
** However, non-secure servers, check if the first 2 bytes are non-zero. If they
** are non-zero, they assume that the user want's to login a secure server and
** reject the login.
*/
    TDS_BYTE        loldsecure[TDS_OLDSECURE];
    TDS_BYTE        lseclogin;
    TDS_BYTE        lsecbulk;
/*
** The following 2 fields were added in specification revision 3.2 to support High
** Availability failover. The lhalogin byte and the 6 lhasessionid bytes were taken from
** the lsecspare bytes, The TDS_SECURE value was reduced from 9 to 2 accordingly.
*/
    TDS_BYTE        lhalogin;
    TDS_BYTE        lhasessionid[TDS_HA];
    TDS_BYTE        lsecspare[TDS_SECURE];
    TDS_BYTE        lcharset[TDS_MAXNAME];
    TDS_BYTE        lcharsetlen;
    TDS_BYTE        lsetcharset;
    TDS_BYTE        lpacketsize[TDS_PKTLEN];
    TDS_BYTE        lpacketsizelen;
    TDS_BYTE        ldummy[4];
} LOGINREC;
```

Comments

- When a client wants to establish a dialog with a server, a TDS packet is sent that contains a login record. This packet is denoted by a packet header type of **TDS_BUF_LOGIN**. Clients may have more than one dialog to a server but each one is established separately in the same way. The dialogs may be established on different transport connections or over the same one (server-to-server).
- When a client sends a login record to a server, the server will respond with a **TDS_LOGINACK** data stream. The status argument in the **TDS_LOGINACK** data stream will indicate success or failure of the login attempt.
- The size of the login record will not be changed in future releases of TDS. Any additional functionality will be implemented using separate token data streams.

Fields

Table 47: Login Record Fields

Field Name	Possible Values	Description
lhostname		Contains the client's host name.
lhostlen		Length, in bytes, of the client's host name in lhostname.
lusername		Client's user name. This field can be used for authentication.
lusernlen		Length, in bytes, of user name in lusername field.
lpw		Client's password. This field can be used for authentication. However, this field is sent as clear text.
lpwnlen		Length, in bytes of the password in the lpw field.
lhostproc		Process identifier associated with client program. The process identifier is specified as a string of ASCII characters.
lhplen		Length, in bytes, of the process identifier in lhostproc.
lint2	**TDS_INT2_LSB_HI (2)** **TDS_INT2_LSB_LO(3)**	Specifies the client byte ordering for two byte integers. **TDS_INT2_LSB_HI** specifies that the least significant byte is in the high byte (68000 byte ordering). **TDS_INT2_LSB_LO** specifies that the least significant byte is in the low byte (VAX and 80x86 byte ordering).
lint4	**TDS_INT4_LSB_HI (0)** **TDS_INT4_LSB_LO(1)**	This field identifies the client byte-ordering for four-byte integers. **TDS_INT4_LSB_HI** indicates that the least significant byte is in the high byte (68000 byte ordering). **TDS_INT4_LSB_LO** indicates that the least significant byte is in the low byte (VAX and 80x86 byte ordering).

Table 47: Login Record Fields

Field Name	Possible Values	Description
lchar	**TDS_CHAR_ASCII (6)** **TDS_CHAR_EBCDIC (7)**	This field identifies the type of character representation being used by the client. **TDS_CHAR_ASCII** indicates that the EBCDIC character set is not being used by the client. The actual character set being used by the client is specified in the lcharset field below. **TDS_CHAR_EBCDIC** indicates that the EBCDIC character set is being used by the client.
lflt	**TDS_FLT_IEEE_HI(4)** **TDS_FLT_VAXD(5)** **TDS_FLT_IEEE_LO(10)** **TDS_FLT_ND5000(11)**	This field identifies the type of floating point representation used by the client. **TDS_FLT_IEEE_HI** indicates IEEE 754 float type with the least significant byte in the high byte (e.g. Sun). **TDS_FLT_VAXD** indicates that VAX 'D' floating point format is being used. **TDS_FLT_IEEE_LO** indicates IEEE 754 float type with the least significant byte in the low byte (e.g. 80x86). **TDS_FLT_ND5000** indicates a ND5000 float byte with the least significant byte in the high byte.
ldate	**TDS_TWO_I4_LSB_HI(8)** **TDS_TWO_I4_LSB_LO(9)**	This field identifies the type of 8-byte datetime representation used by the client. The 8-byte datetime data type is implemented as two four-byte integers. **TDS_TWO_I4_LSB_HI** indicates that the least significant integer is the high integer. **TDS_TWO_I4_LSB_LO** indicates that the least significant integer is the low integer.

Table 47: Login Record Fields

Field Name	Possible Values	Description

linterfacespare	**TDS_LDEFSQL(0)**
	TDS_LXSQL(1)
	TDS_LSQL(2)
	TDS_LSQL2_1(3)
	TDS_LSQL2_2(4)
	TDS_LOG_SUCCEED(5)
	TDS_LOG_FAIL(6)
	TDS_LOG_NEG(7)
	TDS_LOG_SECSESS_ACK(0x08)
		This field is only used in server-server negotiations. Values and meanings here are pulled from SQLServer's version of login header files:
		`server's default SQL will be sent`
		`TRANSACT-SQL will be sent`
		`ANSI SQL, version 1`
		`ANSI SQL, version 2, level 1`
		`ANSI SQL, version 2, level 2`
		`Log in succeeded`
		`Log in failed`
		`Negotiate further`
		`LOGINACK status bit.`
		Note that this bit can be set and one of the above status values may be returned in the same byte. i.e. 0x05, 0x06, 0x07, 0x85, 0x86, and 0x87 are the possible values for the status.

ltype	**TDS_LSERVER(0x01)** **TDS_LREMUSER(0x02)** **TDS_LINTERNAL_RPC(0x04)**	This field specifies the type of dialog. Dialog requests come from two sources; directly from a server, or server-to-server. Server-to-server dialogs are differentiated from normal client connections by the ltype field in the login record. If the dialog is specified as a server-to-server type, the lrempw field contains the actual user name and password. **TDS_LSERVER** indicates that this dialog is a server-to-server type, **TDS_LREMUSER** indicates that this dialog is a user login through another server **TDS_LINTERNAL_RPC** indicates allow an internal RPC to be executed in the connection
lbufsize		This field is not currently specified by TDS. However, it was used in the past by certain Sybase products. Because of this, this field will never be specified by TDS.

Table 47: Login Record Fields

Field Name	Possible Values	Description
lspare		This field is not currently specified by TDS. However, it was used in the past by certain Sybase products. Because of this, this field will never be specified by TDS.
lappname		The client application name. The application name defined by the application program. It is different from the program name which is the name of the library that the client is using to manage the communication with the server.
lappnlen		Length, in bytes, of the lappname field.

Table 47: Login Record Fields

Field Name	Possible Values	Description
lservname		The name of the server to which the client is attempting to establish a dialog. CTlib and DBlib set this field to the interfaces file entry which was specified by the application explicitly or via the $DSQUERY environment variable. This field should correspond with the @@servername of the server for best results. In server-server rpc's this servname field is passed on to the remote server. If that remote server needs to open a connection back to this server for some reason, it will often use this value to access its local interfaces file. With some gateways (DirectConnnect for DB2) this field indicates the name of the desired back-end subsystem. For Adaptive Server Anywhere this field indicates the name of the database which the connection should be made to (the database must already be loaded). For older ASAs, if the name in this field didn't match any currently loaded databases, the connection silently wound up in the "default" database. However, with ASA 7.x and higher, the connection is now refused in this case.
lservnlen		Length, in bytes, of the lservname field.
lrempw		Pairs of remote server name and password fields. This field is used on server-to-server dialogs.See below for a description of the format of this field.
lrempwlen		Length, in bytes, of the lrempw array.

Table 47: Login Record Fields

Field Name	Possible Values	Description
ltds	**TDS_5_0_V1(5)** **TDS_5_0_V2(0)** **TDS_5_0_V3(0)** **TDS_5_0_V4(0)**	The TDS version requested by the client. This is a four-byte array where each byte specifies a number in the TDS version. The requested TDS version is specified with the major version identifier in the high order byte.
lprogname		The name of the client library that is being used to establish the dialog.
lprognlen		Length, in bytes, of the lprogname.
lprogvers	**TDS_CT_5_0_V1(5)** **TDS_CT_5_0_V2(0)** **TDS_CT_5_0_V3(0)** **TDS_CT_5_0_V4(0)** **TDS_DB_5_0_V1(5)** **TDS_DB_5_0_V2(0)** **TDS_DB_5_0_V3(0)** **TDS_DB_5_0_V4(0)**	The version of the client library. This field is a four byte array where each byte specifies a number in the client library version.
lnoshort	**TDS_CVT_SHORT(1)** **TDS_NOCVT_SHORT(0)**	This flag indicates whether 4 byte datetime, money, and floating point data types should be automatically converted to 8 byte equivalents. **TDS_CVT_SHORT** indicates that the short data types should be converted. **TDS_NOCVT_SHORT** indicates that they should not be converted.
lflt4	**TDS_FLT4_IEEE_HI(12)** **TDS_FLT4_IEEE_LO(13)** **TDS_FLT4_VAXF(14)** **TDS_FLT4_ND50004(15)**	This is the format of 4 byte floating point numbers that will be used by the client. **TDS_FLT4_IEEE_HI** IEEE floating point numbers with the least significant byte in the high byte. **TDS_FLT4_IEEE_LO** IEEE floating point numbers with the least significant byte in the low byte. **TDS_FLT4_VAXF** indicate a VAX 'F' floating point number. **TDS_FLT4_ND50004** indicates ND5000 4 byte floating point format.

Table 47: Login Record Fields

Field Name	Possible Values	Description
ldate4	**TDS_TWO_I2_LSB_HI(16)** **TDS_TWO_I2_LSB_LO(17)**	The type of 4 byte datetime representation used by the client. Four byte date time numbers are implemented as two unsigned shorts. **TDS_LOW_I2_LSB_HI** indicates that the least significant short is in the high byte. **TDS_LOW_I2_LSB_LO** indicates that the least significant short is in the low byte.
llanguage		The client's requested national language. The default is "us_english". This is the national language that will be used for error messages. This national language should map to a valid language in locales.dat and is platform dependant.
llanglen		The length, in bytes, of the llanguage field value. If this field is 0 the default server language will be used.
lsetlang	**TDS_NOTIFY(1)** **TDS_NO_NOTIFY(0)**	This field indicates whether the client wants to be notified of language changes. **TDS_NOTIFY** indicates that the client wants to be notified, **TDS_NO_NOTIFY** indicates that the client does not want to be notified.
loldsecure		This field was used by the original secure server. It is not documented by the TDS specification.
lseclogin	**TDS_SEC_LOG_ENCRYPT (0x01)** **TDS_SEC_LOG_CHALLENGE (0x02)** **TDS_SEC_LOG_LABELS (0x04)** **TDS_SEC_LOG_APPDEFINED (0x08)** **TDS_SEC_LOG_SECSESS (0x10)** **TDS_SEC_LOG_ENCRYPT2 (0x20)**	Negotiated login bit mask. See section 20. on page 59 for more information on how these values are used.
lsecbulk	**TDS_SEC_BULK_LABELED (0x01)**	Bulk copy security bit mask.

Table 47: Login Record Fields

Field Name	Possible Values	Description
lhalogin	**TDS_HA_LOG_SESSION(0x01)** **TDS_HA_LOG_RESUME(0x02)** **TDS_HA_LOG_FAILOVERSRV(0x04)** **TDS_HA_LOG_REDIRECT(0x08)** **TDS_HA_LOG_MIGRATE(0x10)**	If the session bit is set, the client is requesting a High-Availability login session. If the server can provide this level of service, it responds with a negotiated login sequence. If login is successful the lhasessionid will be returned to the client. If the HARESUME bit is set then the client is resuming an existing HA session and the lhasessionid has been set. If the HARESUME bit is set, the FAILOVERSRV bit indicates whether this server is the "primary" (FAILOVERSRV is clear) or a "secondary" server (FAILOVERSRV is set) in the cluster. If the failover bit is set, the client is explicitly telling the server that it has already attempted an initial login to the "primary" server for this HA cluster, and is failing over to this, the "secondary" If the redirect bit is set, the client can replace its current server failover information with information provided by the server via a **TDS_EED** message. The client will also, if requested, abort the current login and connect to a server on the failover list. Note that the redirect bit is set independently of the session bit. If the migrate bit is set, the lhasessionid field contains the migration identifier instead of a HA session id. It is considered a fatal error if both **TDS_HA_LOG_MIGRATE** and **TDS_HA_LOG_RESUME** are set. **See the HA negotiated login sequence on page 251.**

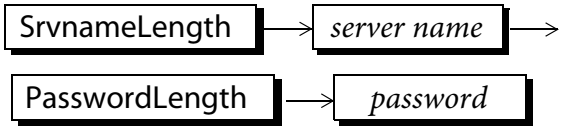
Table 47: Login Record Fields

Field Name	Possible Values	Description
lhasessionid		This field is only meaningful if the **TDS_HA_LOG_SESSION** and **TDS_HA_LOG_RESUME** bits, or **TDS_HA_LOG_MIGRATE** bit are set. The server will attempt to re-establish an existing session which corresponds to this session id.
lsecspare		Spare fields. Not currently used. Reserved for secure server.
lcharset		The name of the character set requested by the client.
lcharsetlen		Length, in bytes, of the lcharset field. If this field is 0 the default server character set will be used.
lsetcharset	**TDS_NOTIFY(1)** **TDS_NO_NOTIFY(0)**	This field indicates whether the client wants to be notified of character set changes. **TDS_NOTIFY** indicates that the client wants to be notified, **TDS_NO_NOTIFY** indicates that the client does not want to be notified.
lpacketsize		This field contains a character array that specifies the client's requested packet size. Each digit of the requested packet size is represented as an ASCII character. The minimum packet size is 256 bytes and the maximum is 9999 bytes.
lpacketsizelen		Length, in bytes, of the lpacketsize field. If this field is 0, the default packet size of 512 bytes is used.
ldummy		pad the login record structure to a longword

Remote Password Array Format

The lrempw field contains an array of remote server name and user password pairs. The length of this array is in the lrempwlen field. This field is used when a server-to-server dialog is established. It is possible for the original client application to pass different passwords to different remote servers.

The format of the lrempw array is:



This pattern is repeated once for each remote server/password pair. If the SrvnameLength is 0, the password which follows is a "universal password" and will be used for any remote server. If the PasswordLength is 0, it means that the password is NULL. The total length of the lrempw array is 255 bytes. This limits the total possible number of server name and password pairs to this length.

See Also

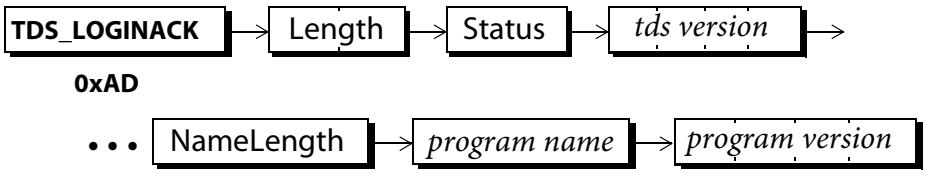
TDS_LOGINACK, TDS_ENVCHANGE

TDS_LOGINACK

Function

The response to token to a login request.

Syntax



Arguments

TDS_LOGINACK This is the token used to acknowledge a client login request.

Length This is the length, in bytes, of the remaining data stream. It is a two-byte, unsigned integer.

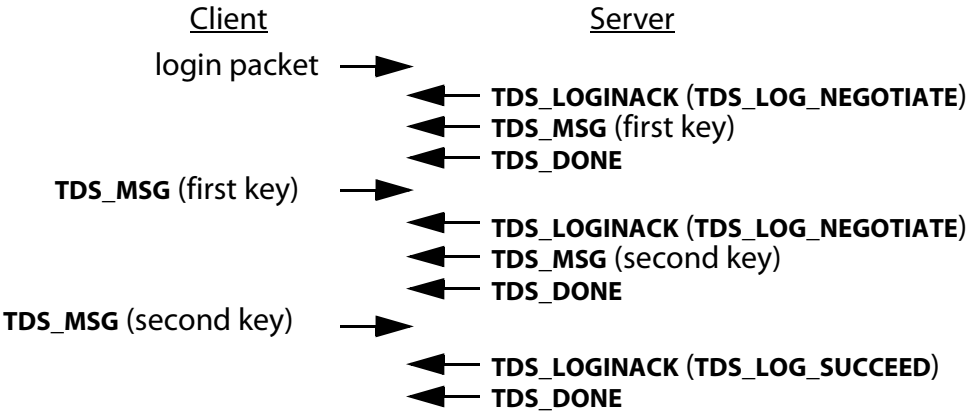
Status The status of the login request. It is a one-byte, unsigned integer. These are the possible status values.

TDS_LOG_SUCCEED - 5
The login request completed successfully.

TDS_LOG_FAIL - 6
The login request failed. The client must terminate the dialog and restart to attempt another login request.

TDS_LOG_NEGOTIATE - 7
The server is requesting that the client complete a negotiation before completing the login request. The login negotiation is done using the **TDS_MSG** token.

For example, if a server uses a double-authentication key to verify logins the sequence of events would be:



COMMENTS:Note that each **TDS_MSG** must be followed by a **TDS_PARAMFMT**/**TDS_PARAM** sequence, even though there are no parameters (paramfmt.#params = 0). This is just how the CTLib state machine is defined.

tds version This is the version of TDS that the server is going to use. This argument is an array of four unsigned, one-byte integers. For example, TDS version 5.0.0.0 is 0x05000000.

NameLength This is the length of the program name argument. It is a one-byte, unsigned integer.

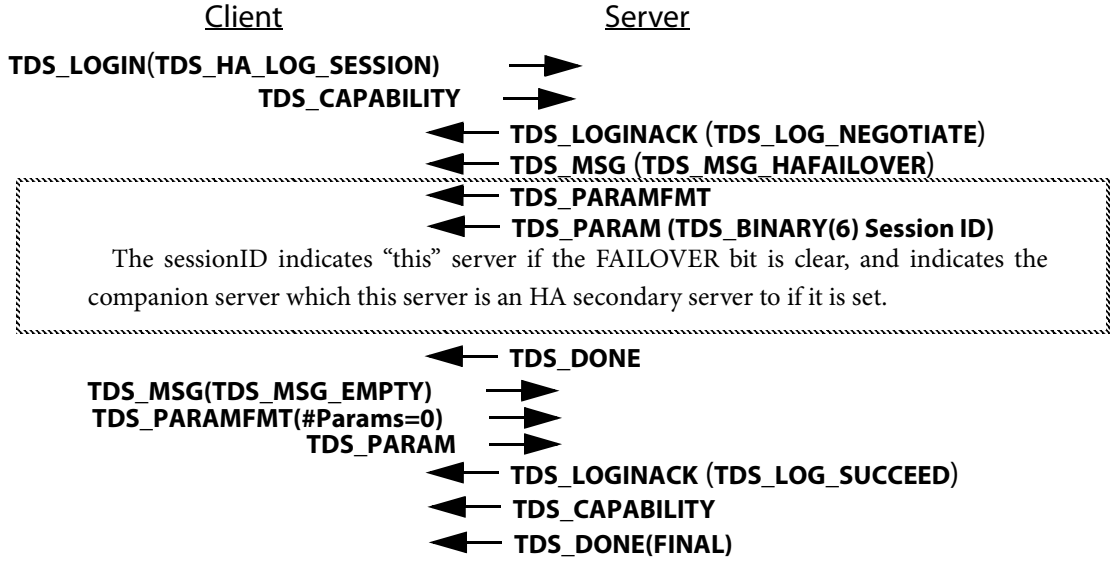
program name This is the name of the server program. It's length is in the NameLength argument.

program version This is the version of the server program. This argument is an array of four unsigned, one-byte integers. For example, SQL Server version 4.0.2 is 0x04000200.

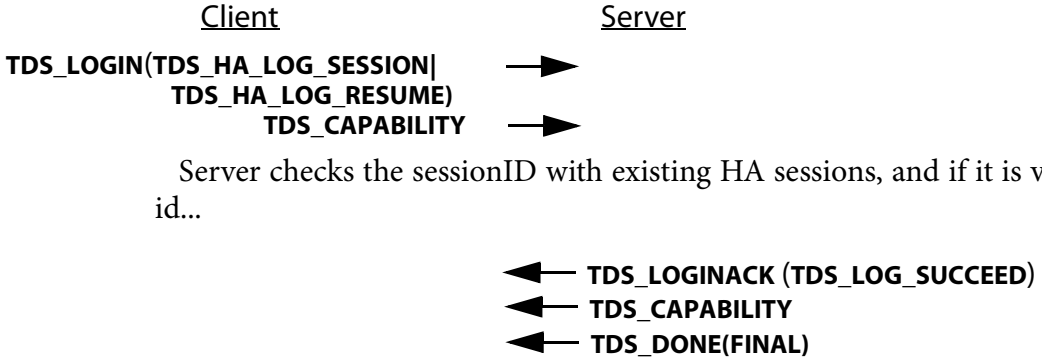
Comments

- A **TDS_LOGINACK** token is always returned to the client whether or not the login attempt has been successful, failed, or is on-going.
- If the login has a status of **TDS_LOG_NEGOTIATE**, the client and server will continue to exchange **TDS_MSG** tokens until the login either succeeds or fails.

- Note that the Interface argument in the data stream has been dropped in TDS 5.0. It has been replaced by the Status argument.
- With the TDS 5.0 Specification revision, the HA Failover login negotiation sequence was added. If the HA_SESSION bit is set and the HA_RESUME bit is clear, then the client is requesting a new HA session. The login negotiation proceeds as:



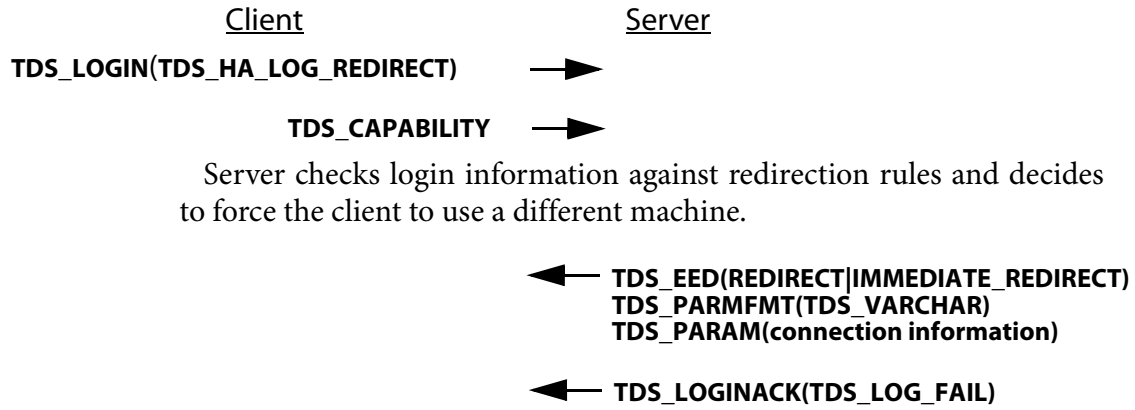
- If the HA_SESSION and HA_RESUME bits are both set then the lhasessionid field in the login request contains the sessionID of the existing session.



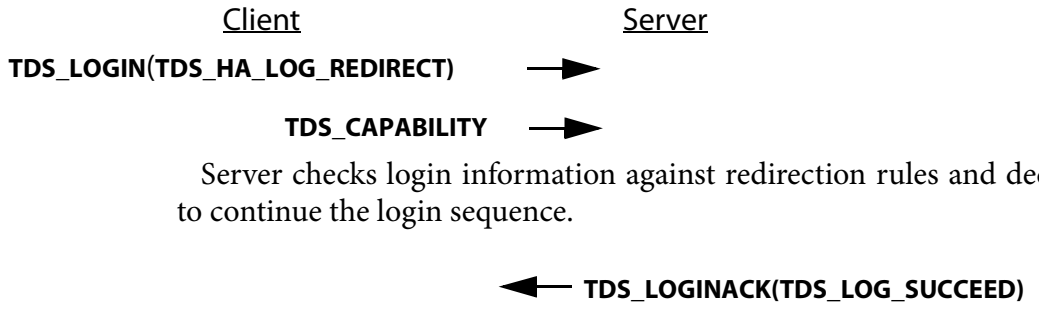
- With revision 3.7 of the TDS 5.0 spec, modifications were made to allow login redirection. This also modified the behavior of HA logins in that an application that set both **TDS_HA_LOG_SESSION** and **TDS_HA_LOG_REDIRECT** can receive, via a **TDS_EED** token sequence, a list of connection information that will be used during a failover situation. The possible combinations are explained in the following examples.

Examples

• In this example the client sets **TDS_HA_LOG_REDIRECT**, but none of the other **TDS_HA_LOG** bits. The server decides to have the client connect to a different host, so it sends **TDS_EED** with status = **TDS_EED_INFO**, the message number set to 1 and **TDS_EED_IMMEDIATE_REDIRECT** set in the state field.

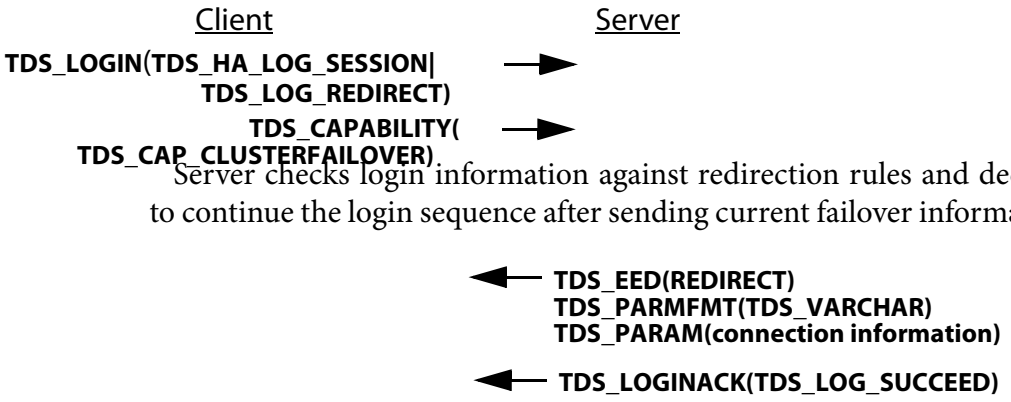


• In this example, the client sets **TDS_HA_LOG_REDIRECT**. The server does not send the redirect message and completes the login sequence.



• In this example, the client set **TDS_HA_LOG_REDIRECT** and **TDS_HA_LOG_SESSION**. Additionally,

TDS_CAP_CLUSTERFAILOVER is set in the capabilities mask. The server sends **TDS_EED** with message set to **TDS_REDIRECT**, but does not set **TDS_EED_IMMEDIATE_REDIRECT**. The login continues normally. Note that if **TDS_CAP_CLUSTERFAILOVER** had not been set, the server would not have sent the



TDS_EED(REDIRECT) to the client.

See Also

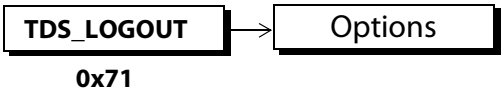
TDS_MSG, login request

TDS_LOGOUT

Function

Client logout request.

Syntax



Arguments

TDS_LOGOUT This token is a client logout request.

Options Options is a one-byte, unsigned integer. There are currently no options defined. This argument must be 0x00.

Comments

- This token is used by a client to logout from the server.
- A **TDS_LOGOUT** is acknowledged by the server with a **TDS_DONE**.

Examples

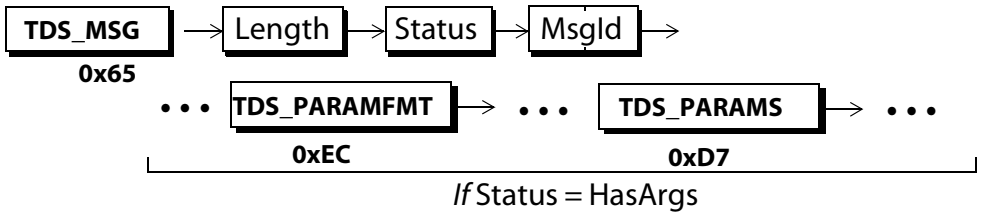
See Also

TDS_MSG

Function

Token to send generic messages between clients and servers.

Syntax



Arguments

TDS_MSG This is token used to send a message to either a client or a server.

Length This is the total length, in bytes, of the remaining data stream. It is a one-byte, unsigned integer.

Status This indicates whether or not the **TDS_MSG** has **TDS_PARAMFMT** and **TDS_PARAMS** following to describe message arguments. If there are no arguments then Status is 0x00. If the MSG has arguments then Status must be **TDS_MSG_HASARGS (0x01)**. Status is a one-byte, unsigned integer.

MsgId This is the id of the message. Ids are two-byte, unsigned integers. Ids 0 through 32,767 are reserved for the CS/I implementation of TDS. The following ids are reserved:

Table 48: Reserved Message Identifiers

Define	Value	Client Visible	Description
TDS_MSG_SEC_ENCRYPT	1	No	Start encrypted login protocol.
TDS_MSG_SEC_LOGPWD	2	No	Sending encrypted user password.

Table 48: Reserved Message Identifiers

Define	Value	Client Visible	Description
TDS_MSG_SEC_REMPWD	3	No	Sending remote server passwords.
TDS_MSG_SEC_CHALLENGE	4	No	Start challenge/response protocol.
TDS_MSG_SEC_RESPONSE	5	No	Returned encrypted challenge.
TDS_MSG_SEC_GETLABEL	6	No	Start trusted user login protocol.
TDS_MSG_SEC_LABEL	7	No	Return security labels.
TDS_MSG_SQL_TBLNAME	8	Yes	**CS_MSG_TABLENAME**
TDS_MSG_GW_RESERVED	9	No	Used by interoperability group.
TDS_MSG_OMNI_CAPABILITIES	10	No	Used by OMNI SQL Server.
TDS_MSG_SEC_OPAQUE	11	No	Send opaque security token.
TDS_MSG_HAFAILOVER	12	No	Used during login to obtain the HA Session ID
TDS_MSG_EMPTY	13	No	Sometimes a MSG response stream is required by TDS syntax, but the sender has no real information to pass. This message type indicates that the following paramfmt/param streams are meaningless
TDS_MSG_SEC_ENCRYPT2	14	No	Start alternate encrypted password protocol.
TDS_MSG_SEC_LOGPWD2	15	No	Return alternate encrypted passwords.
TDS_MSG_SEC_SUP_CIPHER	16	No	Returns list of supported ciphers.
TDS_MSG_MIG_REQ	17	Yes	Initiate client connection migration to alternative server via address provided as message parameter.

Table 48: Reserved Message Identifiers

Define	Value	Client Visible	Description
TDS_MSG_MIG_SYNC	18	No	Client sends to acknowledge receipt of **TDS_MSG_MIG_REQ**.
TDS_MSG_MIG_CONT	19	Yes	Server sends to start actual client migration to alternate server.
TDS_MSG_MIG_IGN	20	Yes	Server sends to abort previous **TDS_MSG_MIG_REQ**.
TDS_MSG_MIG_FAIL	21	No	Client sends to original server to indicate that the migration attempt failed. Optional parameter indicates failure reason.

Comments

- The **TDS_MSG** token is used whenever the client and/or server wish to pass unstructured messages.
- The **TDS_MSG** token is used by both the server and client to implement a negotiated login sequence.
- The **TDS_MSG** token can be interleaved with other TDS tokens. A **TDS_DONE** is not required specifically for the **TDS_MSG** token. If the **TDS_MSG** token is the only token being sent, a **TDS_DONE(FINAL)** is required.
- Message Ids greater than 32k are reserved by TDS for user applications.
- A **TDS_MSG** token from a client is acknowledged by the server with a **TDS_DONE** token.
- For message token values prior to 14, the CTlib state machine requires that a **TDS_MSG** always be followed by a **TDS_PARAMFMT**, **TDS_PARAMS** sequence even if the paramfmt.#params = 0. Note that **TDS_PARAMFMT2** cannot be used instead of **TDS_PARAMFMT**. This restriction has been removed for newer versions of CTlib, so when messages with value > 13 are used, **TDS_PARAMFMT** should only be sent if **TDS_MSG_HASARGS** is set.

- *Table 49: Opaque Message Types* contains the four types of **TDS_MSG_SEC_OPAQUE** messages.

Table 49: Opaque Message Types

Define	Value	Description
TDS_SEC_SECSESS	1	Security session token
TDS_SEC_FORWARD	2	Credential forwarding
TDS_SEC_SIGN	3	Data signature packet
TDS_SEC_OTHER	4	Other security message

- TDS_SEC_SECSESS has five (5) parameters. Parameter 1 is the TDS security version of type TDS_INTN and has the value 50. Parameter 2 is the security message type of TDS type TDS_INTN and has the value TDS_SEC_SECSESS. Parameter 3 is the OID of the security mechanism being used and is of TDS type TDS_VARBINARY. Parameter 4 is an opaque security token of type TDS_LONGBINARY. Parameter 5 is the security services requested and is of type TDS_INTN. These services are a bit mask with the values in the table Security Services.

Table 50: Security Services

Service Name	Value	Description	Login	Per-Packet
Network Authentication	0x0001	Client must provide proof of its identity.	Yes	No
Mutual Authentication	0x0002	Require server to provide proof of its identity to client in addition to client providing proof of its identity to server.	Yes	No
Delegation	0x0004	Allow server to connect to remote server using delegated credentials.	Yes	No
Integrity	0x0008	Requires Network Authentication. All data packets to be sent over the network will generate a signature packet.	No	Yes

Table 50: Security Services

Service Name	Value	Description	Login	Per-Packet
Confidentiality	0x0010	TDS data stream is encrypted.	No	Yes
Detect Replay	0x0020	Attempts to capture and replay data packets are detected. This service will generate a signature packet.	No	Yes
Detect Sequence	0x0040	Detect out of order data packets. This service will generate a signature packet.	No	Yes
Data Origin	0x0080	Perform data origin stamping for each data packet. This service will generate a signature packet.	No	Yes
Channel Binding	0x0100	Both client and server provide a network channel identifier consisting of the network addresses of both the client and server to the security mechanism.	Yes	No
Reserved	0x0200			
Reserved	0x0400			

- **TDS_SEC_FORWARD** has five (5) parameters. Parameter 1 is the TDS security version of type **TDS_INTN** and a value of 50. Parameter 2 is the security message type of type **TDS_INTN** and a value of **TDS_SEC_FORWARD**. Parameter 3 is the server name and is type **TDS_VARBINARY**. Parameter 4 is the channel bind acceptor address family type and is of type **TDS_INTN**. Parameter 5 is the channel bind acceptor bind object and is of type **TDS_VARBINARY**.
- **TDS_SEC_SIGN** has four (4) parameters. Parameter 1 is the TDS security version and is of type **TDS_INTN** and should have a value of 50. Parameter 2 is the TDS security message id and is of type **TDS_INTN** and value **TDS_SEC_SIGN**. Parameter 3 is the TDS security mechanism OID and is of type TDS_VARBINARY. Parameter 4 is the signature as returned by the security mechanism and if of type TDS_LONGBINARY.

- **TDS_MSG_SEC_ENCRYPT2** is followed by two parameters, a **TDS_INT4** and a **TDS_VARBINARY**. The **TDS_VARBINARY** is the server key to use to encrypt the client password. The **TDS_INT4** indicates which cipher suite to use to do the encryption. The valid values are from the TLS Protocol specification, with the provision that the only cipher that part that is important is the processing of the asymmetric cipher. Should dynamic session encryption be added in the future, the information would also be passed via this token.
- **TDS_MSG_SEC_LOGPWD2** is followed by one or more encrypted passwords. The first parameter is a **TDS_VARBINARY** containing the encrypted user password. The subsequent parameters are pairs of **TDS_VARCHAR**/**TDS_VARBINARY** representing remote servers and encrytped passwords.
- **TDS_MSG_SEC_SUP_CIPHER** is followed by on or more **TDS_INT4** parameters indicating what cipher suites the client supports. The values are from the TLS v1 specification. This message is usually sent in response to a **TDS_MSG_SEC_ENCRYPT2** message that uses a cipher suite that the client does not support.
- Connection migration is viewed as an interruption in the normal TDS processing stream. A server can initiate migration at any time. A more detailed explanation of the migration process is contained in the introductory material for TDS.
- **TDS_MSG_MIG_xxx** tokens can only be sent in a TDS buffer type of **TDS_BUF_MIGRATE**.
- **TDS_MSG_MIG_REQ** is used by a server to initiate a client migration request. This request will have a parameter list of addresses formatted as login redirection address are. This list is a server ordered list. See **TDS_EED** comments on page 220 for more information on how individual parameters look. Note that the server is allowed to send the **TDS_MSG_MIG_REQ** to a client and then close down the connection. It is expected that the server will generally implement a timer to wait for the client to send a **TDS_MSG_MIG_SYNC** before doing so, but this is not required (e.g., powerfail signal received and the operating system is shutting down.) In this case, the client is expected to detect the connection close error, read the buffered migration information, and attempt to connect to the first server it can. Should this connection attempt fail, the client would then report a normal network connection failure.

- **TDS_MSG_MIG_SYNC** is sent by a client after receiving **TDS_MSG_MIG_REQ** from the server. Once **TDS_MSG_MIG_SYNC** is sent, the client cannot send anything else to the server prior to receiving **TDS_MSG_MIG_CONT** or **TDS_MSG_MIG_IGN** except an attention or **TDS_LOGOUT**.
- **TDS_MSG_MIG_CONT** is sent by a server to the client at any time after the server receives **TDS_MSG_MIG_SYNC**. Upon receiving this token, the client will connect to the first server in the parameter list provided by **TDS_MSG_MIG_REQ** that it can. After successfully completing the connection to the new server, the client closes the current session and resumes reading the TDS data stream.
- **TDS_MSG_MIG_IGN** is sent by a server to cancel a previous **TDS_MSG_MIG_SYNC**. The **TDS_BUF_MIGRATE** packet can contain a **TDS_MSG_MIG_IGN** followed immediately by **TDS_MSG_MIG_REQ**.
- **TDS_MSG_MIG_FAIL** is sent by a client to the server that sent the **TDS_MSG_MIG_CONT** should a migration attempt fail. This message may have a parameter which the server is free to ignore. The parameter should indicate why the migration failed. The client does not have to send the parameter. The protocol for dealing with this parameter is not defined yet.

Examples

See Also

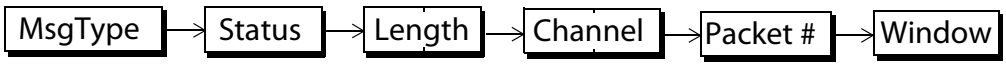
TDS_LOGINACK, TDS_PARAMFMT, TDS_PARAMS

Message Buffer Header

Function

Describes the buffer header used by messages.

Syntax



Arguments

MsgType This one-byte unsigned integer defines the buffer type. The types are:

Table 51: Buffer Types

Define	Value	Description
TDS_BUF_LANG	1	The buffer contains a language command. TDS does not specify the syntax of the language command.
TDS_BUF_LOGIN	2	The buffer contains a login record
TDS_BUF_RPC	3	The buffer contains a remote procedure call command.
TDS_BUF_RESPONSE	4	The buffer contains the response to a command.
TDS_BUF_UNFMT	5	The buffer contains raw unformatted data.
TDS_BUF_ATTN	6	The buffer contains a non-expedited attention request.
TDS_BUF_BULK	7	The buffer contains bulk binary data.
TDS_BUF_SETUP	8	A protocol request to setup another logical channel. This buffer is a header only and does not contain any data.
TDS_BUF_CLOSE	9	A protocol request to close a logical channel.This buffer is a header only and does not contain any data.

Table 51: Buffer Types

Define	Value	Description
TDS_BUF_ERROR	10	A resource error was detected while attempting to setup or use a logical channel. This buffer is a header only and does not contain any data.
TDS_BUF_PROTACK	11	A protocol acknowledgment associated with the logical channel windowing protocol. This buffer is a header only and does not contain any data.
TDS_BUF_ECHO	12	A protocol request to echo the data contained in the buffer.
TDS_BUF_LOGOUT	13	A protocol request to logout an active logical channel. This buffer is a header only and does not contain any data.
TDS_BUF_ENDPARAM	14	What is this???
TDS_BUF_NORMAL	15	This packet contains a tokenized TDS request or response.
TDS_BUF_URGENT	16	This packet contains an urgent tokenized TDS request or response.
TDS_BUF_MIGRATE	17	This packet contains a migration protocol message. Currently these are only **TDS_MSG** tokens.
TDS_BUF_CMDSEQ_NORMAL	24	SQL Anywhere CMDSEQ protocol
TDS_BUF_CMDSEQ_LOGIN	25	SQL Anywhere CMDSEQ protocol
TDS_BUF_CMDSEQ_LIVENESS	26	SQL Anywhere CMDSEQ protocol
TDS_BUF_CMDSEQ_RESERVED1	27	SQL Anywhere CMDSEQ protocol
TDS_BUF_CMDSEQ_RESEVERD2	28	SQL Anywhere CMDSEQ protocol

Status This is a bit field used to indicate the message status. Status is a one-byte unsigned integer.

Table 52: Status Values

Define	Value	Description
TDS_BUFSTAT_EOM	0x01	This is the last buffer in a request or a response.
TDS_BUFSTAT_ATTNACK	0x02	This is an acknowledgment to the last received attention.
TDS_BUFSTAT_ATTN	0x04	This is an attention request buffer.
TDS_BUFSTAT_EVENT	0x08	This is an event notification buffer.
TDS_BUFSTAT_SEAL	0x10	The buffer is encrypted
TDS_BUFSTAT_ENCRYPT	0x20	The buffer is encrypted (SQL Anywhere CMDSEQ protocol)

Length Length is the size of the buffer including the eight bytes in the buffer header. It is the number of bytes from the start of this header to the start of the next buffer header. For example, if there are 504 bytes of data in the buffer, Length will be 512. Length is a two-byte, unsigned integer. Regardless of the hardware architecture of either the server or the client, Length is represented by <MSB, LSB>. The most significant byte is first, followed by the least significant byte.

Channel This is the channel number of the logical dialog. It is used for multiplexing dialogs across the same physical connection. If multiplexing is not being used Channel must be set to 0. Channel is a two-byte, unsigned integer. Regardless of the hardware architecture of either the server or the client, Channel is represented by <MSB, LSB>. The most significant byte is first, followed by the least significant byte.

Packet # This is used for numbering buffers that contain data in addition to the buffer header. It is only significant when multiplexing. Each time a data buffer is sent the value of Packet is incremented, modulo 256. Packet is a one-byte, unsigned integer.

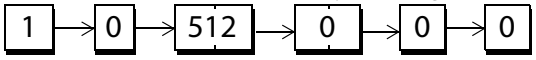
Window This is used to control the number of buffers which will be sent before an acknowledgment is received. Acknowledgments are sent using **TDS_BUF_PROTACK** type buffers. The receiving side defines its buffering limit, which it reports in the Window field of each **TDS_BUF_PROTACK** buffer and in the **TDS_BUF_SETUP** buffer. A **TDS_BUF_SETUP** buffer must always be acknowledged immediately so that the site that initiated the dialog can be informed of the window size it uses. The sending side cannot send a buffer if the receiving side has not acknowledged enough buffers and might have to buffer more than its window size. Window is a one-byte, unsigned integer. If not multiplexing, window size must be set to 0.

Comments

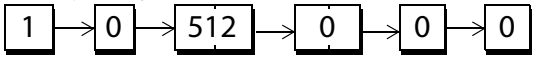
- Requests and responses between clients and servers are passed in buffers. Every buffer has a message buffer header which describes the buffer's type, length, and status information.
- Clients and servers send logical messages to each other. A logical message may consist of multiple buffers. The last buffer in a logical message has the EOM bit set in the Status field.
- All multi-byte fields in the message buffer header are in a fixed byte and bit order. The two-byte integers are represented by <MSB,LSB> which matches the data representation used by the 68000 but is reverse of the 80x86 and the VAX. The most significant byte is first, followed by the least significant byte.
- Packets with **TDS_BUFSTAT_SEAL** set are interpreted differently. The packet is formed by taking the data section of the packet and encrypting it. If the resultant packet size plus two (2) bytes will not fit in the negotiated packet size, the encrypted data is split across as many TDS PDUs as are needed to transfer the packet. The first two (2) bytes after the Message Header of the first packet give the number of bytes in network byte order <MSB,LSB> that make up the encrypted data.

Examples

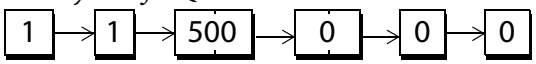
To send a request that is 1500 bytes long the headers sent look like:



504 bytes of SQL command



504 bytes of SQL command



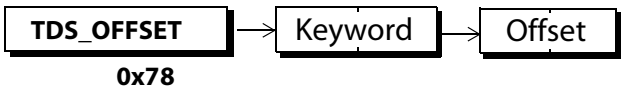
492 bytes of SQL command

TDS_OFFSET

Function

Returns the offset of the specified keyword in the language command buffer.

Syntax



Arguments

TDS_OFFSET This is the token for keyword offset information.

Keyword This is the keyword to which the Offset applies. This argument is a two-byte, unsigned integer. The following keywords are supported:

TDS_OFF_SELECT - 0x016D

TDS_OFF_FROM - 0x014F

TDS_OFF_ORDER - 0x0165

TDS_OFF_COMPUTE - 0x0139

TDS_OFF_TABLE - 0x0173

TDS_OFF_PROC - 0x016A

TDS_OFF_STMT - 0x01CB

TDS_OFF_PARAM - 0x01C4

Offset This is the offset into the command buffer where Keyword begins. The first byte in a command buffer is byte number 0. Offset is a two-byte, unsigned integer.

Comments

- This token is used to tell a client where a particular key word appears in a command buffer. This allows a client to use a server to perform primitive parsing. For example, if a client wants to know each place in a command buffer the keyword from appears the information can be returned via this token.
- The appropriate language option must be set for offsets to be returned.

Examples

See Also

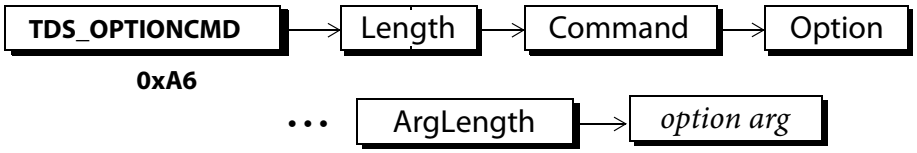
TDS_OPTIONCMD

TDS_OPTIONCMD

Function

Clear, set, and report on options.

Syntax



Arguments

TDS_OPTIONCMD This is the token used to get, set, or clear options.

Length This is the length, in bytes, of the remaining data stream for this token. It is a two-byte, unsigned integer.

Command This is the option. It is a one-byte, unsigned integer. The possible Commands are:

Table 53: Option Commands

Command	Value	Description
TDS_OPT_SET	1	Set an option.
TDS_OPT_DEFAULT	2	Set option to its default value.
TDS_OPT_LIST	3	Request current setting of a specific option.
TDS_OPT_INFO	4	Report current setting of a specific option.

Option The option being manipulated by this option command. A complete list of all supported options is below. Option is a one-byte, unsigned integer.

ArgLength This defines the length, in bytes, of the following option arg. It is an unsigned one-byte integer.

option arg This is the parameter that applies to the option listed in Option. The format of this argument is based on the option. See the table below. The length of this argument is in the ArgLength argument.

Comments

- This is the token used by both the client and server to set, clear, check, or return information about options.
- A Command to **TDS_OPT_SET** must specify the option being set in Option. The value to set it to must be sent in OptionArg. Arglength must be set correctly for OptionArg.
- A Command of **TDS_OPT_DEFAULT** must specify the option to set to the server's default in the Option argument. ArgLength must be set to 0.
- A Command of **TDS_OPT_LIST** must specify the option on which information is being requested in the Option argument. ArgLength must be set to 0.
- A Command of **TDS_OPT_SET** or **TDS_OPT_DEFAULT** is acknowledged with a **TDS_DONE(FINAL)**. The error bit is set in the **TDS_DONE** if the option request was not processed successfully.
- The **TDS_OPT_LIST** command is acknowledged by a server using the **TD_OPT_INFO** command. The **TDS_OPT_INFO** command contains the option specified in the **TDS_OPT_LIST** command in Option, and the current value of this option in OptionArg. ArgLength must be set correctly for OptionArg. A **TDS_DONE(FINAL)** is also sent following the **TDS_OPT_LIST** token.
- There is no way to request a server to return the values for all known options.
- A complete list of all supported options is:

Table 54: Supported Options

Name	Value	Description
TDS_OPT_UNUSED	0	Used to specify no option.
TDS_OPT_DATEFIRST	1	Set first day of week.

Table 54: Supported Options

Name	Value	Description
TDS_OPT_TEXTSIZE	2	Set maximum text size.
TDS_OPT_STAT_TIME	3	Return server time statistics.
TDS_OPT_STAT_IO	4	Return server I/O statistics.
TDS_OPT_ROWCOUNT	5	Set maximum row count to return.
TDS_OPT_NATLANG	6	Change national language.
TDS_OPT_DATEFORMAT	7	Set date format.
TDS_OPT_ISOLATION	8	Transaction isolation level.
TDS_OPT_AUTHON	9	Set authority level on.
TDS_OPT_CHARSET	10	Change character set.
TDS_OPT_SHOWPLAN	13	Show execution plan.
TDS_OPT_NOEXEC	14	Do not execute query.
TDS_OPT_ARITHIGNOREON	15	Set arithmetic exception handling.
TDS_OPT_ARITHABORTON	17	Set arithmetic abort handling.
TDS_OPT_PARSEONLY	18	Parse the query only. Return error messages.
TDS_OPT_GETDATA	20	Return trigger data.
TDS_OPT_NOCOUNT	21	Do not return done count.
TDS_OPT_FORCEPLAN	23	Forces substitution order for joins in the order of the tables provided in this option.
TDS_OPT_FORMATONLY	24	Send format information only.
TDS_OPT_CHAINXACTS	25	Set chained transaction mode.
TDS_OPT_CURCLOSEONXACT	26	Close all open cursors at end of transaction.
TDS_OPT_FIPSFLAG	27	Enable FIPs flagging.

Table 54: Supported Options

Name	Value	Description
TDS_OPT_RESTREES	28	Return resolution trees.
TDS_OPT_IDENTITYON	29	Turn on explicit identity.
TDS_OPT_CURREAD	30	Set session label @@curread.
TDS_OPT_CURWRITE	31	Set session label @@curwrite.
TDS_OPT_IDENTITYOFF	32	Turn off explicit identity.
TDS_OPT_AUTHOFF	33	Turn authority off.
TDS_OPT_ANSINULL	34	Support ANSI null data.
TDS_OPT_QUOTED_IDENT	35	Quoted identifiers.
TDS_OPT_ANSIPERM	36	Check permissions on search columns for update clause.
TDS_OPT_STR_RTRUNC	37	ANSI string right trunc.
TDS_OPT_SORTMERGE	38	Set Sort-Merge for session.
TDS_OPT_JTC	39	Set JTC for session
TDS_OPT_CLIENTREALNAME	40	Set Client Real Name
TDS_OPT_CLIENTHOSTNAME	41	Set Client Host Name
TDS_OPT_CLIENTAPPLNAME	42	Set Client Application Name
TDS_OPT_IDENTITYUPD_ON	43	Turn on explicit update identity on table
TDS_OPT_IDENTITYUPD_OFF	44	Turn off explicit update identity on table
TDS_OPT_NODATA	45	Turn on/off “nodata”option
TDS_OPT_CIPHERTEXT	46	Turn on/off ciphertext (column encryption)
TDS_OPT_SHOW_FI	47	Expose Functional Indexes
TDS_OPT_HIDE_VCC	48	Hide/Show Virtual Computed Columns

- The table below summarizes the option arguments. It includes the defined argument length and defined values for the option value.

Table 55: Option Arguments

Name	Argument Length	Option Argument
TDS_OPT_DATEFIRST	1 byte	**TDS_OPT_MONDAY(1)** **TDS_OPT_TUESDAY(2)** **TDS_OPT_WEDNESDAY(3)** **TDS_OPT_THURSDAY(4)** **TDS_OPT_FRIDAY(5)** **TDS_OPT_SATURDAY(6)** **TDS_OPT_SUNDAY(7)**
TDS_OPT_TEXTSIZE	4 bytes	Size in bytes. XDR is performed on this field.
TDS_OPT_STAT_TIME	1 byte	Boolean
TDS_OPT_STAT_IO	1 byte	Boolean
TDS_OPT_ROWCOUNT	4 bytes	Number of rows. XDR is performed on this field.
TDS_OPT_NATLANG	Arg length	National language string (7 bit ASCII).
TDS_OPT_DATEFORMAT	1 byte	**TDS_OPT_FMTMDY(1)** **TDS_OPT_FMTDMY(2)** **TDS_OPT_FMTYMD(3)** **TDS_OPT_FMTYDM(4)** **TDS_OPT_FMTMYD(5)** **TDS_OPT_FMTDYM(6)**
TDS_OPT_ISOLATION	1 byte	**TDS_OPT_LEVEL0(0)** **TDS_OPT_LEVEL1(1)** **TDS_OPT_LEVEL2(2)** **TDS_OPT_LEVEL3(3)**
TDS_OPT_AUTHON	Arg length	Authorization level string (7 bit ASCII).

Table 55: Option Arguments

Name	Argument Length	Option Argument
TDS_OPT_CHARSET	Arg length	Character set string (7 bit ASCII).
TDS_OPT_SHOWPLAN	1 byte	Boolean
TDS_OPT_NOEXEC	1 byte	Boolean
TDS_OPT_ARITHIGNOREON	4 bytes	**TDS_OPT_ARITHOVERFLOW(0x01) TDS_OPT_NUMERICTRUNC(0x02)**
TDS_OPT_ARITHABORTON	4 bytes	**TDS_OPT_ARITHOVERFLOW(0x01) TDS_OPT_NUMERICTRUNC(0x02)**
TDS_OPT_PARSEONLY	1 byte	Boolean
TDS_OPT_GETDATA	1 byte	Boolean
TDS_OPT_NOCOUNT	1 byte	Boolean
TDS_OPT_FORCEPLAN	1 byte	Boolean
TDS_OPT_FORMATONLY	1 byte	Boolean
TDS_OPT_CHAINXACTS	1byte	Boolean
TDS_OPT_CURCLOSEONXACT	1 byte	Boolean
TDS_OPT_FIPSFLAG	1 byte	Boolean
TDS_OPT_RESTREES	1 byte	Boolean
TDS_OPT_IDENTITYON	Arg length	Table name string.
TDS_OPT_CURREAD	Arg length	Read label string (7 bit ASCII).
TDS_OPT_CURWRITE	Arg length	Write label string (7 bit ASCII).

Table 55: Option Arguments

Name	Argument Length	Option Argument
TDS_OPT_IDENTITYOFF	Arg length	Table name string.
TDS_OPT_AUTHOFF	Arg length	Authorization level string (7 bit ASCII).
TDS_OPT_ANSINULL	1 byte	Boolean
TDS_OPT_QUOTED_IDENT	1 byte	Boolean
TDS_OPT_ANSIPERM	1 byte	Boolean
TDS_OPT_STR_RTRUNC	1 byte	Boolean
TDS_OPT_SORTMERGE	1 byte	Boolean
TDS_OPT_JTC	1 byte	Boolean
TDS_OPT_CLIENTREALNAME	Arg length	Client name string (7 bit ASCII)
TDS_OPT_CLIENTHOSTNAME	Arg length	Client host name string (7 bit ASCII)
TDS_OPT_CLIENTAPPLNAME	Arg length	Client appl name string (7 bit ASCII)
TDS_OPT_IDENTIFYUPD_ON	Arg length	Table name string.
TDS_OPT_IDENTIFYUPD_OFF	Arg length	Table name string.
TDS_OPT_NODATA	1 byte	Boolean
TDS_OPT_CIPHERTEXT	1 byte	Boolean
TDS_OPT_SHOW_FI	1 byte	Boolean
TDS_OPT_HIDE_VCC	1 byte	Boolean

- Boolean option arguments are sent using **TDS_OPT_FALSE(0)** and **TDS_OPT_TRUE(1).**
- Older servers may expect that tablename arguments be 7-bit ASCII.
- Note that clients do not need to enforce any 7-bit ASCII restrictions.

Examples

See Also

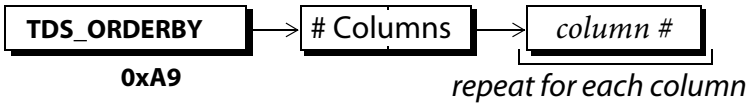
TDS_CAPABILITY

TDS_ORDERBY

Function

Describes the columns in an "order by" clause of a select.

Syntax



Arguments

TDS_ORDERBY This is the token that indicates that this is column order information.

Columns This is the number of columns in the order-by clause. This argument is a two-byte, unsigned integer.

column # This is the number of column that is in the order-by clause. The first column in the select list is number 1. For example, in the statement:

```
select empid, lastname, firstname
    from employees
order by lastname, firstname
```

the order-by columns are columns 2 and 3. This argument is a one-byte unsigned integer.

Comments

- This token is used to describe the columns in an order-by clause of a select list.
- There will always be a least one *column #* defined by a **TDS_ORDERBY** token.

Example

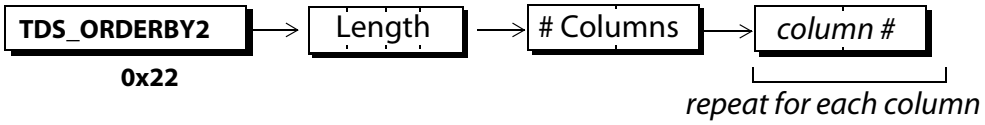
See Also

TDS_ORDERBY2

Function

Describes the columns in an "order by" clause of a select.

Syntax



Arguments

TDS_ORDERBY2 This is the token that indicates that this is column order information.

Length This 4 byte integer indicates the length of the remaining stream.

Columns This is the number of columns in the order-by clause. This argument is a two-byte, unsigned integer.

column # This is the number of column that is in the order-by clause. The first column in the select list is number 1. For example, in the statement:

```
select empid, lastname, firstname
    from employees
order by lastname, firstname
```

the order-by columns are columns 2 and 3. This argument is a two-byte unsigned integer.

Comments

- This token is identical is use to the **TDS_ORDERBY** token, but has was introduced to support > 255 columns in the result set.
- The **TDS_ORDERBY** token does not include a separate Length field since the column# information was being expressed as 1-byte integers - thus the #Columns value correctly indicates the remaining length of the token and was not repeated.

- Servers should only return this token if the **TDS_ORDERBY2** Response Capability bit is true - otherwise the client does not know this token (added in version 3.4 of this specification).

Example

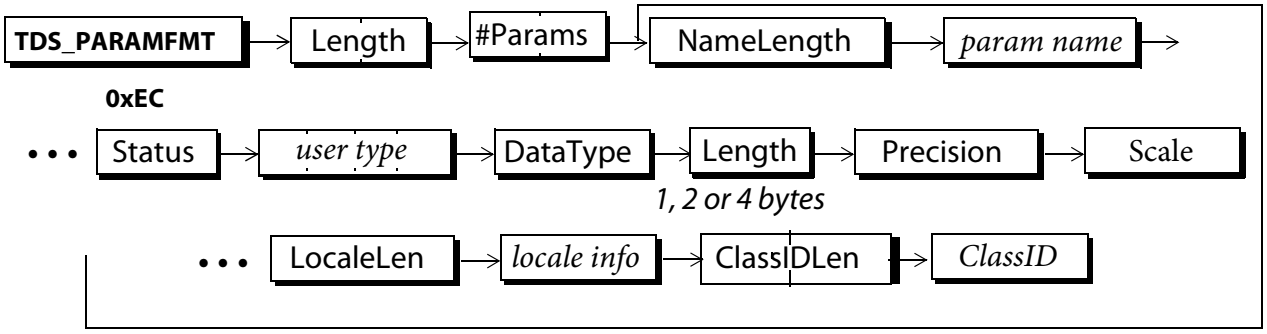
See Also

TDS_PARAMFMT

Function

The token describing the data type, length, and status of **TDS_PARAMS** data.

Syntax



repeat everything but the Length *and* #Params *for each parameter*

Arguments

TDS_PARAMFMT This is the command token used to describe parameter data.

Length This length specifies the number of bytes remaining in the datastream. It is an unsigned, two-byte integer.

#Params This argument specifies the number of parameters being described. It is an unsigned, two-byte integer.

NameLength This is the length of the parameter name which follows. Since parameter names may be NULL, NameLength may be 0. If NameLength is 0, no *param name* argument follows. NameLength is a one-byte, unsigned integer.

param name This is the name of the parameter being described. It's length is described by the preceding parameter. Parameter names are optional.

Status This field is used to describe any non-datatype characteristics of the data. For example, when remote procedure calls use **TDS_PARAMFMT** to describe their parameters' format, the **TDS_PARAM_RETURN** status marks a parameter as an *output* parameter, *i.e.*, passed by reference, in effect. Status is a one-byte, unsigned integer. The valid status bits and values are:

Table 56: Valid Status Values

Name	Value	Description
TDS_PARAM_RETURN	0x01	This is a return parameter. It is like a parameter passed by reference.
TDS_PARAM_COLUMNSTATUS	0x08	This parameter will have a columnstatus byte in its corresponding TDS_PARAM token. Note that it will be a protocol error for this bit to be set when the TDS_DATA_COLUMNSTATUS capability bit is off.
TDS_PARAM_NULLALLOWED	0x20	This parameter can be NULL

user type This is the user-defined data type of the parameter. It is a signed, four-byte integer.

DataType This is the datatype of the data. It is a one-byte unsigned integer. Datatypes which are fixed, standard length (1, 2, 4, or 8 bytes) are represented by a single datatype byte and have no Length parameter following. The *text* and *image* datatypes are not currently supported as parameter datatypes.* DataType is a one-byte, unsigned integer.

The rest of the fields in the repeating datatype descriptions are as described in the Format description for the corresponding DataType see section on page 167

*

Length This is the maximum length, in bytes, of DataType. It is a one-byte unsigned integer or a four-byte, signed integer. The size of Length depends on the DataType. If the preceding DataType is a fixed length datatype of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc*., there is no Length argument.

Precision This is the precision associated with numeric and decimal data types. It is only in the data stream if the parameter is a numeric or decimal data type.

Scale This is the scale associated with numeric and decimal data types. It is only in the data stream if the parameter is a numeric or decimal data type.

LocaleLen This is the length of the localization information, if any, which follows. It is a one-byte, unsigned integer which may be 0. If the length is 0, no localization information follows.

locale info This is the localization information for the parameter. It is character string whose length is given by LocaleLen.

ClassIDLen This is the 2-byte length of the ClassID, if any, which follows. This length field is only present if the **DataType** is **TDS_BLOB**.

ClassID This is the class identification information for BLOB types. Its length in bytes is given by the preceding ClassIDLen value. If ClassIDLen is missing because this is not a TDS_BLOB data format, or if ClassIDLen is 0, then this field is absent.

Comments

- This is the token used to provide a description of data. It is just like the old **TDS_COLNAME** and **TDS_COLFMT** tokens except that it provides a parameter name and Status for each DataType.
- This token is used to describe **TDS_PARAMS** data. Parameter data is sent with parameterized *cursor declares*, *opens*, and *updates* as well as for parameter language statements and messages.
- It is illegal to send a **TDS_PARAMFMT** data stream with zero parameters.

- Each parameter must be described in a **TDS_PARAMFMT** data stream. Only one parameter can be sent for each **TDS_PARAMFMT** description. For example, it is illegal to send a **TDS_PARAMFMT** that contains a description of two parameters, and then send multiple **TDS_PARAMS** data streams, each with two parameters. Each parameter sent from a client or server in a **TDS_PARAMS** data stream must be preceded by a description in a **TDS_PARAMFMT** data stream.
- The **TDS_PARAMFMT** token has exactly the same format as the **TDS_ROWFMT** token. Two tokens are used to provide state information. The formats will remain the same so that client and server code used to encode and decode the tokens can be the same.
- The **TDS_PARAMFMT/PARAMS** tokens are used to send return parameters to a client if the **TDS_RES_NOPARAM** capability bit is false.

Examples

See Also

TDS Datatypes, **TDS_PARAMFMT2**, **TDS_ROW, TDS_ROWFMT, TDS_ROWFMT2, TDS_PARAMS**

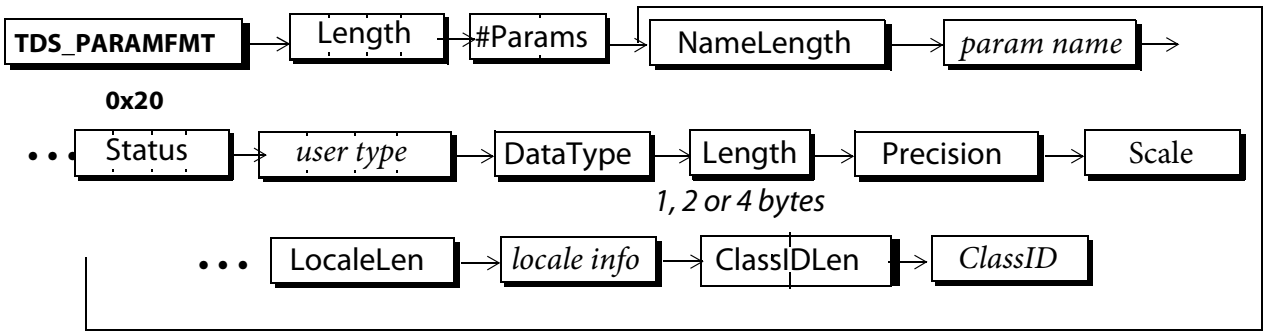
TDS_PARAMFMT2

Function

The token describing the data type, length, and status of **TDS_PARAMS** data.

It is idential to the **TDS_PARAMFMT** token except that the length field is 4 bytes long (to accomodate a greater number of parameters in/out) and the Status field has been expanded to 4 bytes (status bits were nearly used up).

Syntax



repeat everything but the Length *and* #Params *for each parameter*

Arguments

TDS_PARAMFMT2 This is the command token used to describe parameter data.

Length This length specifies the number of bytes remaining in the datastream. It is an unsigned, four-byte integer.

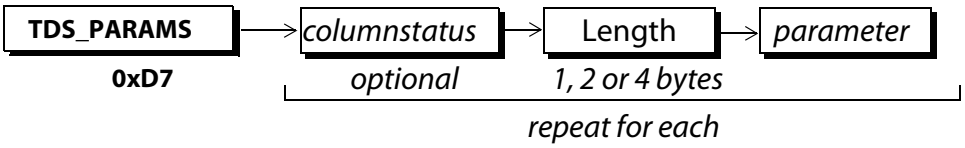
Refer to the **TDS_PARAMFMT** token for further documentation of fields.

TDS_PARAMS

Function

The token for parameter data.

Syntax



Arguments

TDS_PARAMS This is the command token to send parameter data.

columnstatus This byte is present only when the following two conditions have been met: The first is that the client and server have negotiated through the TDS_DATA_COLUMNSTATUS and TDS_DATA_NOCOLUMNSTATUS capability bits that the client can send columnstatus bytes to the server. And secondly, the **PARAMFMT** token for this parameter has the TDS_PARAM_COLUMNSTATUS status bit set. For more information on the columnstatus byte, please see the chapter on datatypes.

Length This is the *actual*, as opposed to *maximum*, data length, in bytes, of the *parameter* data If the parameter data is a fixed length data type of standard length, e.g., ints, floats, datetimes, then there is no Length argument. Length is either a one-byte, unsigned integer, an unsigned, two-byte integer, or a signed, four-byte integer. The size of Length depends on the data types of the data.

parameter This is the actual data for the parameter. Its length, if variable, is indicated by the preceding Length argument. It is in the format specified by the client in the login request. The server always does any translation so that the client receives data in its native format.*

* See previous note.

Comments

- This is the token that contains the parameter data described by a preceding **TDS_PARAMFMT** data stream.
- A **TDS_PARAMS** token consists of Length and *parameter* pairs, one for each parameter described by a preceding **TDS_PARAMFMT** token. The Length component doesn't appear if the data is a fixed data type of standard length, *e.g.*, **INT2**, **MONEY**, **DATETIME**, *etc*. If the data type allows nulls then the data will always be preceded by a Length argument. Fixed length datatypes that are not of a standard length, e.g., **CHAR** and **BINARY** are also preceded by a Length.
- The **TDS_PARAMS** token has exactly the same format as the **TDS_ROW** and **TDS_KEY** tokens. Three tokens are used for the same data stream to provide data stream state information. The formats will remain the same so that client and server code used to encode and decode the data streams can be the same.
- The **TDS_PARAMS** token may appear repeatedly after a **TDS_PARAMFMT** token. A **TDS_DONE** must be sent after all the **TDS_PARAM** tokens for a particular **TDS_PARAMFMT**.

Examples

See Also

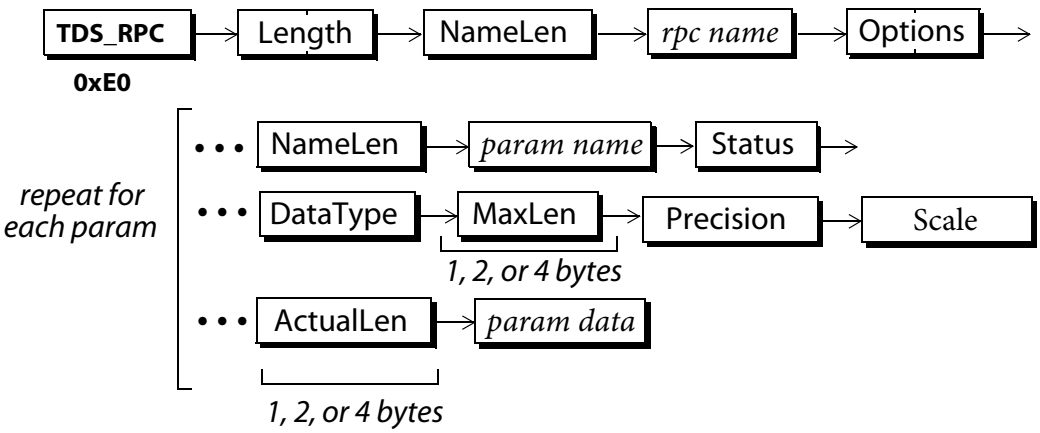
TDS_PARAMFMT, TDS_ROW, TDS_KEY

TDS_RPC

Function

Describes the data stream which contains a remote procedure call request. This token is obsolete.

Syntax



Arguments

TDS_RPC This is the command token to send an RPC request.

Length This is the length, in bytes, of the remaining **TDS_RPC** data stream. It is a two-byte, unsigned integer.

NameLen This is length, in bytes, of the RPC name. It is a one-byte, unsigned integer.

rpc name This is the name of the RPC. Its length, in bytes, is given by the preceding argument.

Options This is a bit mask which contains options related to the RPC. The mask is a two-byte, unsigned integer. The defined options are:

Table 57: RPC Option Values

Name	Value	Description
TDS_RPC_UNUSED	0x0000	Option argument is not used.
TDS_RPC_RECOMPILE	0x0001	Recompile the RPC before execution.

NameLen This the length, in bytes, of the parameter name. It may be 0. The argument is a one-byte, unsigned integer.

param name This the parameter name. Its length, in bytes, is given by the preceding argument. If NameLen is 0, the *param name* argument will not be included in the data stream.

Status This is a one-byte, unsigned integer which is used as a bit field. It indicates any special status for the particular parameter being described. The possible Status values are:

Table 58: Status Field Values

Name	Value	Description
TDS_RPC_STATUS_UNUSED	0x00	The statuss argument is not used.
TDS_RPC_OUTPUT	0x01	This value of this parameter will be returned to the client. It may contain an original value, but it may be changed. Return parameters are returned using the **TDS_RETURNVALUE** token.
TDS_RPC_NODEF	0x02	This indicates that there is no default value for this parameter. The value of this parameter is undefined. This bit is only valid with **TDS_RPC_OUTPUT**.

DataType This is the data type of the parameter and is a one-byte unsigned integer. Datatypes which are fixed, standard length (1, 2, 4, or 8 bytes) are represented by a single data type byte and have no MaxLen or ActualLen parameters following. Variable data types are followed by a length which gives the maximum length, in bytes, for the data type.

MaxLen This is the maximum length, in bytes, of the preceding DataType. The size of MaxLen depends on the data type. If the preceding DataType is a fixed length data type of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc.*, there is no MaxLen argument.

Precision This is the precision associated with numeric and decimal data types. It is only in the data stream if the parameter is a numeric or decimal data type.

Scale This is the scale associated with numeric and decimal data types. It is only in the data stream if the parameter is a numeric or decimal data type.

ActualLen This is the actual length, in bytes, of the following *param data* field. The size of ActualLen depends on the data type.* If the preceding DataType is a fixed length data type of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc.*, there is no ActualLen argument.

param data This is the actual parameter data. Its length, if variable, is indicated by the preceding ActualLen argument. It is in the native format of the client machine. For example, if the client is running on a SUN and the server on a VAX, the representation of the INT4 data type has different byte ordering. The server always does any byte swapping so that the client receives the data in native format.

Comments

- This token is used by a client to make an RPC request to a server.
- Currently, only one **TDS_RPC** token per request is allowed.
- RPC return parameters for the **TDS_RPC** token are returned using the **TDS_RETURNVALUE** token.

*. See previous note.

- Note that the total length of the RPC information is limited to 64k-1. Because of this, this token has been replaced by the **TDS_DBRPC** token. It should not be used in any new products.
- The **TDS_RPC** token should be used by clients only if the **TDS_REQ_PARAM** capability bit is false.

Examples

See Also

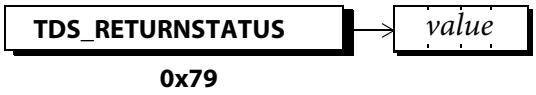
TDS_RETURNVALUE, TDS_DBRPC.

TDS_RETURNSTATUS

Function

Describes the token which is used to return status information to a client.

Syntax



Arguments

TDS_RETURNSTATUS This is the token used to return status information.

value This is the value of the return status. It is a four-byte, signed integer. Note that the value may not be *null*; that is, it must be present in the datastream.

Comments

- This is the token that is used to return a status code to a client.
- When a remote procedure call is executed on a server, a return status value may be returned.
- Only one **TDS_RETURNSTATUS** per RPC is allowed.

Examples

See Also

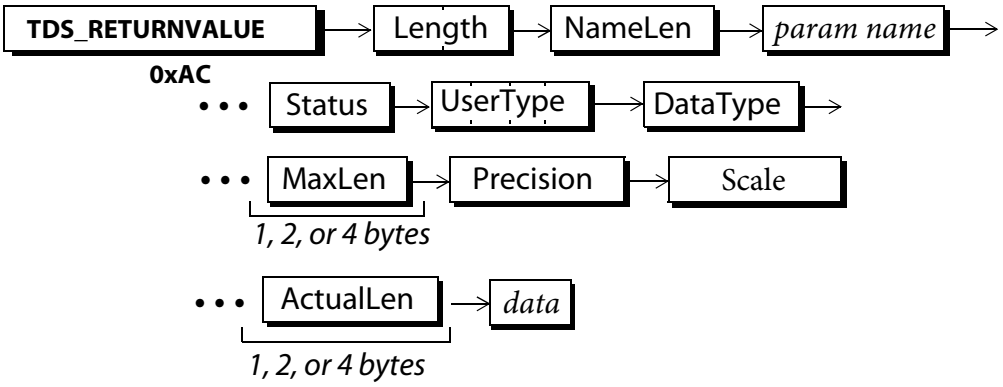
TDS_RETURNVALUE

TDS_RETURNVALUE

Function

Return parameter information to a client.

Syntax



Arguments

TDS_RETURNVALUE This is the token that indicates that is used to return parameter information to a client.

Length This is the length, in bytes, of the remaining **TDS_RETURNVALUE** data stream. It is a two-byte, unsigned integer.

NameLen This is length, in bytes, of the name, if any, of the return parameter. It is a one-byte, unsigned integer.

param name This is the name of the return parameter. Its length, in bytes, is given by the preceding argument. If NameLen is 0, then the *param name* field is omitted from the data stream.

Status This is a one-byte, unsigned integer which is used as a bit field. It indicates any special status for the particular parameter being described. The possible Status values are:

Table 59: Status Values

Name	Value	Description
TDS_PARAM_UNUSED	0x00	The status field is not used.
TDS_PARAM_RETURN	0x01	This indicates that the return value was originally sent to the server as an output parameter in an RPC.
TDS_FUNC_RET	0x02	This indicates that the return value is non-integral. When set, the parameter name length should be 0 and the subsequent data information should be treated as the RPC return value.

UserType This is the user-defined data type, if any, for the returned value. It is a signed, four-byte integer. If there is no UserType for return value UserType will be 0.

DataType This is the datatype of the return value and is a one-byte unsigned integer. Datatypes which are fixed, standard length (1, 2, 4, or 8 bytes) are represented by a single datatype byte and have no Maxlen or Actual Len arguments following. Variable datatypes are followed by a length which gives the maximum length, in bytes, for the datatype.

MaxLen This is the maximum length, in bytes, of the preceding DataType. The size of MaxLen depends on the data type. If the preceding DataType is a fixed length data type of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc.*, there is no MaxLen argument in this data stream.

Precision This is the precision associated with numeric and decimal data types. It is only in the data stream if the parameter is a numeric or decimal data type.

Scale This is the scale associated with numeric and decimal data types. It is only in the data stream if the parameter is a numeric or decimal data type.

ActualLen This is the actual length, in bytes, of the following *param data* argument. The size of ActualLen depends on the data type. If the preceding DataType is a fixed length datatype of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc*., there is no ActualLen argument in this data stream.

data This is the actual data for the parameter. Its length, if variable, is indicated by the preceding ActualLen argument. It is in the native format of the client machine. For example, if the client is running on a SUN and the server on a VAX, the representation of the INT4 data type has different byte ordering. The server always does any byte swapping so that the client receives the data in native format.

Comments

- This is the token that is used by a server to return a value to the client.
- When remote procedure calls (stored procedures) are executed, the parameters may be designated as *output* or *return* parameters. This data stream is used to return a description of the return parameter and the value of the return parameter to the client application.
- There may be multiple return values per RPC. There is a separate **TDS_RETURNVALUE** data stream for each parameter returned.
- Return parameters are sent in the order in which they were defined in the procedure.
- The MaxLen and ActualLen components don't appear if the return value is a fixed data type of standard length, *e.g.,* INT2, MONEY, DATETIME. Parameters that are fixed length data types that are not of a standard length, *e.g.*, CHAR and BINARY include MaxLen and ActualLen.
- The **TDS_RETURNVALUE** data stream limits the total length of return parameters to 64K-1. Because of this restriction this token has been replaced with the **TDS_PARAMFMT/PARAMS** tokens to return parameters to a client.
- The **TDS_RETURNVALUE** token should only be used to return parameters to a client if the **TDS_RES_NOPARAM** capability bit is true.

- This token is obsolete for most uses and should only be used when an RPC return value is not an integer.

Examples

See Also

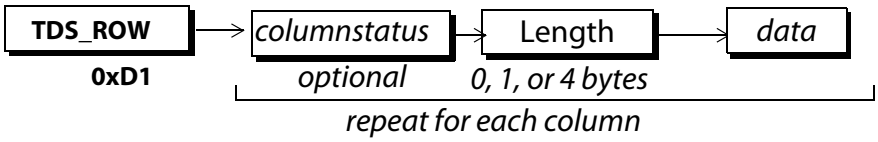
TDS_RPC

TDS_ROW

Function

A row of data.

Syntax



Arguments

TDS_ROW This is the token that is used to send row data.

columnstatus This byte is present only when the following two conditions have been met: The first is that the client and server have negotiated through the TDS_DATA_COLUMNSTATUS and TDS_DATA_NOCOLUMNSTATUS capability bits that the server can send columnstatus bytes to the client. And secondly, the **ROWFMT** token for this column has the TDS_ROW_COLUMNSTATUS status bit set. For more information on the columnstatus byte, please see the chapter on datatypes.

Length This is the *actual*, as opposed to *maximum*, data length, in bytes, of the following *data*. If the following column data is a fixed length data type of standard length, e.g., ints, floats, datetimes, then there is no Length argument. Length is either a one-byte, unsigned integer, an unsigned, two-byte integer, or a signed, four-byte integer. The size of Length depends on the data type of the data.

data This is the actual data for the column data. Its length, if variable, is indicated by the preceding Length argument. It is in the format specified in the login record of the client request. The server always does any translation so that the client receives data in its expected format.

Comments

- This is the token that contains the data for one row.
- A **TDS_ROWFMT** token was used to describe the data sent in the **TDS_ROW** token.
- A **TDS_ROW** token consists of Length and data pairs, one for each column described by a preceding **TDS_ROWFMT** token. The Length argument doesn't appear if the data is a fixed length data type of standard length, e.g., INT2, MONEY, DATETIME, etc. If the data type allows nulls then the data will always be preceded by a Length argument. Fixed length data types that are not of a standard length, e.g., CHAR and BINARY are also preceded by a Length.
- The **TDS_ROW** token has exactly the same format as the **TDS_PARAMS** and **TDS_KEY** tokens.
- A separate **TDS_ROW** token is used for each row in a result set.

Examples

See Also

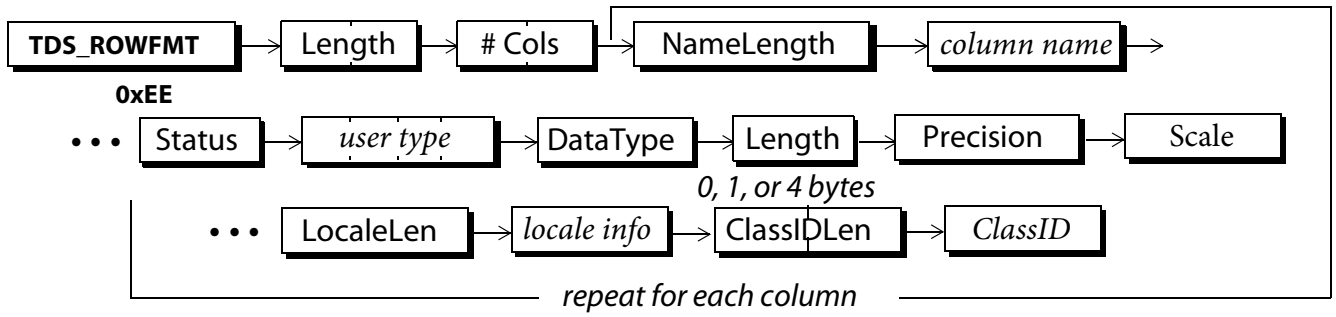
TDS_ROWFMT, TDS_ROWFMT2

TDS_ROWFMT

Function

The token for describing the data type, length, and status of row data.

Syntax



Arguments

TDS_ROWFMT This is the token used to send a description of row data.

Length This length specifies the number of bytes remaining in the data stream. It is an unsigned, two-byte integer.

#Cols This argument contains the number of columns which are being described. It is an unsigned, two-byte integer.

NameLength This is the length of the column name which follows. Since column names may be NULL, ColLength may be 0. If ColLength is 0, no *col name* argument follows. ColLength is a one-byte unsigned integer.

col name This is the name of the column being described. It's length is described by the preceding parameter. Column names are optional.

Status This field is used to describe any non-datatype characteristics for the data. A column may have more than one status bit set. Status is an unsigned, one-byte integer. The valid values are:

Table 60: Valid Status Values

Name	Value	Description
TDS_ROW_HIDDEN	0x01	This is a hidden column. It was not listed in the target list of the select statement. Hidden fields are often used to pass key information back to a client. For example: select a, b from table T where columns b and c are the key columns. Columns a, b, and c may be returned and c would have a status of **TDS_ROW_HIDDEN|TDS_ROW_KEY.**
TDS_ROW_KEY	0x02	This indicates that this column is a key.
TDS_ROW_VERSION	0x04	This column is part of the version key for a row. It is used when updating rows through cursors.
TDS_ROW_COLUMNSTATUS	0x08	All rows in this column will contain the columnstatus byte. Note that it will be a protocol error to set this bit if the TDS_DATA_COLUMNSTATUS capability bit is off.
TDS_ROW_UPDATABLE	0x10	This column is updatable. It is used with cursors.
TDS_ROW_NULLALLOWED	0x20	This column allows nulls.
TDS_ROW_IDENTITY	0x40	This column is an identity column.
TDS_ROW_PADCHAR	0x80	This column has been padded with blank characters.

user type This is the user-defined data type of the data. It is a signed, four-byte integer.

DataType This is the data type of the data and is a one-byte unsigned integer. Datatypes which are fixed, standard length (1, 2, 4, or 8 bytes) are represented by a single data type byte and have no Length argument. Variable data types are followed by a length which gives the maximum length, in bytes, for the datatype.

The rest of the fields in the repeating datatype descriptions are as described in the Format description for the corresponding DataType see section on page 167

Length This is the maximum length, in bytes, of DataType. The size of Length depends on the data type. If the preceding DataType is a fixed length data type of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc.*, there is no Length argument. It the preceding type is *text* or *image*, then the format is a four-byte length argument, followed by a two-byte object name length, and finally the object name.

Precision This is the precision associated with numeric and decimal data types. It is only in the data stream if the column is a numeric or decimal data type.

Scale This is the scale associated with numeric and decimal data types. It is only in the data stream if the column is a numeric or decimal data type.

LocaleLen This is the length of the localization information which follows. It is a one-byte, unsigned integer which may be 0. If the length is 0, no localization information follows.

locale info This is the localization information for the column. It is character string whose length is given by LocaleLen.

ClassIDLen This is the 2-byte length of the ClassID, if any, which follows. This length field is only present if the **DataType** is **TDS_BLOB**.

ClassID This is the class identification information for BLOB types. Its length in bytes is given by the preceding ClassIDLen value. If ClassIDLen is missing because this is not a TDS_BLOB data format, or if ClassIDLen is 0, then this field is absent.

Comments

- This is the token used to provide a description of data. It is just like the old **TDS_COLNAME** and **TDS_COLFMT** tokens except that it provides the column name and Status argument for each DataType.
- This data stream is used to describe **TDS_ROW** data sent in response to a non-cursor or cursor **select**.
- The information in **TDS_ROWFMT** is used to decode the **TDS_ROW** token.
- The **TDS_ROWFMT** token has exactly the same format as the **TDS_PARAMFMT** token. Two tokens are used for the same data stream in order to provide state information. The formats will remain the same so that client and server code used to encode and decode the tokens can be the same.
- The **TDS_COLNAME** and **TDS_COLFMT** tokens are no longer supported with TDS 5.0.

Examples

See Also

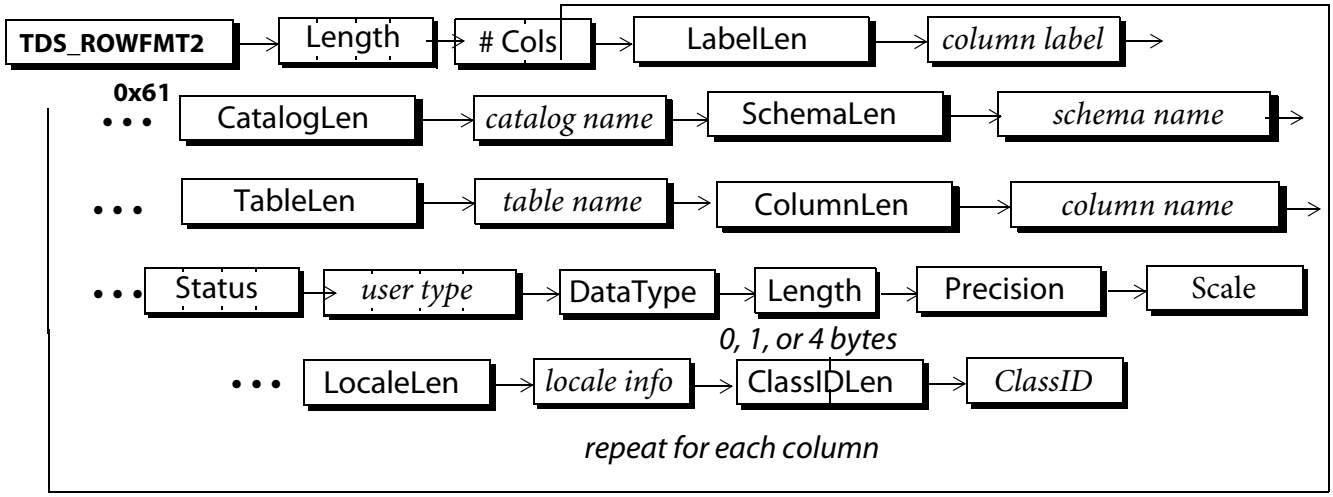
TDS Datatypes, **TDS_ROWFMT2**, **TDS_ROW, TDS_PARAMFMT**

TDS_ROWFMT2

Function

The token for describing the data type, length, and status of row data.

Syntax



Arguments

TDS_ROWFMT2 This is the token used to send a description of row data.

Length This length specifies the number of bytes remaining in the data stream. It is an unsigned, four-byte integer.

#Cols This argument contains the number of columns which are being described. It is an unsigned, two-byte integer.

To describe the next 10 arguments we will look at an example. Suppose that from the pubs2 database one issued the following query:

```
SELECT au_fname AS "FIRST NAME" FROM dbo.authors
```

LabelLen This is the length of the column label which follows. Since column labels may be NULL, LabelLength may be 0. If LabelLength is 0, no *column label* argument follows. LabelLength is a one-byte unsigned integer.

column label This is the name of the column being described. It's length is described by the preceding parameter. Column labels are optional. In the example above this value would be "FIRST NAME", and the LabelLen value would be 10.

CatalogLen This is the length of the catalog name which follows. If CatalogLength is 0, the catalog name field will be absent. It is an unsigned one-byte unsigned integer.

catalog name This is the name of the catalog (database) that the table with this column is in. In the example above this value would be "pubs2" and the CatalogLength value would be 5.

SchemaLen This is the length of the schema name which follows. If it is 0, no *schema name* argument follows. It is a one-byte unsigned integer.

schema name This is the name of the schema (owner) of the table containing the column being described. In the example above this value would be "dbo", and the SchemaLength value would be 3.

TableLen This is the length of the table name which follows. It is a one-byte unsigned integer.

table name This is the name of the table containing the column being described. In the example above this value would be "authors", and the TableLength value would be 7.

ColumnLen This is the length of the column name which follows. It is a one-byte unsigned integer.

column name This is the actual name of the column being described. In the example above this value would be "au_fname", and the ColumnLen value would be 8.

Status This field is used to describe any non-datatype characteristics for the data. A column may have more than one status bit set. Status is an unsigned, four-byte bit field. The valid values are:

Table 61: Valid Status Values

Name	Value	Description
TDS_ROW_HIDDEN	0x01	This is a hidden column. It was not listed in the target list of the select statement. Hidden fields are often used to pass key information back to a client. For example: select a, b from table T where columns b and c are the key columns. Columns a, b, and c may be returned and c would have a status of **TDS_ROW_HIDDEN|TDS_ROW_KEY.**
TDS_ROW_KEY	0x02	This indicates that this column is a key.
TDS_ROW_VERSION	0x04	This column is part of the version key for a row. It is used when updating rows through cursors.
TDS_ROW_COLUMNSTATUS	0x08	All rows in this column will contain the columnstatus byte. Note that it will be a protocol error to set this bit if the TDS_DATA_COLUMNSTATUS capability bit is off.
TDS_ROW_UPDATABLE	0x10	This column is updatable. It is used with cursors.
TDS_ROW_NULLALLOWED	0x20	This column allows nulls.
TDS_ROW_IDENTITY	0x40	This column is an identity column.
TDS_ROW_PADCHAR	0x80	This column has been padded with blank characters.

user type This is the user-defined data type of the data. It is a signed, four-byte integer.

DataType This is the data type of the data and is a one-byte unsigned integer. Datatypes which are fixed, standard length (1, 2, 4, or 8 bytes) are represented by a single data type byte and have no Length argument. Variable data types are followed by a length which gives the maximum length, in bytes, for the datatype.

The rest of the fields in the repeating datatype descriptions are as described in the Format description for the corresponding DataType see section on page 167

Length This is the maximum length, in bytes, of DataType. The size of Length depends on the data type. If the preceding DataType is a fixed length data type of standard length, *e.g.*, *int1*, *int2*, *datetime*, *etc.*, there is no Length argument. It the preceding type is *text* or *image*, then the format is a four-byte length argument, followed by a two-byte object name length, and finally the object name.

Precision This is the precision associated with numeric and decimal data types. It is only in the data stream if the column is a numeric or decimal data type.

Scale This is the scale associated with numeric and decimal data types. It is only in the data stream if the column is a numeric or decimal data type.

LocaleLen This is the length of the localization information which follows. It is a one-byte, unsigned integer which may be 0. If the length is 0, no localization information follows.

locale info This is the localization information for the column. It is character string whose length is given by LocaleLen.

ClassIDLen This is the 2-byte length of the ClassID, if any, which follows. This length field is only present if the **DataType** is **TDS_BLOB**.

ClassID This is the class identification information for BLOB types. Its length in bytes is given by the preceding ClassIDLen value. If ClassIDLen is missing because this is not a TDS_BLOB data format, or if ClassIDLen is 0, then this field is absent.

Comments

It is much like the **TDS_ROWFMT** token, with the following changes

- The Length field is 4 bytes long to allow for wider tables
- The Status byte has been expanded to 4 bytes (most of the original 8 bits had been used up).
- Additional namelen/name pairs have been added to complete the description of each column. The data contained in **TDS_ROWFMT** contains only a single "column name" field. That value would be set to the "alias" from the select query (select *column* AS *alias* ...) if the AS clause or T/SQL equivalent were used. If there was no alias then the value would be the actual name of the column in the table being selected. If the column is the result of an expression and there is no alias, then the value was returned as NULL. With TDS_ROWFMT2 this information has been enhanced as {catalog, schema, table, column-name, column-label}. Addition of this information makes it possible to implement JDBC and ODBC standards compliant client software.

♦ *The "column name" field from* **TDS_ROWFMT** *has changed names to "column label". The new item called "column name" in* **TDS_ROWFMT2** *corresponds to the underlying column name if there is one. Any of these 5 fields may be left empty (but every attempt should be made to fill them in correctly for the sake of standards compliance).*

- This data stream is used to describe **TDS_ROW** data sent in response to a non-cursor or cursor **select**.
- The information in **TDS_ROWFMT2** is used to decode the **TDS_ROW** token.

Examples

See Also

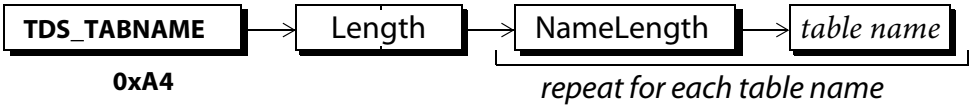
TDS Datatypes, **TDS_ROW, TDS_ROWFMT, TDS_PARAMFMT, TDS_PARAMFMT2**

TDS_TABNAME

Function

The datastream for naming tables referenced in a result set.

Syntax



Arguments

TDS_TABNAME This is the token used to send table names.

Length This is the total length of the remaining **TDS_TABNAME** data stream. It is a two-byte, unsigned integer.

NameLength This is the length, in bytes, of the name of a table.

table name This is the table name. It's length is given by the preceding argument.

Comments

- This is the token sent by a server to the client when it wishes to list the tables that are referenced in a result set. The name of each table which has columns in the select list will be returned using this token.
- Views names are never returned, only the underlying table names.
- This token is always preceded by a **TDS_ROWFMT** token. It is always followed by a **TDS_COLINFO** token.
- This token is only used for browse mode.

Examples

See Also

TDS_ROWFMT, TDS_COLINFO